z/OS
2.5

MVS Programming: Sysplex Services Guide



SA23-1400-50

Note

Before using this information and the product it supports, read the information in "Notices" on page 773.

This edition applies to Version 2 Release 5 of z/OS® (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2023-04-24

© Copyright International Business Machines Corporation 1994, 2023.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures ..... **xvii**

Tables ..... **xxi**

About this information ..... **xxv**
Who should use this information xxv
How this information is organized xxv
Where to find more information xxv

How to send your comments to IBM ..... **xxvii**
If you have a technical problem xxvii

Summary of changes ..... **xxix**
Summary of changes for z/OS MVS Programming: Sysplex Services Guide for Version 2 Release 5 (V2R5) xxix
Summary of changes for z/OS Version 2 Release 4 xxx
Summary of changes for z/OS Version 2 Release 3 xxx

Part 1. Introduction to Sysplex Services **1**

Chapter 1. Introduction to Sysplex Services 3
Sysplex Services for Communication 3
Sysplex Services for Recovery (Automatic Restart Management) 3
Sysplex Services for Data Sharing 3

Part 2. Sysplex Services for Communication (XCF) **5**

Chapter 2. Using the Cross-System Coupling Facility (XCF) 7
XCF Concepts 7
XCF Communication Services 8
Group Services 9
Signaling Services 10
Client/Server Services 10
Status Monitoring Services 11
Member Attributes 11
Permanent Status Recording 12
The Five Member States 12
The User State Field 15
Member Name and Group Name 15
The Member Token 15
The User Routines 16
Member Association 17
XCF-Managed Response Collection 18
Providing Information to Your System Programmer 18
Summary of XCF Communication Macros 18
Defining Members to XCF 22
Changing the Value in a User State Field 25
Using the IXCSETUS Macro 25
Using IXCSETUS for Active Members on Different Systems 26

Using Signaling Services to Send and Receive Messages ... 26
What Is a Message? ... 26
Using the IXCMSGOX Signaling Service ... 27
Using the IXCMSGOX Macro ... 29
Using the IXCMSGIX Macro ... 43
Using the IXCMSGC Macro ... 46
Handling Member Termination ... 52
Coding a Message User Routine ... 52
Coding a Message Notify User Routine ... 59
Requesting XCF Status Monitoring ... 63
Using a Status User Routine ... 63
Updating the Status Field ... 69
Setting and Changing a Status-Checking Interval ... 69
Coding a Status User Routine ... 70
Notifying Members of Changes ... 77
How XCF Works Together with the Group User Routine ... 77
Events that Cause XCF to Schedule a Group User Routine ... 77
Skipping of Events ... 81
Coding a Group User Routine ... 85
Obtaining XCF Information ... 97
Obtaining Sysplex, Group, and Member Information ... 98
Using the IXCQUERY Macro ... 98
Obtaining Tuning and Capacity Planning Information ... 111
Disassociating Members from XCF ... 116
Using the IXCQUIES Macro ... 117
Using the IXCLEAVE Macro ... 117
Using the IXCDELET Macro ... 117
Using the IXCTERM Macro ... 117
Member Termination ... 118
Example of Designing and Implementing a Multisystem Application ... 120
How Does PHONBOOK Work? ... 121
How Does a Member Update its Status Field? ... 122
What Data Structures Does PHONBOOK Use? ... 122
What Do the User Routines Do? ... 123
How Does the Installation Set Up PHONBOOK on Each System? ... 127
How Does PHONBOOK Handle Different Types of Work Requests? ... 128
What Happens When all Processing is Complete? ... 131
What is Another Method for Designating Members? ... 131

Chapter 3. Using XCF for client/server communication ... 133
Overview of XCF client/server processing ... 133
Defining and starting a server ... 135
Overview of IXCSRVR ... 135
Using the IXCSEND macro ... 144
Overview of IXCSEND ... 144
Content of the message ... 145
Identity of the sender ... 146
Time out values ... 146
Receive bind ... 147
Sending a request to a server ... 147
Sending a response to a client ... 150
Summary of IXCSEND functions ... 151
Using the IXCRECV macro ... 153
Overview of IXCRECV ... 153
Requesting responses and IXCRECV ... 154
Receiving responses and IXCRECV ... 154
Processing responses ... 154
Answer area ... 154

Data area....155
Storage considerations for answer and data areas....156
Storage keys....156
Target/response states....156
Message completion and time out values....157
Blocking receives....157
Message completion and IXCMSGC....158
Obtaining message status....158
Obtaining detailed response status....158
Delivered response....158
Competing receives....159
Combining IXCRECV invocations....159
Response codes and the target receiver....159
Summary of IXCRECV function....160
Using the XCF Server....160
Using the IXCREQ macro....161
Sending a request to the XCF Server....161
Receiving responses from the XCF Server....162
XCF Server SERVERINFO requests....162
Example of a client/server application....162
Client/server compatibility....164
Overview of client/server compatibility processing....164
Setting up client/server compatibility....164
Using the CLIENTLEVEL and CRITERIA keywords on IXCSEND....165
Server upgrade....165
Server and client recovery considerations....166
Server exit failures....166
Server failures....167
Receiver failures....167
Coding a server exit routine....167
Environment....167
Entry Specifications....168
Return Specifications....169
User Routine Processing....169
Programming Considerations....171
User Routine Recovery....171
Work area considerations....171

Part 3. Sysplex Services for Recovery (Automatic Restart Management)....173

Chapter 4. Using the Automatic Restart Management Function of XCF....175
Understanding How Your Installation Uses Automatic Restart Management....175
Requesting Automatic Restart Management Services....176
Understanding How MVS Handles Restart Processing....177
Designing Your Application to Use Automatic Restart Management Services....178
Registering as an Element (IXCARM REQUEST=REGISTER)....179
Indicating Readiness for Work (IXCARM REQUEST=READY)....181
Deregistering the Element (IXCARM REQUEST=DEREGISTER)....181
Waiting for Other Work to be Restarted (IXCARM REQUEST=WAITPRED)....181
Associating One Element with Another (IXCARM REQUEST=ASSOCIATE)....182
Designing an Event Exit....182
Gathering Statistical Data....185
Monitoring Restarts through the ENFREQ Macro....185
Displaying Information about Automatic Restart Management....185
IBM-Supplied Automatic Restart Manager Policy Levels....187
Example of Using the IXCARM Macro....187

Part 4. Sysplex Services for Data Sharing (XES)..... 195

Chapter 5. Introduction to Sysplex Services for Data Sharing (XES)..... 197
Data Sharing Concepts and Terminology..... 198
The Coupling Facility from the Point of View of the Programmer..... 199
Types of Coupling Facility Structures..... 199
Using Sysplex Services for Data Sharing..... 200
Guide to Sysplex Services Topics..... 202

Chapter 6. Connection Services..... 203
Overview of Connection Services..... 203
Authorizing Coupling Facility Requests..... 204
Structure Concepts..... 205
Defining the Structure Attributes..... 205
Identifying Connection States..... 205
Understanding Connection Persistence and Structure Persistence..... 207
Allocating a Structure in a Coupling Facility..... 208
Specifying the Required Coupling Facility Attributes..... 209
Selecting a Coupling Facility for Structure Allocation..... 215
Coupling Facility Considerations When Allocating a Structure..... 218
Coupling Facility Resource Allocation “Rules”..... 219
Successful Completion of Structure Allocation..... 221
Connecting to a Coupling Facility Structure..... 222
Overview of Connect Processing..... 222
Specifying structure attributes for all structures..... 229
Connecting to a Cache Structure..... 230
Connecting to a List Structure..... 234
Connecting to a Lock Structure..... 238
Using the IXLCSP Service to Determine Structure Size or Attributes..... 242
Defining the Required Exit Routines..... 243
Determining the Success of a Connection..... 244
Receiving Answer Area Information..... 245
Handling Failed Attempts to Connect to a Structure..... 249
Understanding the Structure Version Numbers..... 252
Reconnecting to a Structure..... 253
Connecting to a Structure During User-Managed Rebuild..... 255
Connecting to a Structure During User-Managed Duplexing Rebuild..... 255
Connecting to a structure during a system-managed process..... 256
Connecting to a Structure That Is Being Altered..... 256
Connecting to a Structure when a Synchronization Point Is Set..... 257
Dumping Considerations..... 257
Handling a Connection's Abnormal Termination..... 257
Deleting Persistent Structures..... 262
Deleting Failed-Persistent Connections..... 262
Using IXLFORCE or the SETXCF FORCE Command..... 263
Structure Rebuild Processing..... 264
Initiating a structure rebuild process..... 267
Overview of user-managed rebuild processing..... 268
User-Managed Rebuild Events and the Event Exit..... 272
Starting the User-Managed Rebuild Process..... 274
Connecting to the New Structure..... 276
Working with structures in the Duplex Established Phase..... 284
Working with structures in the Async Duplex Established phase..... 285
Stopping a Duplexing Rebuild to Forward Complete..... 286
Completing the User-Managed Rebuild Process..... 286
Stopping a User-Managed Rebuild Process..... 288

Handling New Connections During a User-Managed Rebuild Process..... 289
Handling Disconnections During Rebuilding.....291
Handling Failed Connections During Rebuilding.....291
Handling Rebuild Connect Failures..... 292
Handling Failures during Duplexing Rebuild.....292
MVS-Initiated Rebuild Processing.....294
Dumping Considerations..... 297
Summary of User-Managed Structure Rebuild Processing..... 297
User-Managed Rebuild Timeline.....298
Summary of User-Managed Duplexing Rebuild Process..... 298
User-Managed Duplexing Rebuild Timeline.....300
Summary of Rebuild and Duplexing Rebuild Stop Processing..... 300
Overview of System-Managed Rebuild Processing..... 301
Completing or Continuing the System-Managed Process.....312
Working with Structures in a Duplex Established Phase.....313
Working with structures in the Async Duplex Established phase..... 314
Stopping the System-Managed Rebuild Process.....315
Stopping a System-Managed Duplexing Rebuild.....315
Handling Connection Changes During System-Managed Processing..... 315
Handling Loss of Connectivity during System-Managed Processing..... 318
Handling structure failure during system-managed processes..... 320
Dumping Considerations during System-Managed Processes..... 321
Some comparisons between user-managed and system-managed duplexing rebuild..... 321
Summary of System-Managed Rebuild Processing.....322
System-Managed Rebuild Timeline..... 323
Summary of System-Managed Duplexing Rebuild Processing..... 323
System-Managed Duplexing Rebuild Timeline.....324
Altering a Coupling Facility Structure..... 324
Overview of Structure Alter Processing..... 324
Starting the Structure Alter Process..... 328
Completing the Structure Alter Process..... 329
Handling New Connections during Alter Processing..... 333
Responding to Connection Events.....333
Using IXLYEEPL to Provide a Response..... 334
Using IXLYEEPL and the IXLEERSP macro.....334
Events Reported to the Event Exit.....335
Using IXLUSYNC to Coordinate Processing of Events.....344
Overview of IXLUSYNC Processing..... 344
Handling Connection Failures during Synchronization.....345
Disconnecting from a Coupling Facility Structure..... 346
Overview of Disconnect Processing..... 346
Coding the IXLDISC Macro..... 346
Disconnect Events and the Event Exit.....346
Persistence Considerations.....348
Dumping Considerations..... 350
Successful Completion of a Disconnection.....350
Forcing the Deletion of a Coupling Facility Object..... 350
Deleting a Coupling Facility Structure.....350
Deleting a Coupling Facility Connection to a Structure..... 351
Deleting a Structure Dump..... 351
Deleting Structure Dump Serialization.....351
Authorizing the Use of IXLFORCE..... 352
Forcing a Structure with Failed-Persistent Connections..... 352
Coding Exit Routines for Connection Services..... 352
Coding the Event Exit.....352
Using IXLEERSP..... 355

Chapter 7. Using Cache Services (IXLCACHE)..... 357

Benefits of Using Cache Services..... 357
Elements of A Cache System..... 357
Elements of a Cache Structure..... 359
Important Terms..... 361
Using the Cache Structure..... 363
Store-in Cache..... 363
Store-through Cache..... 364
Directory-only Cache..... 365
Summary of IXLCACHE Requests..... 365
Cache Structure Allocation and Connection..... 367
Accessing and Managing Data Within a Cache System..... 369
Managing Local Cache Buffers..... 369
Identifying a Data Item to the Cache Structure..... 370
Changing a Data Item in the Cache Structure..... 371
Casting out Data or Updating Permanent Storage..... 372
Maintaining Data Consistency..... 373
Registering Interest in a Data Item and Validating Local Copies..... 373
Deregistering Interest in a Data Item and Invalidating Local Copies..... 375
Determining the Validity of a Data Item through IXLVECTR..... 377
Serializing and Managing Access to Shared Data..... 377
Using but not Updating Data in a Store-in Cache..... 378
Updating Data in a Store-in cache..... 378
Using but not Updating Data in a Store-through Cache..... 378
Updating Data in a Store-through Cache..... 379
Using but not Updating Data in a Directory-only Cache..... 379
Updating Data in a Directory-only Cache..... 380
Serialization considerations when requesting asynchronous cross-invalidations..... 380
Managing Cache Structure Resources..... 380
Storage Reclaim..... 381
Deleting Data Items and Reclaim Processing..... 384
Casting out Data Items and Reclaim Processing..... 385
Measuring Cache Structure Resource Usage..... 386
Understanding Synchronous and Asynchronous Cache Operations..... 386
The MODE Parameter — Summary..... 387
Using the IXLFCOMP Macro..... 388
Selecting a Data Buffer For a Request..... 388
Receiving Information from a Request..... 395
Requesting Return and Reason Codes..... 395
Defining an Answer Area (ANSAREA)..... 395
Determining Valid Information in the Answer Area..... 397
Specifying the Vector Entry Index on IXLCACHE Requests..... 397
Using Filters for Names on Requests..... 398
Restarting a Request that Ends Prematurely..... 398
Using the Restart Token..... 399
Using an Index Value..... 400
Understanding the Cache Data Entry Version Number..... 401
Other Services Used with IXLCACHE..... 402
WRITE_DATA: Writing a Data Item to a Cache Structure..... 402
IXLCACHE Functions for REQUEST=WRITE_DATA..... 403
WRITE_DATALIST:Writing Multiple Data Items to a Cache Structure..... 412
IXLCACHE Functions for WRITE_DATALIST..... 412
READ_DATA: Reading a Data Item from a Cache Structure..... 417
IXLCACHE Functions for REQUEST=READ_DATA..... 418
REG_NAMELIST: Registering Interest in a List of Data Items..... 421
IXLCACHE Functions for REQUEST=REG_NAMELIST..... 422
CASTOUT_DATA: Casting Out Data from a Cache Structure..... 427
Reasons for Casting out Data..... 428
Cast-out Requests..... 428

IXLCACHE Functions for CASTOUT_DATA....429
CASTOUT_DATALIST: Casting Out a List of Data Items....432
IXLCACHE Functions for REQUEST=CASTOUT_DATALIST....432
UNLOCK_CASTOUT: Releasing Cast-Out Locks....434
IXLCACHE Functions for REQUEST=UNLOCK_CASTOUT....435
UNLOCK_CO_NAME: Releasing a Single Cast-Out Lock....439
IXLCACHE Functions for REQUEST=UNLOCK_CO_NAME....440
DELETE_NAME: Deleting Data Items From a Cache Structure....442
IXLCACHE Functions for REQUEST=DELETE_NAME....443
DELETE_NAMELIST: Deleting a List of Data Items....446
IXLCACHE Functions for REQUEST=DELETE_NAMELIST....446
CROSS_INVAL: Invalidating Other Users' Copies of Data Items....449
Timing and CROSS_INVAL Requests....449
IXLCACHE Functions for REQUEST=CROSS_INVAL....449
CROSS_INVALLIST: Invalidating a List of Data Items....450
IXLCACHE Functions for REQUEST=CROSS_INVALLIST....451
SET_RECLVCTR: Overriding or Restoring the Default Reclaim Algorithm....452
Defining the Reclaim Vector....453
IXLCACHE Functions for REQUEST=SET_RECLVCTR....454
PROCESS_REFLIST: Marking Data Items as Referenced....457
IXLCACHE Functions for REQUEST=PROCESS_REFLIST....458
RESET_REFBIT: Marking Data Items as Unreferenced....459
IXLCACHE Functions for REQUEST=RESET_REFBIT....460
READ_DIRINFO: Reading Cache Directory Entries....461
IXLCACHE Functions for REQUEST=READ_DIRINFO....462
READ_COCLASS: Reading A Cast-Out Class....465
IXLCACHE Functions for REQUEST=READ_COCLASS....466
READ_COSTATS: Reading Cast-Out Class Statistics....468
IXLCACHE Functions for REQUEST=READ_COSTATS....468
READ_STGSTATS: Reading Storage Class Statistics....472
IXLCACHE Functions for REQUEST=READ_STGSTATS....472
Coding a Complete Exit for IXLCACHE....474
Information Passed to the Complete Exit....474
Environment....475
Input Specifications....475
Return Specifications....476
Programming Considerations....476
Managing Cache Structure Utilization....477
Detecting When a Cache Structure Is Becoming Full....477
Responding When the Structure Is Getting Full....478

Chapter 8. Using List Services (IXLLIST)....481
List Structure Concepts....482
What is a List Structure?....483
How Is Data Maintained in a List Structure?....485
What Functions Does the List Structure Provide?....486
Referencing List Entries....489
Understanding the List Cursor....494
Understanding List Structure Monitoring....506
Understanding the Event Queue....508
Understanding Event Monitor Controls....509
Understanding Sublist Monitoring....509
Reviewing Sublist and Event Queue Monitoring....510
Understanding List Entry Controls....511
Understanding List Controls....511
Understanding the List Authority Value....513
Understanding the User Exits....514
Understanding Synchronous and Asynchronous List Operations....514

Understanding the Serialized List Structure.... 516
Understanding the List Entry Version Number.... 523
Selecting the Buffer Format.... 524
WRITE: Writing to a List Entry.... 531
Understanding the Write Operation.... 531
Passing Data for a WRITE Request.... 532
Requesting a Lock Operation as Part of a WRITE Request.... 533
Updating an Existing List Entry.... 533
Creating a New List Entry.... 533
Receiving Answer Area Information from a WRITE Request.... 534
READ, READ_MULT, READ_LIST: Reading List Entries.... 537
READ: Reading a Single List Entry.... 538
READ_LIST: Reading Multiple List Entries from a List.... 541
READ_MULT: Reading Multiple List Entries from One or More Lists.... 547
MOVE: Moving a List Entry.... 550
Understanding the MOVE Operations.... 550
Moving a List Entry Without Performing a Read or Write Operation.... 553
Performing a Read Operation as Part of a Move Request.... 554
Performing a Write Operation as part of a MOVE Request.... 554
Creating a New List Entry as Part of a MOVE Request.... 554
Receiving Answer Area Information from a MOVE Request.... 555
DELETE, DELETE_MULT, DELETE_ENTRYLIST: Deleting List Entries.... 557
DELETE: Deleting a Single List Entry.... 557
DELETE_MULT: Deleting Multiple List Entries.... 560
DELETE_ENTRYLIST: Deleting a List of Entries.... 562
READ_LCONTROLS: Reading List Controls.... 565
Obtaining List Monitoring Information.... 565
Receiving Answer Area Information from a READ_LCONTROLS Request.... 565
WRITE_LCONTROLS: Writing List Controls.... 567
Changing the List Limit.... 567
Effect of Structure Alter on the List Limit.... 567
Receiving Answer Area Information from a WRITE_LCONTROLS Request.... 568
LOCK: Performing a Lock Operation.... 568
Selecting the Lock Operation.... 568
Handling an Incompletely Processed LOCK Request that Specifies LOCKOPER=READNEXT... 569
Receiving Answer Area Information from a LOCK Request.... 569
MONITOR_LIST: Monitoring List Transitions.... 570
The List Notification Vector.... 571
Indicating Your Interest in List Transition Monitoring.... 572
Starting Transition Monitoring of a List.... 572
Stopping Transition Monitoring of a List.... 572
Design Considerations for Using the List Transition Exit.... 572
Receiving Answer Area Information from a MONITOR_LIST Request.... 573
MONITOR_EVENTQ: Monitoring an Event Queue.... 574
Steps to Set Up Event Queue Transition Monitoring.... 574
Indicating Your Interest in Event Queue Transition Monitoring.... 575
Starting Transition Monitoring of an Event Queue.... 575
Stopping Transition Monitoring of an Event Queue.... 575
Receiving Answer Area Information from a MONITOR_EVENTQ Request.... 575
MONITOR_SUBLIST, MONITOR_SUBLISTS: Monitoring Sublists.... 576
Understanding the Event Queue.... 576
Indicating Your Interest in Sublist Transition Monitoring.... 576
Specifying User Notification Controls.... 576
MONITOR_SUBLIST: Monitoring a Single Sublist.... 577
Starting Transition Monitoring of a Sublist.... 577
Stopping Transition Monitoring of a Sublist.... 577
Scenario for Monitoring a Sublist.... 577
Receiving Answer Area Information from a MONITOR_SUBLIST Request.... 577

MONITOR_SUBLISTS: Monitoring Multiple Sublists....578
Receiving Answer Area Information from a MONITOR_SUBLISTS Request....579
READ_EMCONTROLS: Reading Event Monitor Controls....580
Receiving Answer Area Information from a READ_EMCONTROLS Request....580
READ_EQCONTROLS: Reading Event Queue Controls....581
Obtaining Event Queue Monitoring Information....581
Receiving Answer Area Information from a READ_EQCONTROLS Request....581
DEQ_EVENTQ: Retrieving Events from the Event Queue....582
Handling an Incompletely Processed DEQ_EVENTQ Request....582
Receiving Answer Area Information from a DEQ_EVENTQ Request....582
Coding a Complete Exit....583
Information Passed to the Complete Exit....583
Environment....584
Input Specifications....584
Return Specifications....584
Coding a Notify Exit....585
Information Passed to the Notify Exit....585
Environment....587
Input Specifications....587
Return Specifications....587
Coding a List Transition Exit....588
Information Passed to the List Transition Exit....588
Environment....589
Input Specifications....589
Return Specifications....589
Managing List Structure Utilization....590
Detecting When a List Structure Is Becoming Full....591
Responding When the Structure is Getting Full....591
Enhancements to Sublist Monitoring....592

Chapter 9. Using List Services (IXLLSTE, IXLLSTM, IXLLSTC)....595
Additional List Services Provided by IXLLSTE, IXLLSTM, and IXLLSTC....595
Understanding Program-Specified Entry Identifiers....596
Enhancement to List Services Entry Key and Secondary Key Comparison....596
Enhancements to List Monitoring....596
Understanding Secondary Keys....597
Using the New List Services Request Types....597
Specifying 64-bit buffers....598
Comparing IXLLSTC, IXLLSTE, and IXLLSTM with IXLLIST....598
Locating a List Entry....598
Specifying a Range of Values....599
Specifying Entry Keys or Secondary Keys....599
Identifying Individual List Entries for IXLLSTM Requests....599
IXLLSTE: List Structure Single Entry Services....599
Selecting an Entry for Processing by IXLLSTE....600
Receiving Answer Area Information....601
Creating a List Entry....601
Reading a List Entry....602
Writing a List Entry....603
Moving a List Entry....603
Deleting a List Entry....605
IXLLSTM: List Structure Multiple Entry Services....605
Selecting Entries for Processing by IXLLSTM....606
Receiving Answer Area Information....607
Restarting IXLLSTM Requests....607
Reading List Entries from a List....607
Deleting List Entries from a List....609
Reading List Entries from Multiple Lists....611

Deleting List Entries from Multiple Lists ... 612
Moving a List of List Entries ... 613
Deleting a List of List Entries ... 614
IXLLSTC: List Structure Control Services ... 615
Understanding Threshold Counts ... 616
Reading Control Information for a List ... 618
Updating Control Information for a List ... 618
Monitoring a List ... 618
Monitoring a Sublist ... 620
Monitoring a Set of Sublists ... 620
Monitoring a List by Keyrange Values ... 620
Monitoring an Event Queue ... 622
Reading EMC Control Information ... 622
Reading Event Queue Control Information ... 622
Dequeuing Event Monitor Controls from an Event Queue ... 623
Performing Lock Operations ... 623
Reading Structure Counts for a List Structure ... 623

Chapter 10. Using Lock Services (IXLLOCK) ... 625
Resource Concepts ... 625
What Is a Resource? ... 625
State of a Resource Request Queue ... 626
What can you do with the XES lock services? ... 627
Managing Contention ... 628
Defining a Protocol to Handle Contention ... 629
How is Contention Resolved? ... 629
Sample Locking Protocol — Definition ... 630
Sample Locking Protocol — Implementation ... 632
Informing a User of Request Completion ... 633
Using the IXLLOCK MODE Parameter ... 633
Asynchronous duplexing ... 634
Lock Structure Concepts ... 634
The Lock Table ... 635
Record data entries ... 641
Size Considerations for a Lock Structure ... 642
Recovery Considerations ... 643
Designing for recovery ... 643
XES cleanup processing ... 644
Sample Recovery Protocol ... 645
Requesting Lock Services ... 647
Requesting Ownership of a Resource (REQUEST=OBTAIN) ... 648
Determining the Completion of an OBTAIN Request ... 650
Changing Ownership Attributes (REQUEST=ALTER) ... 650
Determining the Completion of an ALTER Request ... 651
Releasing ownership of a resource (REQUEST=RELEASE) ... 652
Determining the Completion of a RELEASE Request ... 652
Processing multiple resource requests (REQUEST=PROCESSMULT) ... 653
Determining the completion of a PROCESSMULT request ... 654
Using Exits for Coupling Facility Lock Services ... 654
General Requirements ... 655
Coding a Complete Exit ... 657
Coding a Contention Exit ... 659
Coding a Notify Exit ... 668
Using the Synchronous Update Service (IXLSYNCH) ... 670
Using the Lock Cleanup and Recovery Service (IXLRT) ... 672
Identifying the User ... 672
Providing areas for returned data ... 672
Identifying the Record Data ... 673

Assigning a Record Data Type to the Record Data....673
What You Can Request with IXLRT....673
Handling an incompletely processed IXLRT request....677

Chapter 11. Supplementary List, Lock, and Cache Services....679
Using the IXLFCOMP Macro....679
Issuing IXLFCOMP During Recovery Processing....679
Purging a Coupling Facility Operation....680
Handling Operations in Progress....680
Handling Operations Yet to be Processed....680
Timing Considerations....680
Using the IXLVECTR Macro....680
List Notification Vector....680
Local Cache Vector....682
Using the IXLADUPX macro....688
Using the IXLAXISN macro....689

Chapter 12. Using Note Pad Services (IXCNOTE)....691
Note pad concepts and terminology....694
Use of the term *connection*....695
Designing your application to use an XCF Note Pad....696
What is a note....698
Note name....698
Note instance number....698
Note tags....699
Note content....700
Note connection identifier....700
Note persistence....701
What is an XCF Note Pad....701
Note pad name....703
Note pad description....706
Note pad information....706
Note limit....706
Duplexing preference....706
Note pad protocols....707
Timestamp when created....710
What is a connection....710
Connection token....711
Connection identifier....711
Connection description....712
Connection information....712
Access scope....712
Connection scope....712
Usage classification....713
Termination scope....715
Using the IXCNOTE macro....716
Answer area....717
System Authorization Facility (SAF) requirements....718
Use of a connection token....719
Quiescing conditions....720
Constrained conditions....720
Timeout conditions....721
Status unknown conditions....723
XCF component failures....725
Note pad requests....725
Create note pad....726
Query note pad....727
Modify note pad....728

Delete note pad ... 729
Connection requests ... 731
Create connection ... 732
Delete connection ... 733
Pause connection ... 733
Resume paused connection ... 734
Single note requests ... 734
Overview ... 734
Create note ... 738
Replace note ... 738
Write note ... 739
Read note ... 739
Delete note ... 740
Multi-note requests ... 740
Overview ... 740
Multi-note selection criteria ... 741
Concurrent request issues ... 743
Read notes ... 744
Delete notes ... 747
Note pad related limits ... 748
Buffer size and single note requests ... 749
Buffer size and multi-note requests ... 749

Chapter 13. Coupling Facility Accounting and Measuring Services ... 751
Using IXLMG ... 751
Specifying the information level ... 751
Types of information available ... 752
Defining an Output Area ... 754
Programming Considerations ... 755
Specifying the Information To Be Returned by IXLMG ... 755

Chapter 14. Dumping Services for Coupling Facility Structures ... 759
Using the IHABLDP Macro ... 759
Using the IXLZSTR Macro ... 759
Requesting Structure Information ... 759
Receiving Information Returned by the IXLZSTR Macro ... 760
Using Component Data in the Dump Data Set ... 760
Associating Macros with the Data Types ... 762

Chapter 15. Documenting your Coupling Facility Requirements ... 767
Specifying the Coupling Facility Structure Requirements ... 767
Naming the Structure ... 767
Determining the Structure Size ... 767
Providing an Exclusion List ... 768
Understanding the Persistence Attribute ... 768
Specifying the Rebuild and/or Alter Attribute ... 768
Providing Connectivity Requirements ... 769
Specifying the coupling facility requirements ... 769
Summarizing Your Requirements ... 770

Appendix A. Accessibility ... 771

Notices ... 773
Terms and conditions for product documentation ... 774
IBM Online Privacy Statement ... 775
Policy for unsupported hardware ... 775
Minimum supported hardware ... 775

Programming Interface Information....776
Trademarks....776

Index....777

Figures

1. Systems, Groups, and Members in an XCF Sysplex 8

2. XCF Member States 14

3. Address Space Restrictions for XCF Macros 20

4. Sending a Message from One Member to Another 27

5. Example of Queue of Message Data Elements 33

6. First Example of Table of Message Data Elements 33

7. Second Example of Table of Message Data Elements 34

8. Third Example of Table of Message Data Elements 35

9. XCF Status Monitoring Service Normal Processing (Part 1 of 2) 66

10. XCF Status Monitoring Service Normal Processing (Part 2 of 2) 67

11. Summary of Group User Routine Logic (Part 1 of 3) 92

12. Summary of Group User Routine Logic (Part 2 of 3) 93

13. Summary of Group User Routine Logic (Part 3 of 3) 94

14. PHONBOOK Multisystem Application 121

15. Data Structures Used by PHONBOOK Routine 123

16. Overview of client/server processing in the sysplex 133

17. Multiple Systems Sharing Data Through a Coupling Facility 199

18. Connection State Transitions: Undefined, Active, Disconnecting, Failing 207

19. Allocating a Structure 221

20. Connecting to an Allocated Structure 221

21. List Structure Space Allocation 237

22. Active Connections 253

23. A Failed-Persistent Connection 254

24. Acknowledging a Failed-Persistent Connection..... 254

25. Reconnection of a Failed-Persistent Connection.....254

26. Deleting a Failed-Persistent Connection using IXLYEEPL..... 263

27. Rebuild Timeline..... 298

28. User-Managed Duplexing Rebuild Timeline.....300

29. Sequence of Events During System-Managed Rebuild.....323

30. Sequence of Events During System-Managed Duplexing Rebuild..... 324

31. Elements of a Cache System..... 358

32. Major Elements of a Cache Structure.....359

33. Registered Interest in Data Items.....374

34. Invalidating Local Cache Copy of a Data Item..... 376

35. Format of Buffer List Specified by the BUFLIST Parameter..... 390

36. Format of Buffer List (BUFLIST) - 64-bit Addresses.....391

37. Three Reclaim Vectors.....453

38. Serialized List Structure.....483

39. Event Queues in a List Structure..... 484

40. List Containing Entries with Various Numbers of Data Elements.....486

41. Use of List Position.....491

42. Use of List Position with Entry Key..... 492

43. Example of Keyed List Entries that Cannot Be Referenced by Entry Key.....493

44. Use of KEYREQTYPE and LISTPOS Parameters.....493

45. Initializing a List Cursor with an IXLLIST WRITE_CONTROLS Request.....495

46. Initializing a List Cursor with Another IXLLIST Request.....495

47. Updating the List Cursor to the Next Entry..... 496

48. Updating the List Cursor Conditionally — Example 1.....497

49. Updating the List Cursor Conditionally — Example 2....497
50. Updating the List Cursor to the Current Entry — Example 1....498
51. Updating the List Cursor to the Current Entry — Example 2....498
52. Conditionally Updating the List Cursor to the Current Entry — Example 1....499
53. Conditionally Updating the List Cursor to the Current Entry — Example 2....500
54. Updating the List Cursor without Using LOCBYCURSOR....501
55. Updating the List Cursor when Creating an Entry with WRITE....502
56. Updating the List Cursor when Creating an Entry with MOVE....502
57. List Cursor After the List Entry is Deleted....504
58. List Cursor When Moved Before the First List Entry....504
59. List Cursor When Moved After the Last List Entry....505
60. List Cursor When Moved Conditionally Before First Entry....505
61. List Cursor When List Entry Is Deleted....506
62. Format of Buffer List Specified by the BUFLIST Parameter....526
63. Format of Buffer List - 64-bit Addresses....526
64. Layout of List Entry Information Returned by READ_LIST Request....544
65. Possible Errors Resulting from Reissue of READ_LIST Request....546
66. List Entry Key Resulting from a MOVE Request....552
67. XES Compatibility Rules....626
68. Resource Request Queue Compatibility....627
69. Lock Structure with Optional Record Data Entries....635
70. Lock Table — Using a Hash Value....637
71. Receiving a Resource Request....660
72. Contention Exit Processing....667
73. Sample Serialization Protocol for Single Data Item....686

74. Sample Serialization Protocol for Multiple Data Items....687

75. Sample Serialization Protocol for a Range of Data Items....688

76. Conceptual model of a note pad....691

77. Connections to a note pad....692

78. Physical embodiment of a note pad....693

79. A note in a note pad....698

80. Note pad connection with USAGE=CONNECTION....713

81. Creating a note pad connection on behalf of a client....713

82. Note pad connection with USAGE=SERVER....714

83. Note pad connection with USAGE=CLIENT....715

84. Answer areas for query note pad....728

85. Multi-note answer area with buffer....745

86. Layout of IXLYAMDA....754

87. Format of Coupling Facility Structure Data in Dump Data Set....761

Tables

1. Summary of XCF Communication Macros..... 20

2. Differences between IXCCREAT and IXCJOIN Macros..... 24

3. Summary of Options on IXCCREAT and IXCJOIN Macros.....25

4. Status User Routine Events Other Than Normal Processing..... 69

5. Events that Cause XCF to Schedule a Group User Routine.....78

6. Skipping of Events Presented to Group User Routines..... 82

7. IXCQUERY Macro Parameters..... 100

8. Summary of IXCMG macros and information XCF provides..... 116

9. Group User Routine Scheduled vs. Status Update Missing..... 125

10. Functions and keywords for IXCSRVR.....144

11. Functions and keywords for IXCSEND..... 151

12. Functions and keywords for IXCRECV.....160

13. Automatic Restart Management Element States.....186

14. Initial Structure Size Allocation.....213

15. User-Managed vs. System-Managed Rebuild Processing..... 264

16. Structure attributes that can be changed with rebuild connect.....277

17. Summary of Events Reported to the Event Exit..... 335

18. Events Monitored by XES.....342

19. IXLYEEPL Data for IXLUSYNC..... 345

20. Comparison of IXLYEEPL and IXLEERSP.....355

21. Data Entry, Data Element, and Adjunct Area Characteristics.....360

22. Terms for Caching..... 361

23. Description of IXLCACHE Services..... 366

24. Two Reclaim Vectors....382
25. Options for IXLCACHE Request Processing and Completion Notification....387
26. When Storage Areas Passed to IXLCACHE Can Be Made Pageable....394
27. Results of Specifying the Number of Data Elements....409
28. IXLCACHE Registration Block Information....423
29. IXLCACHE Registration Block Returned Information....425
30. Identifying Data Items to Mark as Unreferenced....460
31. Identifying Directory Entries to Read....462
32. IXLCACHE Storage Class Statistics Description....472
33. Components of a List Entry....485
34. Summary of IXLLIST Macro Functions....487
35. LISTCNTLTYPE = ENTRY when a list structure is allocated....513
36. LISTCNTLTYPE = ELEMENT when a list structure is allocated....513
37. Options for IXLLIST Request Processing and Completion Notification....515
38. List Structure Lock Operations....518
39. When Storage Areas Passed to IXLLIST Can Be Made Pageable....530
40. Results of Specifying the Number of Data Elements on a WRITE Request....532
41. Rules for Placement of Keyed List Entry for REQUEST=WRITE....534
42. List Structure Lock Operations....569
43. Request Types for IXLLSTE....599
44. Request Types for IXLLSTM....606
45. Request Types for IXLLSTC....615
46. Required Information for Application A....630
47. IXLLOCK Lock Request Block Information....653
48. Return Codes for the Contention Exit....668

49. Coupling Facility Structure COMPDATA Space Descriptions..... 761

About this information

This document supports z/OS (5694-A01).

This document describes the services that MVS™ provides to enable multisystem applications and subsystems to:

- Run in a sysplex
- Share status information
- Send and receive messages using signaling paths
- Automatically restart jobs and started tasks if they or the system on which they are running unexpectedly terminate
- Share data using the coupling facility
- Serialize on resources using the coupling facility

These sysplex services can be used by authorized assembler language programs. In general, an authorized program meets one or more of the following requirements:

- Runs in supervisor state
- Runs under PSW key 0-7
- Resides in an APF-authorized library

Some of the sysplex services, however, are restricted to callers with a PSW key of 0.

Who should use this information

This document is for programmers designing or modifying a multisystem application or subsystem to run in a sysplex and take advantage of the communication, recovery, and data sharing functions available to sysplex members.

Programmers using this document should be extremely knowledgeable about the MVS operating system and assembler language programming.

How this information is organized

This document is divided into the following parts:

- Part 1, “Introduction to Sysplex Services,” on page 1
- Part 2, “Sysplex Services for Communication (XCF),” on page 5
- Part 3, “Sysplex Services for Recovery (Automatic Restart Management),” on page 173
- Part 4, “Sysplex Services for Data Sharing (XES),” on page 195

Where to find more information

Where necessary, this document references information in other documents, using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see *z/OS Information Roadmap*.

Short title used in this information	Title	Order number
PR/SM Planning Guide	*PR/SM Planning Guide*	GA22-7236

© Copyright IBM Corp. 1994, 2023

How to send your comments to IBM

We invite you to submit comments about the z/OS product documentation. Your valuable feedback helps to ensure accurate and high-quality information.

Important: If your comment regards a technical question or problem, see instead "If you have a technical problem" on page xxvii.

Submit your feedback by using the appropriate method for your type of comment or question:

Feedback on z/OS function
If your comment or question is about z/OS itself, submit a request through the IBM RFE Community (www.ibm.com/developerworks/rfe/).

Feedback on IBM® Documentation function
If your comment or question is about the IBM Documentation functionality, for example search capabilities or how to arrange the browser view, send a detailed email to IBM Documentation Support at ibmdocs@us.ibm.com.

Feedback on the z/OS product documentation and content
If your comment is about the information that is provided in the z/OS product documentation library, send a detailed email to mhvrcfs@us.ibm.com. We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information.

To help us better process your submission, include the following information:

- Your name, company/university/institution name, and email address
- The following deliverable title and order number: z/OS MVS Sysplex Services Guide, SA23-1400-50
- The section title of the specific information to which your comment relates
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive authority to use or distribute the comments in any way appropriate without incurring any obligation to you.

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

If you have a technical problem

If you have a technical problem or question, do not use the feedback methods that are provided for sending documentation comments. Instead, take one or more of the following actions:

- Go to the IBM Support Portal (support.ibm.com).
- Contact your IBM service representative.
- Call IBM technical support.

© Copyright IBM Corp. 1994, 2023

Summary of changes

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line to the left of the change.

Note: IBM z/OS policy for the integration of service information into the z/OS product documentation library is documented on the z/OS Internet Library under IBM z/OS Product Documentation Update Policy (www-01.ibm.com/servers/resourcelink/svc00100.nsf/pages/ibm-zos-doc-update-policy?OpenDocument).

Summary of changes for z/OS MVS Programming: Sysplex Services Guide for Version 2 Release 5 (V2R5)

The following content is new, changed, or no longer included in V2R5.

New

The following content is new.

February 2022 refresh

- Updates to XCF Note Pad. See Chapter 12, "Using Note Pad Services (IXCNOTE)," on page 691. (APAR OA60571, V2R3, V2R4 and V2R5.)

Changed

The following content is changed.

March 2023 refresh

- "Determining the Structure Size" on page 767 is rewritten due to previous outdated information.

June 2022 refresh

- Updates made to clarify handling of resources for a disconnection, and IXLDISC REASON=NORMAL or DELTESTR behavior. (APAR OA61848, V2R3, V2R4 and V2R5.) See:
  - "Handling Resources for a Disconnection" on page 348
  - "Disconnection to Delete the Structure" on page 348
  - "Identifying Connection States" on page 205
  - "Connection Persistence" on page 207

May 2022 refresh

- For APAR OA60650, information about CFLEVEL 25 exploitation support has been added. See:
  - "Information returned inline to IXCQUERY" on page 108
  - "Connecting to a Lock Structure" on page 238
  - "Using the IXLCONN macro for rebuilds" on page 277
  - "Lock Structure Concepts" on page 634
  - Chapter 10, "Using Lock Services (IXLLOCK)," on page 625

February 2022 refresh

- A note is added about IXLVECTR in the section, "The List Notification Vector" on page 507.

© Copyright IBM Corp. 1994, 2023

Deleted

The following content was deleted.

- None

Summary of changes for z/OS Version 2 Release 4

The following information has been added, changed, or deleted in z/OS Version 2 Release 4 (V2R4). The most recent updates are listed at the top of each section.

New

September 2020 refresh

- For APAR OA56774, added new QuReqRfAMDALevel4 features bit to the IXCQUERY macro. For more information, see “Information returned inline to IXCQUERY” on page 108.

Prior to July 2020 refresh

- Macro IXLAISN added for APAR OA54688. See .“Using the IXLAXISN macro” on page 689
- New FEATURES bit QUREQRFIXLMSGID added for the IXCQUERY macro for APAR OA54468. See “QUREQRFIXLMGSID” on page 110 and “Coupling Facility Information” on page 755.

Changed

September 2020 refresh

- For APAR OA59813, information about System Recovery Boost is updated. See: “Handling Resources for a Disconnection” on page 348, “Events that Cause XCF to Schedule a Group User Routine” on page 77, “Overview of user-managed rebuild processing” on page 268, “MVS-Initiated Rebuild Processing” on page 294, and “Enabling Support of System-Managed Processes” on page 226.
- For APAR OA56774, updated the list structure usage and control information with IXLAMDSTRL4 and cache structure usage and control information with IXLYAMDSTRC4. For more information, see “Types of information available” on page 752.

Prior to July 2020 refresh

- Changes made to IXLCONN and IXLCACHE macros to reflex APAR OA54688 changes.
- Information for collecting Coupling Facility (CF) statistics using HWSTATISTICS parameter on the IXLMG macro, “Coupling Facility Information” on page 755 for APAR OA54468.

Summary of changes for z/OS Version 2 Release 3

The following changes are made for z/OS Version 2 Release 3 (V2R3).

New

- Updates were added from APAR OA47796.
- Updates to the information returned in the IXCYQUAA mapping macro, see “Information mapped by the IXCYQUAA mapping macro” on page 104.
- Updates to the “Information returned inline to IXCQUERY” on page 108.
- For the IXLCACHE macro, the restriction on the first buffer for a WRITE_DATALIST request having to reside in 31-bit virtual storage and the requirement for all addresses in the BUFLIST be virtual addresses has been eliminated. In V2R3, all buffers of a BUFLIST for a WRITE_DATALIST request can reside in 31- or 64-bit addressable storage and be expressed by real or virtual addresses. See “Selecting a buffering method” on page 413.

- Updates to the information available with the IXLLSTC macro services usage and functionality , see the following:
  - “READ_LCONTROLS: Reading List Controls” on page 565
  - “MONITOR_LIST: Monitoring List Transitions” on page 570
  - “Coding a List Transition Exit” on page 588
  - “Additional List Services Provided by IXLLSTE, IXLLSTM, and IXLLSTC” on page 595
  - “IXLLSTC: List Structure Control Services” on page 615
- New topics added for the IXLLSTC macro. See the following:
  - “Understanding List Monitoring Notification Delay” on page 619
  - “Understanding Keyrange Monitoring Notification Delay” on page 621
- Updates to the information available with the IXLLSTE macro services usage and functionality , see the following:
  - “READ_LCONTROLS: Reading List Controls” on page 565
- Updates to the information available with the IXLVECTR macro services usage and functionality , see the following:
  - Chapter 11, “Supplementary List, Lock, and Cache Services,” on page 679
- Updates to the information available with the IXLMG macro, see the following:
  - “Types of information available” on page 752
  - “Coupling Facility Information” on page 755
  - “Coupling Facility Structure Information” on page 757.

Changed

- Updates to the information available with the IXLALTER macro, see section “Structure Type Considerations” on page 327.

Part 1. Introduction to Sysplex Services

© Copyright IBM Corp. 1994, 2023

Chapter 1. Introduction to Sysplex Services

MVS sysplex services consist of macros that provide communication, recovery, and data sharing services to authorized multisystem applications or subsystems that are running in a sysplex. The services can be used independently or together, depending on the requirements of the application. This introduction assumes that you know what a sysplex is and that you are familiar with its advantages for multisystem applications and subsystems. If you need more information before continuing, see the following information:

- Parallel Sysplex (www.ibm.com/systems/z/advantages/pso)
- *z/OS MVS Setting Up a Sysplex*

Sysplex Services for Communication

Cross-system coupling (XCF) services allow multiple instances of an application or subsystem, running on different systems in a sysplex, to share status information and communicate with each other.

Your application or subsystem might consist of multiple instances, each running on a different system in the same sysplex. Typically, each instance performs certain functions for the application as a whole. For example, one instance of a database application might write changed database information to permanent storage on behalf of all instances of the application. Alternatively, each instance could perform all the application's functions on a given system.

Instances of an application can use XCF services to communicate with each other. They can:

- Inform others of their status (active, failed, etc...)
- Obtain information about the status of the other instances of the application.
- Send messages to and receive messages from each other.

Sysplex Services for Recovery (Automatic Restart Management)

XCF services for recovery allow applications to be restarted automatically when they, or the systems they are running on, terminate unexpectedly. Automatic restart management services allow an application to:

- Request automatic restart in the event of application or system failure
- Wait for another job to restart before restarting
- Indicate its readiness to accept work
- Request that automatic restart no longer be performed
- Indicate that automatic restart should be performed only if the backup copy of the application no longer exists.

Sysplex Services for Data Sharing

Cross-system extended (XES) services allow multiple instances of an authorized application or subsystem, running on different systems in a sysplex, to implement high-performance, high-availability data sharing by using a coupling facility. Applications can maintain and access data in three types of structures (list, lock, or cache). Features of the different structures, available through the use of XES services, include the ability to:

- Share data organized as a set of lists (list structure)
- Determine whether a local copy of cached data is valid (cache structure)
- Automatically notify other users when a data update invalidates their local copies of cached data (cache structure)

© Copyright IBM Corp. 1994, 2023

- Implement efficient, customized locking protocols, with user-defined lock states and contention management (lock structure).

Part 2. Sysplex Services for Communication (XCF)

© Copyright IBM Corp. 1994, 2023

Chapter 2. Using the Cross-System Coupling Facility (XCF)

The cross-system coupling (XCF) services provide the following functions that a multisystem application or subsystem programmer can use:

- A way to define a collection of unique parts of a program, and a way for each part to identify the other parts so they can work together.
- A way for program parts to send messages to or receive messages from other parts on the same MVS system or on a different one, without regard for the I/O considerations involved. Messages can be sent without knowing specifically where the receiving part resides.
- A way to monitor the program parts that you (the programmer) define to XCF. XCF maintains information about the parts you define, and provides notification of changes. Again, these parts can be on the same MVS system or different MVS systems.
- A way to design your program for high availability, such that primary parts are on one system and backup parts are on another system. When the primary system fails, XCF notifies the backup parts on the other system and the backup parts can be designed to take over the function of the primary. The primary and backup parts can also be running in different address spaces on the same system. In this case, the parts running in the backup address space can be designed to take over when the primary address space fails.
- A way to allow batch jobs and started tasks to be restarted automatically. You can use the XCF recovery function, automatic restart management, to design your application for high availability by allowing it to be restarted automatically when it, or the system it is running on, fails. See Chapter 4, “Using the Automatic Restart Management Function of XCF,” on page 175 for more information.

Examples of exploiting XCF appear in various sections of this chapter , as the XCF services are explained. Before learning more about the XCF services and how to use them, you must understand some basic XCF concepts.

XCF Concepts

When you design and implement a **multisystem application** program to exploit XCF, you define one or more **members** to a **group** that resides in a **sysplex**. Figure 1 on page 8 illustrates how the sysplex, group, and members relate to one another. These terms are defined as follows:

- **What is a sysplex?**

  A **sysplex** (systems complex) is the set of one or more MVS systems that is given an XCF sysplex name and in which the authorized programs in the systems can then use XCF services. XCF services are available in both single and multisystem environments. A **multisystem environment** is defined as two or more MVS systems residing on one or more processors. In either environment, as you proceed to design your multisystem application, you need to communicate with the system programmer in your installation about the resources you will need. See “Providing Information to Your System Programmer” on page 18 for more information. System programmers should consult *z/OS MVS Setting Up a Sysplex* for complete information on initializing and managing MVS systems in a sysplex.

- **What is a group?**

  A **group** is the set of related members defined to XCF by a **multisystem application** in which members of the group can communicate (send and receive data) between MVS systems with other members of the same group. A group can span one or more of the systems in a sysplex and represents a complete logical entity to XCF.

- **What is a multisystem application?**

© Copyright IBM Corp. 1994, 2023

A **multisystem application** is defined as a program that has various functions distributed across MVS systems in a multisystem environment. Examples of multisystem applications are:

- Installation applications
- Other products or subsystems that support a multisystem environment.

You can set up a multisystem application as more than one group, but the logical entity for XCF is the group.

- **What is a member?**

A **member** is a specific function (one or more routines) of a multisystem application that is defined to XCF and assigned to a group by the multisystem application. A member resides on one system in the sysplex and can use XCF services to communicate (send and receive data) with other members of the same group. However, a member is not a particular task and is not a particular routine. The member concept applies to all authorized routines running in the address space in which the member was defined. The entire address space has the ability to act as that member. All tasks and SRBs in that address space can request services on behalf of the member.

When you define a member, it is associated with the address space in which the IXCJOIN was issued. The member always ends when the address space ends or when the system ends. If you want the member's existence tied to a more specific unit of work, you can further associate the member with either the task or job step task in which the IXCJOIN was issued. In this case, the member also ends when the associated task (or job step task, if selected) ends. This is explained in more detail in the sections entitled "Member Association" on page 17 and "Member Termination" on page 118.

Members of XCF groups are unique within the sysplex. However, XCF allows you to define more than one member from the same task or address space, and have those members belong to different XCF groups. You might use this option if the number of members you require exceeds the maximum (XCF allows up to 2047 members in a group), and you must define another group. You should be aware, however, that designing a multisystem application with a very large number of members has an associated cost to the system in terms of processor storage.

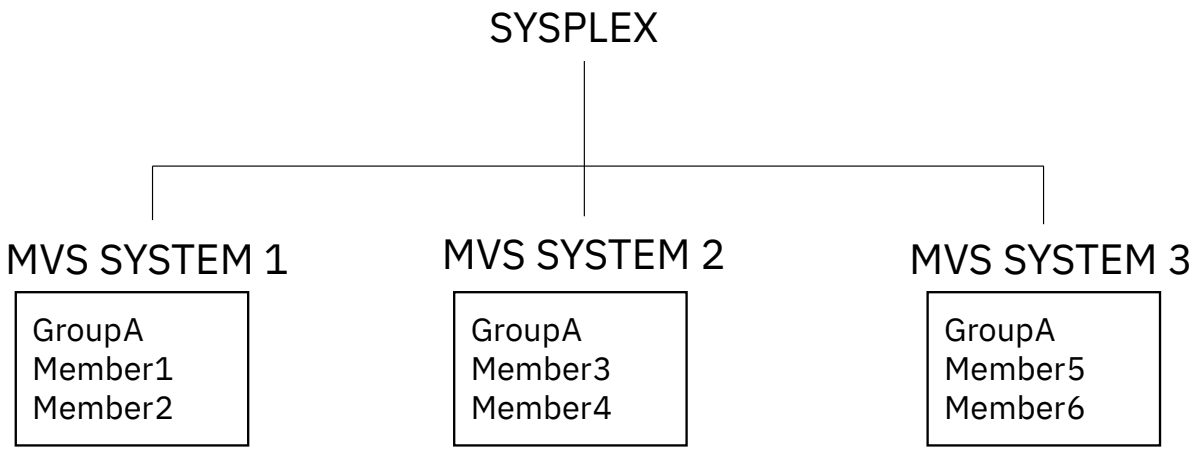


Figure 1. Systems, Groups, and Members in an XCF Sysplex

With these terms defined, you can understand in greater detail the services that XCF provides.

XCF Communication Services

The communication services that XCF provides fall into three broad categories:

- Group services (group and member relationships)
- Signaling services (sending and receiving messages)
- Status monitoring services.

In designing the multisystem application to exploit XCF services, you need to decide the following:

- The structure of the group (group services):
  - Will the group span more than one system or reside on only one system?
  - If the group will span more than one system, will there be one member per system or multiple members per system?
  - What function will each member perform?
  - Will some members duplicate the functions of other members?
  - For which members, if any, will you require a record of the member's existence after the member fails?
  - What name will the group have?
  - What names will the members have?
  - Will the members be associated with a task, a job step task, or only an address space?
  - How will the members be started?
- Which members will be sending messages and which will be receiving messages (signaling services). If messages will be sent, you must plan the size of the messages, how frequently the messages will be sent, and the message content.
- Which members will be notified of changes to other members and changes to systems in the sysplex, which members will take actions based on the notifications, and what those actions will be (group services and status monitoring services).
- Which members will have their activity monitored by XCF (status monitoring services).
- How will your multisystem application handle compatibility in a sysplex made up of varying system release levels?
- How will your multisystem application handle compatibility with varying release levels of itself?

The following sections provide an overview on each of the three categories of XCF services (group, signaling, and status monitoring). XCF provides its services through authorized assembler macros. (See Table 1 on page 20 for a summary of all the XCF macros.) Certain XCF services also require you to identify one or more user routines that you must code. Further details on how to use each of the XCF services, how to code the XCF macros, and how to code the user routines, appear in this chapter .

Group Services

XCF group services provide ways for defining members to XCF, establishing them as part of a group, and allowing them to find out about the other members in the group. Specifically, XCF provides the following for setting up, making changes to, and obtaining information about groups and members:

- The IXCJOIN macro defines a member to an XCF group so the member can use the XCF signaling and status monitoring services.
- The IXCCREAT macro defines a member to XCF to be used later during execution.
- The IXCLEAVE, IXCQUIES, IXCDELET, and IXCTERM macros disassociate members from XCF services. (IXCLEAVE and IXCDELET also disassociate a member from its group.)
- The IXCSETUS macro changes a member's user state value (to be explained in this chapter).
- The IXCMOD macro changes a member's status-checking interval (to be explained in this this chapter).
- The IXCQUERY macro provides you with information about groups, members, and systems in the sysplex.

Through XCF group services, a member can identify an installation-written group user routine. XCF uses this routine to notify the member about changes that occur to members of the group, or systems in the sysplex. With a group user routine, members can have the most current information about the other members in their group without having to query the system.

In providing group services, XCF:

- Guarantees unique identification of each member of a group
- Provides for minimum interference from other multisystem applications when members send messages to one another
- Maintains status information regarding each member of a group.

Signaling Services

XCF signaling services are the primary means of communication between members of an XCF group. XCF provides the following for sending and receiving messages:

- The IXCMSGOX macro, which allows members to send messages to other members in their group, as well as to send responses to messages received.
- The ability to identify two user routines that do processing on behalf of a member. One routine, the message user routine, is for message processing. The other, the message notify user routine, is for processing message responses.
- The IXCMSGIX macro, which allows the message user routine to receive messages from a member and allows the message notify user routine to receive responses from a member.
- The IXCMSGC macro, which allows members to save, discard, reprocess, or obtain information about messages or responses that have been sent.

Note: The IXCMSGOX and IXCMSGIX macro interfaces are the successors to IXCMSGO and IXCMSGI macro interfaces. IBM suggests using IXCMSGOX and IXCMSGIX to send and receive messages between XCF group members, but in most cases you can continue to use IXCMSGO and IXCMSGI. An XCF group member application program needs to be changed to use IXCMSGOX or IXCMSGIX when functions provided by these macro interfaces are wanted.

Client/Server Services

XCF client/server services, which makes use of XCF signaling services, can be used to implement a sysplex-wide protocol that allows clients and servers to send and receive messages in a sysplex. XCF provides the following for sending and receiving messages between clients and servers:

- The IXCSRVR macro, which allows for the definition of servers and server instances
- The IXCSEND macro, which allows clients to send requests to servers as well as allows servers to send responses to client requests
- The IXCRECV macro, which allows programs to obtain the state of messages sent through IXCSEND and as well as receive responses to requests sent by servers
- The IXCREQ macro to format server request messages to be sent to the XCF Server.

The use of the IXCSEND, IXCRECV and IXCSRVR client/server macros to establish sysplex-wide communications between units of work offers an alternative to the existing XCF signalling services for sending and receiving messages in a multi-system XCF group application. Because of the extra interface layer, the traditional XCF signaling services might be better suited to applications with high signaling rates or with stringent performance requirements. Client/server services are well suited for application developers who are interested in using a basic request and response protocol for a sysplex-wide application that results in reduced complexity and implementation. Client/server services are built as a layer on XCF signaling services and provide a simple implementation model that makes use of XCF and sysplex programming best practices. Note that the processing might not be suitable for some performance-sensitive applications.

Some of the differences between using XCF client/server services and XCF signaling services to communicate in a sysplex are:

- A client/server application does not have the development expense or complexity of joining an XCF group and establishing an association with a designated address space or unit of work to send server requests (messages) to systems within the sysplex.
- Client/server applications can develop their applications without concern for XCF member cleanup and termination considerations, writing message and completion exits, and status recording and reporting.

- SRB mode processing is not required to receive messages or process message completion notifications.
- Servers and clients can run in any address space in the system, and send messages and perform all client/server related processing in task mode.
- The IXCRECV macro provides the capability to have a unit of work suspended and immediately released when the server responses have been received and are available for processing by the client. XCF manages all resources associated with message delivery, response monitoring, and status handling.

For details, see Chapter 3, "Using XCF for client/server communication," on page 133.

Status Monitoring Services

XCF status monitoring services provide a way for members to actively participate in determining their own operational status, and to notify other members of their group when that operational status changes. To accomplish this, XCF provides the following:

- The ability to identify an installation-written status user routine, which determines whether a member is operating normally.
- The ability to identify an installation-written group user routine, which allows a member to maintain current information about other members in the group, and systems in the sysplex.

The status user routine and the group user routine work together with XCF in the following sense:

- Specifying a status user routine, status field, and status checking interval on the IXCJOIN macro causes XCF to begin monitoring a specific field that the member identifies. When the member fails to update the field within the specified time interval, or resumes updating after a failure, XCF schedules the status user routine to check on the member.
- When the status user routine confirms that the member's status changed (either it failed to update its status field or resumed updating), XCF notifies the group user routines of other members in the group about the change in the member's status. The group user routines can then take the appropriate actions.

Before proceeding with detailed information about how to use each of the XCF services, you must understand more about the attributes that members of an XCF group can have.

Member Attributes

Members of XCF groups have one or more of the following attributes associated with them:

- Permanent status recording (see "Permanent Status Recording" on page 12)
- Member state (see "The Five Member States" on page 12)
- User state (see "The User State Field" on page 15)
- Member name and group name (see "Member Name and Group Name" on page 15)
- Member token (see "The Member Token" on page 15)
- One or more of the following user routines (see "The User Routines" on page 16):
  - Message user routine
  - Status user routine (implies status monitoring is active)
  - Group user routine
  - Message notify user routine.
- Member Association (see "Member Association" on page 17)
- Participation in XCF-managed response collection (see "XCF-Managed Response Collection" on page 18)

This section explains each of these attributes and, where appropriate, explains how they relate to one another. This section also explains what member and group information you should provide to the system programmer in your installation. (See "Providing Information to Your System Programmer" on page 18.) Details on how to define a member with specific attributes are in "Defining Members to XCF" on page 22.

Permanent Status Recording

The concept of permanent status recording is closely related to both member states and user states. When a member has permanent status recording, XCF:

- Maintains a record of the member's existence (including the member's current member state and user state values) even when the member is dormant or has failed.
- Recognizes five member states for that member. The five member states are:
  - Active
  - Created
  - Quiesced
  - Failed
  - Not-defined.

  For members without permanent status recording, XCF recognizes only two member states: active and not-defined.

When it is important to know what happened to a member the last time it was running, choose permanent status recording for that member. To fully appreciate this concept, you must understand what it means when a member is in each of the five member states, and what it means to have a user state field.

The Five Member States

This section describes each of the five member states and, where appropriate, why a member might choose that state. Figure 2 on page 14 summarizes how the member states relate to each other, and how they can change. Table 1 on page 20 summarizes all the XCF macros and how they relate to member states. From this point forward in the text, when a member is said to be **active** or in the **active state**, that means the member is in the **active member state**. The same is true of the other four member states (created, quiesced, failed, or not-defined).

The Active State

An active member is known to XCF and can use XCF services. Specifically, the active member can:

- Send and receive messages
- Have its status monitored by XCF
- Be notified of status changes to other members of the group.

When a member becomes active (that is, joins a group using IXCJOIN), the member is associated with the address space in which the IXCJOIN was issued. To tie the member to a more specific unit of work, you can further associate the member with either the task or job step task in which the IXCJOIN was issued (see “Member Association” on page 17 for more information.)

Members choose the active state when they need to use XCF services. Members can be active with or without permanent status recording.

The Created State

A member in a created state is known to XCF, but cannot use XCF services. Specifically, the created member cannot:

- Send or receive messages
- Have a status field monitored by XCF
- Be notified by XCF of status changes to other members of the group.

XCF does not associate the created member with a particular task, job step task, address space, or system. (If queried, XCF returns 0 for the system name and job name.)

Members in the created state do, however, have permanent status recording, and can:

- Be queried (member name, member state, and user state field for a created member are available through the IXCQUERY macro)
- Define a user state field on IXCCREAT
- Have their user state field changed by an active member of the same group through IXCSETUS
- Become active through IXCJOIN.

You might place members in a created state, with the intent that they will subsequently become active, under the following circumstances:

- You want to prepare a member with some information before the member becomes active. For example, you can designate a primary member and a backup member by putting an indication in the user state field. As each member is started, it checks its user state field to determine if it is the primary member or the backup member. If it is the primary member, it can then issue IXCJOIN to become active with status monitoring. The backup member would also issue IXCJOIN with the intent of being notified through its group user routine if the primary member experiences problems.
- You need a record of a member's existence even before the member becomes active, in the event that other members require a knowledge of all existing group members.

You might want to place a member in the created state with no intention of making it active. The created member cannot use XCF services, but the other members of the group can use the created member's user state field for shared virtual storage. Other members in a group could use the created member as a focal point for tracking a group state or attribute that is not specifically related to any one active member. (A complete explanation of the user state field appears later in this section .)

The Quiesced State

Only members with permanent status recording can become quiesced. A member in the quiesced state is disassociated from XCF services, but XCF still maintains a record of the member's existence. The IXCQUIES macro places a member in the quiesced state. The following are true for a quiesced member:

- The quiesced member can no longer send or receive messages, and XCF stops scheduling the member's message user routine.
- The quiesced member can no longer have its status monitored by XCF, and XCF stops scheduling the member's status user routine.
- The quiesced member can no longer be notified by XCF of status changes to other members of the group (XCF stops scheduling the member's group user routine).
- XCF no longer associates the quiesced member with a task, job step task, address space, or system, although XCF maintains a record of the system that the member was associated with when it was last **active**.
- A quiesced member can become **active** with permanent status recording once again through IXCJOIN. When this happens, the member is treated as a new member because XCF:
  - Resets the user state field (see “Changing the Value in a User State Field” on page 25 for more information)
  - Deletes any history information
  - Assigns the member a new unique member token.
- A quiesced member can have its user state value changed by another active member of the same group through IXCSETUS.

A member might choose the quiesced state to avoid unnecessary recovery action. When a member becomes quiesced, other members can infer that the member cleaned up its own resources (closed any open data sets, released all serialization on shared data sets, etc.) before terminating.

The Failed State

Only members with permanent status recording can become failed. A member in the failed state is one whose associated task, job step task, address space, or system terminated before the member was

explicitly deactivated by invoking an XCF service. When a member is in the failed state, other members can infer that the member did not have an opportunity to clean up its own resources, and another member should take recovery action.

For address space associated members, however, I/O is purged as part of address termination cleanup before the group user routines of the surviving members receive control to inform those members of the failure.

A failed member can:

- Become **active** with permanent status recording once again through IXCJOIN. When this happens, XCF treats the member as a new member. XCF:
  - Resets the user state field (see "Changing the Value in a User State Field" on page 25 for more information)
  - Deletes any history information
  - Assigns the member a new unique member token.
- Have its user state value changed by another active member of the same group through IXCSETUS.

The Not-Defined State

A member in a not-defined state is not known to XCF. Members are in a not-defined state before they are created or active, and after they are completely disassociated from XCF.

XCF treats the not-defined member similarly to the quiesced member in terms of sending and receiving messages, status monitoring, and notification of changes (see "The Quiesced State" on page 13). Once a member is not-defined, XCF no longer associates the member with a particular task, job step task, address space, or system, and no longer maintains any information about the member.

When a member with permanent status recording becomes not-defined, other members know that no recovery action is required.

When a member without permanent status recording becomes not-defined, the other members cannot determine whether recovery action is required.

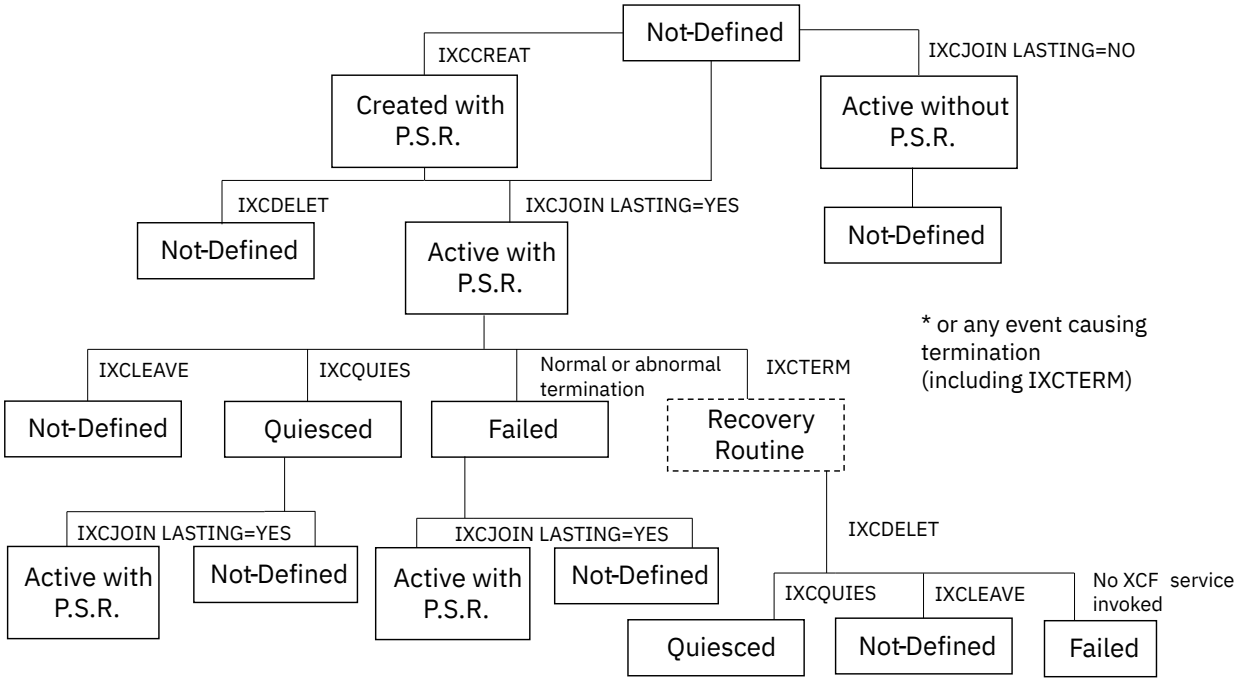


Figure 2. XCF Member States

The User State Field

Every member of an XCF group, regardless of member state, has a user state field associated with it. Each member decides whether it wants to place values in that field. The field is 32 bytes long; a member can use all or part of this field. If a member chooses not to use this field, XCF sets the field to zeros.

While XCF recognizes five member states (not-defined, created, active, quiesced, and failed), the user state field allows you to specify additional, application-defined, internal states that XCF does not use or recognize.

User state information is available to members and other authorized routines in the following ways:

- XCF includes the user state field as part of the parameter list it passes to the group user routines of active members.
- XCF includes the user state field as part of the data it returns to the caller of the IXCJOIN, IXCCREAT, and IXCQUERY macros.

The following are examples of ways you can use the user state field:

- Use the user state field as shared virtual storage. For example, you can use the user state field to keep track of an increasing sequence number across multiple systems. Each member that wants the next sequence number can increment a counter in the user state field.
- Use the user state field to determine which member of the group might perform a particular function. For example, each member places a value in its user state field, and one member examines all the values and chooses the member with the highest value to perform the function.
- Use the user state field to record steps in a process. As each step is completed, update the user state field with a corresponding value. In the event of abnormal termination, other members can determine exactly which steps in the process completed and thereby determine at which point to continue processing.
- Use the user state field to indicate for a failed member that a restart is in progress. For example, when member 1 fails because the system it is running on fails, member 1A can become active in its place. Member 1A can change its user state field to indicate that a restart is in progress. Other members can do recovery for member 1. When recovery is complete, member 1A can change its user state field to indicate it is fully operational.

Members with permanent status recording can reap the most benefit from the user state field. Even if the member ends abnormally, XCF still has a record of the member's existence, and other members can determine what action to take based on the contents of the user state field. Without permanent status recording, if the member ends abnormally, it is no longer defined to XCF, and the contents of the user state field are no longer available to other members.

Details on initializing and changing a user state field are in the sections entitled “Defining Members to XCF” on page 22 and “Changing the Value in a User State Field” on page 25. Refer to “Skipping of Events” on page 81 for user state design considerations related to skipped events.

Member Name and Group Name

Every member of an XCF group has a unique combination of a member name and a group name associated with it. You can specify both the member name and group name on the IXCCREAT and IXCJOIN macros. XCF includes both the member name and group name in the parameter list passed to the group user routine, and in the information provided through the IXCQUERY macro.

The Member Token

When you define a member to XCF through either IXCCREAT or IXCJOIN, XCF assigns a unique (within the sysplex) member token to the member. If a member issues IXCJOIN multiple times, XCF returns a member token for each invocation. In this case, multiple member tokens are associated with the same member.

The member token that XCF returns on IXCCREAT or IXCJOIN can change. XCF assigns a **new** member token when:

- A created member issues IXCJOIN to become active.
- A quiesced or failed member issues IXCJOIN to become active once again.
- A member ends (becomes failed or not-defined) because its associated task, job step task, address space, or system ends; and the member is then restarted.

Aside from these circumstances, a member token remains the same throughout the duration of the sysplex.

Authorized routines use the member token when requesting XCF services on behalf of a member. For the following macros, a member specifies a member token associated with itself:

- IXCLEAVE MEMTOKEN=...
- IXCQUIES MEMTOKEN=...
- IXCMOD TARGET=...
- IXCQUERY MEMTOKEN=
- IXCSYSCL MEMTOKEN=
- IXCMSGC MEMTOKEN=

For the following macros, a member uses two member tokens. The two member tokens can be the same, or can represent two different members of the same group.

- IXCSETUS MEMTOKEN=*the caller's member token*,TARGET=*the target member's member token*
- IXCTERM MEMTOKEN=*the caller's member token*,TARGET=*the target member's member token*
- IXCMSGOX MEMTOKEN=*the sender's member token*,TARGET=*the receiver's member token*
- IXCMSGOX MEMTOKEN=*the caller's member token*,TARGETS=*a table of the member tokens for each receiver*
- IXCMSGC MEMTOKEN=*the caller's member token*,SOURCE=*token*,SOURCE=*the member token for which incoming messages are to be collected*

Any authorized routine can issue IXCDELET TARGET=*the target member's member token*.

The User Routines

Every member of an XCF group can define one or any combination of the following user routines to XCF:

- Message user routine
- Status user routine
- Group user routine
- Message notify user routine.

You are responsible for coding these routines. This section briefly explains each user routine's purpose, and why you might want to code one or more of them. Detailed information on how to code and use each of these routines appears later in this chapter .

- **The Message User Routine**

  The message user routine enables an active member of an XCF group to receive messages from other members of the group. Without a message user routine, a member cannot receive any messages.

  The message user routine also can be used to provide a response to a message, when the member is capable of participating in a protocol that involves sending a response to a sender.

  Code a message user routine when you have one or more members in a group that will be receiving messages from other members and possibly sending responses to those messages. You can use the same message user routine for different members.
- **The Status User Routine**

The status user routine determines whether a member is operating normally. By identifying a status user routine, the member alerts XCF to begin monitoring a field that the member might be updating. If the member fails to update the field within a member-specified time interval, XCF schedules the status user routine to determine if a problem exists. (XCF schedules the status user routine only for active members.) If a problem does exist, XCF notifies other active members of the group through their group user routines.

Code a status user routine when you want XCF to monitor the status field of a member. You can use the same status user routine for different members.

- **The Group User Routine**

  The group user routine enables XCF to notify an active member of a group when there is a change in the operational state of any other member in the group, or a change to the status of any system in the sysplex. If a member does not have a group user routine, XCF cannot notify that member of changes that occur.

  Code a group user routine when you want XCF to notify the member about changes to other members of the same group or about changes to systems in the sysplex. You can use the same group user routine for different members.

- **The Message Notify User Routine**

  The message notify user routine enables XCF to notify a sender about the completion of a message. If the sender specified that a response to the message was required, the message notify user routine is used to process the collected responses. If the sender specified that a response was not required, XCF notifies the sender about the status of the message.

  Your application can specify a message notify user routine when it joins a group, when it sends a message, or when it explicitly calls the user routine to process a response. This allows you to assign a different message notify user routine to different messages.

  Code a message notify user routine if your application includes a protocol for sending messages that require a response or if your application wants to be notified when the message completes. You can use the same message notify user routine for different members.

Member Association

Member association, specified using the MEMASSOC parameter on the IXCJOIN macro, allows you to control the length of time the XCF member will remain in existence. Member association links the newly-created member with a unit of work. When the unit of work ends, so does the member. The member can remain:

- Until the task that issued the IXCJOIN macro ends (MEMASSOC=TASK)
- Until the job step task under which the IXCJOIN macro was issued ends (MEMASSOC=JOBSTEP)
- Until the address space in which the IXCJOIN macro was issued ends (MEMASSOC=ADDRSPACE).

Every member is associated with the address space in which the IXCJOIN was issued. Member association permits you to associate the XCF member with a more specific entity than an address space, namely a task or a job step task.

Note: If a member is termed *address space associated*, the member is associated only with an address space.

When a member is address space associated and the address space ends, all I/O related to the address space is purged before XCF places the member into the failed or not-defined state (whichever is appropriate.) When a surviving member's group user routine receives control, it can assume that the terminated member's I/O has been purged.

The outstanding I/O requests of task or job step associated members might not be purged by the time the group exit receives control.

Note: If the member becomes not-defined using IXCLEAVE, purging of I/O cannot be guaranteed. However, a protocol could be established by the XCF group in which each member purges its own I/O

before issuing the IXCLEAVE macro and sets the user state value on the IXCLEAVE invocation to inform the other members of the group that it has purged its I/O.

The member association also has implications regarding SRB-to-task percolation processing during recovery. For percolation to occur, an associated task (the task that receives control as a result of percolation) must be defined. SRB-to-task percolation, therefore, can be provided only for members that are task or job step associated.

See "Defining Members to XCF" on page 22 for how to specify a member association.

XCF-Managed Response Collection

Your application can request that XCF is to manage the collection of responses to messages you send. Once collected, XCF presents the set of responses to the sender for individual processing.

To exploit this feature, both sending and receiving members must reside on systems running the appropriate z/OS level. The sending member, when sending a message, specifies GETRESPONSE=YES on the IXCMSGOX macro. And, the receiving member must have specified CANREPLY=YES on the IXCJOIN macro to be able to provide a response.

A sending member can use the IXCQUERY service to determine if a target member is capable of participating in a response-collection protocol. See "Specifying Message Response Collection" on page 36 for how to implement XCF-managed response collection.

Providing Information to Your System Programmer

You should provide the system programmer in the installation using your multisystem application with overview information about the application, including a description of the purpose of the groups and the members within each group. Additionally, the following information will help the system programmer plan the sysplex:

- How many groups and members will be defined.
- The names of the groups that will be defined.
- Whether members require permanent status recording.
- The size of the messages that will be sent, how frequently the messages will be sent, and the distribution of the messages amongst the members.
- The consequences that might occur if XCF does not have enough message buffer space to send one or more of your messages.
- The effect of altering the application's configuration on the amount of signaling that might subsequently be generated.
- Specific work load characteristics that might aid in planning transport class definitions.

Providing this type of overview information enables the system programmer to create the proper couple data set(s), to prepare the sysplex configuration, and to establish run-time procedures for the application.

Summary of XCF Communication Macros

XCF provides its services through executable assembler language macros. Members issue some macros on their own behalf, and some macros on behalf of other members. Some macros can be issued by any authorized routine.

Figure 3 on page 20 illustrates these categories by listing the macros that authorized routines can issue from various address spaces on behalf of a particular member. For those macros that have address space restrictions, the key is the IXCJOIN macro. Usually, the primary address space of the caller of an XCF macro must match the primary address space of the caller of IXCJOIN that defined the calling member to the group. The following is an explanation of this figure:

- Address space X represents member 1 of group 1. (IXCCREAT defined member 1 to XCF with permanent status recording, and IXCJOIN made member 1 active.)

- Address space Y represents member 2 of group 1. (IXCJOIN defined member 2 to XCF and made member 2 active. Member 2 issued IXCJOIN with LASTING=YES to request permanent status recording.)
- Address space Z is not a member of any XCF group. (IXCJOIN was not issued from this address space.)
- The following are true regarding address space X:
  - Any authorized routine can issue IXCMOD, IXCMSGOX, IXCQUIES, IXCMSGC, or IXCLEAVE on behalf of member 1, but not on behalf of member 2.
  - Any authorized routine can issue IXCSETUS on behalf of member 1 or on behalf of member 2.
  - A message user routine can issue IXCMSGIX to receive a message on behalf of member 1 but not on behalf of member 2.
  - Any authorized routine can issue IXCTERM to terminate member 1 or member 2.
  - Any authorized routine can issue IXCDELET to delete member 2 (if member 2 is created, quiesced, or failed), but not to delete member 1 (because member 1 is active.
  - Any authorized routine can issue IXCQUERY.
- The following are true regarding address space Y:
  - Any authorized routine can issue IXCMOD, IXCMSGOX, IXCQUIES, IXCMSGC, or IXCLEAVE on behalf of member 2, but **not** on behalf of member 1.
  - Any authorized routine can issue IXCSETUS on behalf of member 2 or on behalf of member 1.
  - A message user routine can issue IXCMSGIX to receive a message on behalf of member 2 but not on behalf of member 1.
  - Any authorized routine can issue IXCTERM to terminate member 1 or member 2.
  - Any authorized routine can issue IXCDELET to delete member 1 (if member 1 is created, quiesced, or failed), but not to delete member 2.
  - Any authorized routine can issue IXCQUERY.
- The following are true regarding address space Z:
  - Any authorized routine can issue IXCDELET to delete member 1 or member 2 (if the member being deleted is created, quiesced, or failed).
  - Any authorized routine can issue IXCQUERY.
- The following is true regarding the master scheduler address space:
  - A member can have an end-of-memory resource manager routine running in the master scheduler address space. The routine can issue IXCMSGOX, IXCQUIES, or IXCLEAVE on behalf of the member.

    Note however, that when invoked from the master scheduler address space, the following IXCMSGOX functions are not available: SENDTO(GROUP), GETRESPONSE(YES), NOTIFY(YES), or TIMEOUT.

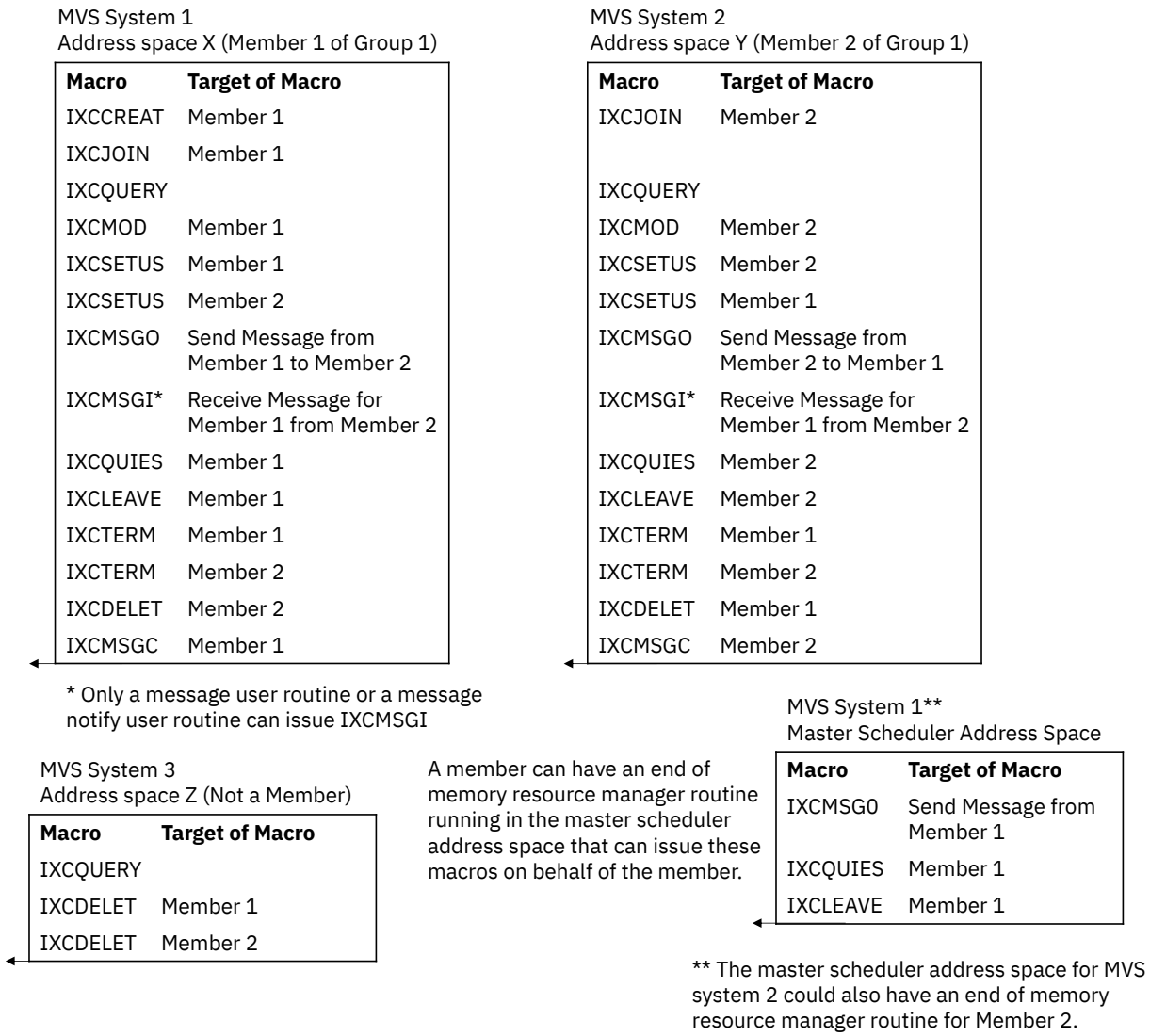


Figure 3. Address Space Restrictions for XCF Macros

Table 1 on page 20 provides a summary of all the XCF macros, the service each macro provides, the effect each macro has on the member state of the target member (where appropriate), what type of routine can issue the macro, and the relationship between the caller of the macro and the target of the macro service. Use the following definitions to interpret the requirements of the caller:

- A *member* is any authorized routine that is running under any task or SRB in the primary address space of the caller of IXCJOIN that defined the member to the group.
- A *calling member* is the member that is invoking the macro.
- A *target member* is the target of the macro service.

Table 1. Summary of XCF Communication Macros

Macro Name	Function	Target Member State Before Macro Executes	Target Member State After Macro Completes	Requirements of Caller
IXCCREAT	Defines a member to XCF, but the member cannot use signaling and status monitoring services.	Not-defined	Created	Any authorized routine in task(1) mode.

Table 1. Summary of XCF Communication Macros (continued)

Macro Name	Function	Target Member State Before Macro Executes	Target Member State After Macro Completes	Requirements of Caller
IXCDELET	Disassociates a member from XCF.	Created Quiesced Failed	Not-defined	Any authorized routine in task(1) mode.
IXCJOIN	Enables a member to join a group and use signaling and status monitoring services.	Not-defined Created Quiesced Failed	Active	Any authorized routine in task(1) mode.
IXCLEAVE	Disassociates a member from XCF.	Active	Not-defined	The calling member equals the target member, or the caller can be any authorized routine running in the master scheduler address space. The caller must be in task(1) mode.
IXCMG	Provides tuning and capacity planning information for the sysplex. Intended for use by system programmers.	N/A	N/A	Any authorized routine in task or SRB mode.
IXCMOD	Changes a member's status-checking interval.	Active	No change	The calling member equals the target member. The caller must be in task(1) mode.
IXCMSGC	Interacts with the XCF signaling service to control message disposition and to obtain information about messages that are held for the user.	Active	No change	Any authorized routine in task or SRB mode. Some request types are valid only when running in task mode, or when running as a message user routine or message notify user routine (SRB mode).
IXCMSGIX	Receives a message on behalf of an active member.	Active	No change	Must be invoked from within either a message user routine or a message notify user routine.
IXCMSGOX	Sends a message to one or more active members in the same group.	Active	No change	The member sending the message can equal the member receiving the message, but if not equal, the sending and receiving member(s) must be active members of the same group. The caller can also be any authorized routine running in the master scheduler address space. The caller can be in task or SRB mode.
IXCQUERY	Returns information about groups, members, and the sysplex.	N/A	N/A	Any authorized routine in task(1) mode. Some request types are valid in both task and SRB mode.
IXCQUIES	Disassociates a member from XCF services, but XCF maintains a record of the member's existence.	Active(2)	Quiesced	The calling member equals the target member, or the caller can be any authorized routine running in the master scheduler address space. The caller must be in task(1) mode.

Table 1. Summary of XCF Communication Macros (continued)

Macro Name	Function	Target Member State Before Macro Executes	Target Member State After Macro Completes	Requirements of Caller
IXCSETUS	Changes the value in a member's user state field.	Active Created Quiesced Failed	No change	The calling member can equal the target member, but if not equal, they must be members of the same group. The caller must be in task(1) mode.
IXCSYSCL	Indicates that a member has completed cleanup processing or that no cleanup was required.	Active	No change	Any authorized routine in task or SRB mode.
IXCTERM	Abnormally ends a task-associated member's task with system completion code 00C and reason code 4. The member's recovery routine cannot retry.	Active	Quiesced, failed, or not-defined(3)	The calling member and target member must be members of the same group. The caller must be in task(1) mode.

1. When the caller must be in task mode, the caller must also be enabled, unlocked, and have no FRRs established.
2. With permanent status recording.
3. The member's state might not have changed when control returns from IXCTERM. The member's recovery routine determines the member's final state, and this processing occurs asynchronously.

Defining Members to XCF

Defining members to an XCF group is a process that requires planning. For each member, you have the following choices:

- Define the member to XCF and immediately make the member **active** (issue the IXCJOIN macro).
- Define the member to XCF in the **created** state with the intent to subsequently make the member **active** (issue the IXCCREAT macro, and later on, issue the IXCJOIN macro).
- Define the member to XCF in the **created** state **without** intending to make the member **active** (issue the IXCCREAT macro).
- Define the member with permanent status recording.
- Define the member with a value in the user state field.
- Define the member with one or more of the following optional user routines on IXCJOIN:
  - Message user routine
  - Status user routine
  - Group user routine.
  - Message notify user routine
- Define the member's association with a task, job step task, or address space.
- Define the member, specifying that the system should wait for the member to clean up resources before the system performs an automatic restart.
- Define the member, specifying that it is capable of participating in XCF's response collection protocols.

Once you define a member, XCF maintains information about the member, along with a timestamp. XCF updates the timestamp on every change to the member's state or the member's user state value.

"Member Attributes" on page 11 provides the information you need to choose attributes for defining each member to XCF. This section provides the information you need to use the IXCCREAT and IXCJOIN macros to define members with the desired attributes.

- **Obtaining Permanent Status Recording**

When you issue the IXCCREAT macro to define a member to XCF, the member automatically has permanent status recording. If, instead, you use the IXCJOIN macro to define a member to XCF, you can choose permanent status recording by coding the LASTING=YES parameter.

If you plan to define members to XCF with permanent status recording, you should inform the system programmer in your installation, because the systems on which your multisystem application will run cannot be in **XCF-local mode**. Permanent status recording requires a sysplex couple data set, which is not supported in XCF-local mode. See *z/OS MVS Setting Up a Sysplex* for further information.

Members requesting permanent status recording should check the return codes from either IXCCREAT or IXCJOIN. Both of these macros provide return codes indicating that the system is in XCF-local mode.

A member cannot discontinue permanent status recording once it is in effect. If a member with permanent status recording is in a created, quiesced, or failed state and then issues IXCJOIN to become active, the member must specify LASTING=YES on the IXCJOIN macro.

- **Initializing a User State Field**

  You can initialize the 32-byte user state field on the IXCCREAT macro or the IXCJOIN macro (USTATE and USLEN parameters). The USLEN parameter indicates the number of bytes of user state data you provided on the USTATE parameter. If you specify a USLEN of fewer than 32 bytes, XCF pads on the right with zeros, thus initializing the remainder of the user state field to zeros.

  If you specify the user state field on the IXCCREAT macro, you do not have to specify it again when you issue IXCJOIN, unless you want to change the value.

  You can also define a member to a group without specifying a value for the user state field on IXCJOIN, and subsequently set the user state field by coding the USTATE and USLEN parameters on IXCLEAVE, IXCQUIES, or IXCSETUS.

  See “Changing the Value in a User State Field” on page 25 for further information.

- **Identifying One or More User Routines**

  Identify user routines on the IXCJOIN macro as follows:

  – Message user routine: code the MSGEXIT parameter.
  – Status user routine: code the STATEXIT, STATFLD, and INTERVAL parameters (you must code all three).
  – Group user routine: code the GRPEXIT parameter.
  – Message notify user routine: code the NOTIFYEXIT parameter.

  When you code one or more user routines, you might want to use the member data field (MEMDATA parameter on IXCJOIN). This is an 8-byte field that can contain whatever information you want to provide to the user routines. XCF includes this information as part of the parameter list that it passes to the message, status, message notify, and group user routines. You might use the member data field to pass the address and ASID or ALET of a particular control structure that the user routine needs to do its work.

- **Associating the Member**

  To specify the unit of work with which the member is to be associated, code the MEMASSOC parameter on the IXCJOIN macro. See “Member Association” on page 17 for more information about member association.

- **Specifying Resource Cleanup for Automatic Restart**

  To specify that the system should wait for the member to clean up resources before automatic resource management restarts batch jobs and started tasks on another system, code the SYSCLEANUPMEM parameter on the IXCJOIN macro. When the group user routine gets control for a system that was removed from the sysplex, the routine must issue the IXCSYSCL macro to indicate that system cleanup has completed.

- **Participating in Message Response Collection Protocols**

To specify that a member is to participate in XCF-managed response collection, code the CANREPLY parameter on the IXCJOIN macro. See "Specifying Message Response Collection" on page 36 for more information about XCF-managed response collection.

- **Summary: Using the IXCCREAT and IXCJOIN Macros**

  Both the IXCCREAT and IXCJOIN macros allow you to:

  – Define a new group name to XCF and define a new member to that group (XCF supports a maximum of 2045 groups and a maximum of 511 members per group, provided the system programmer in your installation defines sufficient capacity in the sysplex.)

  – Define a new member to an existing XCF group (XCF insures that the member name is unique within the group).

  – Initialize a user state field (USTATE and USLEN parameters).

Table 2 on page 24 specifies how the IXCCREAT and IXCJOIN macros differ. Table 3 on page 25 summarizes the parameters you code on each macro to obtain the desired member attributes.

Table 2. Differences between IXCCREAT and IXCJOIN Macros

Area of Difference	IXCCREAT	IXCJOIN
Member state	Defines a member to XCF and places that member in the **created** state. The created member cannot use XCF signaling and status monitoring services.	Defines a member to XCF and places that member in the **active** state. Only members in the **active** state can use XCF signaling and status monitoring services. Issue IXCJOIN to change an existing member from the created, quiesced, or failed state to the active state.
Permanent status recording	Defines a member with permanent status recording.	Defines a member with permanent status recording only if you code LASTING=YES. You can code LASTING=NO if the member does not already have permanent status recording in effect.
Member name	Requires that you provide a member name (MEMNAME parameter).	Requires that you provide a member name (MEMNAME parameter) when you code LASTING=YES. If you code LASTING=NO, you can code MEMNAME, or let XCF generate a unique name for the member.
Member token	Returns a unique (within the sysplex) member token that you can use to code the TARGET parameter on IXCSETUS or IXCDELET. However, when the member becomes active through IXCJOIN, XCF returns a new unique member token, and the old token is no longer valid.	Returns a unique (within the sysplex) member token that you use on subsequent calls to XCF. (If a created, quiesced, or failed member issues IXCJOIN, XCF returns a new unique member token, and the old token is no longer valid.)
User routines	Does not allow you to identify user routines.	Allows you to identify a message, status, group, and message notify user routine.
Member association	Does not allow you to associate a member with any particular task, job step task, address space, or system.	Allows you to associate the member with a task, job step task, or address space on a particular system.
Member cleanup	Does not allow you to specify that the system should wait for the member to clean up resources before the system performs an automatic restart.	Allows you to specify that the system should wait for the member to clean up resources before the system performs an automatic restart.

Table 3. Summary of Options on IXCCREAT and IXCJOIN Macros		
Option	**Macro**	**Parameter**
Permanent status recording	IXCJOIN	LASTING=YES
	IXCCREAT	Automatic
User state value	IXCCREAT or IXCJOIN	USTATE and USLEN
Message user routine	IXCJOIN	MSGEXIT
Status user routine	IXCJOIN	STATEXIT, STATFLD, and INTERVAL
Group user routine	IXCJOIN	GRPEXIT
Message notify user routine	IXCJOIN	NOTIFYEXIT
Member association	IXCJOIN	MEMASSOC
Member cleanup	IXCJOIN	SYSCLEANUPMEM
Message response collection	IXCJOIN	CANREPLY=YES

Changing the Value in a User State Field

Once you have assigned a value to a user state field for a member, you can change it. When you change the value in a user state field, XCF broadcasts the change to those active members that have group user routines.

You can change the user state value for a member by using the IXCSETUS macro or by coding the USTATE and USLEN parameters on any of the following macros:

- IXCJOIN
- IXCLEAVE
- IXCQUIES

For IXCLEAVE and IXCQUIES, if you do not specify the USTATE and USLEN parameters, XCF does not change the existing value in the user state field. For IXCJOIN, if you do not specify the USTATE and USLEN parameters, XCF retains the existing value in the user state field unless the joining member was previously not-defined. In this case, XCF clears the user state field to zeroes.

For IXCLEAVE, IXCQUIES, and IXCSETUS, you specify on USLEN the number of bytes of the user state field you want.

Using the IXCSETUS Macro

Two ways to use the IXCSETUS macro are as follows:

- An active member can issue IXCSETUS to change its own user state value.
- An active member can issue IXCSETUS to change the user state value of another member in the same group. (The target member can be in any of these states: created, active, quiesced, or failed.)

When you issue the IXCSETUS macro, you provide the value (NEWUS parameter) that you want XCF to place in the user state field of the target member. Before making the change, you can take advantage of the compare and swap capability of the IXCSETUS macro to serialize the user state field.

If you code IXCSETUS and attempt to change a user state value without first doing a compare, XCF still checks the current value. If the current value equals the new value you specify on NEWUS, XCF does not make the change, does not notify the other members, and returns an unsuccessful completion code.

Example of Using the IXCSETUS Macro

In this example, the caller of IXCSETUS checks the value in the user state field before attempting to change it.

- Member A and member B are members of the same XCF group. Member C is a member in the created state.
- Member B (the caller of IXCSETUS) expects that member C's current user state value is ***x*** and wants to change it to ***y***, the new user state value (NEWUS parameter).
- Member B makes ***x*** the user state compare value (COMPUS parameter) and issues IXCSETUS.
- XCF compares member C's current user state value with the user state compare value.
- If member C's current user state value is ***x*** (the current value equals the compare value), XCF changes member C's current user state value to ***y*** (the value member B specified on the NEWUS parameter).
- If member C's current user state value does not equal ***x*** (the current value does not equal the compare value), some other member (member A) might have already changed member C's user state value, so XCF does **not** change the current value.
  - If member B specified the OLDUS parameter, XCF places member C's current user state value in that field. Member B can now update its COMPUS parameter to member C's current user state value and try again.
  - If member B specified the OLDUS parameter to be the same storage area as the COMPUS parameter, XCF will place member C's current user state value in the OLDUS parameter, thus automatically updating the compare value for the retry.

Using IXCSETUS for Active Members on Different Systems

When a member on one system (member A on system 1) uses IXCSETUS to update the user state field of a member on a different system (member B on system 2), a problem could occur. If member A updates the user state field of member B, this causes XCF to notify the group user routines of the active members of the group that member B's user state field changed. However, system 1 might fail before XCF on that system has a chance to make the notification. The other members on the other systems receive a notification that member A terminated because system 1 failed, but will not know about member B's user state change. To avoid this problem, you can:

- Have a member issue IXCQUERY (to obtain the latest user state values of the other members in the group) whenever XCF notifies the member's group user routine that another member terminated because of a system failure.
- Have a member change only its own user state value and not that of another member.

Using Signaling Services to Send and Receive Messages

Members of an XCF group can send messages to each other and receive messages from each other using a set of macros:

- IXCMSGOX for sending messages
- IXCMSGIX for receiving messages
- IXCMSGC for saving, discarding, reprocessing, forcing a message to completion, or obtaining information about messages.

Note: The IXCMSGOX and IXCMSGIX macro interfaces are the successors to IXCMSGO and IXCMSGI macro interfaces. IBM suggests using IXCMSGOX and IXCMSGIX to send and receive messages between XCF group members, but in most cases you can continue to use IXCMSGO and IXCMSGI. An XCF group member application program needs to be changed to use IXCMSGOX or IXCMSGIX when functions provided by these macro interfaces are wanted.

What Is a Message?

A message is any piece of information that you want to transmit in an active group from one member to another. The data that makes up the message is of interest to the multisystem application only and not to z/OS.

A message can vary in length up to 128M bytes and resides in the storage area of the sending or receiving member. The data that makes up the message can reside in a single buffer or in multiple buffers. An additional 32 bytes of control information can be associated with each message.

Using the IXCMSGOX Signaling Service

The IXCMSGOX service allows your multisystem application to:

- Send messages, including those up to 128M bytes in length
- Broadcast a message to members of a group
- Request that XCF consolidate responses to a message
- Provide for ordered delivery of messages
- Specify a timeout value for message delivery.

An overview of each of these services follows.

Sending and Receiving Messages

Messages can be sent from one member to one or more other members as follows:

- A member sends a message by invoking the IXCMSGOX macro.
- The system gives control to the message user routine of the member that is to receive the message and passes the member a parameter list containing information about the message to be received. To receive the message the receiving XCF member must be active and must have provided the appropriate message user routine.
- The message user routine invokes the IXCMSGIX macro to receive the message.

Figure 4 on page 27 illustrates the process of sending and receiving a message. In this figure, MEM1 sends a message to MEM2 in the same XCF group, but on a different system.

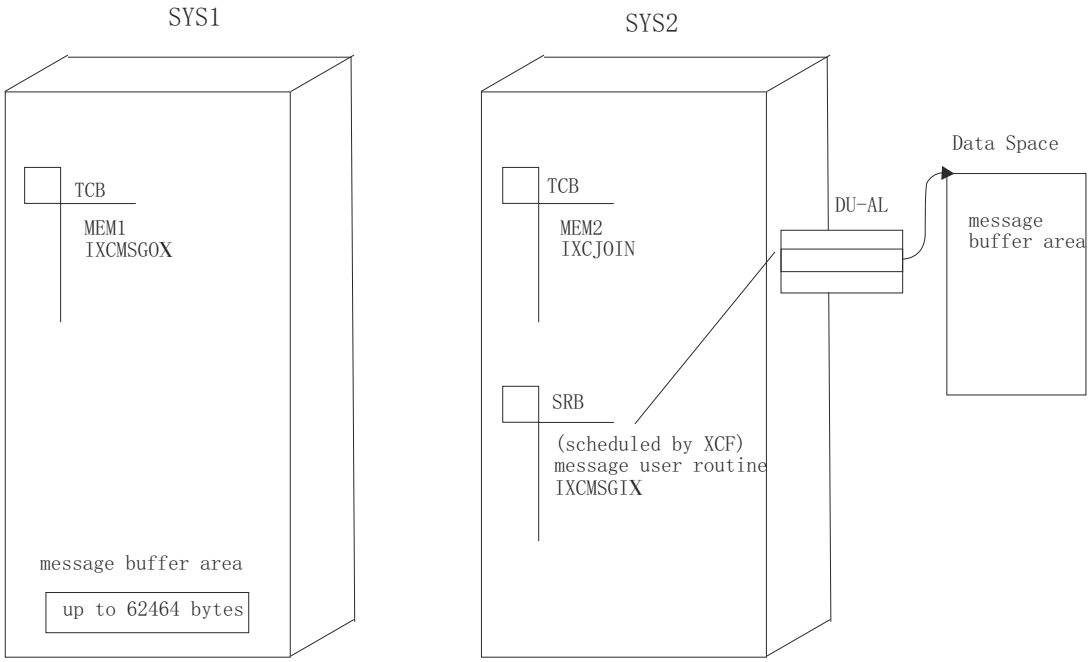


Figure 4. Sending a Message from One Member to Another

Sending Large Messages

Prior to OS/390® Release 8, the maximum length of a message was 62464 (61K) bytes. Release 8 and higher provide support for the delivery of “large messages”, where a large message is defined to be one

that is greater than 61K bytes up to a maximum of 128M bytes. XCF imposes the following restrictions when sending large messages:

- Both the sending and the target systems must be running OS/390 Release 8 or higher.
- Both the sending and the target members must have specified when they joined the XCF group that they supported the large message delivery protocol.

The IXCMSGOX service has been extended to support the large message delivery protocol. There is no change to the IXCMSGIX service for receiving the large messages. Specific IXCMSGOX requirements for large message delivery are the ability of XCF to access the message-related storage even after returning to the IXCMSGOX caller and an enforced need for a timeout value.

It is the responsibility of the application using large message delivery to determine whether a particular target member is capable of receiving such a message. To determine whether a member is able to receive a large message, use the IXCQUERY service. IXCYQUAA contains fields that indicate whether the member resides on a system that supports large message delivery (QUAMPROGT61KDELIVERY) and whether the member specified that it was capable of large message delivery when it joined its XCF group (QUAMPROGT61KMSG).

IXCMSGOX returns reason code IXCMSGORSNTARGETMAXMSGLEN61K when either the target member or the system on which the target member resides does not support messages larger than 61K.

Saving and Discarding Messages

A member can save a message for later processing or discard a message by invoking the IXCMSGC macro. A member can also invoke the IXCMSGC macro to call a user routine to process a message that was previously saved.

Broadcasting a Message to Members of a Group

XCF broadcast is a function that allows a member to send message data and the message control information to multiple target members. The member can specify the target members that are to receive the message in a user-defined table that contains the member tokens of the target members. The number of target members that the sender can specify depends on the maximum number of members per group that is currently supported in the sysplex.

IXCMSGOX support in z/OS V1R2 adds new options ALL and OTHER. ALL specifies that XCF is to send the message to all active members of the group. OTHER specifies that XCF is to send the message to all active members excluding the sender.

The broadcast function of XCF is not atomic. An atomic broadcast would guarantee that if one target member receives the message then all target members receive the message. The XCF broadcast function allows some members to receive their copy of the message while other target members might not receive their messages. There are instances when XCF is unable to deliver a message, such as when the sender has specified a timeout value that expires before message delivery (see “Specifying a Timeout Value for Message Completion” on page 40).

Consolidating Responses to a Message

A sender can request a response to a message and specify that XCF is to manage the collection of responses. The target member must have indicated its ability to participate in response collection by specifying CANREPLY=YES when it invoked IXCJOIN to join the group.

When XCF invokes a member's message user routine to receive a message, the message exit user routine parameter list (MEPL) contains an indication that the sender has requested that XCF consolidate responses to the message. To respond, the member:

- Uses the IXCMSGOX macro to send a response to the sender.
- Identifies the sender by specifying that the response is to be sent to the message “originator”.
- Identifies the message by specifying a “response ID” passed in the MEPL.

The system then

- Collects the response and holds it until all responses (if applicable) to the message are received.
- When all responses are collected, gives control to the originator's message notify user routine, passing it a parameter list containing information about the responses received.

The message notify user routine invokes the IXCMSGIX macro to receive the response(s) or the IXCMSGC macro to save or discard them.

Providing Ordered Delivery of Messages

Ordered message delivery means that messages are presented for input to the receiving member's message user routine in exactly the same order that they were accepted by the XCF Message Out service. Ordered message delivery is only provided between a particular pair of members; the sequencing is performed for a particular sender/receiver pair. The sender must ensure that the message-out requests are made in the desired order. The sender should wait for successful return from the XCF Message Out service before initiating the next ordered request for the same sender and receiver pair.

Specifying a Message Timeout Value

A timeout value is the amount of time a sender is willing to wait for its IXCMSGOX request to complete. When the timeout value has been exceeded, XCF can notify the sender and discontinues trying to send the message.

Using the IXCMSGOX Macro

Use the IXCMSGOX macro to specify the following:

- The sending member
- The target member or members
- Information about the storage area(s) in which the message resides
- Whether the application supports allowing XCF to access the message data asynchronously to the IXCMSGOX call
- Delivery options for the message
- Whether XCF is to manage the collection of responses to a message
- Whether message completion notification is necessary.

Identifying the Sending Member

To identify the XCF member sending the message, specify your member token using the MEMTOKEN parameter. You receive your member token as output when you issue the IXCJOIN macro to join an XCF group. You can also obtain your member token by:

- Issuing the IXCQUERY macro. The token is returned in the QUAMTOKN field of the answer area mapped by IXCYQUAA, which is used for both IXCJOIN and IXCQUERY.
- Using the target member token (MEPLTARGETMEMTOKEN) passed in the version 1 level of the MEPL when responding to a message.

Identifying the Target Member or Members

To identify the one or more members to receive a message, use the member's member token. You can obtain the member token in a variety of ways. Some of the options are:

- Issue the IXCQUERY macro to obtain information about the target member. The token is returned in the QUAMTOKN field of the answer area mapped by IXCYQUAA.
- Save the member token that your group user routine receives when it gets control to notify you of a change to that member's state. The token is passed in the GEPLMTOK field of the parameter area mapped by IXCYGEPL.

- Save the member token that your message user routine receives when it gets control to receive a message from that member. The token is passed in the MEPLSRCE field of the parameter area mapped by IXCYMEPL
- Establish a table, accessible to all members of the XCF group, where each member stores the token it receives from IXCJOIN.
- Save the member token that your message notify user routine receives when it gets control to notify you of message completion. The token is passed in the MNPLMEMTOKEN field of the parameter list mapped by IXCYMNPL.
- Save the member token that your message notify user routine receives when it gets control to notify you that response collection has completed. The token is passed in the MNPLTRRESPSRCE field of the parameter list mapped by IXCYMNPL.

Identifying a Single Target Member

To identify a single target member, specify its member token using the IXCMSGOX TARGET parameter.

Identifying Multiple Target Members using a Table

To identify multiple target members, create a table containing the member token of each member that is to receive the message. Use the IXCMSGOX SENDTO=GROUP,MEMBERS=TABLE option and identify the table with the TARGETS parameter.

Programming Note — Single Target Member Processing

There is a difference between sending a message to a single target member and broadcasting a message to a list of one target member. For example, if the one target member is not active, the result will vary as follow:

- For a send to a single target member that is not active, XCF rejects the send request with a failing return code.
- For a broadcast to a list of one target member that is not active, XCF returns a warning return code to the sender, but does not reject the request.

Creating the Table of Target Members

The table containing the member tokens of the target members is an array of entries. Each entry has the same fixed size (specified by NEXTTARGETOFF) and can contain data other than the target member token. The number of entries in the table is specified with the #TARGETS keyword. XCF iteratively processes the table for the specified number of entries, sending the message to the member indicated by each member token in the entry.

To maintain a fixed size table that allows for members to join and leave an XCF group, XCF ignores member tokens with a value of X'0'. The sender then can use these “slots” at a later time for insertion of a new member token. The sender is assured that the table of target members has a one-to-one correspondence with any other XCF table constructed for this request. Be aware however, that a table containing a significant number of null entries can cause additional system overhead.

Maintaining the Integrity of the Table of Target Members

The member invoking the broadcast service is responsible for maintaining the integrity of the TARGETS table until the message-out service returns. If a table changes while being processed by the message-out service, the system might send the message to a different set of target members than expected. Also, the content of the entries in tables that XCF constructs for this request might no longer correspond to the contents of the entries in the TARGETS table.

Sending a Message to Active Group Members

With z/OS V1R2, support is added to IXCMSGOX to allow an application to broadcast a message to its group members. Use the IXCMSGOX SENDTO=GROUP,MEMBERS=ALL option to send the message to all

active members of the group. Use the IXCMSGOX SENDTO=GROUP,MEMBERS=OTHER option to send the message to all active members of the group excluding the sender.

The set of active members is determined in a way that is functionally equivalent to the set that would be returned by an IXCQUERY REQINFO=GROUP,GRPNAME(sender'sgroupname),REQTYPE=IMMEDIATE.

When using MEMBERS=ALL or MEMBERS=OTHER, the application may not necessarily know the list of members that XCF uses. For example, a simple successful broadcast (one without XCF-managed response collection and without notification) would simply receive RC=0. The application would not be told the list of members that XCF used. If it is important for the application to know the exact set of members, they will need to provide their own table or specify keywords that will cause their message notify exit to be driven.

Identifying the Message Data

The data that is to be sent as one message can reside in one contiguous storage area or in multiple, discrete storage areas. If in discrete storage areas, the message can be organized as a queue of elements or as a table of elements. An element can contain the text of the message or pointer to locate the message text. IXCMSGOX keywords allow you to describe how the message data is organized so that XCF can access the data to be sent to the target member.

Note that there is no requirement for messages to be sent and received using the same number of buffers or the same buffer format. However, XCF members that communicate with each other must know the format in which they are to receive message data so they can interpret the message correctly.

Sending Message Data Using a Single Buffer

You can send up to 62464 bytes of message data (or up to 128M bytes for large messages) in a single buffer you specify with the MSGBUF parameter. Use the MSGLEN parameter to indicate the size of the message.

Each message buffer can be in your primary address space, in an address space accessible through your dispatchable unit access list (DU-AL), or in a common area data space.

Sending Message Data Using Multiple Buffers

This topic describes the formats you can use to pass message data to IXCMSGOX in multiple buffers. Illustrations of the formats are shown in the figures Figure 5 on page 33 through Figure 8 on page 35.

Use the ELEMFORM parameter to indicate how the message is maintained in multiple data buffers. Create either a queue (ELEMFORM=QUEUE) or a table (ELEMFORM=TABLE) of message data elements, each representing a buffer containing message data.

- **Specifying the Message Data Elements**

  Message data elements contain:

  – Either:

    - A buffer containing message data
    - A pointer to a buffer containing message data.

  – The length of the buffer (optional)

  – The ALET to qualify the address of the buffer if message data elements contain pointers to the buffers (optional).

  Buffer lengths and ALETs can be passed separately as described below instead of including them in each message data element.

- **Specifying the Location of Each Buffer**

  Specify the location of the buffer or buffer pointer within each message data element using one of the following parameters:

- If the message data elements contain the buffers, use the PARTOFF parameter to specify the offset of the buffer area from the start of each message data element.
- If the message data elements contain pointers to the buffers, use the PARTPTROFF parameter to specify the offset of the buffer address from the start of each message data element.

- **Specifying the Location of Each ALET**

  Specify the ALETs to qualify the buffer addresses using one of the following parameters:

  - The PARTALET parameter to specify a single ALET to qualify each buffer address.
  - The PARTALETOFF parameter to identify a location in each message data element that contains the ALET to qualify the associated buffer address.
  - The PARTALETTBL parameter to specify a separate table of ALETs.

- **Specifying the Size of Each Buffer**

  Specify the lengths of the buffers using one of the following parameters:

  - The PARTLEN parameter to specify a single length for all buffers.
  - The PARTLENOFF parameter to identify a location in each message data element that contains the length of the associated buffer.
  - The PARTLENTBL parameter to specify a separate table of buffer lengths.

- **Specifying the Location of the Next Message Data Element**

  If you have a table of message data elements, use the NEXTOFF parameter to specify, in each element, the location of the next element. NEXTOFF contains the length in bytes of each entry in the table.

  If you have a queue of message data elements, use the NEXTPTROFF parameter to specify, in each element, the pointer to the next element.

  IXCMSGOX processes message data elements in consecutive order, copying message data from each buffer until either the number of bytes copied matches the specified buffer length or the entire message has been copied.

  Processing of message data continues until one of the following occurs:

  - All message data has been copied, as determined by the value specified by the MSGLEN parameter.
  - IXCMSGOX has processed the number of buffers specified by the #MSGPARTS parameter.
  - IXCMSGOX has reached the end of the queue of message data elements as specified by the ENDOFQUEUE parameter or its default.
  - IXCMSGOX finds more than 65536 consecutive buffers of length 0 and does not know how many message parts to search because you did not specify the #MSGPARTS parameter. IXCMSGOX assumes an error has occurred. The message is not sent and you receive a return code and reason code indicating the error.

Note that if the receiver is going to receive the message into multiple buffers and requires that the sender provide the length of each message part, the sending and receiving member must devise a protocol for transmitting this information. For instance, the length of each message part could be sent in the message data itself or as part of the message control data.

Examples of Message Data Element Formats for Multi-Buffer Messages

The figures Figure 5 on page 33 through Figure 8 on page 35 show examples of various message data element formats. Fields within each element need not be in any particular order. The examples show only a few of the possibilities.

Figure 5 on page 33 shows a queue of message data elements in which each element contains a buffer address and a pointer to the next element in the queue. All buffers reside in the same address space and are to be accessed using the ALET specified by the PARTALET parameter. All buffers are of the length specified by the PARTLEN parameter.

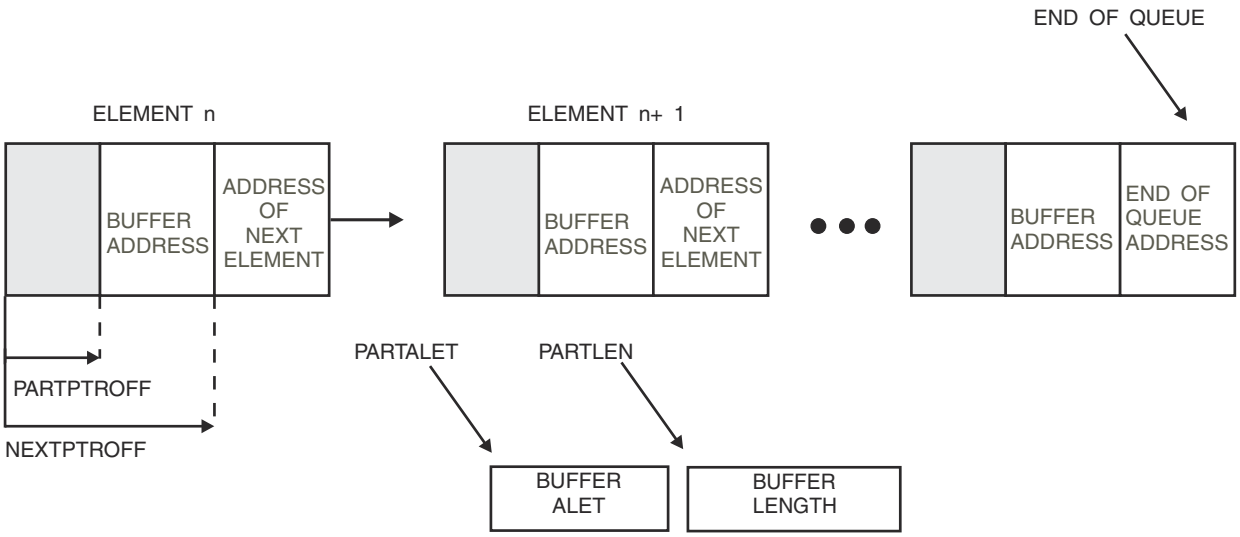


Figure 5. Example of Queue of Message Data Elements

Figure 6 on page 33 shows a table of message data elements in which each element includes the following information relating to the buffer it describes:

- The ALET to qualify the buffer address
- The length of the buffer
- The address of the buffer.

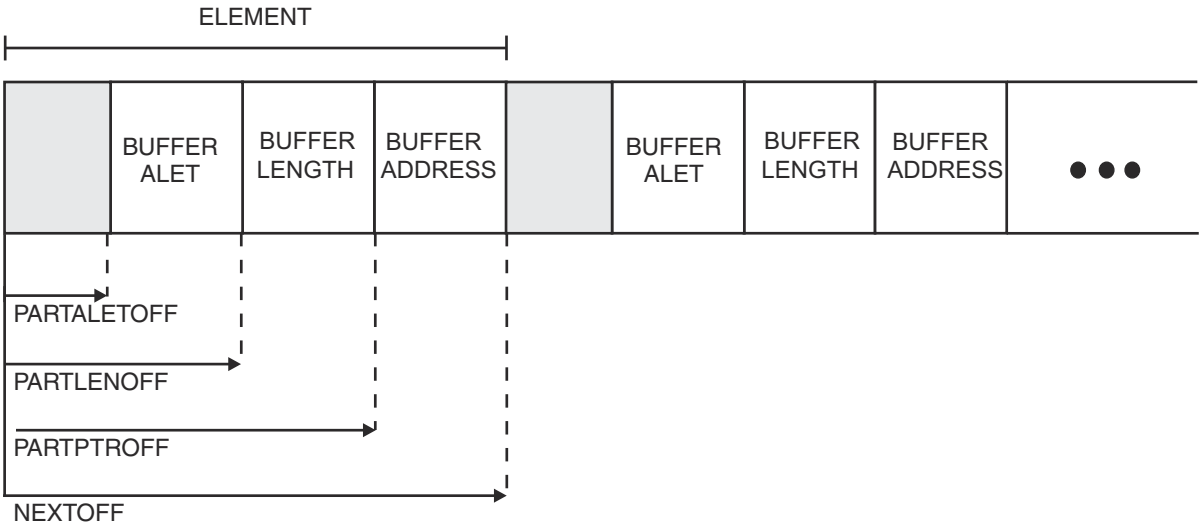


Figure 6. First Example of Table of Message Data Elements

Figure 7 on page 34 shows a table of message elements in which each element contains a buffer. No ALETs are specified because the buffers reside in the table itself. A separate table, specified by PARTLENTBL, contains the length of each buffer.

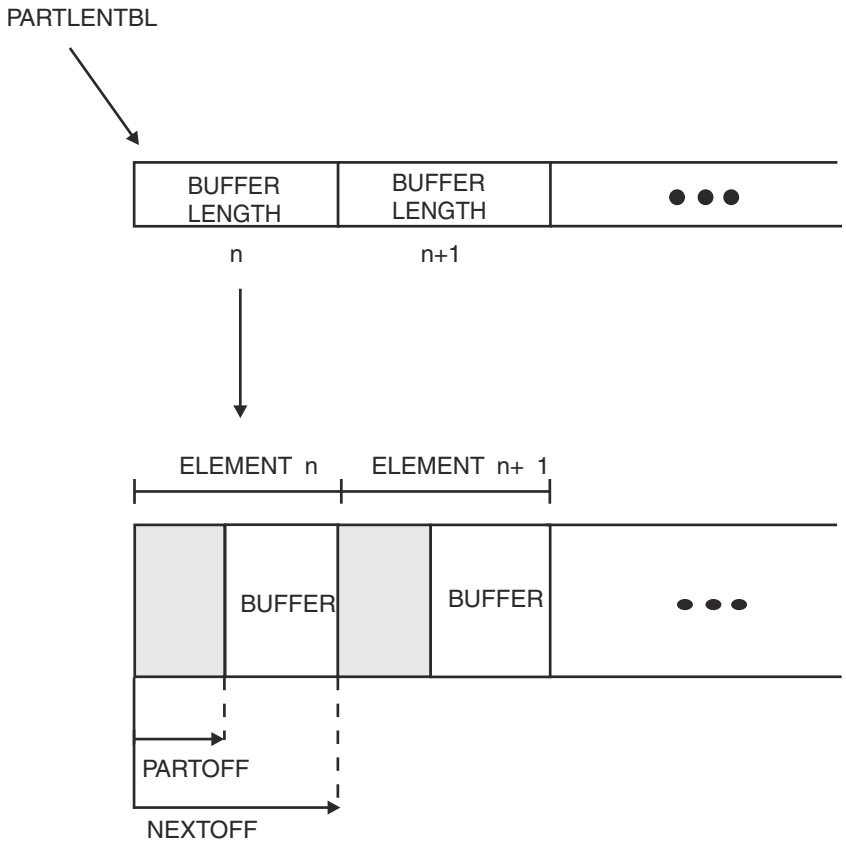


Figure 7. Second Example of Table of Message Data Elements

Figure 8 on page 35 shows a table of message elements in which each element contains a buffer address. A separate table, specified by PARTALETTBL, contains the ALETs to be used with each buffer address. A separate table, specified by PARTLENTBL, contains the length of each buffer.

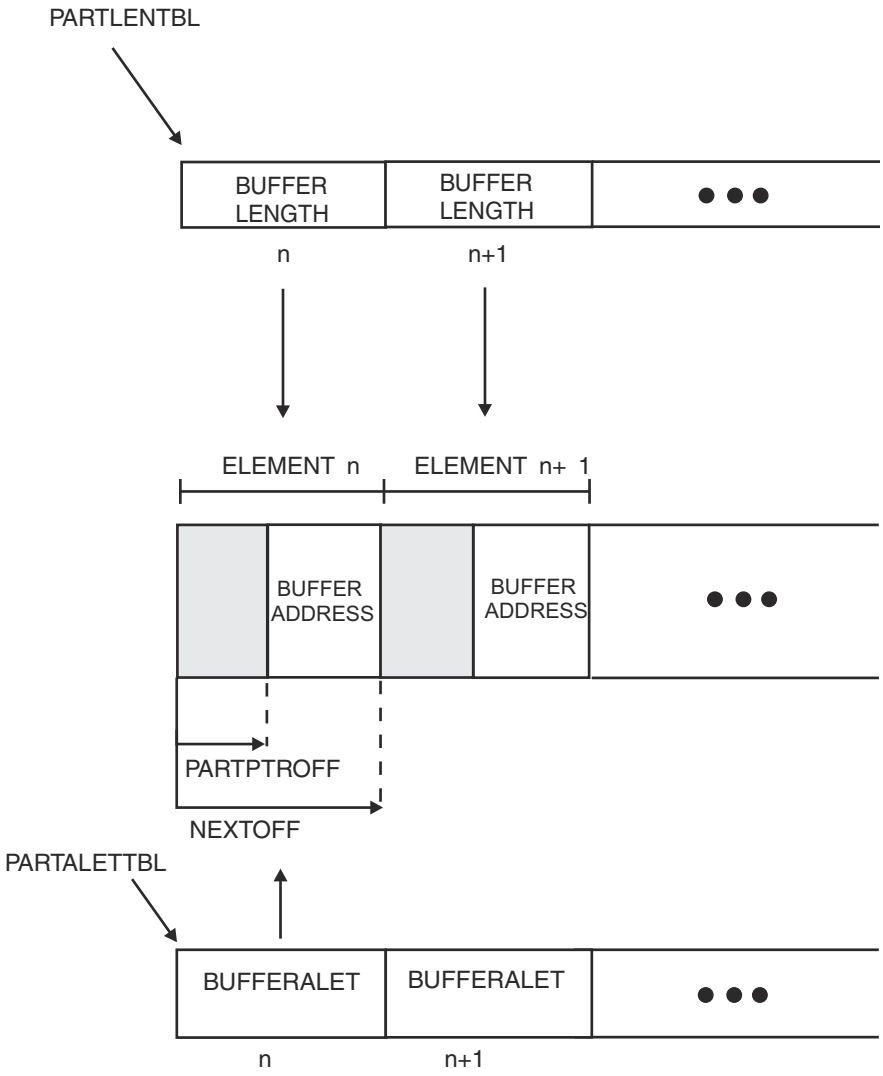


Figure 8. Third Example of Table of Message Data Elements

Specifying the Storage Key

Specify the storage key of the buffers containing the message data by using the MSGSTGKEY parameter. If you specify the storage key of your buffers, the system transfers data from your buffers using a MOVE WITH KEY (MVCK) instruction. The operation is successful only if the buffers are in the storage key specified or if the buffers are not fetch protected.

If you omit the MSGSTGKEY parameter, the system transfers the data from your buffers regardless of their storage key.

Supporting Asynchronous Message Access by XCF

XCF can access the storage area in which the message resides or that contains information identifying the message data either synchronously or asynchronously, as specified by the MSGACCESS parameter. The default access method, MSGACCESS=SYNC, implies that the storage is accessed synchronously with the IXCMSGOX request. When IXCMSGOX returns to the sender after accepting the message for delivery, XCF has already made a copy of the message data and the sender can dispose of the storage containing and identifying the message. Messages up to 61K (62464) bytes long can be sent using MSGACCESS=SYNC. XCF will always attempt synchronous access even if the sender requests the asynchronous protocol.

With MSGACCESS=SYNCSUSPEND, a request to send a message in the decimal range of 0 to 134,217,728 (128M) bytes long can be made synchronously. As needed, XCF suspends the current unit of work for a specified amount of time in order to complete accessing storage areas associated with message text. As

with MSGACCESS=SYNC, when IXCMSGOX returns to the sender, the sender can dispose of the storage that contained either the message or the queue or table entries that defined the parts of the message. MSGACCESS=SYNCSUSPEND is useful for senders that want to send messages that are greater than 61K in length, can tolerate having their unit of work possibly suspended, and want to release storage resources that contained message text and queue or table elements that defined parts of the message after IXCMSGOX returns to the sender (as opposed to waiting for the notify exit to be driven when the message completes).

Asynchronous access to the message data allows XCF to access the storage containing or identifying the message even after IXCMSGOX returns to the sender. Messages up to 128M bytes long can be sent using MSGACCESS=ASYNC. The sender must not dispose of the message-related storage until the message is completed. The return code provided by XCF indicates whether the sender must preserve the storage:

- With IXCMSGOX X'4', reason code X'410', the application must preserve the message and related storage areas. In this case, the notify exit is most likely responsible for freeing the message and related storage when appropriate.
- With IXCMSGOX return code X'0', the application can dispose of the message storage because XCF is finished with the message and its related storage. If the sender disposes of the storage areas, the application must coordinate responsibility for freeing the message-related storage between the sender and the notify exit. The notify exit is driven regardless of whether the sender receives return code X'0' (and disposed of the message storage) or return code X'4' reason code X'410' (and preserved the message storage). The notify exit can distinguish the two cases by checking the MNPLMSGOASYNCMSGACCESS flag.

For any other IXCMSGOX return and reason codes, the sender can dispose of the storage as with MSGACCESS=SYNC.

MSGACCESS=ASYNC or MSGACCESS=SYNCSUSPEND is required when sending a message longer than 61K bytes. If MSGACCESS=SYNC is specified for a message longer than 61K bytes, the message is rejected.

Specifying Message Control Information

In addition to the message buffers, IXCMSGOX allows you to pass 32 bytes of user-defined message control information using the MSGCNTL parameter. The message control information is included in the message exit parameter list (MEPL) passed to the receiving member's message user routine. If message control information is provided when sending a response, the information is included in the message notification parameter list (MNPL) passed to the originator's message notify user routine.

You can use the message control information area to hold whatever information you like. XCF members planning to send messages to one another should agree on how the message control information is to be used, for example to:

- Indicate where to store the message if multiple users will share access to the message data
- Identify the format of the message or message parts so the message data can be accessed correctly
- Pass additional information needed to receive the data
- Identify the message so it can be acknowledged by the receiver
- Hold the message data, if the message is 32 bytes or less.

Specifying Message Response Collection

A sender can specify whether XCF is to manage the collection of responses to a message with the IXCMSGOX GETRESPONSE parameter.

- GETRESPONSE=NO indicates that XCF is not to manage response collection for the message. Either there is no response expected or the member is managing its own response collection.
- GETRESPONSE=YES indicates that a response is expected and XCF is to manage the collection of responses.

A sender can request that XCF is to manage the collection of responses to a message sent to a single target or broadcast to many targets. The target member can recognize that XCF is managing responses (by checking the MEPLNEEDSRESPONSE field in the MEPL) and reply in a way that allows XCF to do so. XCF gathers all responses and when all that are expected have been collected, schedules an SRB to call the sender's message notify user routine. The message notification parameter list (MNPL), mapped by the IXCYMNPL macro, contains information about the collected responses. In the message notify user routine, the member can issue the IXCMSGIX macro to receive each response.

The sender can determine whether the target member is able to participate in response collection by issuing the IXCQUERY macro requesting member information.

- The QUAMPROCANREPLY field indicates whether the target member specified IXCJOIN CANREPLY=YES when it joined the group.
- The QUAMPRORESPONSECOLLECTION field indicates whether the target member resides on a system at a level that supports XCF-managed response collection.

When response collection is complete, XCF notifies the sending member by calling the message notify user routine specified by the IXCMSGOX NOTIFYEXIT parameter when the message was sent.

When Does XCF Not Expect a Response?

XCF does not expect to receive a response from a target member that is running on a system that does not support message response collection or from a target member that specified (or defaulted to) CANREPLY=NO when it invoked IXCJOIN to join its group. A member must specify IXCJOIN CANREPLY=YES to indicate to XCF that it can participate in the XCF response collection protocol.

When expecting a response from a target member, XCF automatically detects situations for which the target member is unable to supply a response. For example,

- The target member resides on a system that is removed from the sysplex.
- The target member becomes not active.

In the latter case, XCF will wait for the arrival of any response that was “in flight” when the member became not active. A response is in flight if XCF has initiated the send of the response. (IXCMSGOX returned return code X'0'. Note that if IXCMSGOX returns return code X'4', that code indicates that the send is pending, but has not yet been initiated nor is in flight. At some later time, XCF might initiate the send.)

Note that, even though a member did not specify CANREPLY=YES when joining the group (and therefore XCF is not expecting a response), the target member is not prevented from responding. However, XCF will not wait for the response to arrive. If the response arrives before message completion occurs, XCF will include the unexpected response with those presented to the sender in the MNPL; otherwise, the unexpected response data is not included in the MNPL information.

Providing a Response for XCF Response Collection

When a target member's message user routine receives a message for which XCF is managing responses, the MEPLNEEDSRESPONSE flag in the message exit parameter list (MEPL) is set and a response identifier (MEPLRESPONSEID) is provided. The target member uses the response identifier when responding to the message with IXCMSGOX SENDTO=ORIGINATOR. The response identifier allows XCF to identify the message for which the response is intended. You must specify the SENDTO=ORIGINATOR keyword when sending a response that is to be collected by XCF. Otherwise, XCF is unable to identify the message as a response to be collected, and the will deliver the message as an ordinary message.

A target member can respond to the message in the message user routine or under some other unit of work. XCF accepts only the first response that it receives on the originating system. Responses can be delivered in any order, so there is no guarantee that the first response sent will be the first to be received.

Providing a Response on Behalf of Another Member

XCF does not require that the target member be the member that responds to a message. Any member of the XCF group can use the response identifier to respond to the message. However, XCF **expects**

the original target member to be the one to respond. If that target member becomes inactive, XCF no longer expects a response from that member. When no further responses are expected, XCF considers the message to be complete. Any response that arrives after the message completes is discarded.

If your application protocol has some other member send a response on behalf of the original target member, you must be aware that XCF has no ability to recognize this situation. If the original target member becomes inactive, the timing could be such that the message will complete (because XCF no longer expects the response) before the other member's response arrives. The message notify user routine would be missing a response even though the member that your application expects to respond is active and still has plenty of time remaining before the message times out.

Calling the Message Notify User Routine

XCF presents the collection of responses to the message notify user routine of the originating member, in the message notification parameter list (MNPL).

MNPL maps a table that contains information about the response, if any, that was collected from each target member. When more than one target member exists, the entries in the table are in one-to-one correspondence with the entries in the table of targets specified (with the TARGETS= parameter, members=all or members=others) when the original broadcast message was sent. The one-to-one correspondence applies to all entries in the table of targets, including any entries containing a null member token. If any target table entry contains a null member token, the corresponding entry in the MNPL identifies that state with one of the following flags:

- MNPLTOSENDSKIPPED
- MNPLTRSENDSKIPPED

When either of these flags is set to B'1', the corresponding target member token is hexadecimal zero, indicating that the sender wanted to skip an entry in the message-out target table.

If no response was received, XCF provides a code in the table. (The MNPLTRRESPCODE field indicates why the expected response was not received.) Note that failure to receive a response does not imply that the target member did not receive its copy of the message. Some reasons why an expected response might not be received are:

- The send of the message-out request has yet to be initiated.
- The target member is not coded to support XCF-managed response collection.
- The target member is not running on a system that supports XCF-managed response collection.
- The system on which the target member resides was removed from the sysplex before the response was sent.
- The target member became inactive before it could send the response.
- The message was not sent because the target member is not active.

If a response was received, use the XCF Message In service (IXCMSGIX) to retrieve the data or the XCF Message Control service (IXCMSGC) to save or discard the data. Note that IXCMSGIX and IXCMSGC must be invoked from within the message notify user routine.

Response collection is considered complete when all the expected responses have arrived, when IXCMSGC is used to force completion, or when the timeout value expires, whichever occurs first. (See “Specifying a Timeout Value for Message Completion” on page 40 for a description of the timeout value.)

Specifying the Delivery Options for Messages

The XCF signaling service supports options that allow ordered message delivery between members and the specification of a time limit within which message delivery must complete.

- Ordered message delivery means that messages are presented for input to the receiving member's message user routine in exactly the same order that they were accepted by the XCF Message Out service. Ordered message delivery is only provided between a particular pair of members; the sequencing is performed for a particular sender and receiver pair. The sender must ensure that the message-out requests are made in the desired order. The sender should wait for successful return from

the XCF Message Out service before initiating the next ordered request for the same sender/receiver pair.

- A timeout value is the amount of time a sender is willing to wait for its IXCMSGOX request to complete. XCF can notify the sender when the timeout value has been exceeded and the message is no longer available for processing.

  The timeout value can also be used to indicate that XCF is to queue any messages that cannot be sent because of a lack of XCF resources (such as buffers or signaling paths). See "Queueing Messages for Later Delivery" on page 42 for a description of this service.

Requesting Unordered Delivery

Unordered delivery (IXCMSGOX DELIVERMSG=UNORDERED) is the default mode and means that messages can be delivered in any order, regardless of the order in which they were sent. The delivery of unordered messages might even be interspersed among messages for which ordered delivery was requested.

Unordered delivery occurs if a message is sent to a target member that resides on a system that does not support ordered message delivery.

Requesting Ordered Delivery

Ordered delivery (IXCMSGOX DELIVERMSG=ORDERED) can be used to send messages between a particular pair of members in independently ordered streams. Both the sending and the receiving member must reside on systems that support ordered message delivery. If a message is sent to a member on a system not at the required level, the message is delivered as an unordered message. To avoid sending messages to a downlevel target member, the sender can issue IXCQUERY and examine the QUAMPROORDEREDDELIVERY field to determine if the target member resides on a system that supports ordered message delivery.

The sender can define multiple streams of ordered messages. Identify the stream to which a message belongs with the IXCMSGOX STREAMID parameter. The messages in any given stream are delivered in the order in which they were accepted for delivery; the messages in each stream are delivered independently from those in another stream.

To process ordered messages, XCF passes control to the message user routine. The routine must either receive the message or indicate that XCF is to save the message in XCF-managed storage. If the routine does neither, XCF discards the message when the message user routine gives up control. Once the message user routine gives up control, the next ordered message is eligible for delivery (regardless of whether the previous message was received, saved, or discarded). Note that the message user routine needs to process the message and return to XCF as quickly as possible. If the message user routine is suspended for any reason while processing an ordered message, the routine will delay the delivery of subsequent messages in the ordered delivery stream. Such delays could allow a large backlog of messages to accumulate. This, in turn, might cause XCF to reject message-out requests (local or on remote systems) that are targetted to this member.

Note that each stream is processed as a separate entity, so that a delay in processing messages in one stream does not affect the processing of messages in another stream nor of unordered messages.

Within the sequence of ordered messages to be delivered, "gaps" in the sequence might occur. These gaps could be the result of either XCF or application processing. In both cases, the multisystem application must be able to recognize that a gap has occurred. Two examples are:

- XCF processing

  A member sends messages A1, A2, A3 in order. Suppose that messages A1 and A3 are successfully transferred to the target system. XCF delivers message A1, but delays delivery of message A3 because message A2 has not yet arrived at the target system. In the meantime, suppose that the sending member decided to cancel message A2 before XCF was able to initiate the send or that the sending system failed before the transfer of message A2 was completed. XCF recognizes that message A2 **cannot** be delivered to the target system and so delivers message A3.
- Application processing

A member sends messages A1, A2, A3 in order and all messages are transferred successfully to the target system. XCF delivers message A1, which is processed successfully by the receiving member. XCF then delivers message A2, but the receiving member fails to process the message correctly. XCF then delivers message A3. From XCF's perspective, message A2 had already been delivered, but from the application's perspective, message A2 is missing.

If the presence of gaps in the sequence is not tolerable, the application must provide its own protocols to react to a gap's occurrence. For example, the application might need to discard incoming ordered messages until the target member resynchronizes with the sender.

There could be multiple instances of a message user routine running, each under a different unit of work. It is not predictable which instance of the message user routine or which work unit will be given control to process the next ordered message in the stream.

Understanding Message Completion

The XCF signaling services use the concept of message completion. Message completion occurs in the following circumstances:

- **Any** message is considered complete if it times-out, if the XCF Message Control Completion service (IXCMSGC) is used to force its completion, or in the case of a send to a single target member, the target member becomes not active.
- A message **without response** is considered complete as soon as XCF has initiated the send of the message. For a message broadcast to multiple targets, the message is considered complete when XCF has initiated the send of the message to every valid target member. Message completion for a message without response implies nothing about whether the message was actually delivered to a target member or whether the message data was transferred to the system on which the target member resides.

  Note that a message is considered complete even if the initial send of the message fails and XCF has to resend the message.
- A message **with response** is considered complete when its expected response arrives. For a message broadcast to multiple targets, the message is considered complete when the expected response from each target member arrives.

  XCF does not expect a response to a message from a member residing on a system that does not support XCF-managed response collection, or from a member that did not specify CANREPLY=YES on IXCJOIN when joining the group. Note that if a response to a message is not expected, the message is considered complete.

Message completion might trigger the invocation of a user routine based on the requirements specified on the XCF Message out service. See “Requesting Notification of Message Completion” on page 41 for information about the IXCMSGOX parameters to specify to be notified of message completion.

Specifying a Timeout Value for Message Completion

With the XCF Message Out service, the sender can specify a timeout value after which the message is to be considered complete. The timeout value is the number of seconds the sender is willing to allow for the message-out request to complete. XCF declares the message to be complete if it does not otherwise complete during the timeout interval. The sender is notified of the message completion in the manner requested when the message was sent. The system discards any response that is received after the message is considered complete.

A nonzero timeout value is required when MSGACCESS=ASYNC is specified on the Message Out service. Users sending large messages should allow sufficient time for the message to be sent. As a general rule, allow at least one millisecond per 4K of message data for data transfer time. Multiply the value by the number of targets and add additional time for system delays, such as spin loops, dumps, or response collection.

When selecting a timeout value, consider the following:

- The timeout value should conform to the service goals established for the application.

- The timeout value should be approximate. XCF does not guarantee that notification of timeout completion occurs precisely at the specified interval.

Consider also the rate at which the member processes its messages and the size of the messages. If a large number of messages arrive at a rate greater than the system's ability to process them, the system might need large amounts of storage to maintain the backlog. If a broadcast with response to a large number of target members requires a large response from each member, the system might need large amounts of storage to maintain the responses. Although XCF uses pageable data spaces to manage the message traffic, your application must be aware of the system resource impact to the system and the sysplex.

Requesting Notification of Message Completion

With the XCF Message Out service, the sender can specify that the system is to provide notification when the message completes. The notification occurs automatically through the message notify user routine. Notification is relevant only if the message-out request is accepted for delivery. XCF does not provide notification for rejected message-out requests.

XCF maintains status information about the request. Even if notification is not requested, this information might be temporarily available to the XCF Message Control service.

Specifying Data to be Associated with the Message

The sender optionally can specify eight bytes of user data to be associated with the message-out request with the IXCMSGOX USERDATA parameter. When the message completes, the system passes a copy of this user data in the MNPL presented to the message notify user routine. The field MNPLMSGOUSERDATA contains this data. You can also use the IXCMSGC QUERYMSG service to retrieve this data. The field MQAMOSUSERDATA in the answer area contains this data.

Identifying the Message

With the IXCMSGOX RETMSGOTOKEN parameter, the sender optionally can specify that the system is to return a 16-byte token that can be used to identify the message to the XCF Message Control service.

Qualifying the Notification Request

With the NOTIFYIF parameter of the XCF Message Out service, the sender can specify the circumstances under which notification should occur. The sender can choose to be notified regardless of how the message completes (either successful or failed) or to be notified only if the message is considered to have failed.

- A completed message without response is considered successful if XCF initiated a send to every possible target member.
- A completed message with response is considered successful if a response was received from every possible target member.
- If neither case applies, the message is considered to have failed with regard to message completion.

When the message completes, XCF determines whether the message succeeded or failed. If the sender requested notification for the type of completion that occurred, XCF begins the requested type of notification. Otherwise, all information related to the request is discarded and no notification occurs.

Specifying the Notification Process

With the NOTIFYBY parameter of the XCF Message Out service, the sender specifies that XCF is to give control to the message notify user routine upon completion of the message. When message completion occurs, XCF schedules an SRB to the address space that was primary when the sending member invoked IXCJOIN to join the XCF group. The message notify user routine is called while running under this SRB (comparable to how the message user routine runs). The message notify user routine can be specified either when the member joins the XCF group or when the member invokes the Message Out service for the request.

On entry to the message notify user routine, register 1 contains the address of a message notification parameter list (mapped by IXCYMNPL). The parameter list contains information about the message-out request, a table of information describing the result of the send to each target, and if relevant, information about each response. From within the message notify user routine, an individual response is received by invoking the XCF Message In service to copy the text of the response message from XCF-managed storage to user-supplied storage. See “Coding a Message Notify User Routine” on page 59 for detailed information about using a message notify user routine.

It is the responsibility of the sender to ensure that completed messages are processed in a timely manner so as to avoid undue consumption of resources within the sysplex. Failure to do so might cause XCF to reject message-out requests with return code X'0C', reason code X'0C', indicating that there is no user message space left. XCF might also reject a Save Message request of the XCF Message Control service.

Handling Error Conditions

When you issue IXCMSGOX to send a message, it is possible that XCF will be unable to deliver the message, either because message buffers are temporarily unavailable or because signaling paths to the target member's system are temporarily unavailable.

- IXCMSGOX returns return code X'0C', reason code X'04' when XCF has used up the installation-specified amount of message buffer space allotted for queueing the signals targeted to the destination system or if the target member is "message isolated" and the sending member did not specify MSGISO=MSGORSN when it invoked the IXCJOIN macro to join its group.
- IXCMSGOX returns return code X'0C', reason code X'08' when XCF has lost all signaling path(s) to the destination system.
- IXCMSGOX returns return code X'0C' reason code X'3C' when the target member is "message isolated". XCF isolates a member when it fails to make adequate progress with respect to the processing of its messages. Message isolation helps keep problematic group members from impeding the delivery of messages to other members. While a member is isolated, XCF delays or rejects messages targeted to that member. When a sending member has a message delayed or rejected because the target member appears to be isolated, the sending member is said to be "impacted".

  This reason code applies only if the sending member specified MSGISO=MSGORSN when the IXCJOIN macro was invoked to become an active group member. If MSGISO=NONE is specified (or taken as the default), a request to send a message to target member that is "message isolated" is rejected with return code X'0C' reason code X'04' (no buffer).

As a general rule, these are temporary conditions for which recovery routines are not required. In both instances, you can try sending the message again after a short period of time (such as ten milliseconds).

- If the condition is a message buffer shortage, it might be necessary for the installation to allocate additional message buffers.
- If signaling paths are unavailable, something might be wrong with the target member's system or with its signaling paths.

  You can keep trying periodically to send the message until your group user routine receives a system-status-update-missing notification (event type GESYSSUM). You could then try again when your group user routine receives the system status update resumed notification (event type GESYSSUR). See “Events that Cause XCF to Schedule a Group User Routine” on page 77 for an explanation of the GESYSSUM and GESYSSUR events.

Queueing Messages for Later Delivery

With the IXCMSGOX support for the specification of a timeout value, you can request that XCF queue a message that might otherwise have been rejected because of a lack of XCF resources. Specifying a non-zero timeout value tells XCF to save the message in XCF-managed storage until it can be sent. IXCMSGOX returns a return and reason code to the sender to indicate that it has accepted and queued the message for delivery. Note that if you do not specify a non-zero timeout value, XCF will reject the message if resources are not available.

The queued message remains pending until either

- XCF automatically sends the message when the resource constraint is resolved.
- The message completes.

XCF notifies the sender of message completion as specified when IXCMSGOX was invoked.

Handling IXCMSGOX Requests When a Member Terminates

A member can voluntarily leave its XCF group by using the IXCLEAVE macro or the IXCQUIES macro. See "Disassociating Members from XCF" on page 116 for general information about member termination. If the member is using the IXCMSGOX service, XCF discards any completed IXCMSGOX requests that are pending presentation to a message notify user routine and any incomplete IXCMSGOX requests. The message notify user routine does not receive control for these messages.

If XCF has initiated a send for a message-out request, XCF continues to attempt delivery of the message. XCF does not guarantee that the message will be delivered, even if it has already initiated the send, because a system can be removed from the sysplex before a message is transferred to its target system. Despite the potential for non-delivery, a member might want to take steps to ensure that XCF has initiated the send of its messages before the member becomes (voluntarily) not active.

See "Handling Member Termination" on page 52 for a description of how XCF handles messages that might still be associated with the member that is terminating and a suggested method for ensuring that XCF has initiated the member's message out requests.

Using the IXCMSGIX Macro

The IXCMSGIX macro allows an active member of an XCF group to receive messages that were sent by another active member of the group and to be notified of XCF signaling related events for processing.

- To receive messages sent to your member, you must
  - Write a message user routine that invokes the IXCMSGIX macro
  - Specify the address of your message user routine using the MSGEXIT parameter when you issue the IXCJOIN macro to join the XCF group.

  The message user routine receives control when the message is ready for delivery. The message user routine can receive, save, or discard the message.
- To receive response messages collected by XCF for your member, you must
  - Write a message notify user routine that might invoke the IXCMSGIX macro
  - Specify the address of your message notify user routine using the NOTIFYEXIT parameter when you issue the IXCJOIN macro to join the XCF group or the NOTIFYEXIT parameter when you issue the IXCMSGOX macro to send a message.

  The message notify user routine receives control when message completion occurs for a message-out request that specified a broadcast or message response collection. The message notify user routine can receive, save, or discard an individual response and can also save or discard an entire collection of responses and message control information.

The IXCMSGIX user routines must be prepared to receive whatever data is sent by its peer members. Prior to OS/390 Release 8, the maximum length of a message that a member could send was 61K bytes. With Release 8, support for messages up to 128M bytes in length was added. To allow its peer members to send a message that is greater than 61K bytes, the receiving member must specify GT61KMSG=YES on its invocation of IXCJOIN to join the XCF group.

The IXCMSGIX service copies message data from XCF-managed storage to a user-specified storage area. The IXCMSGIX service can place the message data received from the IXCMSGOX service into either a single buffer or a user-specified number of buffers.

When your user routine (either a message user routine or a message notify user routine) receives control from XCF, GPR 1 points to a parameter list containing information about the message or signaling-related event to be received. The parameter list for the message user routine is mapped by the IXCYMEPL macro. The parameter list for the message notify user routine is mapped by the IXCYMNPL macro.

Identifying the Message to Be Delivered

Identify the message to be delivered with the TOKEN parameter of IXCMSGIX. The value of TOKEN is passed in the MEPL or MNPL parameter list — MEPLMSGITOKEN, the token for a message presented to a message user routine or MNPLTRMSGITOKEN, the token for a response message presented to a message notify user routine. Note that this message token is valid as input to IXCMSGIX while the user routine is in control, as long as the message has not been saved or discarded using the IXCMSGC service. When the user routine gives up control, or saves or discards the message using the IXCMSGC service, XCF invalidates the message token.

If the message is to be delivered on a system running a release prior to OS/390 Release 3, you can use the MSGTOKEN parameter to identify the message. The value of MSGTOKEN is passed in the MEPL parameter list — MEPLMTOK, the 32-bit token for a message presented to a message user routine. Use the TOKEN parameter instead of the MSGTOKEN parameter on systems running OS/390 Release 3. The TOKEN parameter is required when IXCMSGIX is issued within a user routine that gained control through the IXCMSGC Call Exit service or within a message notify user routine.

Determining the Length of the Message to be Received

Depending on the parameter list that is passed to the user routine, the length of the message to be received is obtained as follows:

Using the MEPL

Field MEPLMLEN in IXCYMEPL contains the length of the message. Use this information to determine how much buffer storage you need to accommodate the message being received. See "Coding a Message User Routine" on page 52 for information on writing a message user routine to issue the IXCMSGIX macro.

Using the MNPL

Field MNPLTYPE contains the type of signaling-related event notification being presented. The contents, and therefore the length, of IXCYMNPL can vary depending on the type of notification. See "Coding a Message Notify User Routine" on page 59 for information on writing a message notify user routine to issue the IXCMSGIX macro.

See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for more information about the IXCYMEPL and the IXCYMNPL mapping macros.

Determining Message Disposition

Within a user routine, an active member can receive or save a message.

- Use IXCMSGIX to receive the message and have XCF copy it into a user-specified storage area.
- Use IXCMSGC to save the message and have XCF copy it into XCF-managed storage.

If a message is neither received nor explicitly saved or discarded, XCF automatically discards the message when the user routine gives up control.

Specifying the Storage Key

Specify the storage key of the buffers to receive the message data by using the MSGSTGKEY parameter or by taking its default. If you specify the storage key of your buffers, the system transfers data into your buffers using a MOVE WITH KEY (MVCK) instruction. The operation is successful only if the buffers are in the storage key specified.

If you omit the MSGSTGKEY parameter, the system uses as a default the value of the PSW key at the time you issued the IXCJOIN macro.

Receiving Message Data into a Single Buffer Area

Use the MSGBUF parameter to specify the one contiguous buffer that is to receive the message. Please note, however, that **you must provide sufficient storage to receive the entire message**. If the storage

cannot hold the entire message, the system will either program check or overlay storage. Furthermore, you cannot reissue IXCMSGIX to obtain the message data that couldn't fit in the buffer when you issued IXCMSGIX the first time.

Receiving Message Data into Multiple Buffers

To have XCF distribute the message data into multiple buffer areas, you must ensure that those areas can be described in either a table structure or a queue structure. This topic describes the different formats you can use to receive message data from IXCMSGIX into multiple buffers. Illustrations of the different formats are shown in the figures Figure 5 on page 33 through Figure 8 on page 35. The formats and parameters for using multiple buffers are the same for both IXCMSGOX and IXCMSGIX. Note that there is no requirement that a message sent using multiple buffers be received using multiple buffers. The formats of the buffers used to send and receive a given message are completely unrelated.

If the receiver is going to receive the message into multiple buffers and requires that the sender provide the length of each message part, the sending and receiving member must devise a protocol for transmitting this information. For instance, the length of each message part could be sent in the message data itself or as part of the message control data.

To receive a message using multiple buffers, you must create a queue (ELEMFORM=QUEUE) or a table (ELEMFORM=TABLE) of message data elements, each representing a buffer that is to receive message data.

Specifying the Message Data Elements

Message data elements contain:

- Either:
  - A buffer that is to receive message data
  - A pointer to a buffer that is to receive message data.
- The length of the buffer (optional)
- The ALET to qualify the address of the buffer if message data elements contain pointers to the buffers (optional).

Buffer lengths and ALETs can be passed separately as described below instead of including them in each message data element.

Specifying the Location of Each Buffer

Specify the location of the buffer or buffer pointer within each message data element using one of the following parameters:

- If the message data elements contain the buffers, use the PARTOFF parameter to specify the offset of the buffer area from the start of each message data element.
- If the message data elements contain pointers to the buffers, use the PARTPTROFF parameter to specify the offset of the buffer address from the start of each message data element.

Specifying the Location of Each ALET

Specify the ALETs to qualify the buffer addresses using one of the following parameters:

- The PARTALET parameter to specify a single ALET to qualify each buffer address.
- The PARTALETOFF parameter to identify a location in each message data element that contains the ALET to qualify the associated buffer address.
- The PARTALETTBL parameter to specify a separate table of ALETs.

Specifying the Size of Each Buffer

Specify the lengths of the buffers using one of the following parameters:

- The PARTLEN parameter to specify a single length for all buffers.

- The PARTLENOFF parameter to identify a location in each message data element that contains the length of the associated buffer.
- The PARTLENTBL parameter to specify a separate table of buffer lengths.

Specifying the Location of the Next Message Data Element

If you have a table of message data elements, specify the location in each message data element of the next message data element using the NEXTOFF parameter.

If you have a queue of message data elements, specify the location in each message data element of the pointer to the next message data element using the NEXTPTROFF parameter.

When the Message Can't Fit in the Buffer Storage Provided

If you provide less total message buffer storage than is needed to receive the entire message, IXCMSGIX fills the available buffers and returns a return code of X'4' with a reason code of X'224' to indicate that more message data remain. Reissue the IXCMSGIX macro **while your message user routine is still running** and continue to do so until you have received all the message data.

If you want to receive and process a message in pieces, you can deliberately provide less buffer space than is needed for the entire message and issue IXCMSGIX repeatedly until you have received the whole message.

IXCMSGIX processes message data elements in consecutive order, copying message data into each buffer until either the receiving buffer is full or all the message data has been stored.

Processing of message data continues until one of the following occurs:

- All message data has been copied.
- IXCMSGIX has processed the number of buffers specified by the #MSGPARTS parameter
- IXCMSGIX has reached the end of the queue of message data elements as specified by the ENDOFQUEUE parameter or its default
- IXCMSGIX finds more than 65536 consecutive buffers of length 0 and does not know how many message parts to search because you did not specify the #MSGPARTS parameter. You do not receive the message; you receive a return code and reason code indicating the error.

Examples of Message Data Element Formats for Multi-Buffer Messages

Figure 5 on page 33 shows a queue of message data elements in which each element contains a buffer address and a pointer to the next element in the queue. All buffers reside in the same address space and are to be accessed using the ALET specified by the PARTALET parameter. All buffers are of the length specified by the PARTLEN parameter.

Figure 6 on page 33 shows a table of message data elements in which each element includes the following information relating to the buffer it describes:

- The ALET to qualify the buffer address
- The length of the buffer
- The address of the buffer.

Figure 7 on page 34 shows a table of message elements in which each element contains a buffer. No ALETs are specified since the buffers reside in the table itself. A separate table, specified by PARTLENTBL, contains the length of each buffer.

Figure 8 on page 35 shows a table of message elements in which each element contains a buffer address. A separate table, specified by PARTALETTBL, contains the ALETs to be used with each buffer address. A separate table, specified by PARTLENTBL, contains the length of each buffer.

Using the IXCMSGC Macro

The IXCMSGC macro allows you to interract with the XCF signaling services by providing services that:

- Save a message or response into an XCF-managed storage area for later processing (REQUEST=SAVEMSG)
- Discard a saved message or response, cancel an incomplete message-out request, or discard a completed message-out request (REQUEST=DISCARDMSG)
- Request information about incomplete message-out requests, completed message-out requests that are pending notification, or messages that have been saved with IXCMSGC (REQUEST=QUERYMSG)
- Force a message to be immediately considered complete (REQUEST=COMPLETION)
- Allow a user routine to receive control to process a saved message or a completed message-out request (REQUEST=CALLEXIT).

Understanding the Programming Environment

You can invoke IXCMSGC either while running in task mode or in SRB mode. In both cases, the caller's address space must be the primary address space that was current at the time the member invoked IXCJOIN to join its XCF group.

Identifying the Requestor

An active member of an XCF group is eligible to use the IXCMSGC services. To identify the member making the request, specify the member token that was returned when the member joined the XCF group. See "Identifying the Target Member or Members" on page 29 for a list of sources for obtaining a member token.

Requesting that a Message or Response Be Saved

IXCMSGC allows a user to request that XCF save messages or responses into XCF-managed storage areas if the user does not want to process the data immediately. The IXCMSGC Save Message request can be invoked only from a message user routine or a message notify user routine.

Identifying a Message to be Saved

The message exit parameter list (MEPL), mapped by IXCYMEPL, or the message notification parameter list (MNPL), mapped by IXCYMNPL, passed to the user routine contains the 16-byte token that the system uses to identify the message or response to be saved. This token is valid only for input to IXCMSGC; once the user routine returns control to XCF, the token is no longer valid. The system also invalidates the message token if the message is discarded or if the Message In service finished delivering all the message data. Once invalidated, the message token will not be accepted by either the IXCMSGC service or the IXCMSGIX service.

Saving a Message

A message that is saved by a message user routine can be processed at a later time by invoking the IXCMSGC Call Exit service. The Call Exit service passes control to a message user routine from which you can invoke the IXCMSGIX service to obtain the message data associated with the message. Once this message data is saved, the message data is no longer accessible to the instance of the user routine that saved it. Information about the saved message that was passed in the input parameter list remains accessible. However, the text of the saved message is accessible only by invoking the IXCMSGC Call Exit service to give control to a new instance of a user routine from which the IXCMSGIX can be invoked to retrieve the message data.

When a message is saved, whatever data is needed to create a new MEPL is also saved along with the message. Some of the data saved with the message includes the source member token, the target member token, and the sender's message control information.

Saving a Partially Delivered Message

A partially delivered message is one in which only part of the message data has been moved from XCF-managed storage to user-specified storage. It is not possible to save that portion of the message that has been received by the user. Only the undelivered portion of the message can be saved by invoking the

IXCMSGC Save Message service. The portion of the message that was saved can then be retrieved the next time an instance of the message user routine processes the message.

Saving a Message and its Associated Responses

From within a message notify user routine, one or more responses are eligible to be saved. The message and any responses still associated with it can be saved as a single entity, or each individual response can be saved independently of the message. The saved message/response entity can be processed by a message notify user routine at a later time by invoking the IXCMSGC Call Exit service to pass control to a message notify user routine.

When saving a message and its responses (if any) as a single entity, the data saved is sufficient to create a new MNPL that contains a descriptor for the message itself and a table of target/response information. Once the message/response entity is saved, the individual responses will not be accessible to the instance of the message notify user routine that saved the entity. To access the responses that remain associated with the saved message/response entity, you can invoke the IXCMSGC Call Exit service to give control to a new instance of a message notify user routine, from which the IXCMSGIX service can be used to retrieve the response message data or the IXCMSGC service can be used to save or discard an individual response.

Saving an Individual Response

When a response is saved independently of the message/response entity, it becomes an independent message. The message notify user routine will not be able to use the IXCMSGIX service to access the response data after it has been saved independently of the message/response entity. Note, however, that it is only the response data that becomes unavailable to the message notify user routine. Information about the response, such as the message control information and who sent the response, is still available to the instance of the message notify user routine that saved the response as well as to any new instances of the message notify user routine that might be called at a later time to process the saved message/response entity.

The message token for an individual response (MNPLTRMSGITOKEN) that is passed in the MNPL is invalidated when the response is saved or discarded, or if the XCF Message In service delivers all the message data. The message tokens for all the associated responses are also invalidated if the message/response entity is saved or discarded.

When saving a response independently of the message/response entity, the undelivered portion of the response is saved along with whatever data is needed to create a new MEPL. A saved response can be processed by invoking the IXCMSGC Call Exit service to call a message user routine.

Performance Implications

You should process or discard a saved message as soon as possible so that XCF can release the storage used for the message. An active member can discard a saved message by invoking the IXCMSGC Discard Message service.

Requesting that a Message Be Discarded

IXCMSGC allows a user to request that XCF discard its messages or its message/response items. The IXCMSGC Discard Message service also allows a user to cancel incomplete message-out requests or to discard completed message-out requests.

A discarded message is:

- No longer available for processing
- Not presented to any user routine
- Not visible to the IXCMSGC Query Message service.

The system discards any response associated with a message/response entity, but does not discard any response that has been saved. (Saving a response causes it to be disassociated from the message/response entity. See “Saving an Individual Response” on page 48.)

Identifying the Message To Be Discarded

Identify the message to be discarded with the TOKEN parameter of IXCMSGC. This 16-byte token can be obtained from one of the following:

- For an incoming message
  - The RETMSGTOKEN parameter on the IXCMSGC Save Message service
  - The IXCMSGC Query Message service (specify MSGIN for DATATYPE)
  - The MEPLMSGITOKEN token from the message exit parameter list (MEPL), if the routine has not yet finished processing the message
  - The MNPLTRMSGITOKEN token from the message notification parameter list (MNPL), if the routine has not yet finished processing the response message.

  Note that the system invalidates these tokens when the user routine gives up control.
- For a message/response entity
  - The RETMSGOTOKEN parameter on the IXCMSGOX service
  - The RETMSGTOKEN on the IXCMSGC Save Message service
  - The IXCMSGC Query Message service (specify MSGOUT for DATATYPE)
  - The MNPLMSGOTOKEN token from the message notification parameter list (MNPL), if the routine has not finished processing the message.

  Note that the system invalidates these tokens when the user routine gives up control.

Cancelling a Message-Out Request

IXCMSGC can be used to discard an incomplete message-out request before the message completes, thus having the effect of cancelling the message-out request. For a broadcast message to multiple targets, any remaining messages will not be sent. The system discards any responses that have been collected for the cancelled message as well as any responses that subsequently arrive for the cancelled message.

It might be necessary to cancel a message-out request to recover from a situation in which a target member (who has not failed nor left the group) fails to send the necessary response. XCF does not expect a response from a member who has terminated or left the group.

Letting the System Discard a Message

If a message is not explicitly saved by a user routine nor received by invoking the IXCMSGIX service, XCF discards the message when the user routine gives up control. This automatic discard is preferable to a user-specified discard in an exit routine because it incurs less system overhead.

Timing Considerations

If a message is discarded while an user routine is processing that same message, the system rejects subsequent attempts by the user routine to process the message with a return and reason code indicating that the message has been discarded. Depending on the timing, the current operation being performed by the user routine might be allowed to complete before the message is discarded. The system returns from the IXCMSGC Discard Message service with the discard of the message left pending.

Requesting Information about Messages

IXCMSGC allows a user to query information about incomplete message-out requests, about completed message-out requests that are pending notification, or about messages that have been saved using the IXCMSGC Save Message service.

The IXCMSGC Query Message service allows the user to request the following type of information:

- Summary information about messages sent by the member using the Message Out service
- Summary information about messages sent by the member using the Message In service

- Detail information about a particular message.

The system returns the information requested in a storage area that the requestor provides. The storage area, specified by the ANSAREA parameter, must either be in the caller's primary address space, or in an address or data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL), or in a common area data space. The contents of the answer area are mapped by the IXCYMQAA macro.

Requesting Message-out Information

To obtain information about messages sent by the member with the IXCMSGOX service, specify DATATYPE=MSGOUT. You can request information about messages that are incomplete, completed, or saved. Only one option can be specified.

- An incomplete message-out request is one for which a send or a response is still pending (MQAMOSSENDPENDING or MQAMOSRESPPENDING bit is set).
- A completed message-out request is one for which XCF is no longer trying to initiate a send and is no longer waiting for a response (MQAMOSCOMPLETED bit is set). Such a message is eligible for processing by a message notify user routine.
- A saved message-out request is one that was presented to, and saved by, a message notify user routine (MQAMOSSAVED bit is set).

The system returns data for each of the member's message-out requests that is in the specified state. The data for each message includes a token that identifies the message, user data associated with the message, and the status of the message, including whether XCF had to access user storage asynchronously to the IXCMSGOX request. This data is mapped by the MQAMSGOUTSUMMARY record in IXCYMQAA.

Requesting Message-in Information

To obtain information about messages saved by the message user routine or responses saved by the message notify user routine, specify DATATYPE=MSGIN. You can request information about messages from a particular member (by specifying the member's token on the SOURCE parameter) or from all members.

The system returns data for each message that includes a token that identifies the message, user data associated with the message, and the member token of the member that sent the message. This data is mapped by the MQAMSGINSUMMARY record in IXCYMQAA.

Requesting Detail Information about a Specific Message

To obtain detailed information about a particular message, specify DATATYPE=DETAIL and include the token that identifies the message. The data that the system returns depends on the type of message.

- For a message/response entity, the data includes a token that identifies the message, user data associated with the message, the number of targets for the message, and a table of target/response data with an entry for each possible target. This table describes the result of the send and the associated response collection (if applicable). This data is mapped by the MQAMSGOUTDETAIL record in IXCYMQAA.
- For a message saved by a message user routine or for a response saved by a message notify user routine, the data includes a token that identifies the message, user data associated with the message, the member token of the member that sent the message, message length, and message control information from the sender. This data is mapped by the MQAMSGINDETAIL record in IXCYMQAA.

Retrieving Information from the Answer Area

The information returned in the user-provided answer area is mapped by the IXCYMQAA macro. The data consists of a header record (mapped by MQAHEADER) and zero or more records appropriate to the type of query. See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for a description of the IXCYMQAA macro.

When retrieving information from the answer area, do not hardcode any length values for the header or various record types. Use the lengths and offsets that are included in the MQAA record itself.

Requesting that a Message Be Completed

IXCMSGC allows a user to force the message to be immediately considered complete. The member invoking the IXCMSGC Completion service must be the same member that sent the message to be completed. Use the IXCMSGC Completion service for a message-out request that XCF has accepted for delivery, but does not consider complete. See "Understanding Message Completion" on page 40 for a description of when a message is considered complete.

When the IXCMSGC Completion service returns to the member, the message is known to be complete and processing for the message continues just as it would have had the message completed without IXCMSGC intervention. That processing includes:

- XCF no longer attempts to send the message to any intended target.
- XCF discards any responses to the message that arrive subsequent to its completion. (Responses that are not collected are identified with the MNPLKRESPCODETOOLATE return code in the MNPL for the appropriate target entry.)
- If XCF was to initiate notification upon completion of the message, the message notify user routine receives control. If not, the member can invoke the IXCMSGC Call Exit service to call a message notify user routine to process the completed message.

Identifying the Message

The 16-character token that identifies the message for which completion is requested can be obtained from the IXCMSGOX service with the RETMSGOTOKEN parameter or from the IXCMSGC Query Message service (specify DATATYPE=MSGOUT).

Replacing the User Data

When using the IXCMSGC Completion service to force a message to completion, the member can replace the user data currently associated with the message. If the invocation of IXCMSGC causes the message to be considered complete, the MNPLMSGOUSERDATA field in the MNPL contains the new user data.

Requesting that a User Routine Is To Process a Message

IXCMSGC allows a user to specify a user routine to receive control to process a saved message or a completed message-out request. The user routine receives control under the same unit of work as the IXCMSGC invoker. The IXCMSGC Call Exit service can call a message user routine or a message notify user routine, whichever is appropriate for the message to be processed.

Identifying the Message

Identify the message to be processed by a user routine with the TOKEN parameter of IXCMSGC. This 16-byte token can be obtained from one of the following:

- For a message user routine
  - The IXCMSGC QUERYMSG service (specify DATATYPE=MSGIN)
  - The RETMSGTOKEN parameter on the IXCMSGC SAVEMSG service
- For a message notify user routine
  - The RETMSGOTOKEN parameter on the IXCMSGOX service
  - The IXCMSGC QUERYMSG service (specify DATATYPE=MSGOUT)
  - The RETMSGTOKEN parameter on the IXCMSGC SAVEMSG service

Identifying the User Routine

The user routine that you identify to be called must be appropriate for the type of message being processed. If you specify an inappropriate user routine, IXCMSGC fails with reason code IXCMSGCRSNINAPPROPEXITROUTINENAME.

Passing Information to the User Routine

You can pass up to 64-bits of information to the user routine with the EXITPARMS parameter. The contents of this area are user-defined, and might, for instance, be used to pass the address and ALET of a storage area containing information that determines how the exit routine should perform its processing.

- When passed to a message user routine, MEPLUSERPARMS in the message exit parameter list (MEPL) contains this information.
- When passed to a message notify user routine, MNPLEXITPARMS in the message notification parameter list (MNPL) contains this information.

Handling Member Termination

A member can become not active unexpectedly as the result of a failure. A member also can become not active voluntarily by invoking the XCF Leave service (IXCLEAVE) or the XCF Quiesce service (IXCQUIES). This section describes how XCF handles messages that might still be associated with the member that is terminating.

When a member becomes not active, XCF ensures that it deletes any member message data space. XCF also discards any messages that could not be presented to the member. The discarded messages include those saved by the member, those pending delivery to a message user routine, and completed message-out requests pending presentation to a message notify user routine. Incomplete message-out requests are declared to be complete when the sending member becomes not active, and are then discarded (without notification).

If XCF has initiated a send for a message-out request, XCF continues to attempt delivery of the message. XCF does not guarantee that the message will be delivered, even if it has already initiated the send, because a system can be removed from the sysplex before a message is transferred to its target system. Despite the potential for non-delivery, a member might want to take steps to ensure that XCF has initiated the send of its messages before it becomes (voluntarily) not active.

Coding a Message User Routine

Your message user routine provides a mechanism for receiving messages from other members of your XCF group. When you join an XCF group, you must specify the address of a message user routine to be given control when another member sends you a message. You also can specify a message user routine when invoking the XCF Message Control service (IXCMSGC REQUEST=CALLEXIT). This section presents the following information to help you code a message user routine:

- The environment in which it receives control
- The information it receives as input
- The actions it might perform
- Programming considerations to bear in mind

Environment

The message user routine receives control in the following environment:

Minimum authorization:	Supervisor state and PSW key 0
Dispatchable unit mode:	SRB
Cross memory mode:	PASN=HASN=SASN. The primary address space is equal to the primary address space of the caller of IXCJOIN, and can be swappable or non-swappable.

AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held

Restrictions

The message user routine **cannot** issue any macros that issue an SVC or that require the caller to be in task mode.

The message user routine can be called by an IXCMSGC REQUEST=CALLEXIT invocation. However, no FRRs can be established when making the CALLEXIT request in task mode.

Entry Specifications

XCF passes information to the message user routine in a parameter list and in registers.

Version 0 of the message exit parameter list (MEPL), mapped by IXCYMEPL, contains the following information:

MEPLMTOK
The message token to be passed to IXCMSGIX using the MSGTOKEN parameter. This field is maintained for version 1 parameter list users to be compatible with version 0 parameter list users.

MEPLMDAT
User-specified data provided by the sending member when it issued IXCJOIN (MEMDATA parameter).

MEPLMLEN
The length (number of bytes) of message data to be received by IXCMSGIX.

MEPLSRCE
The member token of the member sending the message. Use this token to reply to the message.

MEPLCNTL
The contents of the field specified by the MSGCNTL parameter on the IXCMSGOX macro when the message was sent, or zeros if the parameter was omitted. This field could be used to provide information about the message being sent.

Version 1 of the MEPL contains the following information:

MEPLVERSION
Version number of the MEPL.

MEPLFLAGS
Flags describing the characteristics of the message or its delivery.

MEPLTARGETMEMTOKEN
Member token of the member to which this message was sent.

MEPLMSGITOKEN
Token to identify the message being delivered. Use this token when invoking:

- IXCMSGIX to receive the text of the message.
- IXMCSGC to save the message for later processing.

MEPLRESPONSEID
Message response identifier. Use this value for the RESPONSEID parameter when replying to the message with IXCMSGOX SENDTO=ORIGINATOR.

MEPLEXTENSIONADDR
Address of additional data provided to the message user routine.

MEPLSTREAMID
Stream identifier for this message, if specified on the sending IXCMSGOX request.

MEPLLEN
Length in bytes of the latest version of the MEPL. Note that this name is maintained for compatibility with version 0 of the MEPL.

The additional data pointed to by MEPLEXTENSIONADDR contains the following information:

MEPLEXUSERDATA
Data associated with the saved message by the target member. Contains a copy of the data specified for the USERDATA parameter when the message was saved with IXCMSGC. Otherwise, it contains X'0'.

MEPLEXFLAGS
Flags describing characteristics of the MEPL extension record.

MEPLEXEXITPARMS
User parameters. Contains a copy of the data specified for the EXITPARMS parameter when the user routine was called with IXCMSGC. Otherwise, it contains X'0'.

See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for more information about the IXCYMEPL mapping macro.

Registers at Entry: w hen the message user routine receives control, the GPRs contain:

Register
Contents

0
Used as a work register by the system.

1
Address of the message exit routine parameter list (MEPL).

2-12
Used as work registers by the system.

13
Address of a 144-byte work area. The message user routine does not have to save and restore XCF's registers in this work area. The message user routine can use this work area in any way it chooses.

14
Return address

15
Entry point address of message user routine.

When the message user routine receives control, the ARs do not contain any information for use by the message user routine.

Return Specifications

On return to XCF, the message user routine does not have to set any return codes or place any information in the GPRs. The message user routine returns control to the system by branching back to the address in GPR 14.

User Routine Processing

When an active member of an XCF group issues the IXCMSGOX macro to send a message to another active member of the same group, XCF asynchronously passes control to the message user routine of the target member. The message user routine runs in SRB mode in the target member's primary address space (the joiner's address space).

You are responsible for writing a message user routine that can:

- Determine if the member's initialization is complete. A member might issue IXCJOIN and its message user routine could get control before IXCJOIN returns to the caller. To determine if the member's initialization is complete, the message user routine might examine a bit that the member sets on or off. The member might want its message user routine to ignore or defer any messages until the routine determines that initialization is complete.

- Check the message control information area (32 bytes of data passed as part of the parameter list).
- Determine the following from the contents of the message control information area:
  - Whether there is a message to be received. (The message user routine could also determine this from the parameter list. If the length of the message is zero (MEPLMLEN=0), then the message buffer does not contain any data.)
  - Whether to receive the message if there is one in the message buffer area.
  - Whether to receive the message into a single buffer or into multiple buffers.
  - Where to place the data from the message buffer area.
  - The type or format of the data in the message buffer area.
- If the message user routine elects to receive the message, it can:
  - Check the member data to determine which member the message was sent to, if more than one member is using the same message user routine. The member data would have been defined when the member joined the group (MEMDATA parameter on IXCJOIN). XCF passes that information to the message user routine as part of the parameter list.
  - Obtain enough storage to contain all the data from the message buffer area or obtain less storage but plan to specify MULTIPART=YES on the IXCMSGIX macro so you can reissue IXCMSGIX multiple times to receive the entire message. If you receive messages very frequently, you might use a pre-allocated buffer. For less frequent messages, you can obtain storage using one of the system services.
  - Receive the message by invoking the IXCMSGIX macro.
  - Process the data in the message, or queue the message to a task for processing and post the task.
  - Issue IXCMSGOX if the sender requires an acknowledgment, possibly using the MSGCNTL field to contain the acknowledgment.

If the message user routine chooses not to issue IXCMSGIX to receive the data, XCF discards the data as soon as the message user routine returns to XCF, and does not notify the sender that the message was discarded.

Programming Considerations

Consider the following when writing your message user routine:

- The message user routine must be a reentrant program. There could be multiple instances of your message user routine running concurrently.
- The message user routine should return to XCF as soon as possible, because system resources are held until the message user routine gives up control. To avoid performance degradation in the XCF signaling service, and the system as a whole, do not issue the SUSPEND macro within the message user routine.
- XCF does not provide any acknowledgment that a target member has received a message. The target member or its message user routine must provide an acknowledgment if required. However, XCF will either deliver the message, or provide notification that the target member or the target member's system failed.
- Because the message user routine runs in SRB mode, it cannot issue any SVCs. You might want to queue work to one or more tasks for processing and post the tasks when needed.

User Routine Recovery

XCF does not provide any recovery for the message user routine. Routines that require recovery must establish their own. XCF does place sufficient information into the SDWA to identify the message user routine that was in control. The multi-system application must provide whatever diagnostic data is required for problem determination for the message user routine.

If XCF cannot access the parameter list generated by the IXCMSGIX macro, or the parameter list is improperly set, the message user routine's recovery routine will get control provided the message user routine sets up its recovery before invoking IXCMSGIX.

Members that identify a message user routine should allow for SRB-to-task percolation. (**Note:** SRB-to-task percolation does not work for address space associated members. See “Member Association” on page 17 for more information. If XCF processing fails, and XCF does not retry, XCF abnormally ends the task that the member is associated with (either the task or the job step task as specified on IXCJOIN) with a retryable system completion code 00C and one of the following reason codes:

- Reason code 02070000 means that XCF successfully delivered the message and the message user returned control to XCF. The task's recovery routine does not have to take any action.
- Reason code 02070001 means that XCF did not successfully deliver the message. The task's recovery routine might do one of the following:
  - Determine which message, if any, is lost, and notify the sender of the message to send the message again.
  - Back up to some logical point and continue processing from that point.
  - Allow the task to abnormally end.

To ensure that the member's recovery can intercept SRB-to-task percolation, the task that the member is associated with must ensure that its recovery routine always receives control when a task abnormally ends. To accomplish this, the associated task should issue the WAIT macro and continue waiting indefinitely while other tasks perform the member's work.

SRB-to-task percolation does not occur while the task's recovery routine is running. XCF waits until the task is not in recovery before abnormally ending the task.

Timing Considerations

You should be aware of the following possible events related to timing:

- XCF does not necessarily deliver messages in the order in which they were sent.
- Message delivery occurs asynchronously. It is possible for a message to be received by the target member using IXCMSGIX before XCF returns control from IXCMSGOX to the member that sent the signal. If the receiving member provides an acknowledgement signal back to the sender, it is even possible that the acknowledgement signal will be received by the sender before the sender receives control back from issuing the IXCMSGOX invocation that sent the message.
- A target member could become inactive while its message user routine is executing. In that case, the message user routine completes normally, but XCF does not deliver any more messages for that member.
- A target member could become inactive after the SRB for its message user routine is scheduled, but before the message user routine runs. In that case, XCF discards the message because the member cannot receive it, and does not deliver any more messages for that member. XCF does not notify the sender that the message was discarded, but notifies the sender's group user routine that the target member's state has changed.
- A sending member's system might fail while the member is trying to send a message, and the target member might not receive the message. In this case, if the target member has a group user routine, the target member would receive notification of the system failure. If the target member was expecting a message, this notification might explain why the message was not received.
- A target member's system might fail before XCF can deliver its message. In this case, if the sender has a group user routine, the sender would receive notification of the system failure. If the sender was expecting an acknowledgment, this notification might explain why no acknowledgment was received.
- XCF might deliver a message to a target member, but the member, or its system, might fail before the member can take any action on the message (if some action was required). If the sender has a group user routine, the sender would receive notification of member or system failure. If the sender was expecting some action to take place, this notification might explain why the action did not occur.
- A loss of signaling connectivity between systems might occur. The operator might be able to start or restart additional signaling paths and reestablish connectivity. In this case, XCF delivers messages normally, but with some delay.

- A new member might send a message that XCF could deliver before the target member's group user routine receives notification that the new member exists.
- A target member's system might be temporarily non-operational, causing delivery of messages to be delayed until the system resumes activity.
- A member invoking IXCMSGOX might get a return code indicating that the target of the message was not found. This could occur before the member's group user routine was notified that the target member (or the target member's system) is gone.
- When a message user routine runs, the member that sent the message, or the system on which the sender resides, might no longer be in the sysplex. In that case, if the receiving member sends a response, XCF indicates that the original sender (now the target) does not exist.

Coded Example

Here is an example of a message user routine, the members (member 1 and member 2) of a group have established a protocol for the use of the message control information (MSGCNTL parameter on DIDXCMDSGOX):

- If member 1 sends a message to member 2 and places zeros in the first byte of the message control field, member 1 is indicating that the data in the message buffer area (MSGBUF parameter on IXCMSGOX) is an initial message. Member 2's message user routine then reads in the data contained in the message buffer.
- If member 1 sends a message with anything other than zeros in the first byte of the message control field, member 1 is confirming that it received a prior message. Member 2's message user routine then does not have to read in any information from the message buffer.

```
*****************************************************************
*                                                               *
*  MESSAGE USER ROUTINE                                         *
*                                                               *
*****************************************************************
MEXIT    CSECT
MEXIT    AMODE 31
MEXIT    RMODE ANY
@MAINENT DS    0H
         USING *,R15
         B     @PROLOG
         DC    AL1(16)
         DC    C'ME 89360 MEXIT'
         DROP  R15
*****************************************************************
*                                                               *
*  ENTRY LINKAGE                                                *
*                                                               *
*****************************************************************
@PROLOG  STM   R14,R12,12(R13)
         LR    R12,R15
@PSTART  EQU   MEXIT
*
*  SET UP BASE REGISTER TO 12
*
         USING @PSTART,R12
         SLR   R15,R15
         IC    R15,@SIZDATD
         SLR   R0,R0
         ICM   R0,7,@SIZDATD+1
         STORAGE OBTAIN,LENGTH=(0),SP=(15)
         LR    R10,R1
         USING @DATD,R10
         ST    R13,4(,R10)
         ST    R10,8(,R13)
         LM    R15,R1,16(R13)
         LR    R13,R10
*****************************************************************
*                                                               *
*  MESSAGE USER ROUTINE CODE                                    *
*                                                               *
*  IF THE FIRST BYTE OF MSGCNTL CONTAINS ZEROS, THEN READ       *
*  IN THE MESSAGE; OTHERWISE, DO NOT READ IN THE MESSAGE.       *
*                                                               *
*****************************************************************
```

```
         LR    R6,R1                   SAVE THE PARAMETER LIST
         USING MEPL,R6                 GET ADDRESSABILITY TO MEPL
         L     R4,MEPLMLEN             PLACE THE LENGTH OF MSG IN REG 4
         LA    R5,MEPLCNTL             LOAD ADDRESS OF MSGCNTL TO REG 5
         USING CHKTYPE,R5              GET ADDRESSABILITY TO MSGCNTL
         CLI   MSGTYPE,X'00'           SEE IF MESSAGE SHOULD BE READ IN
         BNE   @DONREAD                IF NO, BRANCH
         STORAGE OBTAIN,LENGTH=(R4),SP=0  IF YES, GET STORAGE FOR MSG
         LR    R3,R1                   SAVE THE ADDRESS IN REG 3

*
*  SET UP DYNAMIC AREA AND ISSUE IXCMSGIX TO RECEIVE MESSAGE
*
         L     R7,MSGILNTH
         BCTR  R7,0
         EX    R7,@SETPARM
         IXCMSGIX MSGTOKEN=MEPLMTOK,MSGBUF=(R3),                       X
               RETCODE=RETURN,RSNCODE=REASON,MF=(E,MSGILSTD)

*************************************************************
*                                                           *
*  NOTE:  THIS IS A SIMPLIFIED EXAMPLE OF A MESSAGE USER    *
*  ROUTINE.  NORMALLY, AT THIS POINT, THE MESSAGE WOULD     *
*  BE PLACED ON A WORK QUEUE OR OTHER APPROPRIATE ACTION    *
*  TAKEN WITHIN THE MESSAGE USER ROUTINE, AND               *
*  THE STORAGE WOULD NOT BE RELEASED.                       *
*                                                           *
*************************************************************
*
*  RELEASE THE STORAGE AND WRITE TO OPERATOR
*
         STORAGE RELEASE,LENGTH=(R4),ADDR=(R3),SP=0
         WTO   'READ IN THE MESSAGE',ROUTCDE=11,LINKAGE=BRANCH
         B     @FINI
*
*  BRANCH HERE WHEN MSGTYPE DOES NOT CONTAIN ZEROS (NO MESSAGE
*  TO RECEIVE)
*
@DONREAD WTO   'MESSAGE CONFIRMATION',ROUTCDE=11,LINKAGE=BRANCH
*
*  RELEASE DYNAMIC AREA
*
@FINI    LR    R1,R10
         L     R13,4(,R13)
         SLR   R15,R15
         IC    R15,@SIZDATD
         SLR   R0,R0
         ICM   R0,7,@SIZDATD+1
         STORAGE RELEASE,LENGTH=(0),ADDR=(1),SP=(15)
         LM    R14,R12,12(R13)
         BR    R14                 RETURN TO XCF
         DS    0F
@SETPARM MVC   MSGILSTD(0),MSGILSTS
@PSIZE   EQU   ((*-MEXIT+99)/100)*5
         DC    C'PATCH AREA - MEXIT 89.360'
         PUSH  PRINT
         PRINT ON,GEN,DATA
@PSPACE  DC    25S(*)
         ORG   @PSPACE
         DC    ((@PSIZE+1)/2)S(*)
         ORG   ,
         POP   PRINT
MEXIT    CSECT ,
         LTORG
         DS    0D
@SIZDATD DS    0A
         DC    AL1(0)
         DC    AL3(@DYNSIZE)
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
```

```
R14      EQU   14
R15      EQU   15

MSGILST1 IXCMSGIX  MF=(L,MSGILSTS)    LIST FORM OF IXCMSGIX MACRO
MSGILNTH DC   A(*-MSGILST1)
@ENDDATA EQU   *
@DATA    DS    0H
@DATD    DSECT
         DS    0F
SAVEAREA DS    18F
RETURN   DS    1F                    RETURN CODE
REASON   DS    1F                    REASON CODE
TOKENMSG DS    CL4                   MESSAGE TOKEN
MSGILST2 IXCMSGIX  MF=(L,MSGILSTD)    LIST FORM OF IXCMSGIX MACRO
@ENDDATD DS    0X
@DYNSIZE EQU   ((@ENDDATD-@DATD+7)/8)*8
CHKTYPE  DSECT
MSGTYPE  DS    X
RESTCNTL DS    XL31
*****************************************************************
*                                                               *
*  MAPPING MACROS                                               *
*                                                               *
*****************************************************************
         IXCYMEPL
         END   MEXIT
```

Coding a Message Notify User Routine

Your message notify user routine provides a mechanism for XCF to notify members of events related to the use of the XCF signaling service. When you join an XCF group, you can specify the address of a message notify user routine to be given control when XCF needs to provide this notification. You can also specify the address of a message notify user routine when invoking the IXCMSGOX service to send a message or on the IXCMSGC Call Exit service to call a user routine. The system gives preference to the user routine specified on the invoked IXCMSGOX or IXCMSGC service when a message notify user routine is also specified on IXCJOIN.

This section presents the following information to help you code a message notify user routine:

- The environment in which it receives control
- The information it receives as input
- The actions it might perform
- Programming considerations to bear in mind.

Environment

The message notify user routine receives control in the following environment:

Minimum authorization:	Supervisor state and PSW key 0
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN. The primary address space is equal to the primary address space of the caller of IXCJOIN, and can be swappable or non-swappable.
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held

Restrictions

The message notify user routine can be called by an IXCMSGC REQUEST=CALLEXIT invocation. However, no FRRs can be established when making the CALLEXIT request in task mode.

Entry Specifications

XCF passes information to the message notify user routine in registers and in a parameter list.

Registers at Entry: when the message notify user routine receives control, the GPRs contain:

Register
Contents

0
Used as a work register by the system.

1
Address of the message notification parameter list (MNPL).

2-12
Used as work registers by the system.

13
Address of a 144-byte work area. The message notify user routine does not have to save and restore XCF's registers in this work area. The message notify user routine can use this work area in any way it chooses.

14
Return address

15
Entry point address of message notify user routine.

When the message notify user routine receives control, the ARs do not contain any information for use by the message notify user routine.

The message notification parameter list (MNPL), mapped by IXCYMNPL, contains a header record followed by zero or more data records. Information in the header record indicates the type of notification data that follows.

The **header record** contains the following information:

MNPLVERSION
The version number of the parameter list.

MNPLTYPE
The type of notification that is being presented. Note that new types of notification might be provided in future releases. Your message notify user routine should be written to tolerate any future changes or additions. In the initial version of the MNPL, the type of notification is the completion of a message-out request.

MNPLFLAGS
Flags to describe characteristics of the notification or its presentation.

MNPLMEMTOKEN
The member token of the member to which this notification is presented.

MNPLMEMDATA
A copy of the contents of the field specified by the MEMDATA parameter on the IXCJOIN macro when this member joined the XCF group, or zeros if the parameter was omitted.

MNPLEXITPARMS
User exit parameters. If the member invoked the IXCMSGC Call Exit service to call the message notify user routine, this is a copy of the data specified by the EXITPARMS parameter on the IXCMSGC macro. Otherwise, it contains zeros.

MNPL#DATARECORDS
The number of data records provided.

MNPLDATARECOFFSET
Offset from the start of the header record at which the first data record can be found.

The **data record** contains the following information:

MNPLRECTYPE
The type of data described in this record. In the initial version of the MNPL, the types of data supported are MSGOUT data and MEMBER data.

MNPLRECLEN
The number of bytes in this data record.

MNPLRECDATA
The variable content of the data record. The contents are a MSGOUT data record.

For a **MSGOUT data record**, the record contains the following information:

MNPLMSGOTOKEN
Token to identify this message and any associated responses to XCF services, such as IXCMSGC.

MNPLMSGOUSERDATA
User data associated with the message. This is a copy of the contents of the USERDATA parameter when the IXCMSGOX macro was invoked to send the message or as modified by the IXCMSGC macro when the message was saved or completed.

MNPLMSGOFLAGS
Flags to describe characteristics of the message. The information includes:

- Whether the sender requested notification of message completion by an XCF-scheduled message.
- Whether a broadcast request completed successfully.
- Whether the message was saved.
- Whether XCF had to access user storage describing or containing the message even after IXCMSGOX returned to the caller.

MNPLMSGOMLEN
Number of bytes of message data for the message-out request.

MNPLMSGOSOURCE
The member token of the sending member.

MNPLMSGOMSGCNTL
The message control information from the message-out request.

MNPLMSGO#TARGETS
Number of targets for the message (including skipped targets).

MNPLMSGOTBLPTR
Address of the table containing target/response information for this message. MNPLMSGOENTTYPE indicates which type of entries the table contains.

MNPLMSGOENTTYPE
Code that identifies which mapping to use for the entries in the table of target/response data. The entries are either target only entries or target/response entries. A target only entry describes the result of a send to one particular target member. A target/response entry describes the result of a send to and response from one member.

MNPLMSGOENTLEN
Length in bytes of an individual entry in the table containing target/response information.

For a **MEMBER data record**, the record contains the following information:

MNPLMEMBERMNAME
The member name.

MNPLMEMBERSYSNAME
The member token.

MNPLMEMBERSYSID
The name of the system on which the member resides. The system name is made up of the system token and the system slot number.

The table of target/response information contains either target only entries or target/response entries.

A **target only entry** contains the following information:

MNPLTOTARGET
Target member token.

MNPLTOSENDSTATUS
Status of the message send request.

MNPLTOSENDRETCODE
Return code from the IXCMSGOX macro about the send message request to this particular target member.

MNPLTOSENDRSNCODE
Failing reason code from the IXCMSGOX macro. Only valid if MNPLTOSENDRETCODE is nonzero.

A **target/response entry** contains the following information:

MNPLTRTARGET
Target member token.

MNPLTRSENDSTATUS
Status of the message send request.

MNPLTRSENDRETCODE
Return code from the IXCMSGOX macro about the send message request to this particular target member.

MNPLTRSENDRSNCODE
Failing reason code from the IXCMSGOX macro. Only valid if MNPLTRSENDRETCODE is nonzero.

MNPLTRRESPSTATUS
Status of response message.

MNPLTRRESPCODE
Code to explain why XCF believes the response was not received. Only valid if XCF did not receive a response.

MNPLTRRESPMLEN
Total number of bytes of message data remaining for delivery with the IXCMSGIX macro. The length is accurate only on entry to the message notify user routine. It is not updated while the user routine is running to reflect any partial deliveries performed by the routine. Only valid if the associated response is still available, that is, it has been received and has not been delivered, saved, or discarded.

MNPLTRRESPSRCE
Member token of the originator of the response. Only valid if XCF received a response.

MNPLTRRESPCNTL
The contents of the MSGCNTL parameter from the originator of the response. Only valid when XCF received a response.

MNPLTRMSGITOKEN
Token to identify the response message. Specify this value for the TOKEN parameter when invoking the IXCMSGIX macro or the IXCMSGC macro to process this response message. Only valid if the associated response is available.

MNPLTRRESPONSEID
Message response ID. Specify this value for the RESPONSEID parameter when invoking the IXCMSGOX macro to reply to this response message. Only valid if the sender requested that XCF manage the gathering of responses to this message.

See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for more information about the IXCYMNPL mapping macro.

Return Specifications

On return to XCF, the message notify user routine does not have to set any return codes or place any information in the GPRs. The message notify user routine returns control to the system by branching back to the address in GPR 14.

User Routine Processing

The message notify user routine might receive control at the completion of a message-out request. The information available to the user routine through the message notification parameter list (MNPL) includes the user data provided on the IXCMSGOX request that sent the original message, information about the send to each potential target member, and, if relevant, information about the responses to the message. Note that it is possible for the message notify user routine to receive control before the IXCMSGOX service that processed the request returns control to the caller.

The message notify user routine can do one or both of the following:

- Invoke the IXCMSGIX service to receive any responses
- Invoke the IXCMSGC Save Message service to save any responses in the XCF-managed data space

If a response is neither received nor saved, the system discards the response when the message notify user routine gives up control.

User Routine Recovery

XCF does not provide any recovery for the message notify user routine. Routines that require recovery must establish their own. XCF does place sufficient information into the SDWA to identify the message notify user routine that was in control. The multi-system application must provide whatever diagnostic data is required for problem determination for the message notify user routine.

- For a message notify user routine that receives control as a standard XCF-managed SRB routine, SRB-to-task percolation might occur, just as for a message user routine.
- For a message notify user routine that receives control as a result of invoking the IXCMSGC Call Exit service, percolation from any recovery routine established by the message notify user routine causes XCF's recovery to percolate to the recovery established by the IXCMSGC invoker.

Note that for notification of message completion, XCF discards all information related to the message unless the user routine has used the IXCMSGC Save Message service to save the information.

Requesting XCF Status Monitoring

By identifying a status user routine to XCF, members can request that XCF monitor their activity. In general, XCF status monitoring works as follows:

- XCF monitors the member's activity by regularly checking a status field that the member identifies.
- The member also identifies a status-checking interval. When XCF detects that the member did not update its status field within the status-checking interval, XCF schedules the member's status user routine.

  By scheduling the status user routine, XCF gives the routine the opportunity to:

  – Check the member and determine whether the member is operating normally
  – Decide whether to notify other members if the member is not operating normally.

XCF's actions are limited to checking the status field and scheduling the status user routine, unless the status user routine requests notification to other members, or the status user routine does not run.

This section contains information on the following topics related to XCF status monitoring:

- Using a status user routine
- Updating the status field
- Setting and changing a status-checking interval
- Coding a status user routine.

Using a Status User Routine

This section contains the following information related to using a status user routine:

- A detailed description of normal status user routine processing, including a diagram (Figure 9 on page 66 and Figure 10 on page 67)
- A summary of the important concepts related to normal status user routine processing
- A discussion of events that can occur other than normal processing.

Normal Processing

The following is an overview of how the XCF status monitoring service interacts with the status user routine during normal processing. See Figure 9 on page 66 and Figure 10 on page 67 for a summary of the process.

- The member identifies a status user routine, a status field (which must be in fixed or disabled reference (DREF) common storage), and a status-checking interval on the IXCJOIN macro (STATEXIT, STATFLD, and INTERVAL parameters).
- The member is responsible for regularly updating its own status field (see "Updating the Status Field" on page 69 for suggestions on how to do this).

  Note: Updating the status field is not mandatory. A member can request status monitoring and never update its status field. However, this results in XCF scheduling the status user routine every time the member's status-checking interval expires, and could consume system resources unnecessarily.
- XCF starts monitoring the status field. This monitoring might begin before the IXCJOIN service returns to the caller.
- If the member fails to update the status field within the status-checking interval, XCF schedules the status user routine as a local SRB to run in the member's primary address space.
- XCF passes a parameter list (mapped by the IXCYSEPL mapping macro and pointed to by GPR 1) to the status user routine. The SEPLSTCH field indicates that XCF is checking for a status update missing (SEPLSTCH=SEUPDMIS, where SEUPDMIS is a system-defined constant).

Checking for Status Update Missing

The status user routine determines whether the member is operating normally, and if not, whether it wants XCF to notify other members, through their group user routines, of a status change. For XCF to notify the group user routines, the status user routine must set a return code that matches the value in SEPLSTCH. The status user routine should set the return codes as follows:

- A return code of SEUPDMIS indicates that the member is not operating normally. In this case:
  - The return code matches the value in SEPLSTCH, causing XCF to schedule the group user routines to notify the other members that the member's status update is missing (event type = GEMSUMSE, where GEMSUMSE is a system-defined constant). XCF does not issue an event type of GEMSUMSE to the group user routine of the member whose status update is missing.
  - The status user routine can elect to place user data in GPR 0 to be passed to the group user routines in the parameter list (GEPLUDAT field mapped by the IXCYGEPL mapping macro).
  - XCF continues to monitor the status field to see if the member resumes updating, and continues scheduling the status user routine:
    - As long as the status user routine runs successfully
    - Until the status user routine confirms a status update resumed
    - Until the member becomes inactive.
- A return code of SEUPDRES (system-defined constant) indicates that the member is actually operating normally, even though it might have missed a status update. In this case:
  - The return code does not match the value in SEPLSTCH, so XCF will not schedule the group user routines. (XCF considers this response to be the same as a status update, and does not schedule the status user routine again until another status-checking interval expires with no update to the status field.)

  - The status user routine should not place user data in GPR 0 because XCF will not be scheduling the group user routines.
  - XCF continues to monitor the status field, and continues scheduling the status user routine:
    - As long as the status user routine runs successfully
    - Until the status user routine confirms a status update resumed
    - Until the member becomes inactive.
- Once the status user routine confirms a status update missing, XCF continues monitoring the status field and does the following:
  - If XCF detects that the status field changed, XCF schedules the status user routine again. This time, XCF sets the SEPLSTCH field to SEUPDRES to indicate checking for status update resumed.
  - If XCF detects that the status field did not change, XCF waits a period of time (system-defined, and might be less than the member's status-checking interval). If the status field still does not change, XCF schedules the status user routine. Again, XCF sets the SEPLSTCH field to SEUPDRES to indicate checking for status update resumed.
  - Note that in **either case**, whether XCF detects the status field to be changed or unchanged, XCF schedules the status user routine to determine if the member is operating normally.

Checking for Status Update Resumed

Once XCF schedules the status user routine to check for status update resumed, the routine must again set a return code to let XCF know whether it wants other members to be notified of a status change. The status user routine should set the return code as follows:

- A return code of SEUPDRES indicates that the member is operating normally, even if it did not resume updating its status field. In this case:
  - The return code matches the value in SEPLSTCH, causing XCF to schedule the group user routines to notify the other members that the member's status update resumed (event type = GEMNOSUM, where GEMNOSUM is a system-defined constant).
  - The status user routine can place user data in GPR 0 to be passed to the group user routines in the parameter list (GEPLUDAT field mapped by the IXCYGEPL mapping macro).
  - XCF continues to monitor the status field in case the member misses another update.
- A return code of SEUPDMIS indicates that the member is not operating normally, even though it might have resumed updating its status field. In this case:
  - The return code does not match the value in SEPLSTCH, so XCF will not schedule the group user routines.
  - The status user routine should not place user data in GPR 0 because XCF will not be scheduling the group user routines.
  - XCF continues to monitor the status field, and continues scheduling the status user routine:
    - As long as the status user routine runs successfully
    - Until the status user routine confirms a status update resumed
    - Until the member becomes inactive.

    Note: XCF reports the status update missing condition to the group user routines only once per occurrence, rather than continuously informing the group user routines that the status update is still missing.

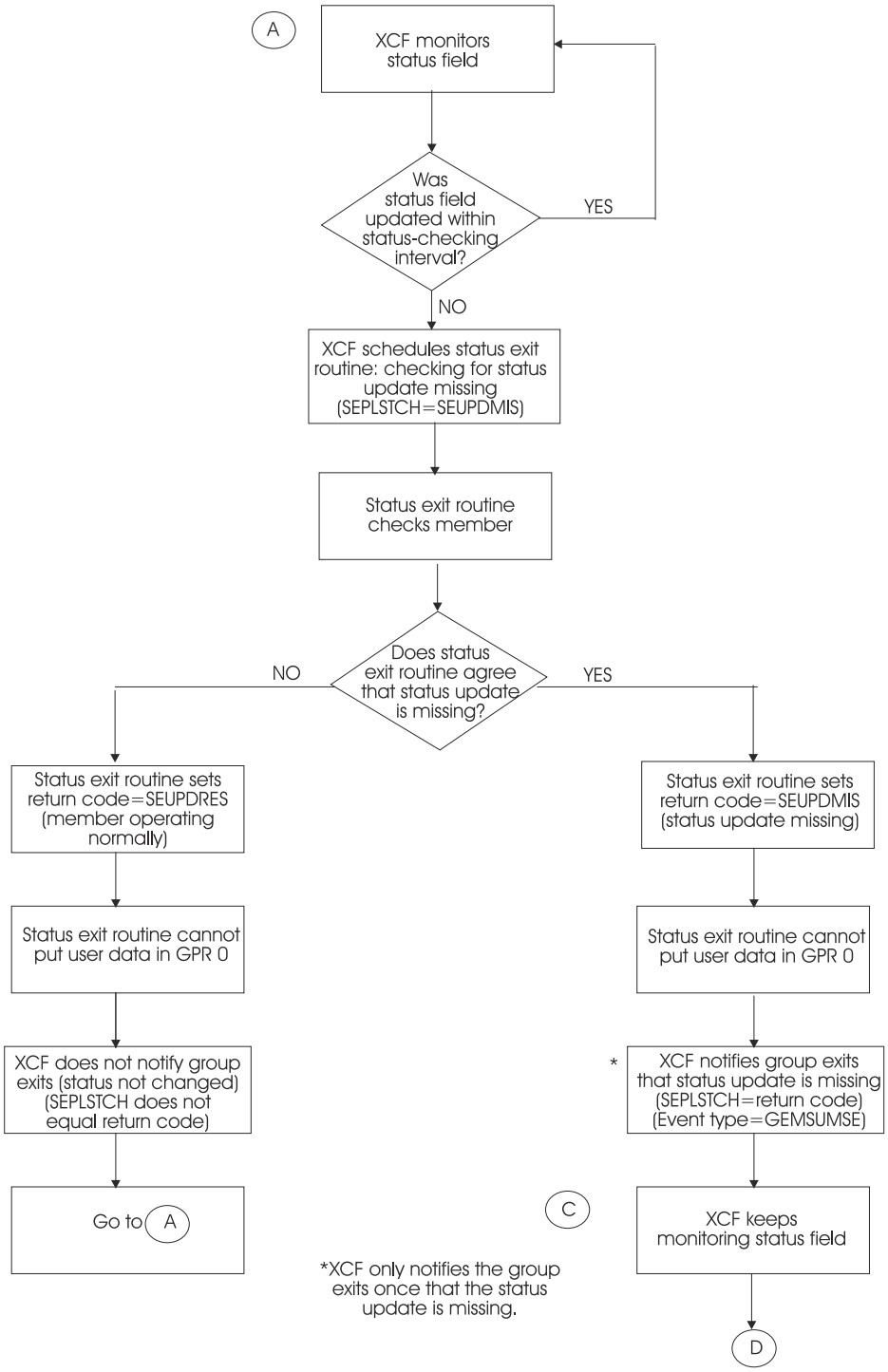


Figure 9. XCF Status Monitoring Service Normal Processing (Part 1 of 2)

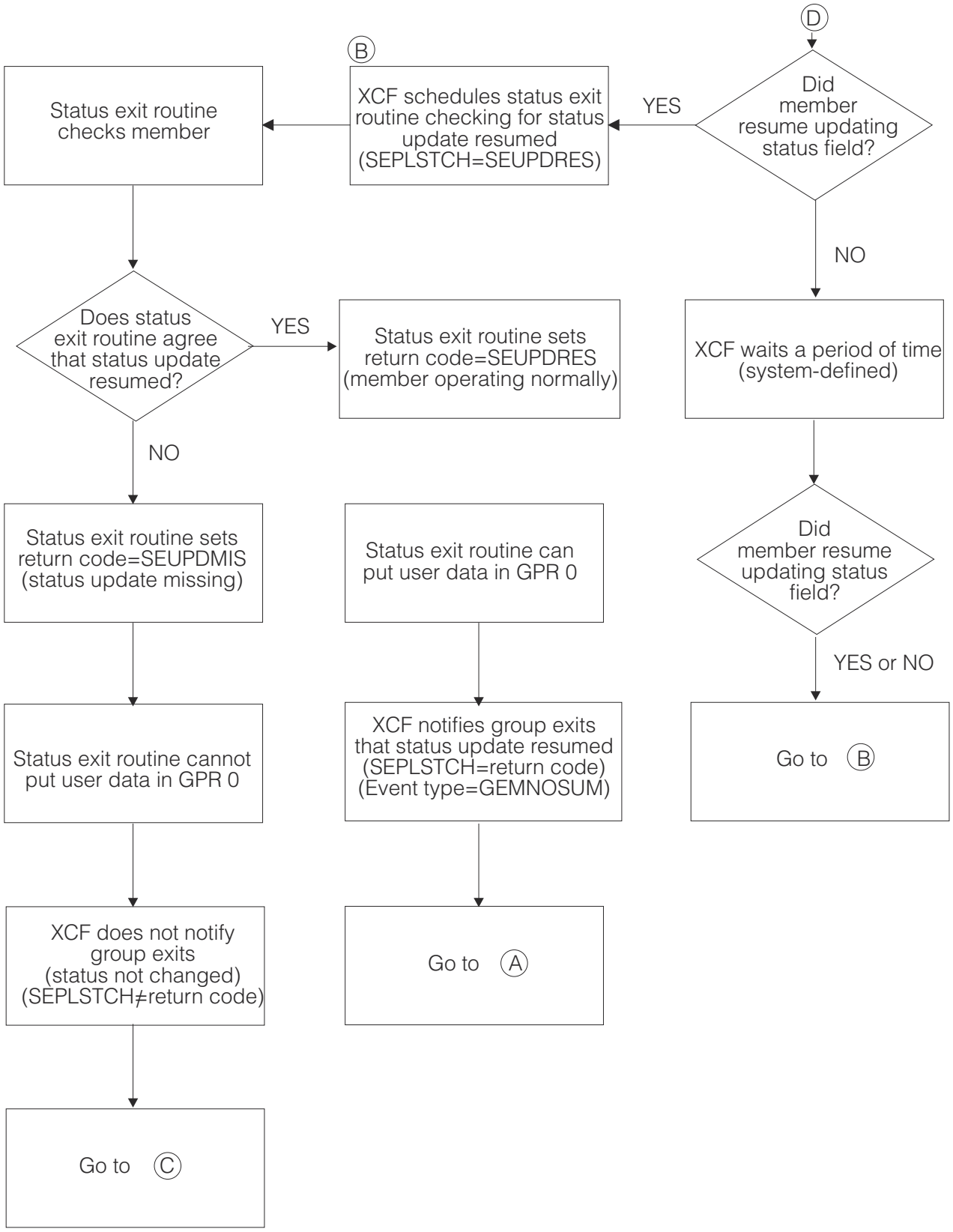


Figure 10. XCF Status Monitoring Service Normal Processing (Part 2 of 2)

Summary of Important Concepts

The preceding section gave an overview of the normal processing that takes place when a member identifies a status user routine to XCF. The following is a summary of the most important concepts related to normal status user routine processing:

- XCF schedules the status user routine in two general cases:
  - After the member misses an update to its status field (XCF is checking for status update missing).
  - After the member's status user routine confirms a status update missing (XCF is checking for status update resumed).
- XCF schedules the group user routines **only** to notify them of a **status change**. A status change is one of the following:
  - The member was operating normally (or does not have a status update missing condition outstanding) and then indicated that it was not operating normally (status update missing).
  - The member had a confirmed or assumed status update missing and then indicated that it was operating normally (status update resumed).

  In both cases:
  - The value in SEPLSTCH must match the return code set by the status user routine.
  - XCF does not schedule the group user routine of the affected member.

  Note: Many other events cause XCF to schedule group user routines. This discussion pertains only to events related to status monitoring. See the section entitled "Events that Cause XCF to Schedule a Group User Routine" on page 77 for a complete discussion.
- Once the status user routine confirms a status update missing, XCF schedules the status user routine to check for status update resumed. XCF does this whether or not the member resumes updating the status field, because:
  - When XCF detects that the status field changed, the status user routine can indicate that the member is not operating normally even though the member updated its status field.
  - When XCF detects that the status field did not change, the status user routine can indicate that the member is operating normally even though the member still did not update its status field.
- The status user routine does not have to confirm that a member's status update is missing, even if the member is not operating normally. The status user routine can elect to post a task to do recovery on behalf of the member and set a return code to XCF indicating that the member is still operational.

Events Other than Normal Processing

Certain events cause XCF to do the following:

- Assume a status update missing condition for a member, even when one is not confirmed by the status user routine (event type = GEMSUMDI).
- Stop monitoring a member (event type = GEMONREM).

In both of these cases, XCF notifies the group user routines. In the case where XCF stops monitoring a member, XCF schedules the group user routine of the affected member as well as the other members.

Note: To reinstate monitoring after XCF stops monitoring a member, the member must issue IXCLEAVE or IXCQUIES, and then issue IXCJOIN once again with the STATEXIT, STATFLD, and INTERVAL parameters.

Table 4 on page 69 summarizes the events that cause XCF to assume a status update missing condition for a member, or to stop monitoring the member:

Table 4. Status User Routine Events Other Than Normal Processing		
Event	**Assume Status Update Missing**	**Stop Monitoring Member**
The member's status user routine did not execute in time (system-defined). **Note:** XCF does not schedule the status user routine if a schedule is already outstanding. If a status user routine becomes deadlocked or is in an infinite loop, XCF cannot schedule it again.	X	
The member's status user routine terminated abnormally for the first time.	X	
XCF rescheduled the member's status user routine and it terminated abnormally for the second time.		X
XCF tried to invoke the member's status user routine for the first time, but failed (for example, an error could occur trying to invoke the routine because of an incorrect address or the routine not being loaded.)	X	
XCF tried to invoke the member's status user routine a second time, and failed again.		X
The member changed to a quiesced, failed, or not-defined state.		X
XCF could not access the member's status field.		X

Updating the Status Field

The member requesting XCF status monitoring must provide to XCF a way to identify whether the member is operating normally. The status field serves this purpose. Whether or not the member chooses to regularly update its status field, a missing update causes XCF to schedule the member's status user routine. If the member chooses to regularly update its status field, the member must determine the method of updating.

Updates to the status field should be done in mainline code that is invoked whenever work is being done. Examples are:

- Work unit changes (for example, the program updates the field every time it finishes doing a defined piece of work)
- Inserting messages in a queue
- Initiating transactions
- Writing to a log
- Accessing a database.

One way to update the status field is to store the clock (STCK instruction), which provides a unique, ever-increasing value.

If an effective means of updating the status field is not available, but monitoring is critical, the member can elect not to update the field at all. XCF will keep scheduling the status user routine, allowing the routine to check on the member. However, system performance degradation could occur.

Setting and Changing a Status-Checking Interval

When you identify a status user routine to XCF, you must set a status-checking interval (INTERVAL parameter on IXCJOIN). If the member will be updating the status field, you should set the interval such that the member can update its status field at least once within that interval. The interval is expressed in hundredths of seconds, but must represent full seconds; that is, the value must be greater than 0 and must be a multiple of 100.

Once the interval is set, an active member can change its own interval by using the IXCMOD macro. To use IXCMOD, the member must code the TARGET parameter to provide its own member token, and the INTERVAL parameter to indicate the new value for the interval. The new value must still be greater than zero and a multiple of 100.

Examples of when and how you might use the IXCMOD macro to change a status-checking interval are:

- Synchronizing with the system failure detection interval:
  - The operator changes the system failure detection interval through the SETXCF command. (The system failure detection interval is similar to the status-checking interval, except at a system level rather than a member level. See *z/OS MVS Setting Up a Sysplex* for further information about the system failure detection interval.)
  - XCF notifies all active members on all systems in the sysplex, through their group user routines, of the change to the system failure detection interval.
  - Active members can then issue IXCMOD to change their interval to be a multiple or fraction of the system failure detection interval. A member might do this because the system failure detection interval is an indication of how frequently the system is updating its own status field. A long interval might indicate that the system is running slowly, and consequently, the member's unit of work might also be running slowly.
- Tuning the status-checking interval:

  A member might need to tune its status-checking interval based on how frequently XCF schedules the member's status user routine. If XCF is scheduling the routine many times, only to find that the member is operating normally, the member might want to increase the status-checking interval to lessen system overhead.

Once a member modifies its status-checking interval by invoking IXCMOD, XCF broadcasts the change to the group user routines of the other active members in the group. XCF does not schedule the group user routine of the member requesting the change.

Coding a Status User Routine

When a member wants XCF to monitor its activity, it identifies a status user routine on the IXCJOIN macro (STATEXIT parameter). The member also identifies a status field for XCF to monitor (STATFLD parameter, which must be in fixed or disabled reference (DREF) common storage) and a status-checking interval (INTERVAL parameter). When the member does not update its status field within the status-checking interval, or resumes updating after a confirmed failure, XCF schedules the status user routine. The status user routine determines whether a problem exists, and takes the appropriate action.

User Routine Environment

The status user receives control in the following environment:

Authorization:	Supervisor state and PSW key 0
Dispatchable unit mode:	SRB
Cross memory mode:	PASN = HASN = SASN. The primary address space equals the primary address space of the caller of IXCJOIN, and can be swappable or non-swappable.
Amode:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held

User Routine Recovery

XCF does not provide any recovery for the status user routine. Routines that require recovery must establish their own. XCF does place sufficient information into the SDWA to identify the status user

routine that was in control. The multisystem application must provide whatever diagnostic data is required for problem determination for the status user routine.

XCF will reschedule a status user routine that suffers an error. However, the status user routine should not rely on this as a means of recovery.

Members that identify a status user routine should allow for SRB-to-task percolation. (**Note:** SRB-to-task percolation does not work for address space associated members. See "Member Association" on page 17 for more information.) If XCF processing fails, and XCF does not retry, XCF abnormally ends the task that the member is associated with (either the task or the job step task as specified on IXCJOIN) with a retryable system completion code 00C and reason code 05070000. However, the task's recovery routine does not have to take any action, because the status user routine completed its function before giving up control.

To ensure that the member's recovery can intercept SRB-to-task percolation, the task that the member is associated with must ensure that its recovery routine always receives control when a task abnormally ends. To accomplish this, the associated task should issue the WAIT macro and continue waiting indefinitely while other tasks perform the member's work.

SRB-to-task percolation does not occur while the task's recovery routine is running. XCF waits until the task is not in recovery before abnormally ending the task.

User Routine Processing

XCF invokes a member's status user routine by scheduling a local SRB to the member's primary address space. You are responsible for writing the status user routine, which can do the following:

- Determine if the member's initialization is complete. A member might issue IXCJOIN and its status user routine could get control before IXCJOIN returns to the caller. To determine if the member's initialization is complete, the status user routine might examine a bit that the member sets on or off. The member might want its status user routine to automatically set a return code indicating that the member is operating normally, until the routine determines that initialization is complete.
- Determine whether XCF was checking for a status update missing (first call to the status user routine) or a status update resumed (subsequent call to the status user routine). The status user routine determines this by checking the SEPLSTCH field in IXCYSEPL. SEPLSTCH=SEUPDMIS means checking for status update missing; SEPLSTCH=SEUPDRES means checking for status update resumed.
- If this is the first call, determine if the member is no longer operating normally and whether XCF should broadcast a status change to the other members of the group. If this was not the first call, determine if the member is now operating normally. The status user routine might do one of the following to make this determination:
  - Access the member data value provided through IXCJOIN (SEPLMDAT in IXCYSEPL). The member data might contain addresses of control structures. The status user routine could check the control structures to determine if they are damaged. The control structures might also contain an indication of the member state or user state value for the member.
  - Examine a work queue to determine if the member missed its status update because there was no work to do.
- Take an appropriate action, such as:
  - Keep a count of how many times XCF invokes the status user routine within a particular time interval (for example, within one hour). From this, determine whether the member's status-checking interval should be modified.
  - If the status user routine determines that the member is not operating normally, it can post a task to do recovery for the member.
  - If the status user routine determines that the member is not operating normally, it can issue the SYMREC macro to create a symptom record with diagnostic data.
  - If recovery is not possible, the status user routine might insure that the member is stopped in the event other members are resuming the member's work.
  - Issue IXCMSGOX to provide another member with recovery data.

- Set the appropriate return code. The following summarizes what effect each return code has:
  - A return code of SEUPDRES indicates that the member is operating normally.
  - A return code of SEUPDMIS indicates that the member missed its status update.
  - A return code that matches the value in SEPLSTCH causes XCF to schedule the group user routines and notify them of a status change for that member (either a status update missing or a status update resumed).

See "Using a Status User Routine" on page 63 and Figure 9 on page 66 and Figure 10 on page 67for a complete explanation of how XCF interacts with the status user routine.

Programming Considerations

Consider the following when writing your status user routine:

- To cause XCF to schedule the group user routines of the other active members, the status user routine must set a return code equal to the value in SEPLSTCH.
- Because the status user routine runs in SRB mode, it cannot issue any SVCs. You might want to queue work to one or more tasks for processing and post the tasks when needed.
- The member can pass data to its status user routine in the member data field (MEMDATA parameter on IXCJOIN). This data might be a pointer to some type of communication area, such as a control structure or an ECB. XCF passes member data to the status user routine as part of the parameter list (SEPLMDAT field).
- When XCF is checking for status update missing, XCF will infer the status update missing condition if the status user routine does not complete in time. For example, if the status user routine tries to do recovery for the member, the routine might take too long trying to repair control structures or take a dump. For this reason, you should limit processing in the status user routine. (When XCF is checking for status update resumed, XCF does not make this assumption, so the status user routine can take longer.)

Restrictions

The status user routine **cannot** issue any macros that issue an SVC or that require the caller to be in task mode.

Timing

You should be aware of the following possible events related to timing:

- XCF does not guarantee that it will notify other members of the status update missing condition within a specific time period. The time elapsed depends on the dispatching priority of the address space in which each group user routine is to run.
- XCF maintains information about the member's status changes (status update missing or resumed) and this information is available to other members through IXCQUERY. XCF does not synchronize updates through IXCQUERY with scheduling of the group user routines. However, XCF does provide a timestamp with the information obtained through IXCQUERY.
- The status user routine might receive control after the member issues an IXCLEAVE or IXCQUIES macro, but before the leave or quiesce service completes processing and returns to the caller. In this case, XCF discards any group user routine invocations.
- A member that was reported as having its status update missing might resume execution and begin sending signals to the other active members before the other members receive notification of the status update resumed.

Entry Specifications

XCF passes information to the status user routine in a parameter list and in registers.

Registers at Entry

On entry to the status user routine, the registers contain the following information:

Register	Contents
GPR 0	Used as a work register by the system
GPR 1	Address of the status user routine parameter list (SEPL) mapped by the IXCYSEPL macro.
GPRs 2 - 12	Used as work registers by the system.
GPR 13	Address of a 72-byte work area for use by the status user routine. The user routine does not have to save and restore XCF's registers in this work area. The user routine can use this work area in any way it chooses.
GPR 14	Return address (the status user routine must return control to XCF through a BR 14 or a BSM 0,14.)
GPR 15	Entry point address of the status user routine.
ARs 0 - 15	Used as work registers by the system.

Parameter List Contents

The parameter list that XCF passes to the status user routine is mapped by the IXCYSEPL mapping macro and pointed to by GPR 1. The parameter list is addressable from the primary address space in which the status user routine runs, and includes the following information:

- Member data value provided by the IXCJOIN macro (MEMDATA parameter).
- Address of the member's status field.
- Whether XCF was checking for status update missing or status update resumed.
- The member's token.

See SEPL in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for complete field names and lengths, offsets, and descriptions of the fields mapped by the IXCYSEPL mapping macro.

Return Specifications

On return to XCF, the status user routine sets return codes and puts information in registers.

Registers on Exit

Register	Contents
GPR 0	Can contain application-specified data to be provided to the group user routines if the return code in GPR 15 matches the value in SEPLSTCH. Otherwise, no requirement. See the description of the GEPLUDAT field in the GEPL for information about how register 0 is used to pass application-specified data.
GPR 1 - 13	No requirement.
GPR 14	Unchanged.
GPR 15	Return code.
ARs 0 - 15	No requirement.

Return Codes

Hexadecimal Return Code	Meaning
0 (SEUPDRES)	The member is operating normally.
8 (SEUPDMIS)	The member's status update is missing.

Coded Example

For this example, assume that the member updates its status field each time it completes an item of work on its work queue. The status user routine uses the work queue to determine if the member is operating normally. An empty queue means that the member missed its status update because it had no further work to do.

The status user routine first determines if XCF is checking for status updating missing or status update resumed.

Checking for status update missing

If XCF is checking for status update missing, the routine checks the member's work queue:

- If the work queue is empty, the routine sets a return code of SEUPDRES to indicate the member is operating normally.
- If the work queue is not empty, the routine sets a return code of SEUPDMIS, and places user data (UDATACD) in register 0 to be passed to the group user routines of the other active members of the group.

Checking for status update resumed

If XCF is checking for status update resumed, the routine checks a bit that the member turns on when it resumes updating its status field:

- If the bit is on, the status user routine sets a return code of SEUPDRES and issues the WTO macro to alert the operator that the member's status update has resumed.
- If the bit is off, the routine sets a return code of SEUPDMIS to alert XCF that the member still has not resumed updating its status field.

```
*****************************************************************
*                                                               *
*  STATUS USER ROUTINE                                          *
*                                                               *
*****************************************************************
SEXIT3   CSECT
SEXIT3   AMODE 31
SEXIT3   RMODE ANY
@MAINENT DS    0H
         USING *,R15
         B     @PROLOG
         DC    AL1(16)
         DC    C'SE 90006 SEXIT'
         DROP  R15
*****************************************************************
*                                                               *
*  ENTRY LINKAGE                                                *
*                                                               *
*****************************************************************
@PROLOG  STM   R14,R12,12(R13)
         LR    R12,R15
@PSTART  EQU   SEXIT3
*
*  SET UP BASE REGISTER TO 12
*
         USING @PSTART,R12
         SLR   R15,R15
         IC    R15,@SIZDATD
         SLR   R0,R0
         ICM   R0,7,@SIZDATD+1
         STORAGE OBTAIN,LENGTH=(0),SP=(15)
         LR    R10,R1
         USING @DATD,R10
         ST    R13,4(,R10)
         ST    R10,8(,R13)
         LM    R15,R1,16(R13)
         LR    R13,R10
*****************************************************************
*                                                               *
*  STATUS USER CODE                                             *
*                                                               *
*****************************************************************
```

```
*
*  GET ADDRESSABILITY TO THE PARAMETER LIST
*
         USING SEPL,R1
*
*  IS XCF CHECKING FOR STATUS UPDATE MISSING? IF SO, BRANCH
*
         CLI   SEPLSTCH,SEUPDMIS
         BZ    @MISSING
*
*  XCF IS CHECKING FOR STATUS UPDATE RESUMED
*  GET ADDRESSABILITY TO THE MEMBER DATA, WHICH CONTAINS
*  THE ADDRESS OF MDATASTR.  MDATASTR CONTAINS THE BYTE
*  (RESUMEB) THAT THE MEMBER TURNS ON IF IT RESUMED
*  UPDATING ITS STATUS FIELD.
*
@FOUND   L     R2,SEPLMDAT
         USING MDATASTR,R2
         CLI   RESUMEB,X'01'         IS THE RESUME VALID?
         BNZ   @NOGOOD               IF NOT, BRANCH
*
*  LOAD INDICATOR FOR STATUS UPDATE RESUMED INTO REGISTER 7. THIS
*  WILL BE TRANSFERRED TO REGISTER 15 DURING EXIT LINKAGE.
*  THEN CLEAR REGISTER 8 TO INDICATE NO USER DATA BEING PASSED.
*  REGISTER 8 GETS TRANSFERRED TO REGISTER 0 DURING EXIT LINKAGE.
*
         LA    R7,SEUPDRES           IF VALID, SET RETURN CODES
         SR    R8,R8
         WTO   'STATUS UPDATE HAS BEEN RESUMED',ROUTCDE=11,           X
               LINKAGE=BRANCH,MF=(E,WTOLST1)
         B     @OVER
*
*  STATUS UPDATE IS MISSING SO CHECK QUEUE.  TO DO SO,
*  GET ADDRESSABILITY TO THE MEMBER DATA, WHICH CONTAINS
*  THE ADDRESS OF MDATASTR.  MDATASTR CONTAINS THE WORK QUEUE
*  ADDRESS (WRKQADDR) AND THE ADDRESS OF THE NEXT ITEM OF
*  WORK (NXTWRKAD).
*
@MISSING L     R2,SEPLMDAT
*
*  IF WORK ADDRESS IS ZERO, THE QUEUE IS EMPTY.
*
         CLC   WRKQADDR(4),ZERO      IS THE QUEUE EMPTY?
         BE    @NOWORK               IF YES, BRANCH
*
*  LOAD INDICATOR FOR STATUS UPDATE MISSING INTO REGISTER 7
*
         LA    R7,SEUPDMIS
*
*  LOAD USER DATA INTO REGISTER 8
*
         L     R8,UDATACD
         DROP  R2
         B     @OVER
*
*  BRANCH HERE WHEN THE QUEUE IS EMPTY
*
*
*  LOAD INDICATOR FOR STATUS UPDATE RESUMED INTO REGISTER 7
*
@NOWORK  LA    R7,SEUPDRES           SET RETURN CODE
         B     @OVER
*
*  BRANCH HERE WHEN STATUS UPDATE DID NOT RESUME
*
*
*  LOAD INDICATOR FOR STATUS UPDATE MISSING INTO REGISTER 7
*
@NOGOOD  LA    R7,SEUPDMIS           SET RETURN CODE
*
*  RELEASE DYNAMIC AREA
*
@OVER    LR    R1,R10
         L     R13,4(,R13)
         SLR   R15,R15
         IC    R15,@SIZDATD
         SLR   R0,R0
         ICM   R0,7,@SIZDATD+1
         STORAGE RELEASE,LENGTH=(0),ADDR=(1),SP=(15)
*************************************************************
*                                                           *
*  EXIT LINKAGE                                             *
```

```
*                                                               *
*****************************************************************
*
*  LOAD REGISTER 15 WITH THE RETURN CODE AND REGISTER 0 WITH
*  THE USER DATA TO BE PASSED TO THE GROUP USER ROUTINES
*
         LR    R15,R7
         LR    R0,R8
         L     R14,12(R13)
         LM    R1,R12,24(R13)
         BR    R14
         DS    0H
         DS    0H
@PSIZE   EQU   ((*-SEXIT3+99)/100)*5
         DC    C'PATCH AREA - SEXIT3 90.006'
         PUSH  PRINT
         PRINT ON,GEN,DATA
@PSPACE  DC    25S(*)
         ORG   @PSPACE
         DC    ((@PSIZE+1)/2)S(*)
         ORG   ,
         POP   PRINT
@DATA    DS    0H
@DATD    DSECT
         DS    0F
SAVEAREA DS    18F
@ENDDATD DS    0X
@DYNSIZE EQU   ((@ENDDATD-@DATD+7)/8)*8
SEXIT3   CSECT ,
         LTORG
         DS    0D
UDATACD DC     F'16'
ZERO     DC    F'0'
WTOLST1  WTO   'STATUS UPDATE HAS BEEN RESUMED',ROUTCDE=11,MF=L
WTOLNTH  DC    A(*-WTOLST1)
@SIZDATD DS    0A
         DC    AL1(0)
         DC    AL3(@DYNSIZE)
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
@ENDDATA EQU   *
*****************************************************************
*                                                               *
*  MAPPING OF THE DATA STRUCTURE (MDATASTR) POINTED TO BY       *
*  MEMDATA (SEPLMDAT FIELD IN PARAMETER LIST)                   *
*                                                               *
*  THIS SAME DATA STRUCTURE IS USED BY THE GROUP USER           *
*  ROUTINE.  SOME FIELDS ARE NOT USED BY THE STATUS USER        *
*  ROUTINE, BUT ARE USED ONLY BY THE GROUP USER ROUTINE.        *
*                                                               *
*  TBLADDR         ADDRESS OF TABLE MAINTAINED BY               *
*                    GROUP USER ROUTINE                         *
*  NEXTITEM        ADDRESS OF NEXT AVAILABLE SLOT IN            *
*                    TABLE                                      *
*  WRKQADDR        ADDRESS OF MEMBER'S WORK QUEUE               *
*  NXTWRKAD        ADDRESS OF NEXT AVAILABLE SLOT IN            *
*                    MEMBER'S WORK QUEUE                        *
*  TASKECB         ADDRESS OF THE ATTACHED TASK'S ECB           *
*                    (THIS TASK IS ATTACHED BY THE MAIN         *
*                    ROUTINE.  THIS FIELD IS USED BY            *
*                    THE GROUP USER ROUTINE.)                   *
*  MAINECB         ADDRESS OF ECB USED FOR SYNCHRONIZING        *
*                    (THE MAIN ROUTINE WAITS ON THIS ECB,       *
*                    WHICH THE ATTACHED TASK POSTS WHEN         *
*                    IT COMPLETES ITS WORK.)                    *
*  FUNCTON         GROUP USER ROUTINE TURNS THIS SWITCH ON      *
*                    WHEN CALLED FOR THE FIRST TIME FOR A       *
*                    STATUS UPDATE MISSING.                     *
```

```
*  RESUMEB         MAIN ROUTINE TURNS THIS SWITCH ON          *
*                    WHEN IT RESUMES UPDATING ITS STATUS      *
*                    FIELD.                                   *
*                                                             *
***************************************************************
MDATASTR DSECT
TBLADDR  DS    1F
NEXTITEM DS    1F
WRKQADDR DS    1F
NXTWRKAD DS    1F
TASKECB  DS    1F
MAINECB  DS    1F
FUNCTON  DS    X
RESUMEB  DS    X
***************************************************************
*                                                             *
*  MAPPING MACROS                                             *
*                                                             *
***************************************************************
         IXCYSEPL
         END   SEXIT3
```

Notifying Members of Changes

A member requests XCF to notify it of changes to other members in the group or to systems in the sysplex by identifying a group user routine to XCF on the IXCJOIN macro. A member or a multisystem application can also request information about changes to systems in the sysplex by identifying an ENF event code routine. ENF code 35 provides function codes to notify listeners when a system has joined a sysplex or has been removed from a sysplex.

This section contains information on the following topics related to the group user routine:

- How XCF works together with the group user routine
- Events that cause XCF to schedule a group user routine
- How to code a group user routine.

For information about using an ENF event code routine, see “Using ENF Event Code 35” on page 250.

How XCF Works Together with the Group User Routine

XCF works together with the group user routine in the following manner:

- XCF schedules the group user routines of active members of the group when specific events occur (such as a member changing state or a missing status update). For an event that affects multiple members, XCF provides a separate notification regarding each affected member.
- XCF passes information to the group user routine through a parameter list (mapped by the IXCYGEPL mapping macro and pointed to by GPR 1).
- The group user routine takes action based on the information in the parameter list.

Events that Cause XCF to Schedule a Group User Routine

The events that cause XCF to schedule an active member's group user routine fall into these categories:

- Events about which the member expects to be notified and must act upon. Examples of such events are:
  - If a member joins the group, XCF notifies the group user routines of the other active members in the group. If a notified member is keeping a table of all the members in the group, the notified member would update its table. Also, the notified members might need to include the new member in any group dialogue.
  - If the operator changes the system failure detection interval, a member that wants its status-checking interval synchronized with the system failure detection interval would issue IXCMOD to modify its interval.

    Note: The group user routine itself cannot issue IXCMOD, but can post a task to do so.

- Events about which the member expects to be notified, but is not concerned. For example:
  - If the operator changes the system failure detection interval, XCF notifies the group user routines of all active members on all systems in the sysplex. If the member is not concerned about keeping its status-checking interval synchronized with the system failure detection interval, the member might ignore this notification.
- Events about which the member does not expect to be notified. Examples of such events are:
  - If no members in the group are using the user state field, the member would not expect notification of a user state value change.
  - If no members in the group are using the XCF status monitoring service, the member would not expect notification of member status changes. (The member would, however, expect to receive notification of **system** status changes.)
- Events from which the member must infer that other events have occurred. XCF insures that members are notified of only the most current events by skipping the notification of events that have been superseded by later events. To fully understand this concept, consider these examples:
  - If a member issued three changes to its user state value in a short time, it is possible that XCF will notify the group user routines of only the latest change. This is because XCF might not have had the chance to deliver the first two notifications before the third change occurred. The group user routines might then need to infer that the other two changes occurred, depending on how the user state field is being used.
  - If a member changes to an active state from some other state, XCF might not be able to notify the group user routines of the member state change before some other event (such as a user state change) occurs indicating the member is active. In that case, XCF does not issue the member state change notification, and the group user routines have to infer that it occurred.

  See "Skipping of Events" on page 81 for more information, including:
  - A table (Table 6 on page 82) you can use to determine, based on the event type presented to the group user routine, what events XCF might have skipped.
  - A discussion of how the skipping of events relates to designing the user state field.
- Unknown events, which the member can ignore. By ignoring unknown events, rather than allowing them to abnormally terminate your program, you make your program independent of future MVS releases that might introduce additional event codes.

Thinking of the events in terms of the categories just described will help you design and code the group user routine. See "Coding a Group User Routine" on page 85 for further details.

Table 5 on page 78 summarizes the events (GEPLTYPE field in the parameter list) that cause XCF to schedule the group user routines of active members, along with the corresponding event type (IXCYGEPL constant), and which group user routines are scheduled. The **Member- or System-Related** column indicates whether the **notification** is about a member or a system. In some cases, an event occurs that affects the system a member is running on, consequently affecting the member. The notification actually pertains to the member affected by the system event (see GESYSSUM, GESYSSUR, GESYSDM, and GESYSGO). Following this figure is a detailed description of each event type.

Table 5. Events that Cause XCF to Schedule a Group User Routine

Event	Event Type (IXCYGEPL Constant)	Member- or System-Related	XCF schedules the group user routines for the following members:
Member state changed	GEMSTATE	Member	Other active members of the group.
User state value changed	GEUSTATE	Member	All active members of the group, including the affected member.
Member status update missing	GEMSUMSE	Member	Other active members of the group.
	GEMSUMDI		

Table 5. Events that Cause XCF to Schedule a Group User Routine (continued)

Event	Event Type (IXCYGEPL Constant)	Member- or System-Related	XCF schedules the group user routines for the following members:
Member status update resumed	GEMNOSUM	Member	Other active members of the group.
System reported active	GESYSACT	System	All active members of all groups in the sysplex.
System status update missing	GESYSSUM	Member	Other active members of the group on other systems.
System status update resumed	GESYSSUR	Member	Other active members of the group on other systems.
System reported going	GESYSGO	Member	Other active members of the group on other systems.
System reported gone	GESYSGON	System	All active members in the sysplex, regardless of whether they have group members on the removed system. (1)
System detected missing	GESYSDM	Member	The active member whose system stopped and then resumed; XCF might have issued a GESYSSUM and GESYSSUR.
System detected gone	GESYSDG	System	All active members in the sysplex, regardless of whether they have group members on the removed system. (1)
System failure detection interval changed	GESYSFDI	System	All active members of all groups on all systems in the sysplex.
Member status-checking interval changed	GESUBFDI	Member	Other active members of the group.
System being removed from the sysplex	GESYSPRT	System	All active members on the system that is about to be removed from the sysplex.
Member status monitoring removed	GEMONREM	Member	All active members of the group, including the affected member.

Note:

1. Those members that have group members on the removed system also receive notification of member state changes for the affected members (from active to not-defined, failed, or quiesced).

The following is an explanation of each event type that causes XCF to schedule the group user routines of active members:

GEMSTATE
Any member state change, other than a change from quiesced or failed to not-defined, could have occurred through IXCCREAT, IXCJOIN, IXCQUIES, IXCLEAVE, or IXCDELET.

For members that terminate without issuing IXCQUIES or IXCLEAVE to explicitly disassociate from XCF, the following could have occurred:

- An active member with permanent status recording became failed.
- An active member without permanent status recording became not-defined.

Note: The member's termination could be either normal or abnormal. If abnormal, the member's termination could have been caused by task, address space, or system failure.

GEUSTATE
A member's user state value changed through the IXCSETUS macro. The member could have changed its own user state value, or it could have been changed by another member.

GEMSUMSE
A member's status user routine reported that the member's status update was missing.

GEMSUMDI
The XCF status monitoring service assumed a status update missing for the member. Either of the following could have occurred:

- The status user routine did not execute in time.
- The status user routine terminated abnormally.

GEMNOSUM
A member's status user routine reported that the member's status update resumed after a confirmed or assumed status update missing.

GESYSACT
A system joined the sysplex.

GESYSSUM
A member's system missed updating its system status field, causing XCF to consider the **member** as missing. XCF reports the GESYSSUM event type when a system does not update its status field within:

- The member's status-checking interval (INTERVAL parameter defined on IXCJOIN or modified on IXCMOD), if the member requested XCF status monitoring service
- The system failure detection interval, if the member did not request the XCF status monitoring service.

XCF on each system in the sysplex monitors every other system in the sysplex. However, this monitoring is not synchronized between systems, so every system in the sysplex might not be simultaneously aware when a particular system misses a status update. Also, a problem on one system in the sysplex might prevent that system from detecting a missing status update on another system. The following example illustrates which group user routines XCF notifies for a system status update missing (GESYSSUM) and a system status update resumed (GESYSSUR):

- System 1, system 2, system 3, and system 4 all reside in the same sysplex.
- System 1 misses a status update.
- System 2 detects that system 1 missed its status update, causing XCF on system 2 to consider the members on system 1 as missing. XCF on system 2 issues the GESYSSUM event to the group user routines of the active members on system 2.
- Then system 1 resumes updating its status field.
- System 2 detects that system 1 resumed updating its status field, causing XCF on system 2 to consider the members on system 1 as no longer missing. XCF on system 2 issues the GESYSSUR event to the group user routines of the active members on system 2.
- System 1 resumed updating its status field before system 3 detected that the update was missing. To system 3, it appears as though system 1 never missed its status update. XCF on system 3 does not issue the GESYSSUM or GESYSSUR events to the group user routines of the active members on system 3.
- System 4 is spinning to obtain a lock, and so does not detect that system 1 missed its status update and later on resumed. XCF on system 4 does not issue the GESYSSUM or GESYSSUR events to the group user routines of the active members on system 4.

GESYSSUR
A member's system resumed updating its status field following a system status update missing condition, causing XCF to consider the **member** no longer missing. See GESYSSUM for an explanation of the circumstances under which XCF issues a system status update missing condition, and for an explanation of which group user routines XCF notifies for the GESYSSUM and GESYSSUR events.

GESYSGO
A member on another system is about to be terminated because its system is about to be removed from the sysplex. Other members can begin recovery for the member. (The member may still have access to multisystem resources.) XCF will also issue a member state change relative to the member's final state (a change from active to not-defined, failed, or quiesced).

GESYSGON
A system was removed from the sysplex and XCF already issued an event type of GESYSGO.

If the SYSCLEANUPMEM parameter was coded on the IXCJOIN macro, then the IXCSYSCL macro must be issued by this routine when cleanup for the removed system has completed or if no cleanup is required.

When z/OS is running on a z15™ server that supports recovery process boosts, to facilitate the actions that a group member performs for system-wide cleanup after a system is removed from the sysplex, XCF notification will request System Recovery Boost's recovery process boost support on all surviving systems in the sysplex as they take on the additional workload of sysplex partitioning related recovery.

For more information on System Recovery Boost, see: IBM Z System Recovery Boost Content Solution (www.ibm.com/support/z-content-solutions/system-recovery-boost/) or IBM Documentation (www.ibm.com/docs/en/zos).

GESYSDM
A member's system stopped updating its status field and then resumed. XCF issues the GESYSDM event type to notify the **resuming member's group user routine** that other members might have taken some action on the member's behalf while it was stopped.

GESYSDG
A system was removed from the sysplex before XCF could issue an event type of GESYSGO.

If the SYSCLEANUPMEM parameter was coded on the IXCJOIN macro, then the IXCSYSCL macro must be issued by this routine when cleanup for the removed system has completed or if no cleanup is required.

When z/OS is running on a z15™ server that supports recovery process boosts, to facilitate the actions that a group member performs for system-wide cleanup after a system is removed from the sysplex, XCF notification will request System Recovery Boost's recovery process boost support on all surviving systems in the sysplex as they take on the additional workload of sysplex partitioning related recovery.

For more information on System Recovery Boost, see: IBM Z System Recovery Boost Content Solution (www.ibm.com/support/z-content-solutions/system-recovery-boost/) or IBM Documentation (www.ibm.com/docs/en/zos).

GESYSFDI
The system failure detection interval changed.

GESUBFDI
A member issued the IXCMOD macro to change its status-checking interval (INTERVAL parameter on IXCJOIN).

GESYSPRT
A member's system is about to be removed from the sysplex. Members receiving this notification should clean up any resources they are using, and issue IXCLEAVE or IXCQUIES as soon as possible.

GEMONREM
XCF stopped monitoring a member for one of the following reasons:

- XCF could not access the member's status field.
- The member's status user routine terminated abnormally two consecutive times.
- XCF tried to issue the member's status user routine two consecutive times and failed.

Note: When XCF stops monitoring a member, this does not cause the member to change states. To reinstate monitoring after XCF stops monitoring a member, the member must issue IXCLEAVE or IXCQUIES, and then issue IXCJOIN once again with the STATEXIT, STATFLD, and INTERVAL parameters.

Skipping of Events

It is possible for XCF to skip notification of certain events when they are superseded by later events. This allows XCF to present only the latest information to the group user routines. For example, a member might join a group and then change its user state value (through IXCSETUS) within a very short time. XCF

might not be able to notify the group user routines of the member state change before the user state change occurs. In that case, XCF skips the member state change notification and presents the user state change notification. The group user routine can then infer, if it needs to, that the member state change occurred. (Only active members can issue IXCSETUS to change their user state field, so the group user routine can infer that the member became active.)

This section includes the following information related to skipping of events:

- What events XCF might skip
- How to determine when events are skipped
- How the skipping of events relates to designing a user state field.

Events That XCF Might Skip

Table 6 on page 82 lists the events that XCF might present to a group user routine and the corresponding events that XCF might have skipped. Events are listed by their IXCYGEPL constant and decimal equivalent. You can use this table to help you design your group user routine. One technique for using this table is to ignore those columns that pertain to events your routine does not expect to receive, or expects to receive but is not concerned about. Most group user routines will need to provide code for only a small subset of this table.

Table 6. Skipping of Events Presented to Group User Routines

Event Presented	**Events XCF Might Have Skipped**															
	1	**2**	**7**	**8**	**9**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**21**	**22**	**23**
GEMSTATE (1)	X	X	X	X	X									X		X
GEUSTATE (2)	X	X	X	X	X									X		X
GEMSUMSE (7)	X	X	X	X	X									X		X
GEMSUMDI (8)	X	X	X	X	X									X		X
GEMNOSUM (9)	X	X	X	X	X									X		X
GESYSACT (11)																
GESYSSUM (12)		X	X	X	X		X	X						X		X
GESYSSUR (13)		X	X	X	X		X	X						X		X
GESYSGO (14)		X	X	X	X		X	X						X		X
GESYSGON (15)																
GESYSDM (16)		X*														
GESYSDG (17)																
GESYSFDI (18)																
GESUBFDI (21)	X	X	X	X	X									X		X
GESYSPRT (22)																
GEMONREM (23)	X	X	X	X	X									X		X

* When XCF presents the GESYSDM event to a member's group user routine, XCF might have skipped the GEUSTATE event only if it was a notification of a change in the member's **own** user state. XCF will not skip GEUSTATE events about other members of the group due to the GESYSDM.

Note:

1. Event types 3, 4, 5, 6, 10, 19 and 20 are not used.
2. XCF skips events on a member basis. For example, XCF does not skip notification about an event that happened to member A based on an event that happened to member B.

Determining When Events Are Skipped

The following explains how a group user routine can determine if XCF skipped notification of the indicated event:

GEMSTATE (member state change)
Compare the current value of GEPLOLDS with the previous value. To do this comparison, the routine had to save the value of GEPLOLDS on a previous invocation. For example, if the last invocation of the group user routine indicated a GEPLOLDS of created, and now its value is quiesced, then XCF must have skipped a GEMSTATE with a GEPLOLDS of active. See "The Five Member States" on page 12, which illustrates the transition of XCF members from one state to another, for help in determining if a state was skipped.

The routine might not be able to determine if XCF skipped some member state transitions. For example, if this is the first invocation of the group user routine and the current value of GEPLOLDS is active, the routine might not be able to determine which GEMSTATE events were skipped. A member can become active from a not-defined, created, quiesced, or failed state. The routine can look at the GEPLHSTY for additional information. (See "Parameter List Contents" on page 88.)

GEUSTATE (user state value change)
Compare the currently presented user state value with the previously presented user state value. However, a difference in the user state values does not necessarily indicate a skipped GEUSTATE event. Another service (such as the IXCQUIES macro), rather than the IXCSETUS macro, might have changed the user state.

The routine might detect a difference in the user state values, and subsequently receive the GEUSTATE notification. For example, a member might change its status-checking interval, and then change its user state value before XCF delivers the notification of the changed interval. When XCF presents the changed interval, the parameter list passed to the group user routine contains the new user state value. So the group user routine could detect the new user state value before XCF presents the GEUSTATE event.

GEMSUMSE (member status update reported missing)
Check the GEPLMISR bit in the parameter list. The GEPLMISR bit is on when a member's status user routine confirms that the member's status update is missing. The GEPLMISR bit is off when a member's status update was never reported missing, or after a member's status user routine confirms that the member's status update resumed. This bit indicates the current monitored state and does not necessarily indicate that the event was skipped.

If XCF presents a GEMNOSUM event without having first presented a GEMSUMSE, the routine can assume that XCF skipped the GEMSUMSE.

The routine might not be able to determine that XCF skipped a GEMSUMSE event, because XCF might skip both the GEMSUMSE and GEMNOSUM events.

GEMSUMDI (member status update assumed missing)
Check the GEPLMISD bit in the parameter list. The GEPLMISD bit is on when XCF assumes that a member's status update is missing. The GEPLMISD bit is off if the member's status update was never assumed missing, or after the member's status user routine confirms that the member's status update resumed. This bit indicates the current monitored state and does not necessarily indicate that the event was skipped.

If XCF presents a GEMNOSUM event without having first presented a GEMSUMDI, the routine can assume that XCF skipped the GEMSUMDI.

The routine might not be able to determine that XCF skipped GNAMMSUMDI event, because XCF might skip both the GEMSUMDI and GEMNOSUM events.

GEMNOSUM (member status update resumed)
Check whether XCF failed to present a GEMNOSUM event after it presented a GEMSUMDI or GEMSUMSE event and one of the following is true:

- A GEMSUMSE event occurred and the GEPLMISR bit in the parameter list is off. The GEPLMISR bit is off if the member's status update was never reported missing, or after a member's status user routine confirms that the member's status update resumed.
- A GEMSUMDI event occurred and the GEPLMISD bit in the parameter list is off. The GEPLMISD bit is off if the member's status update was never assumed missing, or after the member's status user routine confirms that the member's status update resumed.

The GEPLMISR and GEPLMISD bits indicate the current monitored state. They do not by themselves indicate that an event was skipped.

The routine might not be able to determine that XCF skipped a GEMNOSUM event, because XCF might skip both the GEMSUMSE (or GEMSUMDI) and GEMNOSUM events.

GESYSACT (system reported active)
GESYSACT events are not skipped.

GESYSSUM (system status update missing)
If XCF presents a GESYSSUR event without having presented a GESYSSUM, assume XCF skipped the GESYSSUM.

GESYSSUR (system status update resumed)
If XCF presents two GESYSSUM events without an intervening GESYSSUR, assume the GESYSSUR event was skipped.

GESYSGO (system reported going)
GESYSGON (system reported gone)
GESYSDM (system detected missing)
GESYSDG (system detected gone)
XCF does not skip these events.

GESYSFDI (system failure detection interval changed)
XCF does not skip GESYSFDI events. However, XCF always presents the most current value of the system failure detection interval. Thus, two or more GESYSFDI events might present the same system failure detection interval even though the interval was changed two or more times. This situation is similar to that described under GEUSTATE.

GESUBFDI (member status-checking interval changed)
Compare the current value of GEPLINTV to the previously received value for which the member was active. However, the routine might detect that these two values are different, and subsequently be presented with the GESUBFDI event. This situation is similar to that described under GEUSTATE.

GESYSPRT (system being removed from the sysplex)
XCF does not skip GESYSPRT events.

GEMONREM (member status monitoring removed)
Check the GEPLMONR bit in the parameter list. The GEPLMONR bit is on if XCF removed monitoring for the member. However, this bit indicates that the event occurred and does not necessarily indicate that the event was skipped.

User State Field Design Considerations

The fact that XCF skips certain events is an important consideration in designing the user state field. In general, members should not use the user state field to communicate critical information to other members of the group, because the possibility exists that XCF might not present every user state change.

To communicate critical information to other members of the group, a member should use the XCF signaling service. XCF will either deliver messages, or provide notification that the target member or the target member's system failed. However, XCF might not deliver messages in the sequence in which they were sent. If the sequence of the messages is important, members can save the timestamp, or a sequence number, associated with each message.

The following examples illustrate how the skipping of events relates to the user state field:

- If you use the user state field to contain a data set name, and the current data set name is all your program needs, it is of no consequence if a user state change is skipped.

- If you use the user state field to record events with corresponding data, and your program needs only the latest event, the skipping of events does not cause a problem. If your program is tracking every event in a table, then the skipping of events will result in an incomplete table.

Coding a Group User Routine

When a member wants to be notified of changes to other members in the group, or of changes to systems in the sysplex, the member identifies a group user routine on the IXCJOIN macro (GRPEXIT parameter). When an event occurs that causes XCF to schedule a group user routine, the routine should take the appropriate action based on the event.

User Routine Environment

The group user routine receives control in the following environment:

Authorization:	Supervisor state and PSW key 0
Dispatchable unit mode:	SRB
Cross memory mode:	PASN = HASN = SASN. The primary address space equals the primary address space of the caller of IXCJOIN, and can be swappable or non-swappable.
Amode:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held

User Routine Recovery

XCF does not provide any recovery for the group user routine. Routines that require recovery must establish their own. XCF does place sufficient information into the SDWA to identify the group user routine that was in control. The multisystem application must provide whatever diagnostic data is required for problem determination for the group user routine.

A member's group user routine is given two chances to complete event notification processing of a particular event. The first time the group user routine ends abnormally while handling an event, XCF returns an abend code of X'00C' with a reason code X'060B0000' to the recovery routine of the task that the member is associated with (either the task or the job step task as specified on IXCJOIN.

Note: SRB-to-task recovery cannot be provided for members that are address space associated. See “Member Association” on page 17 for more information. The member's group user routine might or might not have completed event notification processing before terminating abnormally. The task's recovery routine does not have to take any action.

XCF will give the member's group user routine control again and set the GEPLSECC bit in the group user routine parameter list to show that this is the second time the member's group user routine is being given control for this event. (See “Parameter List Contents” on page 88 for an explanation of the fields in the group user routine parameter list.) Members that cannot handle processing the same event twice should check the GEPLSECC bit on entry to the group user routine. If the bit is on, the group user routine should determine if the event had already been processed before the routine terminated abnormally.

If the member's group user routine ends abnormally a second time or if XCF cannot give the member's group user routine control again, XCF abnormally ends the task (that the member is associated with) with a retryable abend code of X'00C' and a reason code of X'060B0001'. This abend code and reason code combination indicates that the member's group user routine could not finish processing the event notification information so at least some of the information may have been lost. The task's recovery routine can do one of the following:

- Issue the IXCQUERY macro to determine any lost information.
- Back up to some logical point and continue processing from that point.

- Allow the task to abnormally end.

To ensure that the member's recovery can intercept SRB-to-task percolation, the task the member is associated with must ensure that its recovery routine always receives control when a task abnormally ends. To accomplish this, the associated task should issue the WAIT macro and continue waiting indefinitely while other tasks perform the member's work.

SRB-to-task percolation does not occur while the task's recovery routine is running. XCF waits until the task is not in recovery before abnormally ending the task.

User Routine Processing

When an event occurs, such as a member state change or a status change, XCF invokes the group user routines of active members. Each group user routine is scheduled as a local SRB to the owning member's home address space. You are responsible for writing the group user routine. You can design the routine to:

- Determine if the member's initialization is complete. A member might issue IXCJOIN and its group user routine could get control before IXCJOIN returns to the caller. To determine if the member's initialization is complete, the group user routine might examine a bit that the member sets on or off. The member might want its group user routine to ignore all events until the routine determines that initialization is complete.
- Categorize the events as described in "Events that Cause XCF to Schedule a Group User Routine" on page 77. With these categories in mind, the group user routine should:
  - Determine what events occurred. The group user routine must consider both the event for which XCF is providing notification, and the events that XCF might have skipped. See Table 5 on page 78 and Table 6 on page 82 for more information.
  - Take the appropriate action.

To determine what event occurred, the group user routine can check the GEPLTYPE field in the parameter list. Here are some examples of what the group user routine might do based on what it finds in GEPLTYPE:

- If the GEPLTYPE field indicates a member state change (GEPLTYPE=GEMSTATE), the routine can then check the GEPLOLDS and GEPLNEWS fields to determine the member state before the event occurred and the member state after the event occurred. For example, if the member went from **active** to **failed**, the group user routine might do recovery for the failed member, cleaning up any resources the member was using. Then the group user routine could post a task to issue IXCDELET to disassociate the member from XCF.

  Note: XCF schedules a group user routine for only one event at a time. If you perform recovery in the group user routine rather than posting a task, you could delay XCF's notification of other events and XCF might skip those notifications.
- If the GEPLTYPE field indicates a user state field change (GEPLTYPE=GEUSTATE), the group user routine might then check the user state field, which is passed as part of the parameter list, and take the appropriate action based on the contents of the field.
- If the GEPLTYPE field indicates that the member's status update is missing, the group user routine might:
  - Post a task to change the member's user state field to indicate that another member took over the member's activities.
  - Post a task to do the takeover for the member.

Programming Considerations

Consider the following when writing your group user routine:

- Because the group user routine runs in SRB mode, it cannot issue any SVCs. You might want to queue work to one or more tasks for processing and post the tasks when needed.

- A member might consider having a group user routine, even if the member is the only member of a group that is running on a single-system sysplex. With a group user routine, the member can be notified of system events.
- If you plan to have your group user routine maintain a table of systems, groups, and members in the sysplex, you should consider serializing the use of this table with other units of work that require access to it.
- XCF schedules the group user routine for only one event at a time. By having other units of work available to process actions that are time consuming, you can avoid missing events.
- You should design your group user routine to ignore unknown event codes rather than allowing an unknown event code to abnormally terminate your program. This makes your program independent of future MVS releases that might introduce additional event codes.

Restrictions

The group user routine **cannot** issue any macros that issue an SVC or that require the caller to be in task mode.

Timing

You should be aware of the following possible events related to timing:

- XCF does not notify the group user routines about every event. XCF skips some events, and the group user routines must be able to infer events if necessary. Examples appear in "Events that Cause XCF to Schedule a Group User Routine" on page 77 and more detail is provided in Table 6 on page 82.
- Once a member issues IXCLEAVE or IXCQUIES, XCF no longer schedules the member's group user routine (XCF schedules the group user routines of **active** members only). If the group user routine is in control when IXCLEAVE or IXCQUIES is issued, the routine completes normally before the leave or quiesce service returns control.
- When a member's group user routine receives notification that the member's system is about to be removed from the sysplex (event type GESYSPRT), the member should clean up any resources it is using, and issue IXCLEAVE or IXCQUIES as soon as possible.

Entry Specifications

XCF passes information to the group user routine in a parameter list and in registers.

Registers at Entry

On entry to the group user routine, the registers contain the following information:

Register	Contents
GPR 0	Used as a work register by the system.
GPR 1	Address of the group user routine parameter list (GEPL), mapped by the IXCYGEPL macro.
GPRs 2 - 12	Used as work registers by the system.
GPR 13	Address of a 72-byte work area for use by the group user routine. The user routine does not have to save and restore XCF's registers in this work area. The user routine can use this work area in any way it chooses.
GPR 14	Return address (the group user routine must return control to XCF through a BR 14 or a BSM 0,14.)
GPR 15	Entry point address of the group user routine.
ARs 0 - 15	Used as work registers by the system.

Parameter List Contents

The parameter list that XCF passes to the group user routine is mapped by the IXCYGEPL mapping macro and pointed to by GPR 1. The parameter list is addressable from the primary address space in which the group user routine runs. The group user routine interprets the information in the parameter list according to the type of event that caused XCF to schedule the routine, and according to the contents of the user state field. Note that XCF does not always provide the contents of all IXLYGEPL fields in the parameter list. For example, some information in IXLLYGEPL might not be provided for events related to a system. Events that are related to a system are:

- A system left or joined the sysplex (event type=GESYSGON, GESYSDG, GESYSPRT, or GESYSACT.
- A system changed its failure detection interval (event type=GESYSFDI).

The following describes each field in the parameter list, and how to interpret the field depending on the event:

GEPLTYPE (event type)
Contains the event type. XCF provides this information for all events.

GEPLETIM (event time)
Contains the clock time (in Greenwich mean time) at which the event occurred. XCF provides this information for all events.

GEPLMDAT (member data)
Contains member data that the member owning the group user routine specified on IXCJOIN (MEMDATA parameter). You might use this field to contain pointers to control structures, or other information that the group user routine needs. XCF provides this information for all events.

GEPLGNAM (group name)
Contains the group name of the affected member's group. XCF provides this information for all events except those related to a system.

GEPLMNAM (member name)
Contains the member name of the affected member. XCF provides this information for all events except those related to a system.

If XCF is unable to determine the member name, this field contains blanks. In that case, check the GEPLMTOK field for the member's token.

GEPLMTOK (member token)
Contains the member token of the affected member. XCF provides this information for all events except those related to a system.

Note: XCF issues a new member token when a created, quiesced, or failed member issues IXCJOIN to become active.

GEPLOLDS (old member state)
Contains the member state of the affected member before the event occurred. XCF provides this information for all events except those related to a system.

GEPLNEWS (new member state)
Contains the member state of the affected member after the event occurred. XCF provides this information for all events except those related to a system.

GEPLSYS (system name)
For all events related to a member, this field contains the system name of the affected member's system (except for **created** members, which are not associated with a system).

For events related to a system, this field contains the system name of the affected system.

GEPLSID (system token)
For all events related to a member, this field contains the system token of the affected member's system.

For events related to a system, this field contains the system token of the affected system.

GEPLUSOF (user state offset)
For all events related to a member, this field contains the offset from GEPL of the 32-byte user state field containing the affected member's current user state value. Regardless of the length the member specified for the user state field, the group user routines receive all 32 bytes.

When XCF places a terminated member in the not-defined state, and XCF could not determine the user state value, the user state field contains 32 bytes of X'FF'.

GEPLHSTY (history)
Contains a list of the last eight events that affected the member, including event time and expected duration, in LIFO order. If fewer than eight events occurred for the member, the unused fields contain zeros. XCF provides this information for all events except those related to a system.

GEPLUDAT (user data)
When the affected member's status user routine indicates that the member's status update is missing or has resumed (event types GEMSUMSE and GEMNOSUM), this field contains the data that the affected member's status user routine placed in GPR 0. GEPLUDAT contains valid data when GEPLMONR and GEPLMISD are both off and the user data value in GEPLUDAT is not zero. This assumes that the affected member's status user routine does not pass a data value of zero, which would be make it impossible to determine whether a value was passed in GEPLUDAT.

GEPLINTV (interval)
When a system updates its failure detection interval (event type = GESYSFDI), this field contains the new system failure detection interval.

GEPLMEME (member-related event)
This bit is on when XCF presents a member-related event, and off when XCF presents a system-related event.

GEPLMONR (monitoring removed)
This bit is on when XCF removes monitoring for the member.

GEPLMISR (member status update reported missing)
This bit is on after the member's status user routine confirms that the member's status update is missing. This bit is off if the member's status update was never reported missing, or after the member's status user routine confirms that the member's status update resumed.

GEPLMISD (member status update assumed missing)
This bit is on after XCF assumes a status update missing for the member. This bit is off if the member's status update was never assumed missing, or after the member's status user routine confirms that the member's status update resumed.

GEPLSECC (member group user routine called a second time for the event)
This bit is on after the member's group user routine percolates to XCF's recovery routine, and XCF calls the group user routine a second time for the same event. This bit is off when XCF calls a member's group user routine the first time for a particular event.

Systems at z/OS V1R2 and higher support an extended version of IXCYGEPL. The following describes each field in the extended GEPL (GEPL1):

GEPL1_VERSION
Version number of GEPL/

GEPL1_TARGETMEMTOKEN
Member token of the member whose group exit is being driven.

GEPL_KONLYBASE
Compare this value to GEPLUSOF to determine whether the GEPL1 is available for use by the group exit.

GEPL_KVERSION1
Version 1 of GEPL.

To determine which version of IXCYGEPL is being used, compare the contents of GEPLUSOF with the constant GEPL_KONLYBASE. If the comparison is equal, then only the fields defined by the base version of IXCYGEPL exist (GEPL1 does not exist). If the comparison is not equal, then the fields in the extended version (GEPL1) do exist along with the fields defined by the base version of IXCYGEPL.

See GEPL in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for complete field names and lengths, offsets, and descriptions of the fields mapped by the IXCYGEPL mapping macro.

Return Specifications

On return to XCF, the group user routine does not have to set any return codes or place any information in the GPRs. GPR 14 should still contain the return address, because the group user routine must return control to XCF through a BR 14 or a BSM 0,14.

Coded Example

In this example of a group user routine, the routine has two functions to perform:

- Maintain a table containing the member names and member tokens of members in the group. When a member becomes active, the group user routine adds the member to the table.
- Initiate® a takeover if the primary member in the group fails. This routine runs in a group where one member is the primary member and another member is the backup member.

To accomplish these functions, the group user routine is concerned with two types of events:

- Member state changes (GEPLTYPE=GEMSTATE).
- Member status update missing. To test for this event, the routine uses a test-under-mask operation, with a mask of X'30'. This covers the following conditions:
  - GEPLFLG2 = X'20', indicating that the GEPLMISR bit is on (the member's status user routine reported a status update missing for the member)
  - GEPLFLG2 = X'10', indicating that the GEPLMISD bit is on (XCF assumed a status update missing for the member).

The group user routine also has to be concerned with skipping of events. The logic of the routine covers the possibility that either a status update missing or a member state change event might have been skipped.

The following describes the group user routine logic (see Figure 11 on page 92,Figure 12 on page 93, and Figure 13 on page 94 for a summary).

- The routine first checks to see if the event is member-related. If not, the routine takes no action.
- If the event is member-related, the routine then checks to see if one of the status update missing flags (GEPLMISR or GEPLMISD) is on.

If the GEPLMISR and GEPLMISD flags are both off

The routine checks to see if the table needs to be updated by doing the following:

- Checks to see if the member is currently active. If so, the routine loops through the table. If the member is already in the table, the routine updates the member token. If the member is not already in the table, the routine adds the member. This covers the case where a member might have become active, but the GEPLTYPE=GEMSTATE event was skipped.
- If the member is not currently active, the routine checks to see if it is being called for a member state change.
  - If this is not a member state change, the routine takes no action.
  - If this is a member state change, the routine checks the previous member state to see if the member was active. (This covers another case where a GEPLTYPE=GEMSTATE event might have been skipped.) If the member was active at one time, the routine checks to see if the member is in the table, and adds the member or updates the member as appropriate. Otherwise, the routine takes no action.
- If the member is currently active, the routine checks to see if the member is in the table, and adds the member or updates the member as appropriate.

If either the GEPLMISR flag or the GEPLMISD flag is on

This indicates a status update missing for the member. When the member's status update is missing, the routine has to initiate a takeover so that another member can assume the member's work. However, once either of these flags is turned on, XCF does not turn it off until the member resumes updating its status field. The group user routine might be called several times, and this flag might continuously be on. So, the routine has a switch (FUNCTON) that it turns on the first time it is called for a status update missing.

The routine does the following when first called with either the GEPLMISR flag or the GEPLMISD flag on:

- The routine turns on its own switch (FUNCTON).
- The routine posts a task whose job is to change user state values so that the backup member becomes the primary member. The main routine attached this task with an ECB, and passed a parameter list containing the address of the member's data structure. The data structure contains the information the task needs to do the takeover.
- The routine then checks to see if the member must be added to the table.

If this is not the first time the routine is called with either the GEPLMISR flag or the GEPLMISD flag on, the routine knows that no takeover work needs to be done. The routine then checks to see if the member must be added to the table.

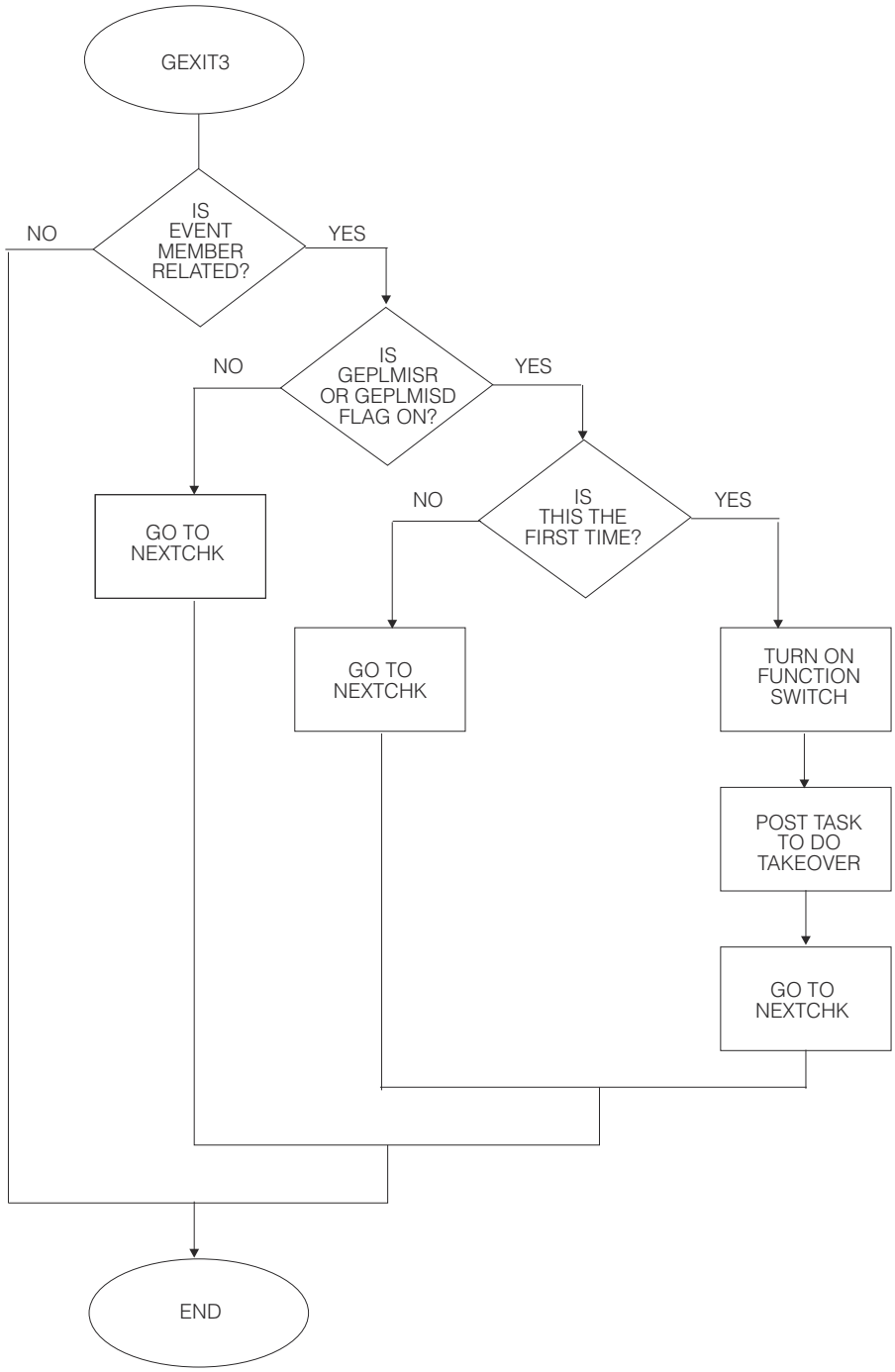


Figure 11. Summary of Group User Routine Logic (Part 1 of 3)

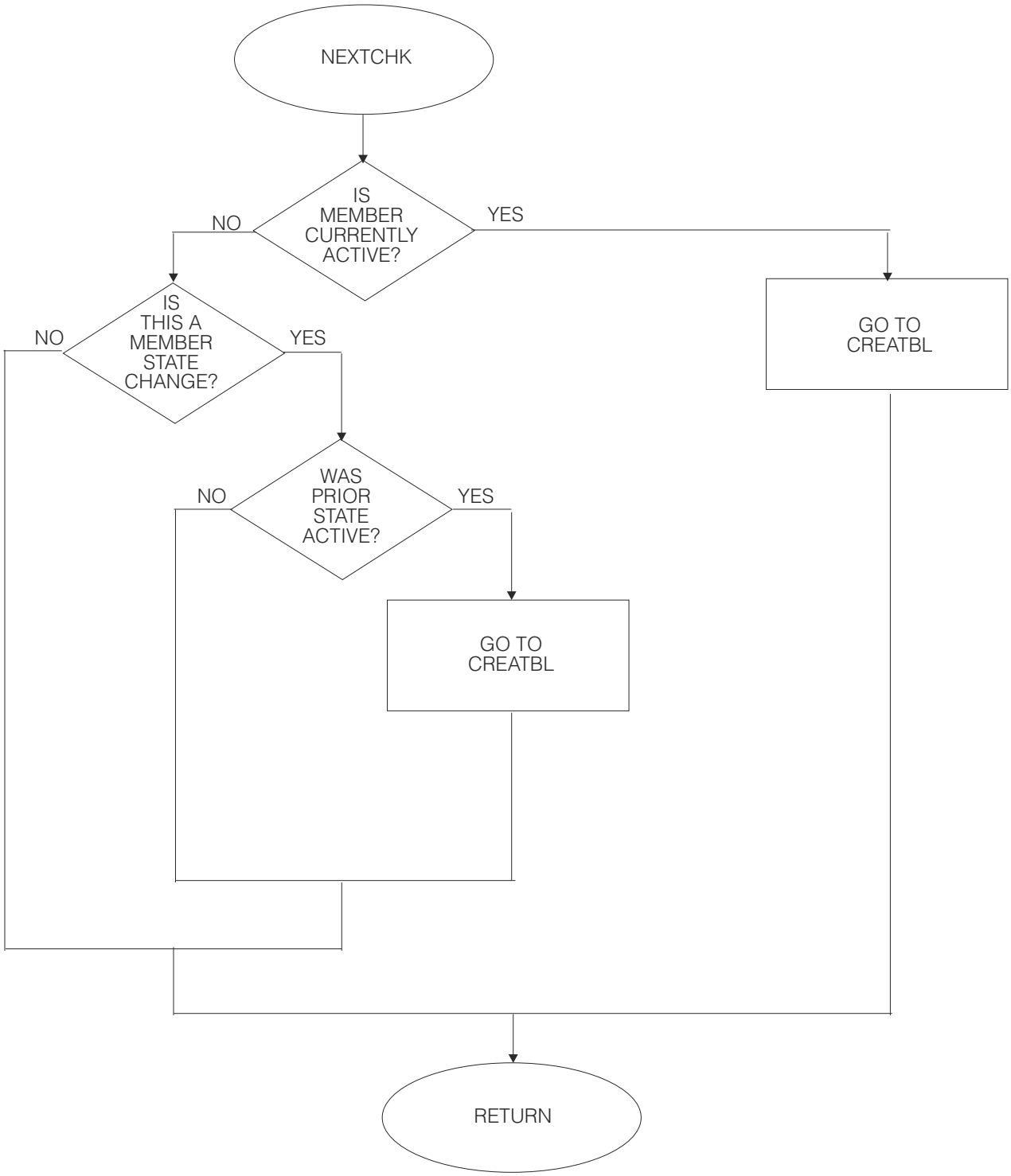


Figure 12. Summary of Group User Routine Logic (Part 2 of 3)

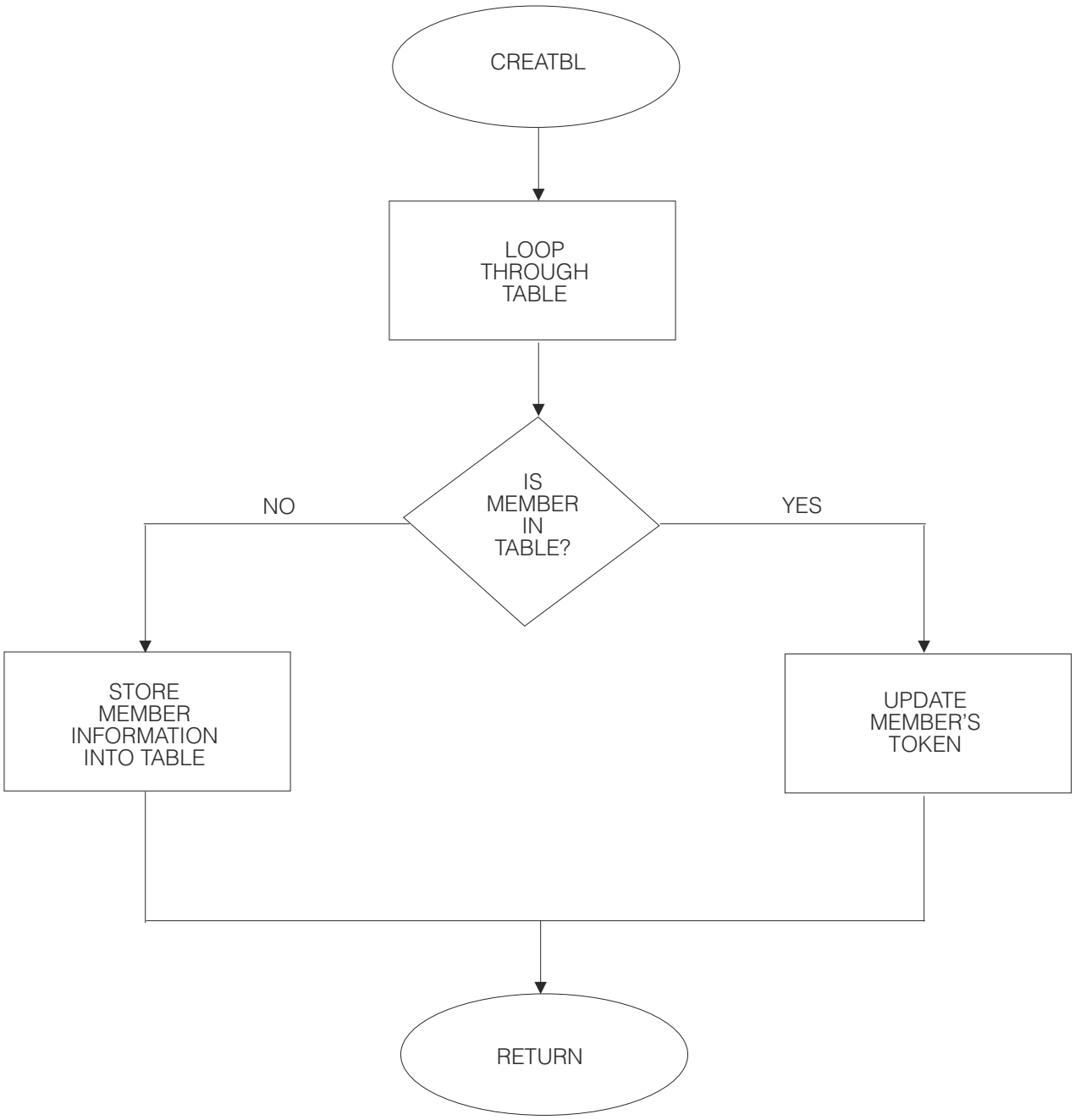


Figure 13. Summary of Group User Routine Logic (Part 3 of 3)

Here is an example of the group user routine code.

```
*****************************************************************
*                                                               *
*  GROUP USER ROUTINE                                           *
*                                                               *
*****************************************************************
GEXIT3   CSECT
GEXIT3   AMODE 31
GEXIT3   RMODE ANY
@MAINENT DS    0H
         USING *,R15
         B     @ENTRY
         DC    AL1(16)
         DC    C'GR 90010 GEXIT3'
         DROP  R15
*****************************************************************
*                                                               *
*  ENTRY LINKAGE                                                *
```

```
*                                                               *
*****************************************************************
@ENTRY   STM   R14,R12,12(R13)
         LR    R12,R15
@PSTART  EQU   GEXIT3
*
*  SET UP BASE REGISTER TO 12
*
         USING @PSTART,R12
*
*  SET UP DYNAMIC AREA
*
         SLR   R15,R15
         IC    R15,@SIZDATD
         SLR   R0,R0
         ICM   R0,7,@SIZDATD+1
         STORAGE OBTAIN,LENGTH=(0),SP=(15)
         LR    R10,R1
         USING @DATD,R10
         ST    R13,4(,R10)
         ST    R10,8(,R13)
         LM    R15,R1,16(R13)
         LR    R13,R10

*****************************************************************
*                                                               *
*  GROUP USER CODE                                              *
*                                                               *
*****************************************************************
*
*  GET ADDRESSABILITY TO THE PARAMETER LIST
*
         LR    R8,R1
         USING GEPL,R8
*
*  CHECK THE GEPLFLG2 FIELD FOR MEMBER-RELATED EVENT
*  (GEPLMEME BIT).  IF NOT MEMBER-RELATED, BRANCH BECAUSE
*  NO ACTION REQUIRED.
*
         TM    GEPLFLG2,X'80'
         BNO   @FINI
*
*  CHECK THE GEPLFLG2 FIELD FOR STATUS UPDATE MISSING
*  (GEPLMISR or GEPLMISD BITS).  IF NEITHER BIT IS ON,
*  BRANCH TO CHECK FOR TABLE UPDATES.
*
         TM    GEPLFLG2,X'30'
         BZ    @NEXTCHK
*
*  GET ADDRESSABILITY TO THE MEMBER DATA, WHICH CONTAINS
*  THE ADDRESS OF MDATASTR.  MDATASTR CONTAINS THE BYTE THAT
*  THE GROUP USER ROUTINE TURNS ON IF THIS IS THE FIRST TIME
*  CALLED FOR STATUS UPDATE MISSING.  IF THE BYTE IS ON ALREADY,
*  BRANCH TO CHECK FOR TABLE UPDATES.  IF THE BYTE IS NOT ON,
*  TURN IT ON.
*
         L     R5,GEPLMDAT
         USING MDATASTR,R5
         CLI   FUNCTON,X'00'
         BNE   @NEXTCHK
         MVC   FUNCTON(1),HEX11
*
*  GET THE ADDRESS OF THE TASK'S ECB AND POST THE TASK.
*  THE TASK WAS ATTACHED BY THE MAIN ROUTINE.  THE MAIN ROUTINE
*  PASSED THE ADDRESS OF THE MDATASTR IN THE PARAMETER LIST
*  ON THE ATTACH MACRO.  THE TASK'S JOB IS TO SWITCH THE
*  PRIMARY AND BACKUP MEMBERS.
*
         L     R7,POSTLNTH           MOVE LENGTH OF STATIC AREA TO R7
         BCTR  R7,0
         EX    R7,@SETPARM
         L     R9,TASKECB
         POST  (R9),LINKAGE=SYSTEM,MF=(E,POSTLSTD)
*
*  BRANCH HERE TO CHECK FOR TABLE UPDATES.  ONLY MEMBERS WHO
*  ARE CURRENTLY ACTIVE OR HAVE BEEN ACTIVE SHOULD BE IN
*  THE TABLE.  IF THE MEMBER'S CURRENT STATE IS ACTIVE,
*  BRANCH SO THE TABLE CAN BE UPDATED IF NECESSARY.
*
@NEXTCHK CLI   GEPLNEWS,GEACTIVE
         BE    @CREATBL
*
```

```
*  CHECK TO SEE IF THE MEMBER WAS PREVIOUSLY ACTIVE.  IF NOT
*  PREVIOUSLY ACTIVE, BRANCH TO THE END OF THE PROGRAM.
*
         CLI   GEPLOLDS,GEACTIVE
         BNE   @FINI
*
*  BRANCH HERE TO SEE IF A MEMBER IS CURRENTLY IN THE TABLE.
*  IF SO, UPDATE THE MEMBER'S TOKEN.  IF NOT, ADD THE MEMBER.
*
*  GET ADDRESSABILITY TO THE MEMBER DATA, WHICH CONTAINS
*  THE ADDRESS OF MDATASTR.  MDATASTR CONTAINS THE ADDRESS OF
*  THE TABLE.  LOOP THROUGH THE TABLE TO SEE IF THE MEMBER
*  IS THERE.  IF THE MEMBER IS THERE, BRANCH OUT OF THE
*  TABLE TO UPDATE THE MEMBER.  IF THE LOOP COMPLETES WITHOUT
*  FINDING THE MEMBER, BRANCH OUT OF THE LOOP TO ADD THE MEMBER.
*
@CREATBL L     R5,GEPLMDAT
@CHKTBL  L     R6,TBLADDR         START OF THE TABLE
         L     R7,NEXTITEM        NEXT AVAILABLE SLOT IN TABLE
@TBLLOOP CR    R6,R7              IS THIS THE END OF THE TABLE?
         BE    @STORTBL           YES
         USING ITEM,R6
         CLC   NAME(16),GEPLMNAM  MEMBER ALREADY IN THE TABLE?
         BE    @FOUND             IF YES, BRANCH
         A     R6,INCREM          MOVE TO NEXT MEMBER IN THE TABLE
         B     @TBLLOOP
*
*  BRANCH HERE WHEN THE MEMBER IS ALREADY IN THE TABLE
*
@FOUND   MVC   TOKEN(8),GEPLMTOK  UPDATE THE CURRENT TOKEN
         B     @FINI
*
*  BRANCH HERE WHEN THE MEMBER IS TO BE ADDED TO THE TABLE
*
@STORTBL L     R7,NEXTITEM        PLACE THE NEXT SLOT AVAILABLE IN R7
         USING ITEM,R7            USE MAPPING OF TABLE CONTENTS
         MVC   NAME(16),GEPLMNAM  STORE THE MEMBER NAME
         MVC   TOKEN(8),GEPLMTOK  STORE THE MEMBER TOKEN
         A     R7,INCREM          INCREMENT THE POINTER
         ST    R7,NEXTITEM        STORE THE POINTER
         DROP  R7
         B     @FINI
*
*  RELEASE DYNAMIC AREA
*
@FINI    LR    R1,R10
         L     R13,4(,R13)
         SLR   R15,R15
         IC    R15,@SIZDATD
         SLR   R0,R0
         ICM   R0,7,@SIZDATD+1
         STORAGE RELEASE,LENGTH=(0),ADDR=(1),SP=(15)

*****************************************************************
*                                                               *
*  EXIT LINKAGE                                                 *
*                                                               *
*****************************************************************
         LM    R14,R12,12(R13)
         BR    R14
@SETPARM MVC   POSTLSTD(0),POSTLST1  SET UP DYNAMIC AREA FOR POST
GEXIT3   CSECT ,
         LTORG
         DS    0D
INCREM   DC    F'24'
HEX11    DC    X'11'
POSTLST1 POST  MF=L
POSTLNTH DC    A(*-POSTLST1)
@SIZDATD DS    0A
         DC    AL1(0)
         DC    AL3(@DYNSIZE)
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
```

```
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
@ENDDATA EQU   *
@DATA    DS    0H
@DATD    DSECT
         DS    0F
SAVEAREA DS    18F
POSTLSTD POST  MF=L
@ENDDATD DS    0X
@DYNSIZE EQU   ((@ENDDATD-@DATD+7)/8)*8

*************************************************************
*                                                           *
*  MAPPING OF THE DATA STRUCTURE (MDATASTR) POINTED TO BY   *
*  MEMDATA (GEPLMDAT FIELD IN PARAMETER LIST)               *
*                                                           *
*  THIS SAME DATA STRUCTURE IS USED BY THE STATUS USER      *
*  ROUTINE.  SOME FIELDS ARE NOT USED BY THE GROUP USER     *
*  ROUTINE, BUT ARE USED ONLY BY THE STATUS USER ROUTINE.   *
*                                                           *
*  TBLADDR         ADDRESS OF TABLE MAINTAINED BY           *
*                    GROUP USER ROUTINE                     *
*  NEXTITEM        ADDRESS OF NEXT AVAILABLE SLOT IN        *
*                    TABLE                                  *
*  WRKQADDR        ADDRESS OF MEMBER'S WORK QUEUE           *
*                    (USED BY STATUS USER ROUTINE)          *
*  NXTWRKAD        ADDRESS OF NEXT AVAILABLE SLOT IN        *
*                    MEMBER'S WORK QUEUE                    *
*                    (USED BY STATUS USER ROUTINE)          *
*  TASKECB         ADDRESS OF THE ATTACHED TASK'S ECB       *
*                    (THIS TASK IS ATTACHED BY THE MAIN     *
*                     ROUTINE.)                             *
*  MAINECB         ADDRESS OF ECB USED FOR SYNCHRONIZING    *
*                    (THE MAIN ROUTINE WAITS ON THIS ECB,   *
*                     WHICH THE ATTACHED TASK POSTS WHEN    *
*                     IT COMPLETES ITS WORK.)               *
*  FUNCTON         GROUP USER ROUTINE TURNS THIS SWITCH     *
*                      ON WHEN CALLED FOR THE FIRST TIME    *
*                      FOR A STATUS UPDATE MISSING.         *
*  RESUMEB         MAIN ROUTINE TURNS THIS SWITCH ON        *
*                      WHEN IT RESUMES UPDATING ITS STATUS  *
*                      FIELD.                               *
*                                                           *
*************************************************************
MDATASTR DSECT
TBLADDR  DS    1F
NEXTITEM DS    1F
WRKQADDR DS    1F
NXTWRKAD DS    1F
TASKECB  DS    1F
MAINECB  DS    1F
FUNCTON  DS    X
RESUMEB  DS    X
ITEM     DSECT             MAPPING OF ELEMENT IN THE TABLE
NAME     DS    CL16           MEMBER NAME
TOKEN    DS    XL8            CURRENT TOKEN OF MEMBER
*************************************************************
*                                                           *
*  MAPPING MACROS                                           *
*                                                           *
*************************************************************
         IXCYGEPL
         END   GEXIT3
```

Obtaining XCF Information

You can obtain information related to XCF from any authorized routine. The routine can be, but does not have to be, a member of an XCF group. This section tells you why you might need, and how to obtain, the following types of information:

- Sysplex, group, and member information (available through the IXCQUERY macro)
- Tuning and capacity planning information (available through the IXCMG macro, and intended for system programmers).

Obtaining Sysplex, Group, and Member Information

A member of an XCF group, or any authorized routine, might need information about members, groups, and systems in an XCF sysplex under these circumstances:

- A member of an XCF group might need to know which other members of the group are currently defined to XCF and what their member states are.
- A member of an XCF group might need to request services (such as invoking IXCTERM or IXCSETUS) on behalf of another member of the group. The member must provide the target member's token to invoke these services.
- A member of an XCF group might need to send a message to another member in the group. The sender must provide the target member's token to issue the IXCMSGOX macro.
- A member of an XCF group did not identify a group user routine to be notified of changes to other members in the group. Occasionally, the member might need to know the status of the other members in the group.
- An authorized routine that maintains or displays XCF data might need information about the systems, groups, and members in the sysplex.
- An authorized routine might need to delete a member of an XCF group (place the member in a not-defined state). The routine must supply the target member's token to issue the IXCDELET macro.
- At initialization, a multisystem application might need to know if the system it is started on is part of a multisystem sysplex, and the maximum number of systems supported in that sysplex.

The above are just a few examples of the many reasons why XCF group members and other authorized routines might request information about the systems, groups, and members in the XCF sysplex by invoking the IXCQUERY macro. The information provided by the IXCQUERY macro is mapped by the IXCYQUAA mapping macro.

Using the IXCQUERY Macro

When you code the IXCQUERY macro, you specify what type of information you want (REQINFO parameter) and where you want the information placed (the output area identified on the ANSAREA parameter). When you code the ANSAREA parameter, you must also code the ANSLEN parameter to tell XCF the size of the output area. You can determine the size by consulting the data structures mapped by IXCYQUAA. If you do not allow enough space, XCF:

- Fills up the space you provided
- Lets you know how many records could not be included
- Lets you know how much space you should have provided
- Sets the QUAARSNRECORDSREMAIN reason code.
- **Handling the QUAARSNRECORDSREMAIN Reason Code**

  The QUAARSNRECORDSREMAIN reason code indicates that the ANSAREA you provided is too small to contain all the requested data. You can reissue the IXCQUERY macro using the value returned in QUAHTLEN (total length of answer area needed to contain all the requested information) as the length of your answer area. However, be aware that the IXCQUERY information returned is a snapshot of the current environment — which might change between one invocation of IXCQUERY and the next. (For example, additional systems might have joined or left the sysplex, thus changing the number of system records in the answer area.)

  You must provide code to handle the QUAARSNRECORDSREMAIN reason code in case the length of the record(s) you are requesting ever changes.
- **Retrieving Information from the Answer Area**

  The answer area mapped by IXCYQUAA can contain one or more instances of many different types of records depending on your IXCQUERY request. To help you reference each of the record types, the answer area contains fields indicating the length of each record type. You must use these length fields to index through the answer area in case the length of the record(s) you are requesting ever changes.

Using the DSECT length of a particular record type is not recommended because the length might have been changed since your program was assembled.

Specifying the Information Level

The Query Answer Area supports several levels of information that IXCQUERY returns. Certain coupling facility and structure requests might provide data that was not returned when the IXCQUERY service was first made available. For these request types, you can specify the level of information you want with the QUAALEVEL parameter on IXCQUERY. The QUAALEVEL parameter is available with version 2 of the IXCQUERY macro. The system returns base QUAA information when you specify QUAALEVEL=0 on your request; the system returns level-1 QUAA information when you specify QUAALEVEL=1 on your request and level-2 QUAA information when you specify QUAALEVEL=2 on your request. You should be aware of the type of output that you are requesting and be able to process it correctly. IBM recommends that you use the level-1 level of IXCYQUAA in case additional new data is returned by the IXCQUERY service. Note that the level-1 IXCYQUAA records are larger than the level-0 IXCYQUAA records.

Table 7 on page 100 lists the IXCYQUAA structures that support the level-1 level of IXCYQUAA information. See the IXCYQUAA macro in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for a description of the information returned.

Specifying the Type of Information

Depending on how you code the IXCQUERY macro, you can obtain information about:

- The name of the sysplex
- Every system in the sysplex
- Every group in the sysplex
- Every member in a specific group
- A specific member in a specific group
- Every application in the sysplex that is using automatic restart management
- Every application on a certain system that is using automatic restart management
- Every application in a certain restart group (a group of applications that the automatic restart manager is to restart together on a certain system)
- A particular application using automatic restart management
- Whether the sysplex is in XCF-local mode
- Whether the sysplex is in monoplex mode
- The maximum number of systems allowed in the sysplex for this XCF release level
- The current maximum number of systems allowed in the sysplex as defined by the installation in the couple data sets
- Software features available on a system in the sysplex
- Every coupling facility in the sysplex
- A specific coupling facility
- Every coupling facility structure in the sysplex
- Every coupling facility structure in a specific coupling facility
- A specific coupling facility structure in a specific coupling facility.
- Pending policy information for a specific coupling facility structure (available with QUAALEVEL=2).
- General couple data set information on all functions.
- Detailed couple data set information for a specific function.
- Site affiliation of the coupling facility, if applicable.

Table 7 on page 100 summarizes the parameters you code on the IXCQUERY macro to obtain the required information.

Table 7. IXCQUERY Macro Parameters

Parameter on IXCQUERY	Information Returned	Structure in IXCYQUAA
REQINFO=SYSPLEX	Header record	QUAHDR
	One record for each system in the sysplex	• QUASYS • QUASYS1 (when QUAALEVEL=1) • QUASYS2 (when QUAALEVEL=2)
REQINFO=GROUP (the default)	Header record	QUAHDR
	One record for each group in the sysplex	QUAGRP
REQINFO=GROUP, GRPNAME=*grpname*	Header record	QUAHDR
	One record for each member of the specified group	QUAMEM, QUAMEM1, and QUAMEM2
REQINFO=GROUP, GRPNAME=*grpname*, MEMNAME=*memname*	Header record	QUAHDR
	One record for the specified member	QUAMEM, QUAMEM1, and QUAMEM2
REQINFO=COUPLE, LOCAL=*local*	Whether the sysplex is in XCF-local mode	N/A
REQINFO=COUPLE, MONOPLEX=*monoplex*	Whether the sysplex is in monoplex mode	
REQINFO=COUPLE, MAXSYS=*maxsys*	The maximum number of systems allowed in the sysplex as specified by the XCF release level	
REQINFO=COUPLE, CURRMAXSYS=*currmaxsys*	The current maximum number of systems allowed in the sysplex as specified in the sysplex couple data set	
REQINFO=COUPLE, SYSPLEXID=*sysplexid*	The unique sysplex identifier for the sysplex	
REQINFO=COUPLE, SYSTEMID=*systemid*	The unique system identifier for the system on which the IXCQUERY was invoked.	
REQINFO=COUPLE, PLEXNAME=*plexname*	The name of the sysplex	
REQINFO=COUPLE, CFLEVEL=*cflevel*	The maximum CFLEVEL supported by the system	
REQINFO=COUPLE, ND=*nd*	The node descriptor of the system	
REQINFO=FEATURES	XCF and XES software features available on this system.	QUREQFEATURES
REQINFO=CF	Header record	QUAHDR
	One record for each coupling facility in the sysplex.	• QUACF • QUACF1

Table 7. IXCQUERY Macro Parameters (continued)		
Parameter on IXCQUERY	**Information Returned**	**Structure in IXCYQUAA**
REQINFO=CF, CFNAME=*cfname*	Header record	QUAHDR
	One record for the specified coupling facility	• QUACF • QUACF1
	System connectivity to the coupling facility	• QUACFSC • QUACFSC1
	Coupling facility structures assigned resources in the coupling facility	• QUACFSTR • QUACFSTR1
REQINFO=CF_ALLDATA	Header record	QUAHDR
	One record for each coupling facility in the sysplex	• QUACF • QUACF1
	One record for each coupling facility in the sysplex for system connectivity information	• QUACFSC • QUACFSC1
	One record for each coupling facility in the sysplex for information about structures assigned resources in the coupling facility	• QUACFSTR • QUACFSTR1
REQINFO=STR	Header record	QUAHDR
	One record for each coupling facility structure in the sysplex	• QUASTR • QUASTR1 • QUASTRPPINFO (when QUAALEVEL=2)

Table 7. IXCQUERY Macro Parameters (continued)		
Parameter on IXCQUERY	**Information Returned**	**Structure in IXCYQUAA**
REQINFO=STR, STRNAME=*strname*	Header record	QUAHDR
	One record for the specified coupling facility structure	• QUASTR • QUASTR1 • QUASTRPPINFO (when QUAALEVEL=2)
	Names of coupling facilities in the structure's preference list	• QUASTRPL • QUASTRPL1 When QUAALEVEL=2, the system returns preference list information for the pending policy also, if applicable.
	Names of structures in the structure's exclusion list	• QUASTRXL • QUASTRXL1 When QUAALEVEL=2, the system returns exclusion list information for the pending policy also, if applicable.
	Names of the coupling facilities where the structure is allocated	• QUASTRCF • QUASTRCF1
	Connectors to the coupling facility structure	• QUASTRUSER • QUASTRUSER1
REQINFO=STR_ALLDATA	Header record	QUAHDR
	One record for each coupling facility structure in the sysplex	• QUASTR • QUASTR1 • QUASTRPPINFO (When QUAALEVEL=2)
	Names of coupling facilities in each structure's preference list	• QUASTRPL • QUASTRPL1 When QUAALEVEL=2, the system returns preference list information for the pending policy also, if applicable.
	Names of structures in each structure's exclusion list	• QUASTRXL • QUASTRXL1 When QUAALEVEL=2, the system returns exclusion list information for the pending policy also, if applicable.
	Names of the coupling facilities where each structure is allocated	• QUASTRCF • QUASTRCF1
	Connectors to each coupling facility structure	• QUASTRUSER • QUASTRUSER1

Table 7. IXCQUERY Macro Parameters (continued)		
Parameter on IXCQUERY	**Information Returned**	**Structure in IXCYQUAA**
REQINFO=ARMSTATUS	Header record	QUAHDR
	One record for each application specified that is using automatic restart management	QUAARMS
REQINFO=ARMS_ALLDATA	Header record	QUAHDR
	One record for each application in the sysplex that is using automatic restart management	QUAARMS
REQINFO=CDS	Header record	QUAHDR
	One record for each couple data set function defined to the system	QUACDSFUN
	One record for each couple data set defined to the system	QUACDS
REQINFO=CDS CDSTYPE=cdstype	Header record	QUAHDR
	One record for the specified couple data set function	QUACDSFUN
	One record for each (primary/alternate) couple data set defined for the couple data set function	QUACDS
	One record for each system using the specified couple data set function	QUACDSSU
	Narrative records describing narrative data specified by the owner of the couple data set function	QUACDSNAR
REQINFO=CDS_ALLDATA	Header record	QUAHDR
	One record for each couple data set function defined to the system	QUACDSFUN
	One record for each couple data set defined to the system	QUACDS
	Records for each system using each couple data set function	QUACDSSU
	Narrative records describing narrative data specified by the owner of each couple data set function defined to the system	QUACDSNAR

You can also specify, on IXCQUERY REQINFO=GROUP, whether you want the status that XCF has readily available (REQTYPE=IMMEDIATE parameter), or whether you want XCF to suspend your work unit while it obtains the most current data available (the default, REQTYPE=DEFER parameter). If you specify REQTYPE=DEFER, XCF serializes updates to the requested group data and retrieves the most current data. However, XCF cannot guarantee that updates will not occur before the requestor uses the data. For example, when the IXCQUERY service returns to the caller, the caller could then be swapped out. By the time the caller is swapped back in, updates could have been made, and the data that was returned by IXCQUERY is no longer the latest data.

Programming Considerations

Depending on the type of information requested, IXCQUERY might reference the CFRM active policy. Multiple IXCQUERY requests could result in a large amount of I/O to the CFRM couple data set, which in turn, could generate a noticeable loss of system performance. When designing an application such as a sysplex monitoring tool that uses the IXCQUERY macro, be aware of the performance effect of multiple macro invocations.

Information mapped by the IXCYQUAA mapping macro

Most of the information that IXCQUERY provides is mapped by the IXCYQUAA macro. IXCYQUAA provides information that is related to:

- Header record that describes the data records returned (QUAHDR).
- Sysplex data (QUASYS, QUASYS1 and QUASYS2)
- Group data (QUAGRP)
- Member data (QUAMEM, QUAMEM1, and QUAMEM2)
- Coupling facility data (QUACF, QUACF1, QUACFSC, QUACFSC1, QUACFSTR, and QUACFSTR1)
- Coupling facility structure data (QUASTR, QUASTR1, QUASTRPL, QUASTRPL1, QUASTRXL, QUASTRXL1, QUASTRCF, QUASTRCF1, QUASTRUSER, and QUASTRUSER1)
- Couple data set data (CDS data)
- Automatic restart management data (QUAARMS topic)
- Software features installed on a system (QUREQREATURES topic)

The information about record contents provided here is only partial. See IXCYQUAA in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for complete information on field names and lengths, offsets and descriptions of the fields mapped by the IXCYQUAA macro.

Header Record

When you request the information mapped by IXCYQUAA, the IXCQUERY macro also provides a header record. This record (QUAHDR section) includes the following information:

- Number of system, group, member, coupling facility, or structure records that will follow
- Number of system, group, member, coupling facility, or structure records not returned because of insufficient space
- Total length of the answer area needed to contain all the requested information (including the area for the records that were successfully returned on this call).

Sysplex Data

When you request information about the systems in the sysplex, the information returned includes the following for each system:

- System name
- Failure detection interval and operator notification interval that is specified at IPL time
- System status
- System token.

When you request a level-1 IXCYQUAA record mapping about a system in a sysplex, the following information is returned in addition to the ones that were previously listed:

- Whether the system is in LPAR mode, and when in LPAR mode, the LPAR number within the CEC in which the system is running.
- Serial number and model number of the CEC on which the system is running.

Group Data

When you request information about the groups in the sysplex, the information that is returned includes the following for each group:

- Group name
- Number of members in the group
- Whether the group has any stalled members

Member Data

When you request information about all members in a specific group, or a specific member in a specific group, the information that is returned includes the following for each member:

- Member name
- Member token
- Member state
- Additional status information (for example, system status update missing, member status update missing, etc.)
- System name for the system on which the member is active or was last active
- System token for the system on which the member is active or was last active
- JOB, STC, MOUNT, or LOGON name from the primary ASID current when IXCJOIN was issued
- Timestamp (in Greenwich mean time) of the last event that affected the member
- Length of the member's user state field (specified on IXCCREAT or IXCJOIN)
- The member's status-checking interval (specified on IXCJOIN or changed on IXCMOD)
- User data that is returned by the member's status user routine in GPR 0
- The member's space token (STOKEN) for the primary address space at the time IXCJOIN was issued.
- Protocols that are supported for the member. Protocols include whether the member can receive message, participate in XCF-managed response collection, and support large message (up to 128M bytes) delivery.
- Whether the member is stalled, causing signaling sympathy sickness, being deactivated, or being terminated by SFM.

Coupling Facility Data

When you request information about all coupling facilities in a sysplex, the information that is returned includes the following for each coupling facility:

- Coupling facility name
- Node descriptor (mapped by the IXLYNDE macro)
- Size of the dump space (in multiples of 4K)
- Status indicators
- Number of coupling facility structures in this facility that cannot be added to the policy
- Number of systems connected to the coupling facility
- Number of coupling facility structures in the coupling facility
- Information about the active policy in effect for this coupling facility, including policy name, the times the policy was last updated and activated, and storage limitations.
- Information about the site affiliation of the coupling facility, if applicable, including the name of the SITE specified in the CFRM policy and the state of the Recovery Manager.

When you request information about a specific coupling facility in a sysplex, the following information is returned in addition to ones that were previously listed:

- Names of the systems that are connected to the specified coupling facility
- Names of the structures in the specified coupling facility
- Allocation status of each structure.

Coupling Facility Structure Data

When you request information about structures in a sysplex, the information that is returned includes the following for each structure:

- Name of the structure
- Size of the structure (as specified in the CFRM active policy)
- Pending size of the structure (if specified in a pending CFRM policy)
- Status indicators
- Number of associated preference list records
- Number of associated coupling facility exclusion list records
- Number of coupling facilities containing the structure
- Number of connectors to the structure
- Active policy data, including the policy name
- Information on structure rebuild
  - Type of processing (rebuild or duplexing rebuild)
  - Method of processing (user-managed or system-managed)
  - Duplexing mode (synchronous or asynchronous)
  - Phase
- Information on structure alter
- User-defined event information.
- Whether message-based processing is being used to coordinate event management for this structure. If so, managing system information, such as the system name and system ID, is also provided.

When you request information about a specific coupling facility structure in a sysplex, the following information is returned in addition to those that were previously listed:

- Name of coupling facilities in the preference list for the structure
- Name of coupling facility structures in the exclusion list for the structure
- Name and node descriptor of the coupling facility in which the structure is allocated
- Structure version numbers (physical and logical)
- Structure data format (encrypted or non-encrypted)
- Allocation status of the structure
- Connection data about each connector to the structure:
  - Connection version
  - Connect data
  - Connect name
  - Connect token
  - System name
  - System token
  - Job name or started task name
  - State of the connection (for example, active, failed persistent, terminating, or keep disposition)
  - Connection identifier
  - Level of information that is returned for the connection
  - Failure/isolation information.
  - Structure rebuild information, including whether ALLOWAUTO has been specified and the SUSPEND= and ASYNCDUPLEX= options, if applicable.

- Structure alter information, including whether ALLOWALTER has been specified and appropriate IXLCONN values.

When you request a level-1 IXCYQUAA record mapping about a structure in a sysplex, the following information is returned in addition to those that were previously listed:

- Percent loss of connectivity when structure is undergoing an MVS-initiated rebuild based on the value of REBUILDPERCENT
- Additional USYNC-related completion code information
- Group name that is associated with the structure, if the structure is being used as a serialized structure
- Name of coupling facility for which the structure is a populate candidate
- Auto version (applicable only to system-managed processes)
- The structure user's current disconnect/failed confirm string (for unserialized stuctures only)
- System-specific information when process is system-managed, including system identification and phase of system-managed process.

Couple data set data

When you request information about all couple data sets in a sysplex, the information returned includes the following for each couple data set:

- Function name
- Policy data
- Policy name
- Policy start time
- Policy update time
- Data set information
  - Primary or Alternate indication
  - Status indicators
  - Couple data set name
  - Couple data set VOLSER
  - Couple data set device address
  - Format Time
  - Maximum number of systems that are supported by this couple data set.
- When you request detailed information about one or more couple data set functions, the following is returned in addition to that were previously listed:
  - Systems using the couple data set function
    - System name
    - System token that identifies the system using the couple data set
  - Narrative data.

Automatic Restart Management Data

When you request information about automatic restart manager elements and restart groups, the information returned includes the following for each element:

- Element name
- Name of the system where the element originally registered
- Name of the system where the element is running (or last ran if the element has failed and has not yet been restarted)
- Replication ID of the system where the element originally registered
- Job or started task name

- STOKEN for the element's address space
- ASID of the element's address space
- Level number associated with this element
- Name of the JES XCF group in which this element must run
- Date and time of initial registration
- Date and time of the first automatic restart manager restart
- Date and time of the most recent automatic restart manager restart
- Element type (or blank)
- Flags indicating:
  - Whether restarts by the automatic restart manager are enabled in the sysplex
  - Whether all systems capable of automatic restarts have connectivity to the ARM couple data set
  - Whether the current ARM policy prohibits a restart by the automatic restart manager for this element
  - The state of the element (for example: available, restarting, and so on)
  - Whether the element is a batch job, started task, or abstract resource
  - Whether the element has override JCL or command text
  - Whether the element is associated with another element
  - Whether the element has a minimum bind to the system on which it registered.
- Name of the element's event exit routine
- The number of restarts by automatic restart manager that have occurred since the element initially registered
- The number of restarts by the automatic restart manager that have occurred in the most recent restart interval for this element
- The number of restart attempts allowed for this element and the interval
- Name of the element to which this element is associated (or blank if no associations)
- An indication of whether the associated element is the element being backed up or is the back up for this element
- The total number of elements that are currently registered with the automatic restart manager
- The maximum number of elements that are able to be registered with the automatic restart manager.

Information returned inline to IXCQUERY

IXCQUERY returns inline information for both REQINFO=COUPLE and REQINFO=FEATURES. The information that IXCQUERY returns when you specify REQINFO=COUPLE is placed in a storage area that you provide.

LOCAL
Whether the sysplex is in XCF-local mode. In XCF-local mode, XCF does not provide signaling services between MVS systems in a multisystem environment. (Members residing on a system in XCF-local mode can exchange messages with one another, but cannot exchange messages with members on other systems.) XCF does not support permanent status recording for a system in XCF-local mode. See *z/OS MVS Setting Up a Sysplex* for further information.

MONOPLEX
Whether the sysplex is in monoplex mode. In monoplex mode, the sysplex is made up of a single system that requires the use of a sysplex couple data set and can use other couple data sets. XCF signaling services can be used between members on this system.

MAXSYS
The maximum number of systems allowed in the sysplex based on the MVS release level and the sysplex configuration. If the sysplex is in XCF-local mode or monoplex mode, then the value of MAXSYS is 1.

The MAXSYS value remains constant throughout the life of the IPL. Use this information to allow your program to be independent of future MVS releases that might increase the maximum number of systems allowed in the sysplex. For example, your program might need to obtain enough storage for a table with one entry per system in the sysplex.

CURRMAXSYS
The current maximum number of systems allowed in the sysplex (specified when the sysplex couple data set was formatted). The CURRMAXSYS value might change during the life of an IPL if you switch to a new sysplex couple data set formatted for a different number of systems. Use this information to minimize storage when your program needs to maintain information about the systems in a sysplex. For example, your program might need to obtain enough storage for a table with one entry per system in the sysplex. If the maximum allowable number of systems in the sysplex is 32, but you have chosen to maintain a sysplex of only 10 systems, the amount of storage that you need to allocate for your table is significantly less if you use that required for only 10 systems.

SYSPLEXID
The sysplex identifier token for the sysplex. XCF creates the token when the first system in the sysplex IPLs. The token remains in existence for the life of the sysplex.

SYSTEMID
The identifier of the system on which the IXCQUERY was invoked. The high order byte contains the XCF slot number. The low order three bytes contain the XCF system sequence number used to identify a unique instantiation of the system in the sysplex.

PLEXNAME
The name of the sysplex in which this system is participating. The sysplex name is specified in the COUPLExx parmlib member and in the couple data sets that support the sysplex.

CFLEVEL
The maximum coupling facility operational level that is supported by the operating system where the IXCQUERY was issued.

ND
The node descriptor of the system where the IXCQUERY macro was issued.

The information that IXCQUERY returns when you specify REQINFO=FEATURES is placed in a storage area that you specify with the FEATAREA parameter. The information is mapped by QUREQFEATURES in IXCYQUAA and includes the following:

QUREQRFPROXYRESPONSE
The ProxyResponse feature is available for:

- IXLUSYNC
- IXLEERSP EVENT=REBLDSTOP
- IXLEERSP EVENT=REBLDCLEANUP

QUREQRFUSYNCCOMPCODE
The IXLUSYNC COMPCODE function is available on this system.

QUREQRFREBUILDPCTLOSSCONN
Percent lossconn is available for rebuild events on this system.

QUREQRFREBUILDDUPLEX
Support for user-managed duplexing is available on this system.

QUREQRFIXLMGHWSTATCF
HWSTATISTICS=CF for IXLMG is supported on this system.

QUREQRFIXLRTRDATATYPE
IXLRT RDATATYPE function is available on this system.

QUREQRFIXLCONNSUSPENDFAIL
IXLCONN SUSPEND=FAIL is supported on this system.

QUREQRFIXLMGSID
IXLMG SID is supported on this system.

QUREQRFRETURNRDATATYPE
IXLRT support to return the RDATATYPE for record data entries that are read is available on this system.

QUREQRFDEMEBUFFERSIZE
Support for the relaxation of buffer size requirements for IXLLSTM REQUEST=DELETE_ENTRYLIST and REQUEST=MOVE_ENTRYLIST is available on this system.

QUREQRFDETAILEDXCFSTATUS
Support for IXCMG TYPE=MEMBER or AMDALEVEL=1 is available on this system.

QUREQRFDISALLOWFORCEFPCONN
IXLFORCE support for new return/reason code is available on this system. The new return/reason code is: RC=04 RSN=xxxx041B — OK to force a structure with only failed-persistent connections. SETXCF FORCE support is available on this system. A SETXCF FORCE,STRUCTURE command will force a structure with only failed-persistent connections. A SETXCF FORCE,CONNECTION command will fail to force failed-persistent connections to a persistent serialized list or lock structure.

QUREQRFDISPLAYSTRTYPE
D XCF,STR,STRNAME=*strname* provides the structure type if set in the CFRM active policy when the allocated structure is ACTIVE, REBUILD OLD/NEW, or DUPLEXING REBUILD OLD/NEW.

QUREQRFQUAALEVEL2
Support for QUAALEVEL 2 and related enhancements is available on this system.

QUREQRFIXCM2DEL
Support for the IXCM2DEL XCF deletion utility is available on this system.

QUREQRFALLSHAREDCPS
Support for IXLMG to return information about CFs shared/dedicated CP status is available on this system.

QUREQRFIXLCONNMONITORSTORAGE
Support for IXLCONN MONITORSTORAGE is available on this system.

QUREQRFIXCCFCM
Support for the IXCCFCM programming interface is installed on this system.

QUREQRFIXCMGGATHERFROM
Support for IXCMG GATHERFROM= is available on the system.

QUREQRFIXCCFCM
Support for the IXCCFCM programming interface is installed on this system.

QUREQRFALLOWREALLOCATE
Support for the ALLOWREALLOCATE CFRM administrative policy option is available on this system.

QUREQRFIXLCMPLLOCKFLAGS
Support for the locking completion exit to receive miscellaneous flags (including real and false contention indications) is available on this system.

QUREQRFALLOCNOTPERMITTED
Coupling facility `ALLOCATION IS NOT PERMITTED` indicator is available on this system.

QUREQRFMAINTENANCEMODE
Coupling facility maintenance mode is supported on this system.

QUREQRFIXLCACHEWRITESUPPRESS
Support for the LOCALREGCNTL keyword on IXLCACHE for write suppression that is based on local cache registration is available on this system.

QUREQRFIXCNOTESERVICEAVAIL
XCF Note Pad Services are available on this system.

QUREQRFIXLCACHEHALTCHGSUPPXI
Support for IXLCACHE HALTONCHANGED and SUPPCROSSINVAL keywords is available on this system.

QUREQRFASYNCDUPLEX
Support for system-managed asynchronous duplexing. Includes IXLADUPX, IXLLOCK ADUPREQSEQNUM, IXLLOCK REQVERSION, and IXLMG AMDALEVEL=2.

QuReqRfListMonitorOptions
List full/not-full monitoring, aggressive list notification and setting list monitoring notification delays supported on this system.

QuReqRfNotFullMonitoring
IXLLSTC full/not-full monitoring option for coupling facility list structure lists supported on this system.

QuReqRfAggressiveNotify
IXLLSTC aggressive list monitoring and notification option for list and key ranges supported on this system

QuReqRfWriteReadMetrics
List and cache structure write/read measurement metrics and IXLMG AMDALEVEL=3 supported on this system.

QuReqRfAMDALevel4
IXLMG AMDALEVEL=4 supported on this system.

QuReqRfAsyncXI
Support for asynchronous cross-invalidation is available on the system.

QuReqRfPctEntryRsv
IXLCONN PCTENTRYRSV keyword supported on this system.

QuReqRfCacheResTime
IXLMG cache residency time metrics supported on this system.

QuReqRfIxcNoteResiliency
IXCNOTE service support for REQUEST=NOTEPAD REQTYPE=MODIFY and the LOSSCONNDELETE keyword is available on this system.

Obtaining Tuning and Capacity Planning Information

Note: The information in this section is intended for use by system programmers in tuning and planning a sysplex. The programmer designing a multisystem application does not need this information.

Installations running multisystem applications that use the XCF signaling services need data for tuning the sysplex, and data for planning the capacity of the sysplex. XCF accumulates information during sysplex processing, and maintains that information. The Resource Measurement Facility (RMF) collects, and produces reports based on, this data. For more information on the reports that RMF produces for tuning a sysplex, see *z/OS MVS Setting Up a Sysplex* and *z/OS Resource Measurement Facility User's Guide*.

Authorized routines can also obtain tuning and capacity planning information by issuing the IXCMG macro. The information provided by the IXCMG macro is mapped by the IXCYAMDA mapping macro.

When you code the IXCMG macro, you specify what type of information you want (TYPE parameter), where you want the information placed (DATAAREA parameter), and which system in the sysplex you want to gather the information from (GATHERFROM parameter). When you code the DATAAREA parameter, you must also code the DATALEN parameter to tell XCF the size of the area that you provided. If you do not allow enough space, XCF does the following:

- Fills up the space you did provide
- Lets you know how much space you should have provided
- Sets a reason code of X'4'.

Handling the X‘4’ Reason Code

The X'4' reason code indicates that the ANSAREA you provided is too small to contain all the requested data. You can reissue the IXCMG macro using the value returned in AMDATLEN (total length of answer area needed to contain all the requested information) as the length of your answer area. However, be aware that the IXCMG information returned is a snapshot of the current environment — which might change between one invocation of IXCMG and the next. (For example, additional systems might have joined or left the sysplex, thus changing the number of system records in the answer area.) You must provide code to handle the X'4' reason code in case the length of the record(s) you are requesting ever changes.

Retrieving Information from the Answer Area

The answer area mapped by IXCYAMDA can contain one or more instances of many different types of records depending on your IXCMG request. To help you reference each of the record types, the answer area contains fields indicating the length of each record type. For each record type, AMDAREA contains the number of entries in IXCYAMDA, the length of each entry, and the offset to the first entry of the record type. You must use these length fields to index through the answer area in case the length of the record(s) you are requesting ever changes. Using the DSECT length of a particular record type is not recommended because the length might have been changed since your program was assembled.

Specifying the Information Level

The XCF Accounting and Measurement Data Area (IXCYAMDA) supports several levels of information that IXCMG returns. Certain XCF requests may provide data that was not returned when the IXCMG service was first made available. For these request types, you can specify the level of information you want with the AMDALEVEL parameter on IXCMG. The AMDALEVEL parameter is available with version 1 of the IXCMG macro. The system returns base AMDA information when you specify AMDALEVEL=0 on your request; the system returns level-1 AMDA inforamtion when you specify AMDALEVEL=1 on your request. You should be aware of the type of output that you are requesting and be able to process it correctly.

Specifying the System from which You Gather the Information

Depending on how you code the IXCMG macro, you can gather information from the following:

- Local system (GATHERFROM=LOCAL)
- Some other system in the sysplex (GATHERFROM=OTHER)

  To use GATHERFROM=OTHER, the requester needs to provide the XCF system ID of the system from which the data is to be gathered and a timeout value. An optional ECB can be provided if the requester wants to be posted when the results arrive. If not posted, the requester is expected to poll for the results. If the request is accepted, the output DATAAREA contains a request token (AMDAGFO_REQTOKEN) that is used to obtain the results of the asynchronous data. Use this token as input to a subsequent IXCMG GATHERFROM=TOKEN request to retrieve the results. If the user does not gather the results before the timeout, XCF discards the results.

When you request information using GATHERFROM=LOCAL or GATHERFROM=TOKEN, the data returned includes the following:

- A header record (AMDAGFD), which includes:
  - Areas mapped by AMDAREA
  - Length of the header
  - IXCMG return and reason codes
  - XCF system ID and name of the system that collected the data
  - TOD when data gathering started
  - AMDALEVEL requested

- Two Gatherer Level Infomation records (AMDGLI) that indicate the AMDALEVEL of the data records returned, and the maximum AMDALEVELs that can be returned.
- The requested data records (AMDPATH, AMDMPEND, AMDSYS, AMDSD, and AMDMEM).

When you request information using GATHERFROM=OTHER, the data returned includes the following:

- A header record (AMDAGFO), which includes the following:
  - Length of the header
  - IXCMG return and reason codes
  - Maximum PLISTVER and AMDALEVEL supported by the target system
  - A request token, if the request was accepted

See IXCYAMDA in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for complete information on field names and lengths, offsets, and descriptions of the fields mapped by the IXCYAMDA mapping macro.

Specifying the Type of Information

Depending on how you code the IXCMG macro, you can obtain information about:

- Outbound and inbound XCF signaling paths (TYPE=PATH parameter)
- Pending message requests (TYPE=MSGPEND parameter)
- System usage information (TYPE=SYSTEM parameter)
- Members sending and receiving messages (TYPE=SRCDST parameter)
- Member data (TYPE=MEMBER parameter)

  Note: For GATHERFROM=LOCAL requests, the member has to be active on the local system; for GATHERFROM=OTHER, the member has to be active on the system that performs the data collection.
- All data types (the default, TYPE=ALL parameter).

  Note: For AMDALEVEL 0 requests, ALL is equivalent to having specified PATH, MSGPEND, SYSTEM, and SRCDST for TYPE. For AMDALEVEL > 0 requests, ALL is equivalent to having specified PATH, MSGPEND, SYSTEM, SRCDST, and MEMBER for TYPE.

  For GATHERFROM=OTHER requests, the target system determines what data types are to be returned when ALL is specified. Because the target system can support more data types than the local system, it can include data records that are not understood by the local system. Also be aware that the amount of data collected when ALL is specified can be quite large.

The information that IXCMG provides is mapped by the IXCYAMDA mapping macro. IXCYAMDA provides six major structures, related to:

- Signaling paths (AMDPATH structure)
- Pending message requests (AMDMPEND structure)
- System usage (AMDSYS structure)
- Members sending and receiving messages (AMDSD structure)
- Member data (AMDMEM structure)
- Header record that describes the data records returned (AMDAREA, AMDAGFD, or AMDAGFO)

See IXCYAMDA in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for complete information on field names and lengths, offsets and descriptions of the fields mapped by the IXCYAMDA mapping macro.

Signaling Paths

When you request information about signaling paths, the information returned includes:

- For each outbound XCF path:

  - The total number of times XCF selected the path for message transfer while the path was not busy. This count includes re-sent signals.
  - The total number of times XCF selected the path for message transfer while the path was busy. This count includes re-sent signals.
  - The current number of signals pending for data transfer over the path.
- For each inbound path, the total number of times the path could not replenish a message buffer because there was not enough message buffer space available.
- For each XCF path, the count of restarts performed against the path (XCF restarts a path when the path fails)

Pending Message Requests

When you request information about pending message requests, the information returned includes the following for each outbound message queued for delivery:

- The message requestor's member token, primary ASID, and home ASID
- The length of the message
- The transport class XCF selected for transferring the message.

System Usage

When you request information about system usage, XCF returns records that describe the message traffic associated with the system on which you issue IXCMG (called the **local** system). XCF describes this message traffic in terms of the messages sent and received by the local system. To describe the messages **sent** by the local system, XCF returns one or more records for each possible **target** system (the system receiving the messages.) The local system is also a target system. The number of records XCF returns per target system equals the number of transport classes defined to the local system.

To describe the messages **received** by the local system, XCF returns one record for each possible **source** system (the system sending messages to the local system). The local system is not considered a source system for this purpose.

For example, if system 1, system 2, and system 3 are systems in a sysplex, system 1 has transport classes A, B, and C, and an authorized routine on system 1 issues IXCMG, XCF returns the following data:

- For messages sent from system 1 to itself, one record for each of system 1's transport classes (A, B, and C)
- For messages sent from system 1 to system 2, one record for each of system 1's transport classes (A, B, and C)
- For messages sent from system 1 to system 3, one record for each of system 1's transport classes (A, B, and C)
- For messages sent from system 2 to system 1, one record
- For messages sent from system 3 to system 1, one record.

The information returned to describe messages **sent** by the local system includes:

- The total number of times message requests in each transport class were refused because of inadequate message buffer space
- The total number of times message requests in each transport class were migrated to an alternate transport class because no signaling paths were available in that transport class
- The total number of times message requests in each transport class exceeded the message length defined for that class.

The information returned to describe messages **received** by the local system includes:

- The total number of messages received from a source system.

Members Sending and Receiving Messages

When you request information about members sending and receiving messages, the information returned includes:

- For each active member on the system on which IXCMG is called:
  - The approximate number of messages sent by the member
  - The number of messages received by the member.
- For each active member on a remote system:
  - The number of messages the member sent that were received by the system on which IXCMG is called
  - The approximate number of message requests sent to the member by the system on which IXCMG is called.

Consider the following example illustrating what counts are incremented when one member sends a message to another member:

- When member 1 on system 1 sends a message to member 2 on system 2, XCF increments the following counts on system 1:
  - The number of messages sent **by member 1**
  - The number of messages sent **to member 2**
  - The number of times the XCF path was selected while busy or while not busy, whichever is appropriate
  - The number of signals queued for delivery on that XCF path.
- When member 2 on system 2 receives the message sent by member 1 on system 1, XCF increments the following counts on system 2:
  - The number of messages received **from member 1**
  - The number of messages received **by member 2**.

Member Data

When you request information about member data, the information returned for each eligible member includes:

- XCF group name, member name, and member token for the indicated member
- Jobname and ASID of the member
- Name and XCF token for the system on which member resides
- Cumulative counts of the following information:
  - Messages accepted for delivery
  - Messages rejected for lack of a message buffer
  - Local and remote messages delivered to the member
  - Group events that were to be delivered to the member
  - Remote signals received for the member
- The average message transfer time for the most recent remote inbound signals
- The number of signal and group work items currently queued for processing
- Member flags, which indicates the following information about the member:
  - If it is considered stalled
  - If it appears to be contributing to signaling sympathy sickness
  - If it is being deactivated
  - If it is being ended by SFM
- Number of currently queued work items that consume an XCF signal buffer, a DREF buffer, or a pageable buffer
- Number of pending, completed, and saved msgout requests
- Array of data items, each of which is mapped by AMDMEMDI

Header Record

When you request the information mapped by the AMDPATH, AMDMPEND, AMDSYS, or AMDSD structures, the IXCMG macro also provides a header record. This record includes:

- The total length of the output data area needed to contain all the requested information (including the area for the records that were successfully returned on this call).
- For path, pending message, system, and member entries:
  - Number of entries
  - Length of data
  - Offset to entries.

The header record is mapped differently depending on the IXCMG GATHERFROM specification. If not specified, AMDAREA is used. For GATHERFROM=LOCAL and GATHERFROM=TOKEN, AMDAGFD is used. For GATHERFROM=OTHER, AMDAGFO is used.

Table 8 on page 116 summarizes the parameters you code on the IXCMG macro and the resulting information that XCF provides.

Table 8. Summary of IXCMG macros and information XCF provides

Parameter on IXCMG	Information Returned	Structure in IXCYAMDA
TYPE=PATH	Header record	AMDAREA
	One record for each XCF signaling path	AMDPATH
TYPE=MSGPEND	Header record	AMDAREA
	One record for each message pending	AMDMPEND
TYPE=SYSTEM	Header record	AMDAREA
	Records describing the message traffic associated with the system on which you issue IXCMG.	AMDSYS
TYPE=SRCDST	Header record	AMDAREA
	One record for each active member	AMDSD
TYPE=MEMBER	Header record	AMDAREA
	One record for each active member on the target system	AMDMEM
TYPE=ALL	Header record	AMDAREA
	All of the above.	AMDPATH, AMDMPEND, AMDSYS, and AMDSD for QUALEVEL=0. If QUALEVEL>0, AMDMEM records are also included.

Disassociating Members from XCF

Similar to defining members to XCF, disassociating members from XCF is a process that requires some planning. Once a member is defined to XCF and is in a **created** or **active** state, there are a number of ways that it can become disassociated from XCF. For each member, you have the following choices:

- Do a controlled stop by placing an active member in the quiesced state through IXCQUIES, and then, at a later time, in the not-defined state through IXCDELET. (A member must have permanent status recording to choose this option.) Placing the member in a quiesced state disassociates the member from XCF services (cannot send and receive messages, cannot be monitored, etc.) but the member is still known to XCF. Placing the member in the not-defined state then disassociates the member completely from XCF.

- Immediately place an active member in the not-defined state through the IXCLEAVE macro. (Permanent status recording is not required for this option.)
- Place a created member in the not-defined state through the IXCDELET macro.
- Allow an active member with permanent status recording to terminate without explicitly disassociating from XCF, causing the member to be placed in the failed state. Then disassociate the member from XCF through the IXCDELET macro. Any event causing termination, either normal or abnormal, will cause an active member with permanent status recording to be placed in the failed state.
- Force an active member to stop using XCF services by issuing the IXCTERM macro. The member's recovery routine then gets control and decides the member's final state. You can use IXCTERM for a member with or without permanent status recording.

The section entitled "The Five Member States" on page 12 provided information you need regarding the quiesced, failed, and not-defined member states to determine how you should disassociate each member from XCF. The information in this section tells you how to use the IXCQUIES, IXCLEAVE, IXCDELET, and IXCTERM macros to achieve the desired results. Also included in this section is information on providing recovery when a member does not explicitly disassociate from XCF.

Using the IXCQUIES Macro

A member must be **active** with permanent status recording to use the IXCQUIES macro to become **quiesced**. The member must supply its member token (MEMTOKEN parameter). This token was provided by the IXCJOIN macro.

Optionally, the member can change its user state value by coding the USTATE and USLEN parameters on IXCQUIES. Changing the user state value on IXCQUIES does not cause XCF to notify the group user routines of the other active members that the user state value is changed. XCF schedules the group user routines because of the **member state change**; the user state field is included as part of the parameter list passed to the routines.

Using the IXCLEAVE Macro

A member must be **active**, with or without permanent status recording, to use the IXCLEAVE macro to become **not-defined**. The member must supply its member token (MEMTOKEN parameter). This token was provided by the IXCJOIN macro.

Optionally, the member can change its user state value by coding the USTATE and USLEN parameters on IXCLEAVE. Changing the user state value on IXCLEAVE does not cause XCF to notify the group user routines of the other active members that the user state value is changed. XCF schedules the group user routines because of the **member state change**, but the user state field is included as part of the parameter list passed to the routines.

Using the IXCDELET Macro

Issue IXCDELET to completely disassociate a created, quiesced, or failed member from XCF. This service allows multisystem applications and installation-provided routines to remove all information about a particular member from the data that XCF maintains.

Any authorized routine can issue the IXCDELET macro to place a created, quiesced, or failed member in the **not-defined** state. The authorized routine calling IXCDELET does not have to be a member of any XCF group. The routine must supply the target member's token (TARGET parameter).

Using the IXCTERM Macro

An active member of an XCF group can issue IXCTERM to force another active member of the same group to terminate. The caller of IXCTERM must be running in the primary address space of the caller of the IXCJOIN that defined the calling member to the group. The target of IXCTERM can be an active member anywhere in the sysplex. The target member must not be associated with an address space; if it is, IXCTERM will not work.

Invoking IXCTERM **does not**:

- Result in an immediate member state change for the target member
- Immediately cause XCF to schedule the group user routines of other active members of the group.

Invoking IXCTERM **does**:

- Abnormally end the target member's associated task or job step task (whichever association was designated on IXCJOIN) with system completion code 00C and reason code 4.

  You should be aware when invoking IXCTERM that XCF ends every member associated with the target member's tasks and its subtasks. Also, issuing IXCTERM against a member that resides on a system that is in the middle of an IPL causes the system to enter a wait-state.
- Pass control to the target member's recovery routine. The recovery routine determines the final state of the member, and this decision determines what notification the group user routines receive. The recovery routine **is not allowed to retry**, because the object of invoking IXCTERM is to terminate the member.

The caller of IXCTERM should be aware that the terminate service runs asynchronously; the target member might still be associated with XCF when the issuing member regains control.

Member Termination

A member is terminated (put in a failed or not-defined state) when the task, job step task, address space, or system that the member is associated with ends. If the member does not explicitly disassociate from XCF, the resulting member state depends on whether the member has permanent status recording. The member with permanent status recording becomes failed. The member without permanent status recording becomes not-defined. In either case, XCF notifies the group user routines of the other active members of the group about the member state change. Another member of the group or authorized routine can then provide recovery (cleanup of resources) for the member.

XCF considers the member as terminated under any of the following conditions:

- The member is task associated and the task, the active job step task, address space, or system ends
- The member is job step task associated and the job step task, address space, or system ends.
- The member is only address space associated (not task or job step task) and its address space or system ends.

Also, consider the following conditions about termination of the member:

- The value of the TERMLEVEL keyword for IXCTERM when the member invoked IXCJOIN to become an active member of the group affects termination. For example, if TERMLEVEL=ADDRSPACE when it joined the group, the member's address space will be terminated and every XCF group member associated with that address space including any other group will be terminated. If TERMLEVEL=SYSTEM, the system on which the target member resides will be removed from the sysplex.
- If the system that is to terminate the member is being initialized (that is, it is running during NIP), the system will be removed from the sysplex.

Note: See “Member Association” on page 17 for more information on member association as it relates to member termination.

For abnormal task termination, members can use MVS recovery services (ESTAEs, FRRs, ARRs, etc.) to retry or perform cleanup of resources. For normal or abnormal task or address space termination, members can also provide resource manager routines to get control. Resource manager routines cannot retry, but can perform cleanup of resources. See *z/OS MVS Programming: Authorized Assembler Services Guide* for information about using resource managers.

Task Termination

When a member's associated task (or associated job step task) ends, the system passes control to whatever end-of-task recovery routines or resource manager routines the member provided. These routines can:

- Clean up any multisystem resources the member was using, as necessary. MVS end-of-task resource managers might not provide sufficient cleanup if the terminated member was accessing shared data under work units other than the terminated task. The member's recovery or resource manager routine should consider:
  - Sysplex serialization requests
  - Outstanding operator communication
  - Shared DASD access.
- Depending on the environment, initiate other task or address space terminations as required to ensure the integrity of shared data resources.
- Optionally issue SDUMPX SDATA=(COUPLE) to include group and member relationships in a dump.
- Determine the final state of the member. For a member with permanent status recording, the routine has the following choices:
  - Issue IXCQUIES to place the member in a **quiesced** state.
  - Issue IXCLEAVE to place the member in a **not-defined** state.
  - Allow the member to terminate without explicitly disassociating from XCF. The member then becomes **failed**.

  For a member without permanent status recording, the recovery routine can issue IXCLEAVE or allow the member to terminate without explicitly disassociating from XCF. In either case, the resulting member state is **not-defined**.
- Determine whether the member's XCF request was completed before the task abnormally terminated. You can obtain this information by issuing the IXCQUERY macro with REQINFO=GROUP,REQTYPE=DEFER and specifying the appropriate group and member names. XCF might or might not have finished processing the member's XCF request before the member's task was terminated.

XCF ensures that all connections to XCF services are broken by checking all members that became active under the terminating task. For those that did not explicitly disassociate, XCF does the following:

- For members with permanent status recording, places the member in a failed state.
- For members without permanent status recording, places the member in a not-defined state.
- For normal termination, generates a symptom record to identify the members that did not explicitly disassociate from XCF. In addition to the required fields in the symptom record, XCF records the group name and member name in the variable recording area. The symptom record in LOGREC alerts the installation of a programming error in the terminating multisystem application.

  XCF generates one symptom record for up to 16 members associated with a terminating task, rather than generating one record for each member.

Members that are accessing multisystem resources should not depend on XCF to provide sufficient cleanup.

Address Space Termination

In some memory termination environments, such as DAT errors, a task's recovery routines and end-of-task resource managers do not get control. The system gives control only to end-of-memory resource managers. To protect against these situations, a member can have an end-of-memory resource manager routine that runs in the master scheduler address space. This routine can do cleanup for the member, and issue IXCQUIES, IXCLEAVE, and IXCMSGOX for the member.

If the member did not provide an end-of-memory resource manager routine, XCF ensures that the member is disassociated.

If the member is address space associated, I/O for the member is cleaned up before XCF ends the member.

Removing Systems from the Sysplex

When the system that a member is running on is removed from the sysplex, XCF notifies the group user routines of the other active members of the group on other systems so that they can take recovery action for the member.

XCF reports to the group user routines the following event types that relate to systems being removed from the sysplex. See "Events that Cause XCF to Schedule a Group User Routine" on page 77 for a complete description of each event type.

- GESYSGO (system reported going)
- GESYSPRT (system being removed from the sysplex)
- GESYSGON (system reported gone)
- GESYSDG (system detected gone).

Example of Designing and Implementing a Multisystem Application

Note: This example illustrates only mainline paths, and does not cover error conditions, serialization, or synchronization. The intent of this example is to illustrate, at a high level, the way members of a group interact, the way the members use the XCF macros, and the way the various user routines interact. Examples of macro invocations are provided where appropriate. See *z/OS MVS Programming: Sysplex Services Reference* for an explanation of the parameters used on each macro.

In this example, an installation has three MVS systems in an XCF sysplex. Users on each of the three systems can obtain phone numbers from a database, and can add, change, and delete phone numbers.

The multisystem application that handles maintaining the database, and providing information to users, consists of a group (called PHONBOOK) with one member on each of the MVS systems. The members consist of identical routines. All three members have identical message, status, and group user routines.

All three group members can accept requests for work, but only the member designated as PRIMARY can perform the work. When the PRIMARY member fails, the BACKUP member takes over the work. This is an example of using XCF to achieve high availability.

Figure 14 on page 121 illustrates the relationship between the members of the group. Member 1 and member 2 have access to the DASD device that contains the PHONBOOK database. Member 3 does not have access to the database.

In the figure, member 1 is shown as the PRIMARY member and member 2 is shown as the BACKUP member. It is also possible for member 2 to be the PRIMARY member and for member 1 to be the BACKUP member.

Member 3 is shown as NO-BACKUP because member 3 cannot access the database. Member 3 cannot be PRIMARY or BACKUP.

The PRIMARY, BACKUP, and NO-BACKUP designations are made by the operator when the tasks are started. The designations are maintained in the member's user state field. These designations can change dynamically if something happens to the PRIMARY member, causing the BACKUP member to take over. See "What is Another Method for Designating Members?" on page 131 for an alternate way to designate the members.

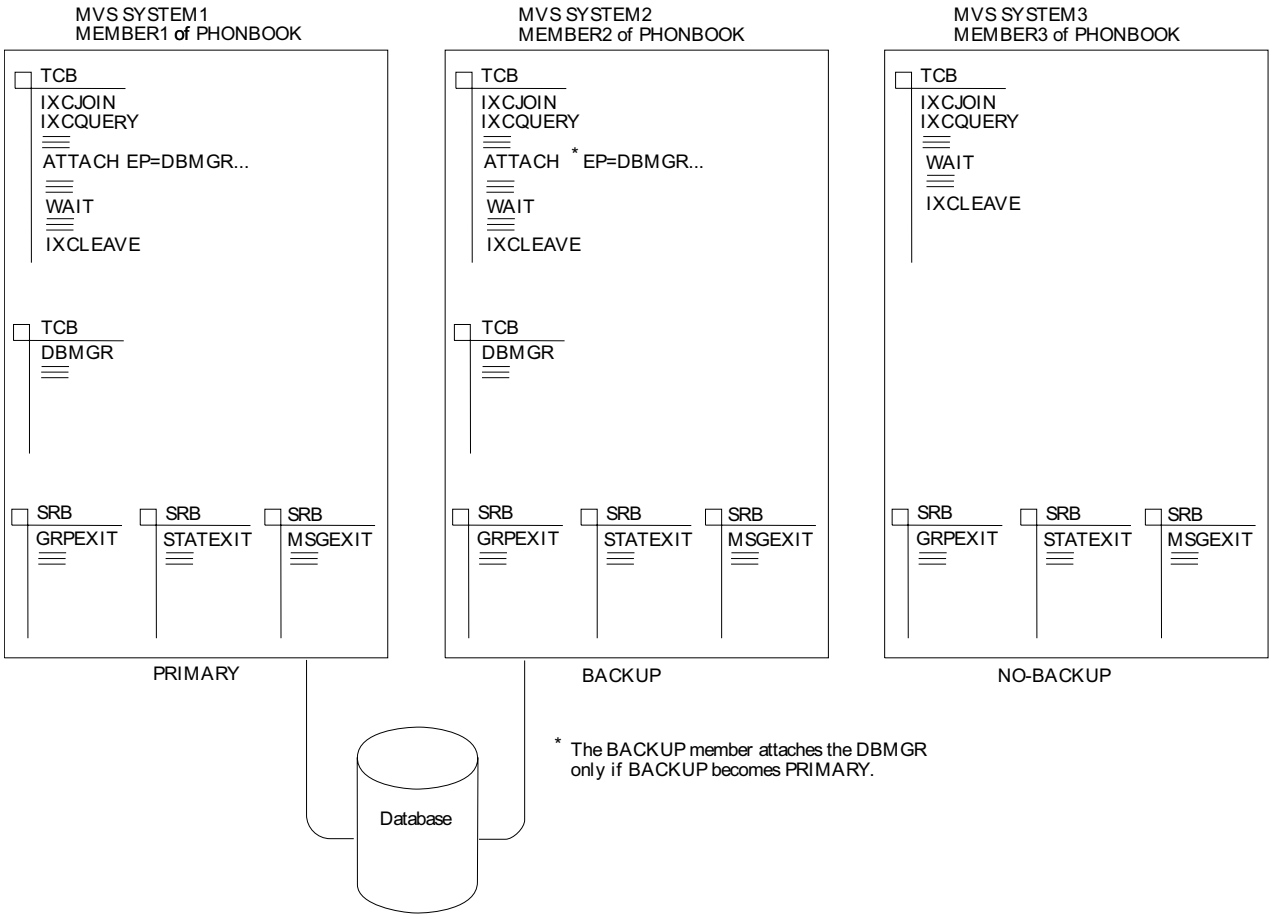


Figure 14. PHONBOOK Multisystem Application

How Does PHONBOOK Work?

In general, the PHONBOOK routine works in the following manner:

- Each time a user on a particular system submits a request for work, the member on that system places an element on its work queue.
- The member then sends a message containing the request to the PRIMARY member.
- The PRIMARY member's message user routine posts an authorized routine called the database manager (DBMGR) and places the request on the DBMGR's work queue. The DBMGR's work queue is separate from the member's work queue.
- DBMGR does the actual updating of, or retrieval of information from, the database.
- When DBMGR is done, it sends the information requested, or an acknowledgment that an update has been made, to the requesting member, and takes the request off its work queue.
- The requesting member then removes that request from its work queue.
- If anything goes wrong with the PRIMARY member, an authorized routine called the CLEANUP task ends the PRIMARY member and changes the BACKUP member to PRIMARY.

  The CLEANUP task waits for two different ECBs (TASKECB1, posted by the message user routine, and TASKECB2, posted by the group user routine.) The group user routine posts TASKECB2 to alert the CLEANUP task to do the takeover. The CLEANUP task cannot do the takeover until the message user routine places needed information into MDATASTR (see “What Data Structures Does PHONBOOK Use?” on page 122). So, the message user routine posts TASKECB1 when this is accomplished.

How Does a Member Update its Status Field?

Members update their status fields by storing the clock (STCK instruction). Members determine when to store the clock as follows:

- The PRIMARY member updates its status field every time the DBMGR deletes an element from the DBMGR's work queue (signifying that the work is completed).
- The BACKUP and NO-BACKUP members update their status fields every time they delete an element from their member work queues.

When the BACKUP member takes over for the PRIMARY member, the BACKUP member must change its method of updating its status field.

What Data Structures Does PHONBOOK Use?

Figure 15 on page 123 illustrates the data structures that the PHONBOOK routine uses to do its work. When each member joins the group, the member specifies as member data (MEMDATA parameter on IXCJOIN) the address of the MDATASTR data structure. MDATASTR contains the following information:

Field name	Contents
TBLADDR	Address of the table created and maintained by the group user routine (the TABLE data structure).
NEXTITEM	Address of next available slot in the table.
MEMWQHDR	Address of the member's work queue.
DBWQHDR	Address of the DBMGR's work queue. (This field used only by the PRIMARY member.)
MAINECB	Address of the ECB that the main routine waits for.
TASKECB1	Address of the ECB that the CLEANUP task waits for, and the message user routine posts.
TASKECB2	Address of the ECB that the CLEANUP task waits for, and the group user routine posts.
XPRIMBU	Indicates a switch from PRIMARY to BACKUP. The group user routine turns this switch on the first time it is called for a status update missing.

The TABLE data structure contains the following information:

Field name	Contents
MEMNAME	The member's name.
MEMTOKEN	The member's token.
MEMSTATE	The member's state.
MEMUSTAT	The member's user state value.

The member's work queue and the DBMGR's work queue both consist of work elements (WRKELEMT). Each WRKELEMT contains a pointer to the next element on the queue, the requesting member, and the work to be done. If the pointer to the next element is zero, the queue is empty.

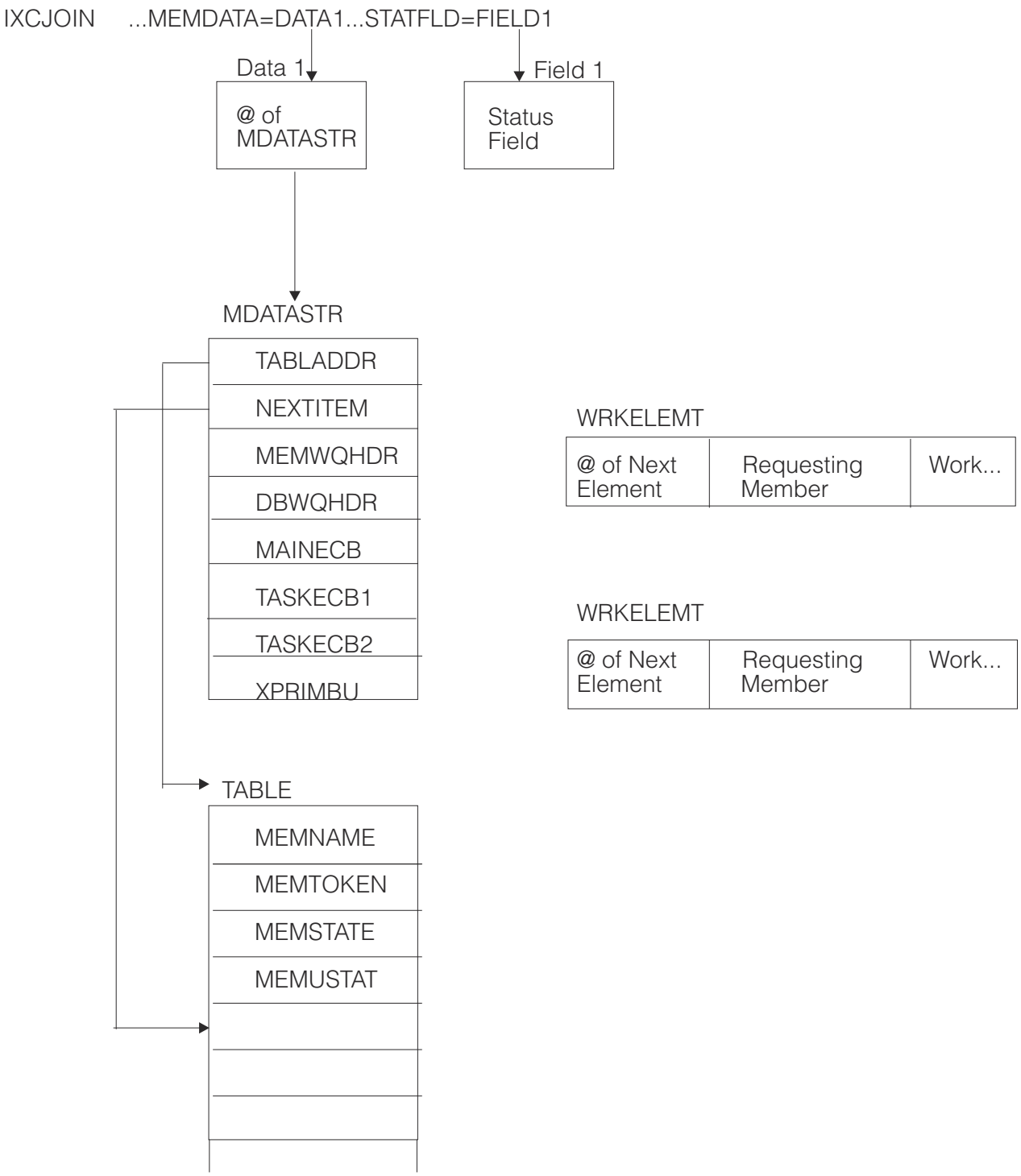


Figure 15. Data Structures Used by PHONBOOK Routine

What Do the User Routines Do?

Each member has a message, status, and group user routine. This section explains what each user routine does.

The Message User Routine

The message user routine for the PRIMARY member receives messages that contain requests for work to be done. The message user routines for the BACKUP and NO-BACKUP members receive messages

that contain acknowledgments of work completed or that contain the requested information. The routines determine what has to be done based on the contents of the message control information.

The members of the group have established the following protocol for the use of the message control information (MSGCNTL parameter on IXCMSGOX):

MSGCNTL Contents	Meaning
REQDATA	Request for data.
RETDATA	Return with data.
REQUPD	Request for update.
GOODRC	Confirmation of successful update.
BADRC	Requested update failed.
WORKXFER	The PRIMARY member's status user routine is transferring the work queue to the BACKUP member. (This is the case where the PRIMARY member's status user routine confirms that the member's status update is missing, and the BACKUP member is taking over.)
WORKREQ	The BACKUP member is now the PRIMARY member, and is requesting the work queue from the NO-BACKUP member. (This is the case where XCF assumed a status update missing for the PRIMARY member, indicating that the member's status user routine did not run successfully, and so could not transfer the work queue.)

Based on the contents of the message control information, the message user routine does the following:

- If the message control information contains REQDATA or REQUPD, the routine reads in the information from the message buffer, places the request on the DBMGR's work queue, and posts the DBMGR routine.
- If the message control information contains RETDATA, the routine reads the information from the message buffer into a pre-established work queue element, and posts a task (the RETINFO task) to return the data to the caller and to notify the requesting member to delete that request from its work queue.
- If the message control information contains GOODRC or BADRC, the routine posts the RETINFO task to inform the caller that the requested update was or was not successful.
- If the message control information contains WORKXFER, the BACKUP member knows it will be taking over for the PRIMARY. The routine reads in the work queue from the message buffer, placing the queue in storage the routine has obtained. The routine then places the address of the queue into MDATASTR, and posts TASKECB1.
- If the message control information contains WORKREQ, it means the following:
  - The BACKUP member took over for the PRIMARY member.
  - The BACKUP member did not receive the DBMGR's work queue.
  - The BACKUP member has to build a new work queue for its DBMGR routine.

  NO-BACKUP's message user routine loops through NO-BACKUP's work queue and issues IXCMSGOX to send each element on the queue to the BACKUP member.

The Status User Routine

When a member misses updating its status field within its prescribed interval, XCF schedules the member's status user routine. The routine determines whether the member is operating normally by checking the following:

- For the PRIMARY member, the DBMGR's work queue
- For the BACKUP and NO-BACKUP members, the member's work queue.

The status user routine's actions depend on whether the member is PRIMARY. The routine determines this by checking a table that is maintained by the group user routine. If the member is PRIMARY (MEMUSTAT=PRIMARY), the status user routine checks the DBMGR's work queue, and does the following:

- If the work queue is empty, the routine sets a return code of SEUPDRES to indicate the member is operating normally.
- If the work queue is not empty, the routine sets a return code of SEUPDMIS, indicating that the member's status update is missing. It then issues IXCMSGOX, sending the DBMGR's work queue to the BACKUP member. To send the work queue, the status user routine places each element into the message buffer so that all the elements are sent as one block of data. XCF then:
  - Schedules the message user routine of the BACKUP member. (This message user routine reads in the work queue and posts TASKECB1.)
  - Schedules the group user routines of the BACKUP and NO-BACKUP members. (The backup member's group user routine is responsible for posting TASKECB2 to alert the CLEANUP task to do the takeover.)

If the member is not PRIMARY, the status user routine checks the member's work queue, and does the following:

- If the work queue is empty, the routine sets a return code of SEUPDRES, indicating that the member is operating normally.
- If the work queue is not empty, the routine sets a return code of SEUPDMIS, indicating that the member's status update is missing.

The Group User Routine

A member's group user routine receives control under a variety of circumstances. In this example, the group user routines have two basic functions:

- To create and maintain a table with entries for each member of the group. The group user routine serializes the use of this resource by obtaining the local lock.
- To initiate a takeover when the PRIMARY member fails.

To accomplish these functions, the group user routines are concerned about the following events:

- Member state changes (GEPLTYPE=GEMSTATE)
- Member status updating missing (GEPLMISR flag is on if the member's status user routine reported the status update missing; GEPLMISD flag is on if XCF assumed a status update missing for the member.)

When a member state change occurs (GEPLTYPE=GEMSTATE), the group user routine loops through its table and does one of the following:

- Adds the member to the table if no entry for that member exists
- Updates the member's entry.

When a status update missing event occurs, the group user routine's actions depend on the following:

- The member whose status update is missing (PRIMARY, BACKUP, or NO-BACKUP)
- The member whose group user routine is being scheduled (PRIMARY, BACKUP, or NO-BACKUP)
- Whether the member's status user routine reported the status update missing condition, or whether XCF assumed that condition for the member.

Table 9 on page 125 summarizes the possible combinations of the member whose group user routine is scheduled and the member whose status update is missing (either reported to or assumed by XCF):

Table 9. Group User Routine Scheduled vs. Status Update Missing

Combination Number	Whose group user routine is scheduled?	Whose status update is missing?
1	BACKUP	PRIMARY
2	BACKUP	NO-BACKUP
3	NO-BACKUP	PRIMARY
4	NO-BACKUP	BACKUP

Table 9. Group User Routine Scheduled vs. Status Update Missing (continued)		
Combination Number	**Whose group user routine is scheduled?**	**Whose status update is missing?**
5	PRIMARY	BACKUP
6	PRIMARY	NO-BACKUP

Note: Remember the following:

- When a member misses its status update, its **own status user routine** runs.
- When a member's status user reports the member's status update missing (or XCF assumes a missing status update for the member), the **group user routines** of the **other active members of the group** run.

The following explains the actions of the group user routine for each of the combinations specified above. For all combinations except combination 1, the group user routine takes the same action whether the member's status user routine reported the status update missing condition or XCF assumed that condition for the member.

Combination 1

When the BACKUP member's group user routine gets control because the PRIMARY member's status update is missing, the routine:

- Posts a task (the CLEANUP task) that:
  - Waits for both TASKECB1 (to be posted by the message user routine) and TASKECB2 (to be posted by the group user routine).
  - Terminates the PRIMARY member by issuing IXCTERM. (The PRIMARY member's recovery routine then gets control and can disassociate the member from XCF through the IXCLEAVE macro.)

```
IXCTERM   MEMTOKEN=MEM2TKN,TARGET=MEM1TKN,RETCODE=RETURN,  X
    RSNCODE=REASON,MF=S
IXCLEAVE  MEMTOKEN=MEM1TKN,RETCODE=RETURN,                X
    RSNCODE=REASON,MF=S
```

  - Change the BACKUP member's user state value from BACKUP to PRIMARY through the IXCSETUS macro.

```
IXCSETUS  MEMTOKEN=MEM2TKN,NEWUS=PRIMARY,TARGET=MEM2TKN,   X
    RETCODE=RETURN,RSNCODE=REASON,MF=S
```

- Turns on the XPRIMBU switch in MDATASTR.

When XCF assumes a status update missing for the PRIMARY member, the primary member's status user routine might never have sent the DBMGR's work queue. The group user routine still posts the CLEANUP task to terminate the PRIMARY member and change the user state values. However, the CLEANUP task does not wait for TASKECB1 to be posted by the message user routine. Additionally, the group user routine issues IXCMSGOX to the NO-BACKUP member, with the message control information containing WORKREQ. This signals the NO-BACKUP member to send its work queue to the BACKUP member for processing.

Combination 2

When the BACKUP member's group user routine gets control because the NO-BACKUP member's status update is missing, the routine checks NO-BACKUP's member state in the table. If NO-BACKUP is still active, the routine takes no action, assuming that the member missed its status update for a valid reason. If NO-BACKUP is not-defined, the routine posts the CLEANUP task to delete the member from the table.

Combination 3

When the NO-BACKUP member's group user routine gets control because the PRIMARY member's status update is missing, the routine takes no action because the BACKUP member will do the work.

Combination 4
When the NO-BACKUP member's group user routine gets control because the BACKUP member's status update is missing, the routine takes no action because the PRIMARY member will do the work.

Combination 5
When the PRIMARY member's group user routine gets control because the BACKUP member's status update is missing, the routine checks BACKUP's member state in the table. If BACKUP is still active, the routine takes no action, assuming that the member missed its status update for a valid reason. If BACKUP is not-defined, the routine posts the CLEANUP task to delete the member from the table.

Combination 6
When the PRIMARY member's group user routine gets control because the NO-BACKUP member's status update is missing, the routine takes no action because the BACKUP member will do the work.

How Does the Installation Set Up PHONBOOK on Each System?

This section describes what happens as each of the three systems are set up to take work requests. The steps are described sequentially. However, you should realize that these events do not necessarily happen sequentially. Once a member issues IXCJOIN, any of its user routines can get control in any order, and can even get control prior to completion of the IXCJOIN service. The example shows the operator starting the tasks on system 1, then system 2, then system 3. However, it is possible, for example, that member 2 might finish initialization prior to member 1 and be the first member in the table. This is one reason why each member issues IXCQUERY to determine which other members are already active.

Setting Up on System 1

The following explains what happens when the operator starts the PHONBOOK routine on system 1:

- The routine prompts the operator, through a WTOR, to designate PRIMARY, BACKUP, or NOBACKUP.
- The operator replies PRIMARY.
- The task on system 1 issues IXCJOIN as follows:

```
SYSTEM1  IXCJOIN  GRPNAME=PHONBOOK,ANSAREA=(R2),ANSLEN=AREALEN,     X
              LASTING=NO,MEMNAME=MEMBER1,GRPEXIT=(R4),              X
              STATEXIT=(R5),MSGEXIT=(R6),USTATE=USTATEP,            X
              STATFLD=(R7),INTERVAL=INTER1,MEMDATA=(R3),            X
              USLEN=LEN,RETCODE=RETURN,RSNCODE=REASON,MF=S
```

  Member 1 is now established as the PRIMARY member.
- Member 1 issues IXCQUERY to determine if any other members have joined the group yet. At this point, no other members are initialized.

```
    IXCQUERY REQINFO=GROUP,GRPNAME=PHONBOOK,ANSAREA=(R2),           X
          ANSLEN=AREALEN,REQTYPE=DEFER,MF=S
```

- Member 1 adds itself to member 1's copy of the table.
- When member 1 becomes active, XCF schedules the group user routines of any other active members in the group. However, at this point, member 2 and member 3 are not started, so their group user routines are not scheduled.
- Member 1 attaches DBMGR as a subtask and waits for work.

Setting Up on System 2

The following explains what happens when the operator starts the PHONBOOK routine on system 2:

- The routine prompts the operator, through a WTOR, to designate PRIMARY, BACKUP, or NOBACKUP.
- The operator replies BACKUP.
- The task on system 2 issues IXCJOIN as follows:

```
SYSTEM2  IXCJOIN  GRPNAME=PHONBOOK,ANSAREA=(R2),ANSLEN=AREALEN,     X
              LASTING=NO,MEMNAME=MEMBER2,GRPEXIT=(R4),              X
```

```
               STATEXIT=(R5),MSGEXIT=(R6),USTATE=USTATEB,            X
               STATFLD=(R7),INTERVAL=INTER1,MEMDATA=(R3),            X
               USLEN=LEN,RETCODE=RETURN,RSNCODE=REASON,MF=S
```

Member 2 is now established as the BACKUP member.

- Member 2 issues IXCQUERY to determine if any other members have joined the group yet. At this point, member 1 is active.
- Member 2 adds itself and member 1 to member 2's copy of the table.
- XCF schedules the group user routine of member 1, notifying member 1 that member 2 is now active. (Member 3 is not active yet, so member 3's group user routine is not scheduled.)
- Member 1's group user routine now updates member 1's table (adds member 2).
- Member 2 does not attach the DBMGR task because member 2 is not PRIMARY.
- Member 2 waits for work.

Setting Up on System 3

The following explains what happens when the operator starts the PHONBOOK routine on system 3:

- The routine prompts the operator, through a WTOR, to designate PRIMARY, BACKUP, or NOBACKUP.
- The operator replies NOBACKUP.
- The task on system 3 issues IXCJOIN as follows:

```
SYSTEM3  IXCJOIN  GRPNAME=PHONBOOK,ANSAREA=(R2),ANSLEN=AREALEN,     X
               LASTING=NO,MEMNAME=MEMBER3,GRPEXIT=(R4),              X
               STATEXIT=(R5),MSGEXIT=(R6),USTATE=USTATEN,            X
               STATFLD=(R7),INTERVAL=INTER1,MEMDATA=(R3),            X
               USLEN=LEN,RETCODE=RETURN,RSNCODE=REASON,MF=S
```

Member 3 is now established as the NO-BACKUP member.

- Member 3 issues IXCQUERY to determine if any other members have joined the group yet. At this point, members 1 and 2 are active.
- Member 3 adds itself and members 1 and 2 to member 3's copy of the table.
- XCF schedules the group user routines of members 1 and 2, notifying them that member 3 is now active.
- The group user routines of member 1 and member 2 update their copies of the table (add member 3).
- Member 3 does not attach the DBMGR task because member 3 is not PRIMARY.
- Member 3 waits for work.

How Does PHONBOOK Handle Different Types of Work Requests?

This section describes scenarios that illustrate how different types of work requests enter each system and are handled by the PHONBOOK routine. An additional scenario describes what happens when the PRIMARY member misses its status update.

Updating the Database - Requestor is the PRIMARY Member

In this scenario, a user on system 1 wants to add a name to the database, causing the following events to occur:

- Member 1 checks the user states in the table to determine which member is PRIMARY.
- Member 1 determines that it is the PRIMARY member.
- Member 1 creates a work element, places it on its own work queue, and issues to send the work element to the PRIMARY member (in this case, itself).

```
      IXCMSGOX  MEMTOKEN=MEM1TKN,MSGBUF=RECORD1,MSGLEN=LENMSG        X
         MSGCNTL=REQUPD,TARGET=MEM1TKN,RETCODE=RETURN,             X
         RSNCODE=REASON,MF=(E,MSGOLSTD)
```

- The message user routine for member 1 gets control. The routine checks the message control information and determines that this is a request to update the database.
- Member 1's message user routine creates a work element to prepare to receive the message.
- Member 1's message user routine issues IXCMSGIX to read in the message.

```
      IXCMSGIX   MSGTOKEN=TOKENMSG,MSGBUF=(R3),RETCODE=RETURN,       X
            RSNCODE=REASON,MF=(E,MSGILSTD)
```

- Member 1's message user routine places the work element on the DBMGR's work queue, and posts the DBMGR.
- When the DBMGR is done with the work, it issues IXCMSGOX to the requesting member (member 1 in this case) stating that the work is completed, deletes the request from DBMGR's work queue, and updates member 1's status field.

```
      IXCMSGOX   MEMTOKEN=MEM1TKN,MSGBUF=RECORD1,MSGLEN=LENMSG       X
            MSGCNTL=GOODRC,TARGET=MEM1TKN,RETCODE=RETURN,          X
            RSNCODE=REASON,MF=(E,MSGOLSTD)
```

- When member 1's message user routine again gets control, the routine determines that the message control information contains GOODRC, indicating a successful update.
- Member 1 returns the results of the operation to the caller.
- Member 1 deletes that request from its own work queue.

Updating the Database - Requestor is the NO-BACKUP Member

In this scenario, a user on system 3 wants to change a name in the database, causing the following events to occur:

- Member 3 checks the user states in the table to determine which member is PRIMARY.
- Member 3 determines that member 1 is the PRIMARY member.
- Member 3 creates a work element, places the work element on its own work queue, and issues IXCMSGOX to send the work element to the PRIMARY member (member 1).

```
      IXCMSGOX  MEMTOKEN=MEM3TKN,MSGBUF=RECORD1,MSGLEN=LENMSG        X
         MSGCNTL=REQUPD,TARGET=MEM1TKN,RETCODE=RETURN,             X
         RSNCODE=REASON,MF=(E,MSGOLSTD)
```

- The message user routine for member 1 gets control. The routine checks the message control information and determines that this is a request to update the database.
- Member 1's message user routine creates a work element to prepare to receive the message.
- Member 1's message user routine issues IXCMSGIX to read in the message.

```
      IXCMSGIX   MSGTOKEN=TOKENMSG,MSGBUF=(R3),RETCODE=RETURN,       X
            RSNCODE=REASON,MF=(E,MSGILSTD)
```

- Member 1's message user routine places the work element on the DBMGR's work queue, and posts the DBMGR.
- When the DBMGR is done with the work, it issues IXCMSGOX to the requesting member stating that the work is completed, deletes the request from DBMGR's work queue, and updates member 1's status field.

```
   IXCMSGOX   MEMTOKEN=MEM1TKN,MSGBUF=RECORD1,MSGLEN=LENMSG     X
         MSGCNTL=GOODRC,TARGET=MEM3TKN,RETCODE=RETURN,         X
         RSNCODE=REASON,MF=(E,MSGOLSTD)
```

- When member 3's message user routine gets control, the routine determines that the message control information contains GOODRC, indicating a successful update.
- Member 3 returns the results of the operation to the caller.
- Member 3 deletes that request from its work queue, and updates member 3's status field.

Finding a Name in the Database - Requestor is the BACKUP Member

In this scenario, a user on system 2 wants to find a name in the database, causing the following events to occur:

- Member 2 checks the user states in the table to determine which member is PRIMARY.
- Member 2 determines that member 1 is the PRIMARY member.
- Member 2 creates a work element, places it on its own work queue, updates its status field, and issues IXCMSGOX to send the work element to the PRIMARY member (member 1).

```
IXCMSGOX  MEMTOKEN=MEM2TKN,MSGBUF=RECORD1,MSGLEN=LENMSG        X
   MSGCNTL=REQDATA,TARGET=MEM1TKN,RETCODE=RETURN,             X
   RSNCODE=REASON,MF=(E,MSGOLSTD)
```

- The message user routine for member 1 gets control. The routine checks the message control information and determines that this is a request for information from the database.
- Member 1's message user routine creates a work element to prepare to receive the message.
- Member 1's message user routine issues IXCMSGIX to read in the message.

```
IXCMSGIX   MSGTOKEN=TOKENMSG,MSGBUF=(R3),RETCODE=RETURN,       X
      RSNCODE=REASON,MF=(E,MSGILSTD)
```

- Member 1's message user routine places the work element on the DBMGR's work queue, and posts the DBMGR.
- When the DBMGR is done with the work, it issues IXCMSGOX to the requesting member sending the requested information, deletes the request from DBMGR's work queue, and updates member 1's status field.

```
IXCMSGOX   MEMTOKEN=MEM1TKN,MSGBUF=RECORD1,MSGLEN=LENMSG       X
      MSGCNTL=RETDATA,TARGET=MEM2TKN,RETCODE=RETURN,          X
      RSNCODE=REASON,MF=(E,MSGOLSTD)
```

- When member 2's message user routine gets control, the routine determines that the message control information contains RETDATA, indicating that the requested information is being returned.
- Member 2's message user routine issues IXCMSGIX to read in the message.

```
IXCMSGIX   MSGTOKEN=TOKENMSG,MSGBUF=(R3),RETCODE=RETURN,       X
      RSNCODE=REASON,MF=(E,MSGILSTD)
```

- Member 2 returns the data to the requestor.
- Member 2 deletes that request from its work queue, and updates member 2's status field.

Member 1 (PRIMARY) Misses its Status Update

If a problem occurs on system 1, causing member 1 to miss updating its status field, the following events occur:

- Member 1's status user routine gets control.
- The status user routine determines that the DBMGR's work queue has work to be done, so the routine sets a return code of SEUPDMIS, and issues IXCMSGOX to member 2, with the work queue in the message buffer. To send the work queue, the status user routine places each element into the message buffer so that all the elements are sent as one block of data.

```
IXCMSGOX   MEMTOKEN=MEM1TKN,MSGBUF=WORKQUE,MSGLEN=LENMSG       X
      MSGCNTL=WORKXFER,TARGET=MEM2TKN,RETCODE=RETURN,         X
      RSNCODE=REASON,MF=(E,MSGOLSTD)
```

- XCF schedules the group user routines of both member 2 and member 3, and the message user routine of member 2. These events can occur in any order.
- When member 3's group user routine receives control, it takes no action because member 2 is the BACKUP.

- Member 2's CLEANUP task waits for TASKECB1 and TASKECB2 (to be posted by the message user routine and group user routine respectively). The CLEANUP task needs the information being passed to the message user routine.
- Member 2's message user routine checks the message control information and determines that it contains WORKXFER.
- Member 2's message user routine reads in the work queue from the message buffer, places the address of the queue into MDATASTR, and posts TASKECB1.
- Member 2's group user routine now gets control.
- Member 2's group user routine posts TASKECB2 to alert the CLEANUP task to terminate member 1 and change member 2's user state value from BACKUP to PRIMARY.
- Work requests coming in will now go to member 2 for processing, because member 2 is now PRIMARY.

What Happens When all Processing is Complete?

At the end of the day, when all processing is complete, each member issues an IXCLEAVE to disassociate from XCF.

```
        IXCLEAVE MEMTOKEN=MEM1TKN,MF=S
        IXCLEAVE MEMTOKEN=MEM2TKN,MF=S
        IXCLEAVE MEMTOKEN=MEM3TKN,MF=S
```

What is Another Method for Designating Members?

In the example just described, the operator starts each member on a different system and designates the PRIMARY, BACKUP, and NO-BACKUP members. Here is another way you can designate these members:

- If, in your installation, member 1, member 2, and member 3 all have access to the database, any member could be PRIMARY and any member could be BACKUP.
- When each member issues IXCJOIN, have the member set its user state value to BACKUP.
- Each member can check the return code from the IXCJOIN macro to determine if it is the first member to join the group.
- The first member can issue IXCSETUS to change its user state value to PRIMARY.
- The second and third members will determine that they are not the first to join the group, so their user state values remain BACKUP.
- Your program would then contain logic to determine which member takes over when the PRIMARY member fails.

Chapter 3. Using XCF for client/server communication

XCF provides services for client/server communication to enable a client to send requests to systems in the sysplex for processing by a server and to receive the results of that processing. XCF provides the communication and failure handling so that clients and servers can focus on developing their own processing protocols.

Overview of XCF client/server processing

Servers and their clients are expected to perform the following actions:

- Servers must register with XCF and provide an exit routine for XCF to call and process client requests.
- Clients must initiate requests and gather results.
- Servers are generally expected to reply to client requests.

A set of XCF macros allows you to perform the following actions:

- "Instantiate" or start the server to process client requests. To define server properties you use the IXCSRVR macro. You can define one or more servers to receive requests from clients and send responses. See “Defining and starting a server” on page 135.
- Allow clients to send requests to servers as well as allow servers to send responses to clients. To send a request or response, you use the IXCSEND macro. See “Using the IXCSEND macro” on page 144.
- Allow programs to obtain the state of messages sent through IXCSEND and also receive responses to requests sent by servers. To obtain message information or responses from a server, you use the IXCRECV macro. See “Using the IXCRECV macro” on page 153.
- Initiate server requests to the XCF Server. XCF has its own server, the XCF Server, which processes client requests that are formatted with the IXCREQ macro. See “Using the XCF Server” on page 160 and “Using the IXCREQ macro” on page 161.

The IXCYSRVR mapping macro defines mappings and constants to be used when writing client/server applications.

Figure 16 on page 133 summarizes the processing for a basic client/server communication in the sysplex using the services of XCF.

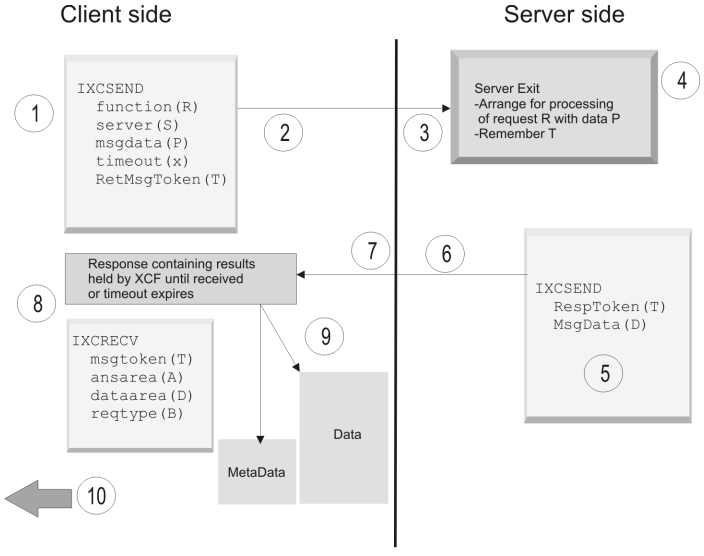


Figure 16. Overview of client/server processing in the sysplex

© Copyright IBM Corp. 1994, 2023

The following steps summarize the processing in the figure:

1. Client issues IXCSEND macro to send request R to server S, providing parameters P. The macro returns a token T that the client later can use to retrieve the results provided by the server. The request R can be at most 100 MB. The target server is identified by its name and the name of the system on which it resides. You can define one or more instances of the same server on each system in the sysplex.
2. XCF builds control blocks to manage the request and invokes IXCMSGOX to send the request and its parameters to one or more systems on which the server resides.
3. XCF on the target system intercepts the signal. If the target server does not exist, the message is discarded and acknowledged by XCF with a "no receiver" response". If the server does exist, XCF invokes the message control service (IXCMSGC) to save the message and queues a work item for the server. As needed, a server is selected and resumed.
4. When the server is resumed, the XCF server exit stub makes suitable preparations for processing the work item, including doing IXCMSGIX to extract the client parameters from the saved message. XCF calls the server exit routine to present the request to the server. The server exit routine inspects R to determine which type of request is to be processed. The server exit can process the request directly or it can arrange for it to be processed asynchronously. It needs to retain the token T that represents the client request for later use when sending the results of the request back to the originating client. Note that the token given to the client and the token given to the server represent the same logical request, but the tokens themselves do not have the same content.
5. After the server exit or its agent processes the request, the IXCSEND macro is invoked to send the results D back to the originating client. The token T identifies the client request to which the results belong. The results D can be at most 100 MB. XCF decodes the token T to determine where the results are to be sent.
6. XCF invokes IXCMSGOX to send the results to the originating system. If the message exceeds 60KB, XCF might suspend the responding thread until the IXCMSGOX service finishes sending the message.
7. XCF on the client system intercepts the server response. XCF locates the control blocks used to manage the original client request. If they are not found, the client request has timed out, and the server response is discarded. Otherwise XCF binds the response message to the client request and holds it until the results are gathered by the client, or the client request times out. If the client is waiting for the results, XCF notifies (that is, resumes) the client.
8. The client issues an IXCRECV request to gather the results of the request identified by token T (which was returned in step 1 when the request was initiated). The client provides an answer area A to contain metadata that describes the response. The data area D contains the response data sent by the server. In the event that the response has not yet arrived when the client attempts to gather it, the IXCRECV service suspends the client thread until the requested result becomes available. Alternatively, the client can avoid being suspended by using RECEIVE=STATUS to poll for message completion.
9. To process the IXCRECV request, XCF locates the control blocks being used to manage the request identified by token T. If not found, the request has timed out (or was cancelled) and no results are available. If the request is found, XCF determines whether the response has arrived. If it has not arrived, XCF either suspends the client thread or returns to it with the message status. If suspended, XCF resumes the thread when the results arrive, or when the request times out, or when the request is cancelled, whichever comes first. Assuming the results have arrived, XCF determines whether the answer area and data area provided by the caller are large enough. If not, XCF returns to the client indicating the required size to receive the results. If the output areas are large enough, XCF fills them with the appropriate data, discards the control blocks used to manage the request, and returns to the client.
10. Client inspects the metadata, the response data, or both, and processes the results of the request. Metadata includes a "response code" to describe the status of the request. For example, the response code might indicate that the request was processed by the server or it might indicate that the server failed. For more details on response code see “Response codes and the target receiver” on page 159.

For reference information about the IXCSRVR, IXCSEND, IXCRECV, and IXCREQ macros, see *z/OS MVS Programming: Sysplex Services Reference*.

Defining and starting a server

You use the IXCSRVR service to define a server to receive, process, and send responses to requests from clients. When defining a server, you must provide a server exit routine for XCF to call to process requests. When the server finishes processing the request, it uses the IXCSEND macro to send the response with the request results back to the client. The client then uses the IXCRECV macro to receive the response provided by the server.

Overview of IXCSRVR

Use IXCSRVR to define the server that is to process requests and send responses. The IXCSRVR macro completes a parameter with data and then calls the XCF service routine. When starting a server, the XCF service routine does not return to the caller until the server stops or fails; that is, the XCF service routine repeatedly calls the server exit routine as long as there are requests to process. Each time it is called, the exit routine of the server processes the request and, as needed, invokes the IXCSEND macro to send a response with the results of the processing. When the server exit routine returns, XCF either calls it again to process the next request or suspends the server task until there is a request to process.

Servers and server instances

You can start one or more servers for the same server name. Each of these servers is a **server instance**. The servers can be started in any address space on any system in the sysplex and each server instance has a server id that uniquely identifies the instance.

A server instance is in essence a task running the **XCF Server stub**. This routine runs in an infinite loop, suspending when there are no requests to process. When a request arrives, the work unit is resumed, and the XCF Server stub calls the server exit routine to process the request.

The term **server** can refer to the collection of all the server instances for a particular server name, or it can refer to a single server instance.

The first call to the server is an initialize server request. The server exit routine can perform any suitable initialization. When initialization is complete, XCF starts the XCF Server stub loop, and the server is eligible to process work.

Naming a server

Consider the following about the server name:

- Each server is identified by a server name
- Clients need to know the name of the server they want to process the request,
- If you are creating services you need to know the format of server names and how that format keeps different client/server applications from using the same server name and provides the flexibility for a particular client/server application to define different servers for different purposes.

The target server is usually identified by a combination of server name and system name. When a client uses the IXCSEND macro to send a request to a server, XCF sends the request to the indicated system and presents the request to a suitable instance of the named server. For details of how XCF uses specifications from the IXCSEND and IXCSRVR macros to determine whether a particular server instance is suitable for processing a request, see “Server selection criteria” on page 149.

Server id: Instead of specifying server name and system name, a client can use the server id to send the request to a particular instance of the server. You might use the server id if the client needs to continue communicating with the same server instance that processed the request that initiates the "conversation" between the client and server. You might also use the server id if the rules by which XCF selects a server to process a request are not appropriate for a particular client/server application. When a client identifies

the target server by its server id, the client is responsible for ensuring that the designated server is suitable for processing the request.

Obtaining information about servers

The XCF Server is an XCF-provided client/server application that provides a query service to let applications obtain information about the servers that exist in the sysplex.

An application might need to discover which servers have been defined, or it might need to determine the attributes of a server. You can use the IXCSEND macro to send a request to the XCF Server (on one, some, or all systems in the sysplex) and have XCF reply with the information about the servers that are defined. You can then process this data to determine which servers exist and by using the suggested application or component naming convention in the first section of the server name, you can find the relevant servers for your application.

Alternatively, a client/server application might define its own protocols and messages to provide information about its servers. For example, you might send a query request to your server to have it respond with relevant information. If the server does not exist, XCF acknowledges the request with a response code indicating "no receiver". However, be aware that, XCF delivers a request to but one instance of a server. If you have multiple server instances, some invention is required if your query request needed to provide information about them all. When an appropriately specified request is sent to the XCF Server, it will respond with information about all the server instances.

Server naming conventions

Server names have a specific format. The ixcysrvr_tName mapping, which is defined in the IXCYSRVR macro, defines the format.

A server name is a 32 byte string consisting of four 8-byte sections. By convention, the first section of the server name should be unique to the software vendor that provides the client/server application. For IBM software this typically is the component name or component prefix. For other software vendors, this is the unique prefix assigned by IBM to the vendor. This convention helps ensure that different client/server applications will not have name conflicts.

The first section of the server name cannot be blank. The remaining sections of the name, which can be blank, provide the flexibility that enables a given software vendor to create different server names for different client/server applications. For example, section 1 of the server name might be the vendor's prefix for every application, with section 2 being used to identify the particular application. As needed, you can define the remaining sections as needed to provide a variety of different servers for the relevant application.

These naming conventions help prevent different client/server applications from using the same server name. However, for a client/server application to work, the client needs to "know" the name of the server that is to process its request. Generally this can be accomplished through the implementation of the client and server processing as done together.

If you create a server that is to process requests from unknown clients, you must document the name of the server as well as the format of the message content so that others can use it. The XCF Server is an example of such a server. The IXCREQ macro is used to format storage with a request that is understood by the XCF Server. It also describes how to send the request to the XCF Server.

Specifying information about the server

When you invoke IXCSRVR to start a server instance, you specify the DESCRIPTION and INFO keywords to provide information about the server. You can also specify USERDATA to pass user-defined parameters to the server exit routine.

The DESCRIPTION parameter, which is required, contains a text string that describes the server and appears in the output of the DISPLAY XCF,SERVER command. The text should describe the role or function of the server so that installations and service personnel can understand its purpose.

The INFO keyword, which is optional, provides information about the server that is intended for use by the client/server application. This information is presented to the server exit routine whenever it is called by XCF and returned when a program uses the XCF Server to get information about the server. INFO might be used to describe the server in a way that allows client programs to discover the attributes of the server. If used in this way, the information might help clients determine whether the server is an appropriate target for its requests.

You can also specify USERDATA when starting the server. A copy of the USERDATA is presented to the server exit whenever it is called. You can use USERDATA to pass-user defined parameters to the server exit routine. USERDATA might, for example, contain the address of a storage area that the starter of the server and the server exit routine can use to communicate with each other.

Note that XCF does not call the server exit with a "termination" request. Because the server exit does not necessarily have an opportunity to clean up resources, you might need to have the program that invoked IXCSRVR REQTYPE=START be responsible for cleanup of server exit resources.

For example, the starter might acquire the resources needed by the server exit and then pass USERDATA to allow the server exit to locate those resources. Alternatively, in cases where the server exit acquires the resources, you might use USERDATA to pass the address of a storage area where the server exit is to anchor those resources, thus making them visible for cleanup processing by the program that started the server. This technique of anchoring resources in a storage area provided by the starter of the server is often used in cases where the server exit routine does not establish its own recovery. If the server exit fails, XCF recovery percolates to the starter's recovery, and the starter can then perform cleanup for the resources acquired by the server exit routine.

XCF Server stub routine and the server exit routine

The XCF Server stub routine gets control when you invoke IXCSRVR to start a server. The purpose of the stub is to wait for client requests to arrive and then call the user's **server exit routine** to process requests as they arrive.

After selecting a server instance to process the request, XCF constructs the **server exit parameter list** (SXPL) and calls the server exit routine to process the request. The SXPL provides information about the client request and identifies the storage location where the content of the client request message was stored.

XCF puts the address of the parameter list in register 1 and calls the server exit routine. Register R13 contains the same value as when the IXCSRVR macro was invoked to start the server (including AR13 if invoked in AR mode). The server exit routine receives control in the same environment as that of the IXCSRVR macro when it was invoked to start the server, except that it is now running under XCF recovery. In particular, the server exit routine receives control in the same addressing mode, the same ASC mode, with the same PSW key, and in the same state (supervisor or problem) as when the IXCSRVR macro was invoked to start the server.

For more information on coding a server exit routine, see “Coding a server exit routine” on page 167.

Server exit parameter list and processing

The server exit parameter list is mapped by ixcysrvr_tSXPL as part of the mapping macro IXCYSRVR. A **server code** within that parameter list identifies the specific type of processing that the server exit is to perform.

The SXPL has a base portion that is common to all server codes, and a variable portion that is unique to each particular server code. The server exit routine will need to process server codes for the following functions:

- Initializing the server
- Providing a work area for XCF
- Processing a request sent by a client

For programming details, see "User routine processing" in “Coding a server exit routine” on page 167.

Initialializing the server

When called to perform initialization, the server exit is to perform whatever initialization is appropriate. The function specific parameters mapped by ixcysrvr_tInitServer contain copies of the keyword values that were specified on the IXCSRVR REQTYPE=START invocation that started the server.

The first call of the server exit routine is always for the initialize function. XCF makes this call once. The processing needed for initialization depends entirely on the implementation of the server. Some implementations might have no processing to do, some might provide a work area to XCF, others might spawn additional tasks or perform other actions necessary to coordinate the server with other cooperating processes.

Providing a work area for XCF

In most cases, the server must provide a work area that XCF can use to store the content of a client request message before the server exit is called to process the request. If XCF does not have such a work area, or if the current work area is not large enough to hold the client message, XCF calls the server exit with a server code indicating that the server needs to provide a work area.

When called to provide a work area, the function specific parameters mapped by ixcysrvr_tGetWorkArea indicate the amount of storage that XCF requires. The server exit routine is expected to obtain the requested storage, update the SXPL_WAD field to indicate the location and size of the storage provided, and return to XCF. If the work area provided by the server is not accessible, XCF stops the server.

Note that on entry to the server exit, the SXPL_WAD field describes the work area that was last given to XCF. If there is an existing work area, the server exit most likely needs to dispose of the storage before updating the SXPL_WAD with information about a new work area.

XCF asks for a work area when it needs space to store the content of a request message. If the server exit does not provide the necessary storage, XCF cannot deliver the request. In such cases, XCF discards the request and sends an acknowledgment to the originator indicating that the request was not delivered because the server failed to provide the necessary work area.

If the server is unable to provide the necessary work area, it can also update the SXPL_RefusalCode with a non-zero value to have XCF send an acknowledgment back to the originator indicating that the server refused the request. In such cases a copy of the SXPL_RefusalCode is provided to the originator. Thus, the server has an opportunity describe the problem or influence how the originator is to proceed.

Every time the server exit is called (regardless of server code), the SXPL_WAD describes the work area that XCF is currently using. The server exit can

- Leave the SXPL_WAD unchanged to have XCF keep using the work area
- Update the SXPL_WAD to provide a new work area for XCF to use when delivering the next client request
- Update the SXPL_WAD to take the work area away from XCF.

Thus, the server can dynamically provide new work areas as needed. For some servers, it might be possible for XCF to use the same work area over again. For others, a new work area might be needed for each request.

For example, if the client messages are a known fixed size and the server exit processes each request synchronously, the server exit can provide a work area when called to perform its initialization. This work area can be used again repeatedly for each client request, and the server exit is never called with a "get work area" server code.

Alternatively, a server might have some requests that can be processed synchronously by the server exit routine but might have others that need to be processed asynchronously by some other work unit. If the asynchronous processing needs to access the content of the message, the server exit needs to take care to preserve that content. The server exit might take the work area away from XCF (or provide a different one) and pass the work area along to the asynchronous work unit. Or the server exit might make a separate copy of the message for the asynchronous work unit to use and allow XCF to continue using the current work area. If both the asynchronous work unit and XCF have access to the same work area, there

is a danger that XCF might store the content of the next client request in the work area and corrupt the copy of the message to be processed by the other work unit.

Processing a request sent by a client

The primary function of a server is to process client requests. Clients invoke the IXCSEND macro to send requests to the server. The content of the client request message, the processing performed by the server, and the content of the server response (if any) are determined by the needs of the client/server application. The server processes the request and sends a response. Understand that the request might be processed by the server exit routine, or the server exit might arrange for some other agent to process the request, or the request might be processed cooperatively by the server exit and various agents.

Responses are sent to provide the results of the processing by a server. In practice, a client might expect a reply from the server. If a reply is expected, the reply might be a message containing data, or it might be a simple acknowledgment indicating whether the request has been processed.

When a server exit is called to process a request, the function specific parameters mapped by ixcysrvr_tRequest contain copies of most of the keyword values that are specified on the IXCSEND macro used by the client to send the request. The function specific parameters also include a "message descriptor" that identifies the size and location of the content of the request message. The server exit routine uses the SXPL parameters and the message content to determine what it needs to do.

Note that the message descriptor contains a flag (md_ExpectReply) to indicate whether the client is expecting a reply. If a reply is expected, the field md_RespToken contains a token that the server (or its agent) must provide as input when it invokes IXCSEND SENDTO=ORIGINATOR to send the expected response. If the server exit does not send the response before it returns to XCF, a copy of the token must be preserved for later use with IXCSEND. Processing that occurs asynchronously to the server exit routine cannot rely on the content of SXPL to remain intact because XCF might re-use the storage containing the SXPL upon return from the server exit. Thus, before it returns to XCF the server exit needs to preserve a copy of the response token, as well as any other information that might be needed from the SXPL, so that the data is available for use by the asynchronous work unit that is to send the results.

The manner in which the request is processed is up to the provider of the server exit routine. If the client expects a reply, a response will be sent. In most cases, the responder invokes IXCSEND SENDTO=ORIGINATOR to send the reply. The originating client obtains the reply by invoking IXCRECV.

When invoking IXCSEND to send a reply, the responder can specify various keywords to provide different kinds of data to the client. The design of the client/server application determines whether any particular keyword is relevant. Specifically, the following keywords might be of interest:

- RESPRETCODE and RESPRSNCODE
- SUPPLIEDLEVEL and SUPPORTSLEVEL
- MSGCNTL
- MSGDATA or MSGDESC

For information about sending a response, see “Sending a response to a client” on page 150. For details on all IXCSEND keywords, see *z/OS MVS Programming: Sysplex Services Reference*.

Acknowledgment of a request

Most servers will likely use IXCSEND to send a message containing the results of its processing back to the client. But for some applications, perhaps depending on the particular request,the reply could be an acknowledgment indicating whether or not the request was processed. You can use IXCSEND to explicitly send a simple acknowledgment. Using IXCSEND allows the acknowledgment to include a return and reason code. An alternative approach is to have the server exit routine update the SXPL to have XCF send the acknowledgment. Setting either the SXPL_ResultCode or the SXPL_RefusalCode to a nonzero value causes XCF to send an acknowledgment on behalf of the server. For a given request, the server exit can set either the result code or the refusal code, not both. XCF stops the server if both codes are set to a nonzero value.

If the server exit arranges for the request to be processed by some other work unit, it is the responsibility of that other work unit to invoke IXCSEND to send the result of the request back to the originating client. In this case, the SXPL "result code" and "refusal code" is not used. If the client expects either some sort of data in response to its request, or some sort of acknowledgment, it is up to the work unit that processes the request to send the response or acknowledgment by explicitly invoking IXCSEND.

Note that for any given request, you either use the technique of setting the result code or refusal code to have XCF send an acknowledgment, or you invoke IXCSEND to send the acknowledgment or results. If both techniques are used, there are race conditions to consider. The client receives at most one of the responses, either the XCF acknowledgment or the response sent using IXCSEND. In general, it is unpredictable which will be delivered to the originator.

Response binds

The server exit can update the SXPL_RespBind field to indicate who has responsibility for sending the response (a response bind). Response binds are relevant in cases where the server exit routine arranges for some other work unit to send the results of the request to the client. If the entity responsible for sending the response fails before doing so, the response bind allows XCF to recognize that the failed entity is no longer capable of sending the response. In such cases, XCF sends an acknowledgment to the originator indicating that the responder terminated before it could send the results. Without this acknowledgment, the client request continues to wait for the expected reply until the request times out. See “Defining response binds” on page 142.

Server exit and work area storage

The storage areas passed to the server exit routine are valid for use only when the server exit routine is called by XCF. When the server exit returns to XCF, the client/server application must assume that the storage areas are in flux. When the server exit returns to XCF, the storage containing the SXPL and the work area storage provided by the server are subject to being updated by XCF as part of its preparations for calling the server exit to process another request.

If the server exit processes the request and invokes IXCSEND to send the response before returning to XCF, there are no concerns because the contents of the storage areas remain intact while the server exit is in control. However, if the server exit arranges for the request to be processed asynchronously by a third party, it needs to take care about using the work area that contains the text of the client message.

When the server exit returns to XCF, XCF assumes it can use the work area for the next request to be presented to the server exit. If the server exit wants the third party routine to access the work area storage for the message content, the server exit needs to update the SXPL_WAD to prevent XCF from using the storage to store the content of the message for the next request.

The exit must either obtain a new work area and update SXPL_WAD to point to it, or zero out SXPL_WAD to "take it away" from XCF. (In that case, XCF must call the server exit with a "get work area" request before it can deliver the next request.) Otherwise, there is a possibility that XCF stores a new message in the work area while the third party routine is accessing the storage, which can lead to unexpected results. Depending on the timing and the third party processing, a request might be lost, processed multiple times, or fail because of corrupted message content.

Alternatively, the server exit can copy the text of the client message to some other storage area, in which case it can leave the work area intact for XCF to use for the next request.

In cases where a response is expected, the md_RespToken field contains the response token to use when you invoke IXCSEND to send the response. Because XCF reuses the SXPL storage when it calls the server exit routine to process each request, if the server exit arranges for another work unit to process the request, the md_RespToken (as well as any other data in the SXPL that the third party needs to know) can be updated with new information. Thus, the server exit routine must take care to preserve whatever SXPL content is needed by the third party to process the request. For example, you can copy the relevant data to another storage area.

Defining feature strings

Different levels of client code and server code might be running in the sysplex. A server might need to support some particular "feature" in order to process a client request. A "feature" is whatever the client and server decide it is, but typically a feature is a function or service, or perhaps the ability to support a particular protocol, or the ability to interpret new parameters or produce new results (as compared to a server that does not have the feature.)

When a server is defined, it can specify a "feature string" to indicate the features it supports. When a client sends a request, it can similarly provide a feature string to indicate what features are required to process the request. If the client does not explicitly target the request to a specific server instance, XCF makes sure that the request is presented to a server that supports the features requested by the client.

You can define feature strings on the IXCSRVR macro using the FEATURES keyword. Using feature strings allows a client to ensure that the client request is presented to a server instance with the necessary function to process the request. When determining whether the server is suitable for the request, XCF compares the features requested by a client to the features supported by the server. A client/server application might use feature strings instead of, or in addition to, client and server levels. See “Specifying client/server compatibility levels” on page 141.

Feature strings can be thought of as a set of flags where each flag corresponds to some feature, function, or protocol that a server supports. With the initial release of a client/server application, no flags need be set. As new releases of the server are installed, or as maintenance is applied, a server supports new functions, features, or protocols. When a server is started, it sets one or more feature flags to indicate that it offers the relevant support.

A feature string is mapped by ixcysrvr_tFeatures, which is defined by the IXCYSRVR mapping macro. ixcysrvr_tFeatures maps both the feature level and the feature flags.

When a client sends a request to a server, it specifies the CRITERIA keyword to indicate which features (and server levels) the server must support in order to process the request. A server is eligible to process the request if it supports all of the features specified by the client. A server supports all of the features requested by the client if either of the following is true:

- The feature level requested by the client is less than the feature level supported by the server.
- The feature level requested by the client equals the feature level supported by the server, and every non-zero feature flag requested by the client is also nonzero for the server.

See “Server selection criteria” on page 149 and “Client/server compatibility” on page 164.

Specifying client/server compatibility levels

You can use the following IXCSRVR keywords to develop protocols that help clients and servers function compatibly with mixed levels of support and functional capabilities:

- MINLEVEL and MAXLEVEL
- MINCLIENT and MAXCLIENT

These specifications provide the criteria that XCF uses to determine whether a server instance is suitable for processing a given request. A request is not presented to a particular server instance if the instance is not suitable for the request.

MINLEVEL and MAXLEVEL keywords on IXCSRVR define the range of **server levels** that are supported by the server. (The interpretation of "levels" is for the client/server application to define.) MINCLIENT and MAXLCIENT define the range of **client levels** whose requests the server is willing to accept.

When a client invokes IXCSEND to send a request to a server, it specifies the CLIENTLEVEL to indicate the "level" of the client, and the CRITERIA keyword (as needed) to indicate what range of levels (and features) the target server must support. A server is eligible to process the request if the client level is within the range of client levels that are acceptable to the server, and if the range of server levels it supports intersects the range of server levels requested by the client. For information about these IXCSEND keywords, see “Server selection criteria” on page 149.

For information about client/server compatibility, see “Client/server compatibility” on page 164.

Defining response binds

When a client expects a response to its request, the server is expected to provide a response to the sender. The response might, for example, always be sent by the server exit routine. Alternatively, the server exit routine might arrange for some other work unit to send the response. Depending on the implementation of the server, the responsibility for sending the response might vary depending on the type of request.

Clients that invoke IXCRECV to get the results of a request are blocked (suspended) until all the expected responses arrive, or a timeout value expires (for example, the RESPTIME keyword on the IXCSEND macro). Thus, if the server or its agent experiences an unrecoverable failure before it sends the expected response, the client remains blocked until the timeout value expires. To mitigate this problem, the server can establish a "response bind" for the request. A response bind indicates who is responsible for sending the response. XCF monitors the designated entity and sends an acknowledgment indicating that the request has failed if the responder terminates without sending a response.

When a server is started, you can use IXCSRVR RESPBIND to establish a default "response bind" to indicate who has responsibility for sending the response to a request. For example, you might indicate that the server exit routine is responsible for sending the response.

Suppose that XCF determines that the server exit failed while processing a request that expects a response. XCF cancels the expected response on behalf of the failed server exit. That is, XCF indicates to the requester that the anticipated response was not provided because the server failed while processing the request. Thus, the requester is not forced to wait the full timeout value before it is allowed to resume processing.

The response bind can be assigned to one of the following:

- The server exit routine. XCF cancels the response if the server instance ends.
- The server address space. XCF cancels the response if the address space ends. Use this option if some work unit running in the server address space (other than the server task) is responsible for sending the response.
- The system on which the server instance resides. XCF cancels the response only if the system ends. Use this option if some work unit running in an address space (other than the server address space) is responsible for sending the response.

Responsibility for sending the response might vary on a request-by-request basis though this depends on the implementation of the server. Thus XCF provides the ability for the server to override the default RESPBIND specification that is set when the server is started. Before returning to XCF, the server exit can set the SXPL_RespBind field to change the default response bind. Note that updating SXPL_RespBind to set a new response bind for a particular request does not become effective until XCF successfully completes its backend processing of the request after the server exit returns.

Stopping servers

A server exit routine can set the SXPL_StopCode field in the server exit parameter list to a valid non-zero value to cause XCF to stop running the server exit stub loop. Upon return from the server exit routine, XCF ends the server stub loop and returns to the caller that invoked IXCSRVR REQTYPE=START to start the server instance. IXCSRVR return and reason codes for the START request are determined by the non-zero "stop code" when the server exit stops in this manner (the macro IXCYSRVR describes the correlation).

Alternatively, you can invoke IXCSRVR REQTYPE=STOP to initiate a stop of a server. Specify the SERVERID keyword to stop a particular instance of the server. Specify SERVER to have XCF stop one or more instances of the named server. Note that stop processing occurs asynchronously to the IXCSRVR request. Thus, there might still be instances of the server running upon return from the stop request. Also note that an IXCSRVR REQTYPE=STOP request must be issued from the system on which the server resides.

When invoking IXCSRVR to stop a server, the server can be stopped in one of two ways:

- Normal: the server is allowed to finish processing any suitable work that was being processed or queued for processing before the arrival and acceptance of the stop request.
- Immediate: the server is allowed to finish the request that is currently being processed (if any). However, the server does not process any queued work that might have been suitable for the server.

When the last instance of a server stops, XCF deletes the server definition. If any work remains pending when the last instance is stopped, the work is cancelled and discarded in an appropriate fashion. For example, if the cancelled work item was a client request that was expecting a reply, the request is cancelled and acknowledged with a "no receiver" response code.

You can only stop servers that are instantiated on the system where the IXCSRVR STOP macro is invoked. To stop servers on other systems, the client/server application must create its own mechanism for performing the stop. For example, one might send the server a request to tell it to shut down or to invoke IXCSRVR REQTYPE=STOP to stop the designated servers.

In order for a server to stop, XCF must stop running the server stub loop. If the server exit is processing a request, of if the server task is hung, or if the server has a long queue of pending requests (in the case of a normal stop), there might be significant delay before the server stops.

When a server is stopped, the invoker of the IXCSRVR REQTYPE=START request receives control back with a return and reason code indicating how the server was stopped. Return code 0 indicates the server exit set a stop code in the SXPL to indicate that it was stopping normally. Return code 4 is used if XCF stops the server because of a stop request, or if the server exit set a stop code indicating that it is stopping because of an error. Either case has its own reason code.

XCF also stops the server and returns to the IXCSRVR REQTYPE=START requester if the server exit violates the XCF interface. For example, the server exit might update the SXPL with invalid data. If so, XCF returns to the starter with return code 8 and a reason to indicate the particular failure.

XCF might also percolate errors to the recovery routine established by the invoker of the IXCSRVR REQTYPE=START request.

Note: The server exit routine can invoke the IXCSRVR macro to stop itself if it chooses. Setting a stop code in the SXPL is the same as requesting an immediate stop in that the server instance will not be called to process any pending work. However, note that the different stop techniques cause a different return code to be presented to the invoker of IXCSRVR REQTYPE=START. If the server wants to finish processing any currently queued work and then stop, it needs to invoke IXCSRVR REQTYPE=STOP MODE=NORMAL.

Startup timing issues

XCF does not call the server exit with a "termination" request. Thus the server exit does not necessarily have an opportunity to clean up resources when it is stopped. In the case of a normal stop where the server exit is being called to finish pending work, the SXPL_StopPending flag is set. However, becsause the server exit is not called if no work is pending, the server exit might never get a chance to inspect the flag. Thus, the program that invoked IXCSRVR REQTYPE=START to start the server instance must accomplish the cleanup of resources acquired by the server exit. When a client sends a request to a server, it might not necessarily know whether the server is up. The request might arrive before any instances of the server have started. In such cases, the request is discarded and XCF sends back an acknowledgment to indicate that there is "no receiver".

To determine if a specific server is active, the client can periodically poll by sending requests to the server or by sending requests to the XCF Server that allows you to obtain information about servers that are already defined. (See the IXCREQ macro in *z/OS MVS Programming: Sysplex Services Reference*.) Alternatively, your application can be structured so that both the client and the server are running servers. The client can start a server whose primary role is to receive messages that a server is up and running from the "real" servers. After such a message is received from the "real" server, the client can start sending requests.

Summary of IXCSRVR functions

Table 10 on page 144 summarizes the functions and keywords for IXCSRVR:

Table 10. Functions and keywords for IXCSRVR	
IXCSRVR function	**IXCSRVR keyword**
Starting the server	REQTYPE=START SERVER to specify the name of the server you want to start. DESCRIPTION to describe the purpose of the server. INFO to contain additional information about the server. USERDATA to pass user defined parameters. SERVEREXIT to define the server exit routine. FEATURES to specify feature strings. MINLEVEL, MAXLEVEL, MINCLIENT, and MAXCLIENT to specify server/client levels FDI to specify the number of seconds that the server can appear to be unresponsive before the system considers it to have failed. RESPBIND to specify the default response bind option that XCF is to establish for the server when processing a response.
Stopping the server	REQTYPE=STOP Identify the server or server instance you want to stop: • SERVERID for a specific server instance • SERVER for one of more instances of the named server MODE to indicate whether the server is to be allowed to finish pending work.

Using the IXCSEND macro

The IXCSEND interface is used to send messages within the sysplex. A client uses IXCSEND to send a message containing a request to one or more servers for processing. (For information on how a server is defined to XCF, see "Defining and starting a server" on page 135.) A server uses IXCSEND to send a message containing the results of its processing back to the client. The client uses the IXCRECV interface to receive those results. (For information on using IXCRECV, see "Using the IXCRECV macro" on page 153.)

Overview of IXCSEND

The following summarizes the use of IXCSEND in a typical client/server message exchange:

1. A client invokes IXCSEND SENDTO=SERVER to send a request to one or more instances of a server.
2. On each target system, XCF presents a copy of the request to a suitable server instance for processing.
3. A server processes the request and responds by invoking IXCSEND with SENDTO=ORIGINATOR to send the request results back to the client.

For information on how the client invokes IXCRECV to receive the results of the request that it sends, see “Using the IXCRECV macro” on page 153. The invoker of IXCSEND can specify the USERDATA keyword to pass user defined data to the work unit that invokes IXCRECV.

All senders must indicate the following on IXCSEND:

- Content of the message. See “Content of the message ” on page 145.
- Identity of the sender. See “Identity of the sender” on page 146.
- Timeout values. See “Time out values” on page 146.
- Receive bind. See “Receive bind” on page 147.

In addition, there are considerations and keywords unique to the type of message being sent.

- For a discussion of considerations unique to the use of SENDTO=SERVER by the client, see “Sending a request to a server” on page 147.
- For a discussion of considerations unique to the use of SENDTO=ORIGINATOR by the server, see “Sending a response to a client” on page 150.

RETMSGTOKEN specifies a storage area where XCF is to store a token that identifies the message. The token is required for use with other XCF client/server interfaces and the XCF message control service (IXCMSGC)

For details on specific IXCSEND keywords, see *z/OS MVS Programming: Sysplex Services Reference*.

Content of the message

The content of the message is determined by the client/server application. The format of the message needs to be well defined. The client needs to format a request message that the server can understand and, in turn, the server needs to format a response message that the client can understand.

When designing the content of the client/server messages, give consideration to the possibility that the content might need to change over time. For example:

- New requests might be created
- Existing requests might be updated with new options.
- Response data provided by servers can be updated with new or changed data.

Thus, include information in the message content that enables the sender to specify, and the receiver to determine, the content included in the message. XCF also provides some support intended to allow multiple versions of a client/server application to coexist in the sysplex.

A client/server message typically has two components:

- Message control data
- Message content or message "payload"

The intended purpose of message control data is to enable the sender to provide metadata that can be used to describe the content of the message. The recipient might use this metadata to determine how the message is to be processed.

The message content, which is sometimes referred to as the message payload, is whatever data the application needs to send.

Typically one thinks of the message payload as being "the" message. When invoking IXCSEND, you use the MSGCNTL keyword to provide message control data. The data to be sent as the payload of the message is specified with either the MSGDATA keyword or the MSGDESC keyword.

- MSGDATA is used when the message data resides in a single contiguous storage area, in which case the MSGLEN keyword indicates the length of the message.
- MSGDESC is used when the sender wants XCF to gather the message data from several different storage areas. The MSGDESC keyword names storage that contains an array of data descriptors. The keyword #MSGDESC indicates how many entries are in the array.

Each data descriptor, which is defined by the ixcysrvr_tDataDescriptor mapping in the IXCYSRVR macro, identifies the location and length of message data that is to be included in the payload. When the message is presented to the target, the message data consists of the various pieces of the message concatenated end to end in the order specified in the MSGDESC array.

In the simplest case, each entry of the array only contains a data descriptor. However, the data descriptor can be embedded within array entries that contain other data as well. In that case, use the LENMDENTRY keyword to indicate the number of bytes in each array entry. XCF uses this length to advance to each successive data descriptor. In some cases, the message control data might contain all the data that needs to be sent.

Specify the NODATA keyword to indicate that the message has no payload. Message control data is always presented to the target. If the MSGCNTL keyword is not coded, every byte of the message control data is zero.

MSGSTGKEY specifies the storage key that XCF is to use when fetching the message data from the indicated storage areas. If not specified, XCF uses the key of the caller.

MSGID is a user defined value that is intended to be unique for each message. The MSGID provides a "tag" for the message that can be used to correlate the processing performed by XCF with processing performed by the client/server application. The MSGID is presented to the target as well. This value can be useful when performing problem diagnosis.

Identity of the sender

The sender of a message specifies the SENDER and SENDERID keywords to identify itself. Copies of the specified values are presented to the target.

The SENDER specification is a text string intended to identify the sender of the message, either the client or the server. In cases where a server is sending a response message, consider specifying the server name. In cases where a client is sending a request, consider specifying text that allows installations and service personnel to identify the client application that sent the message. For example, include the job name as part of the sender name.

The SENDERID is intended to be a token that can uniquely identify the sender. When a server sends a response, consider specifying the server id of the server. When a client sends a request, consider specifying a value that uniquely identifies the client. For example, include the TTOKEN of the sending task as part of the SENDERID.

Time out values

When sending a message, you can specify the following keywords to assign timeout values to various phases of the processing:

- SENDTIME
- RESPTIME
- HOLDTIME

SENDTIME indicates how long the work unit that invokes IXCSEND can be suspended to allow XCF to finish accessing the user storage that contains the message. Whenever IXCSEND returns to the caller, the user is free to reuse or dispose of the storage areas containing the message. To achieve this condition, XCF will either copy the user message into its own buffer or suspend the calling work unit XCF is done accessing the storage containing the user message. The SENDTIME timeout value applies when XCF elects to suspend the sender. If the SENDTIME timeout expires, XCF stops sending the message, in which case some of the targets will not receive the message.

RESPTIME indicates how long the sender is willing to wait for the message to complete:

- If responses or acknowledgments are expected, the message completes when all of the expected responses and acknowledgments have arrived. In this case, the RESPTIME timeout value needs to allow enough time for the message to be sent to the target systems, time for the message to be processed, and time for the response to be sent and delivered to the local system. Responses and

acknowledgments that arrive after the RESPTIME timeout expires are discarded by XCF. An attempt to send a response after the RESPTIME timeout expires might be rejected.

- If responses and acknowledgments are not expected, the message completes when XCF has initiated the send of the message to each of the targets. In this case, the RESPTIME timeout value needs to allow time for XCF to send the message.

  When the RESPTIME timeout value expires, XCF stops trying to send any messages that had not yet been initiated. Thus, some of the targets might not receive the message. The fact that XCF initiated the send of the message does not imply that the message has been delivered to the target.

  The RESPTIME timeout value limits the amount of time that the invoker of IXCRECV can be blocked (suspended) waiting for the message to complete. If the RESPTIME timeout value expires, XCF resumes the blocked receiver and presents the results as of that moment. The metadata that describes the results will indicate whether messages were sent and if applicable, whether responses and acknowledgments were received.

HOLDTIME indicates how long XCF is to hold the results of a message after it completes. The HOLDTIME timeout value needs to allow however much time is needed for the user to invoke IXCRECV to retrieve the results. For example, if the user sends the message, performs other work, and then invokes IXCRECV to get the results, the HOLDTIME value needs to include time for the user to complete the other work. HOLDTIME should allow time to overcome system issues that might prevent the receiving unit of work from being dispatched.

If the HOLDTIME timeout expires, the message and its results, including any responses that might have been sent, are discarded by XCF. At that point, an IXCRECV request is rejected with a reason code indicating that the message no longer exists.

For details, see “Using the IXCRECV macro” on page 153.

Receive bind

RECVBIND is an optional specification used to identify the entity responsible for invoking IXCRECV to receive the results of the message. If the indicated entity ends, XCF discards the message and any associated responses. The RECVBIND specification helps XCF ensure that system resources are recovered in a timely manner.

The receive bind can be assigned to a task, an address space, or the system. The entity responsible for invoking IXCRECV could end before IXCSEND finishes sending the message. In such cases, XCF allows the send process to continue. When send processing is complete, XCF detect that the receiver ended. XCF then discards the message and any associated responses.

Sending a request to a server

A client uses IXCSEND SENDTO=SERVER to send a request message to a server for processing. When using IXCSEND to send a message, you must provide the message content, identify the sender, specify timeout values, and identify the entity responsible for invoking IXCRECV to receive the results. When sending a request to a server, you must also indicate the following:

- The function to be performed. See “FUNCTION keyword of IXCSEND” on page 148.
- A description of the request. See “DESCRIPTION keyword of IXCSEND” on page 148.
- The identity of the target server. See “Identifying the target server” on page 148.
- Server selection criteria. See “Server selection criteria” on page 149.
- Whether a reply is expected. See “Using EXPECTREPLY and RESPONSELEVEL” on page 150.
- The desired level of the response data. See “Specify additional response information” on page 151.

FUNCTION keyword of IXCSEND

The FUNCTION keyword is used to specify eight bytes of user defined data to be presented to the target server. The intended purpose of this data is to allow the client to indicate the function that the target server is to perform.

The designer of the client/server application has several different techniques that can be used to indicate what function the server is to perform. Some applications might make use of FUNCTION data. Others might make use of the message control data or the message content to indicate the function that the server is to perform for the client.

Some applications might make use of various combinations of FUNCTION, message control data, and message content to specify the desired function. Still others might elect to dedicate different servers to performing specific functions, in which case the function to be performed is implicitly understood by virtue of which server name was specified as the target for the request. The client needs to specify the function in whatever form the server.

DESCRIPTION keyword of IXCSEND

The DESCRIPTION keyword is used to specify 32 bytes of text that describes the request. The intended purpose is to provide information that will help installations and service personnel understand how the message relates to the function of the client/server applications. The description might indicate

- Indicate the function, service, purpose, or role of the client.
- Indicate the application with which the client is associated.
- Describe the request or the reason the request is being issued. The description is presented to the target server and can also appear in XCF display output and dump reports.

Identifying the target server

When a client sends a request to a server, it must indicate which server is to receive the request. You can specify either the SERVER keyword to identify the target server by name or the SERVERID keyword to identify a specific server instance as the target. Most clients use SERVER to identify the target.

When the target server is identified by name, XCF selects an instance of the server to process the request. When the target server is identified by its server id, the client designates which instance of the server is to process the request.

- The SERVER keyword provides the name of the server that is to process the request.

  When specifying SERVER, you must also specify SYSTEMS to indicate the set of systems to which the request is to be sent. You can specify SYSTEMS=ALL to send the request to all systems in the sysplex, SYSTEMS=OTHER to send the request to all systems in the sysplex excluding the system on which the sender is running, or SYSTEMS=LOCAL to send the request only to the system on which the sender is running. You can also specify SYSTEMS=NAME or SYSTEMS=SYSID to identify the set of target systems, either by system name or XCF system id, respectively. XCF sends the request to each of the indicated systems.

  On each of the target systems, XCF chooses an instance of the server that is suitable for the request and calls the server exit routine to process the request. For the rules that XCF uses to choose a suitable server instance, see "Server selection criteria" on page 149.
- The SERVERID provides a token that uniquely identifies the particular server instance that is to process the request. XCF sends the request to the system on which the designated server instance was started. On each of the target systems, XCF calls the server exit routine of the designated server instance to process the request. It is up to the client to ensure that the designated server instance is suitable for processing the request.

In general, most clients specify the SERVER keyword to identify the target server by name. However, there might be cases where the client might use SERVERID to send the request to a specific server instance. For example, the client/server application might require the client to communicate with a particular server instance. The client might initially specify SERVER to allow XCF to choose some instance of the server to

process the request. The server instance chosen by XCF might allocate resources on behalf of that client. When it sends its response, the server instance includes its server id in the response message, either as part of the message data, or as part of the message control data, or as the value specified for the SENDERID keyword when invoking IXCSEND. The client then specifies SERVERID with that server id when it invokes IXCSEND to send subsequent requests to ensure that the requests get processed by the server instance that allocated the resources on its behalf.

As another example, a client/server application might need to override the rules that XCF uses to choose a suitable server instance. The application might elect to use the server selection criteria in a way that differs from how XCF uses them. The application might need to consider server information specified via the INFO keyword (see IXCSRVR) when determining a suitable server instance. In such cases, the client might need to obtain information about the server before it sends any requests.

If so, the client can send a query request to the XCF Server to get information about the servers of interest. The information returned by XCF includes the server id and attributes of each of the reported server instances. For information on how to interact with the XCF Server to get information about the servers that have been defined in the sysplex, see "Using the IXCREQ macro" on page 161.

Server selection criteria

When sending a request, the client can specify criteria that influences which server instance gets selected to process the request. When the SERVER keyword is specified to identify the target server by name, the CLIENTLEVEL and CRITERIA keyword values, in conjunction with attributes specified by the server when it was started, are used by XCF to select the application that is operating. The CRITERIA keyword identifies a storage area that describes the range of server levels and the set of features that the target server must support in order to process the request. The storage area identified by CRITERIA is mapped by ixcysrvr_tCriteria, which is declared in the IXCYSRVR mapping macro.

When IXCSRVR is invoked to start a server instance, the MINLEVEL and MAXLEVEL keywords determine the range of server levels that the server supports. The FEATURES keyword indicates the set of features that the server supports. The MINCLIENT and MAXCLIENT keywords indicate the range of client levels whose requests the server is willing to accept.

The designer of the client/server application determines how the various levels and features are to be interpreted. However, XCF has very specific rules as to how levels and features are to used to select a server instance that is suitable for processing the request. These rules apply when the target server is identified by name (SERVER). An instance of the named server is suitable for processing the request if (1) The client level is within the range of client levels that are acceptable to the server, (2) the range of server levels specified by the client intersect the range of levels supported by the server, and (3) the server supports all the features requested by the client.

A feature string is mapped by ixcysrvr_tFeatures, which is defined by the IXCYSRVR mapping macro. ixcysrvr_tFeatures maps both the feature level and the feature flags. A server supports all of the features requested by the client if either of the following are true:

- The feature level requested by the client is less than the feature level supported by the server.
- The feature level requested by the client equals the feature level supported by the server, and every nonzero feature flag requested by the client is also nonzero for the server.

The intended purpose of these rules is to provide a protocol that allows different versions of a client/server application to run compatibly at the same time in the same sysplex. With careful use of levels and features a client/server application can ensure that XCF routes client requests to an appropriate instance of the server. An up-level server can in effect refuse to process requests sent by a downlevel client. An up-level client can in effect prevent its request from being processed by a downlevel server. See "Client/server compatibility" on page 164.

Although the server selection rules for client level, server level, and server features are applied only when the target server is identified by name, the client can still specify the CLIENTLEVEL and CRITERIA keywords on IXCSEND when the target server is identified by server id. XCF always selects the designated server instance to process the request without regard to levels and features. However, the values

specified by the client will be presented to the server instance. There might be cases where the server exit routine might make use of this information to determine whether and how it is to process the request.

Using EXPECTREPLY and RESPONSELEVEL

You can indicate that you expect a reply to the message that you send through EXPECTREPLY=YES. You can only specify this keyword when you specify SENDTO=SERVER. Note that an XCF acknowledgment might be sent as well.

Time out values can affect when you receive an acknowledgement or reply. If an XCF acknowledgment is received before RESPTIME expires, the information returned by IXCRECV provides the status of the delivery. Thus the status information returned by the corresponding IXCRECV macro might indicate that the message was delivered to the target even though a response was not received from the target before the RESPTIME expired.

When you specify EXPECTREPLY=YES, the RESPONSELEVEL keyword allows the client to request the "level" of data it wants to have the server send back. In that way a client has a way to specify which level of data it "understands" so that the server does not send back anything that is not compatible. The content and interpretation of the response level is defined by the receiver. The response level is made available when the message is presented to the target server. For example, when presenting a client request to a server exit, the server exit parameter list contains the level of response data that the sender wants the recipient to send.

Sending a response to a client

After processing a request, the server uses IXCSEND SENDTO=ORIGINATOR to send a response message back to the client. When using IXCSEND to send a message, you must provide the message content, identify the sender, specify timeout values, and identify the entity responsible for invoking IXCRECV to receive the results. When sending a response to a client request, you also:

- Identify the target request. See “Identify the target request” on page 150.
- Optionally, specify additional response information. See “Specify additional response information” on page 151

If a server sends more than one response to a request, at most, one of the responses is presented to the client. The remaining responses are either rejected at the time of the send or discarded (by either the sending system or the target system). The XCF client/server interface does not define which of the responses is to be delivered to the client.

If the client request exceeds ts RESPTIME timeout value or is otherwise completed or cancelled before the results arrive on the client system, XCF discards the server response message. In some cases, XCF rejects the server IXCSEND request outright if the response is sent after the client RESPTIME timeout value has expired.

Identify the target request

Use the RESPTOKEN keyword to specify a token that represents the client request to which the response is being sent. XCF uses the token to determine where the response needs to be sent. When the server exit routine is presented with the client request, the server exit parameter list contains a message descriptor (SXPLRQ_MsgDesc) that provides information about the client request. The message descriptor is mapped by ixcysrvr_tMsgDescriptor, which is declared in the IXCYSRVR mapping macro. Within the message descriptor, the field md_RespToken contains the token to be used for the RESPTOKEN keyword when invoking IXCSEND to send the response.

Note that md_RespToken is valid for use only if the client requests that the server is to send a reply. The md_ExpectReply field indicates whether the client specified EXPECTREPLY=YES to request a response. If the client expects a response, completion of the client request will remain pending until a response or acknowledgment is received. In particular, failure to send a response could cause the client to remain blocked in IXCRECV processing until the RESPTIME timeout value of the client request expires.

Specify additional response information

The server can optionally include additional information with the response message. For some client/server applications, this information could constitute the entire response message. The following keywords can be specified:

- RESPRETCODE and RESPRSNCODE
- SUPPLIEDLEVEL and SUPPORTSLEVEL

The values specified for these keywords are presented to the client when it invokes IXCRECV to receive the response. The interpretation of these values is determined by the client/server application. Use the RESPRETCODE and RESPRSNCODE keywords to specify a return and reason code indicating the result of the processing of the request. If not specified, the values presented to the client will be zero.

The SUPPORTSLEVEL and SUPPLIEDLEVEL keywords are intended for use when responding to requests where the client might have specified the RESPONSELEVEL keyword when sending the request. The intended purpose of these keywords is to enable the client/server application to easily support a protocol that allows different levels of data to be returned by the server.

For example, a client could specify RESPONSELEVEL=0 to request summary information and RESPONSELEVEL=1 to request detailed information. When sending the response, the server can specify SUPPLIEDLEVEL to indicate the level of data that it provided in the response and SUPPORTSLEVEL to indicate the highest level of data that it could have provided.

Summary of IXCSEND functions

Table 11 on page 151 summarizes the functions and keywords for IXCSEND.

Table 11. Functions and keywords for IXCSEND

IXCSEND function	**IXCSEND keyword**
Information about the message content	NODATA to indicate that no message data is associated with the IXCSEND. MSGDATA to indicate that a single contiguous buffer variable contains the message data to be delivered to the receiver of the IXCSEND. You specify the following: • MSGLEN to specify the length in bytes of the message. MSGDESC to indicate that a table with one or more data descriptors each identifying the location and length of a piece of the message data to be delivered to target. You can specify the following: • LENMDENTRY to indicate the length in bytes of each entry in the message descriptor table. MSGCNTL to describe control data for the message. MSGSTGKEY to specify the storage key to be used when fetching the message data. MSGID to specify a message id that identifies this particular message. RETMSGTOKEN to specify a storage area to contain a token that identifies message for later use with IXCRECV.

Table 11. Functions and keywords for IXCSEND (continued)

IXCSEND function	IXCSEND keyword
Information about the sender	SENDER to indicate the name of the sender. When sending a request to a server for processing, the client is the sender. When sending a response containing request results back to the client, the server is the sender. SENDERID to indicate the id of the sender. The sender id is presented to the target of the message.
Time out values	Specify the following values for the message data: • SENDTIME for the amount of time XCF is allowed to suspend the calling unit of work while it accesses the message storage. • RESPTIME for the amount of time the sender is willing to wait for responses to arrive for the message (either a reply from the server or an acknowledgement from XCF) • HOLDTIME for the amount of time XCF is to hold the results for processing by a subsequent IXCRECV request (information about the IXCSEND request, information about one or more targets of the message, or replies from servers or XCF acknowledgments)
Receive bind information	RECVBIND to indicate the entity that is responsible for issuing an IXCRECV to inspect the results of the IXCSEND request. You can specify a task, address space or the local system as the entity that is responsible. You can also specify the following for address space: • HOME to indicate the home address space of the sender. • PRIMARY to indicate the primary address space of the sender. • STOKEN to indicate a space token for the address space.

Table 11. Functions and keywords for IXCSEND (continued)	
IXCSEND function	**IXCSEND keyword**
Information about a client request message and the servers to which it is to be sent	SENDTO=SERVER to indicate that the message is intended for one or more server instances in the sysplex. The message is considered a request. SERVER to indicate the name of the server that is to process the request. You can then specify SYSTEMS to indicate which systems are to receive the request. SERVERID to indicate the particular server instance that is to process the request. FUNCTION to identify the function that target server is to perform. DESCRIPTION to contain a description of the request. EXPECTREPLY to indicate whether a reply to the message is expected from the target. CRITERIA to indicate the range of server levels and set of features that the target server must support in order to process the request CLIENTLEVEL to indicate the level of the client application.
Information about a message sent by a server in response to a client request	SENDTO=ORIGINATOR to indicate that the IXCSEND message is a response a client request. RESPTOKEN to specify the token that identifies the originating message to which this response is being sent. RESPRETCODE and RESPRSNCODE to provide a return and reason code to indicate the result of the request SUPPORTSLEVEL for the maximum level of response data that the sender can provide. SUPPLIEDLEVEL for the level of response data that the sender is providing.

Using the IXCRECV macro

Use the IXCRECV macro to receive the results of a message that was sent using IXCSEND. Most typically, IXCRECV is used by a client to receive the response messages sent by a server in reply to a request. Less typically, IXCRECV is used by a server to determine whether XCF has sent its reply message to the client. IXCRECV has two modes of operation, blocking and non-blocking. A blocking receive will not return to the caller until the subject message is complete. A message is complete, for example, when all of the expected responses have arrived. A non-blocking receive immediately returns to the caller with an indication of whether the subject message is complete.

Overview of IXCRECV

The XCF service routine stores information about the message and its responses in a caller provided **answer area** and stores any actual response data in a caller provided **data area**. If the caller does not

provide enough storage for all the requested data, the XCF service routine returns with a warning return code and indicates how much storage is needed, in which case, the caller needs to obtain the storage and reissue IXCRECV to receive the available data.

Requesting responses and IXCRECV

If a client sends an IXCSEND request and specifies EXPECTREPLY=YES, the server sends a response to the client who then uses IXCRECV to receive the response. Because the client is expecting a response, XCF maintains status for the client message. When the response arrives XCF binds it to the client message and associates it to the appropriate target so that the client can receive the response through IXCRECV.

A client can send requests to multiple targets, in which case there might be multiple responses expected, and thus multiple responses to receive.

IXCRECV allows you to specify the following requests depending on the information that the client wants to receive:

- You can ask for response data for the associated message from the XCF service routine by specifying RECEIVE=RESPONSES on IXCRECV. If you are expecting to receive the message data, you must provide a data area.
- You can also ask that XCF return only status information by specifying RECEIVE=STATUS on IXCRECV. This means that you are not expecting any message metadata or response data.

Receiving responses and IXCRECV

After sending its request to one or more target servers, the client invokes the IXCRECV macro to receive the responses from the servers. As a client, you might choose the following methods to handle the responses:

- Invoke IXCRECV immediately upon return from the IXCSEND service
- Perform other work and then invoke IXCRECV to process the response
- Allow another unit of work to invoke IXCRECV to process the response.

IXCSEND and the IXCRECV do not need to be invoked from the same address space, but for any given message, the IXCSEND request that sends the message and the IXCRECV request that receives the responses to that message must be invoked from the same system.

Processing responses

A message can be sent to multiple targets, which means that the caller must expect multiple responses, one from each target. To receive one or more of the responses with IXCRECV, you must provide an answer area (ANSAREA) and a data area (DATAAREA) on the IXCRECV macro.

Answer area

The answer area (ANSAREA) contains information about the message and its responses. Consider the following for specifying an answer area with IXCRECV:

- The answer area header, which is mapped by ixcysrvr_tAnsArea of the mapping macro IXCYSRVR, summarizes the state of the message and its responses.
- The answer area contains the following information about the sender, the message, and the response:
  - The "send descriptor" for the sender of the request, which is mapped by ixcysrvr_tSendDescriptor, and provides information about the original message sent on the IXCSEND macro.
  - The "target descriptor" for each requested target, which is mapped by ixcysrvr_tTargetDescriptor, and provides information about an individual target.
  - The "response descriptor"for each requested response, which is mapped by ixcysrvr_tResponseDescriptor, and provides information about an individual response.

Note that the order of the target/response descriptors does not necessarily correspond to the order of target systems specified in the originating IXCSEND invocation

Data area

The data area (DATAAREA) contains a copy of the message data that is sent by the responder using the MSGDATA or MSGDESC keywords on the IXCSEND SENDTO=ORIGINATOR request that sends the reply. The data area for receiving the response data can either be one contiguous storage area into which all the responses are stored, or a set of individual storage areas, one for each response.

One data area for all responses

The IXCRECV DATAAREA keyword names one contiguous storage area into which the response data for each requested response is to be stored. Consider the following:

- If there are multiple responses, the response data for each response will be concatenated end to end within the data area.
- The response descriptor in the answer area indicates where within the data area the response data for each response was stored.
- The data area must be large enough to contain the response data for all of the requested responses. If not, no response data is stored and the IXCRECV service routine returns a warning return code.
- The answer area header indicates how much storage is needed for the data area to hold all the response data for the requested responses.
- The caller needs to obtain the necessary storage and reissue the IXCRECV macro.
- The response data is to be stored at offset 0 in the data area. Thus, if the receiver requires that the response data be on a certain storage boundary (for example, a doubleword boundary), the receiver must ensure that the data area resides on that boundary.

One data area per response

Instead of one large contiguous data area, the receiver can provide a separate data area for each response. The IXCRECV DATADESC keyword indicates the location of a data descriptor table. A data descriptor table is an array with an entry for each requested response. Consider the following:

- Each entry of the array contains a "data descriptor" that identifies where the response data for the corresponding response is to be stored.
- There must be an entry in the table for each requested response. Furthermore, the storage area identified by any one data descriptor must be large enough to contain the response data for the corresponding response. If not, no response data is stored and the IXCRECV service routine returns with a warning return code.
- The answer area header indicates the number of response descriptors.
- The response descriptor for each response indicates the size of its response data.
- The caller needs to set up the necessary storage areas and data descriptors and reissue the IXCRECV macro to receive the responses.
- The response data for a response is to be stored at offset 0 in the storage area that is identified by the corresponding data descriptor in the data area. Thus, if the receiver requires that the response data be on a certain storage boundary (for example, a doubleword boundary), the receiver must ensure that the data area resides on that boundary.

When a message has more than one response, the entries in the data descriptor table can be associated with the responses in one of two ways:

- BIND=TARGET indicates that the data descriptor "i" is to be used when storing the response data from target "i" .
- BIND=NEXT indicates that each successive data descriptor is to be used for whatever response XCF happens to process next.

A client might choose to use BIND=NEXT, for example, if it learns from the response descriptors that some targets did not provide response data. This way the client only needs to provide a data descriptor table with enough entries for all available responses rather than for all targets.

Storage considerations for answer and data areas

The answer area and data areas must be large enough to hold the header and the descriptors for all of the requested targets and responses. If the answer area or data area is not large enough, the request is rejected. XCF returns an indication that you require more storage, and you need to reissue the IXCRECV macro if you want to receive the responses.

If you as the client do not want to receive the responses and do not intend to reissue the IXCRECV macro, you need to invoke the XCF message control service (IXCMSGC REQUEST=DISCARDMSG) to discard the message to allow XCF to clean up system resources in a timely manner. (The SENDTOKEN keyword on IXCMSGC allows you to specify the token associated with the XCF client/server request/response entity, and you can obtain the token from the IXCSEND macro.) Otherwise, the resources are held until the HOLDTIME value specified on the corresponding IXCSEND request expires.

For details about the IXCMSGC macro, see *z/OS MVS Programming: Sysplex Services Reference*.

Storage keys

By default, XCF uses the PSW key in effect at the time the XCF receive service was called when storing into the data area. Use the IXCRECV MSGSTGKEY keyword to specify that XCF is to use another key. Thus, any authorized caller of the macro might be able to have XCF store response data directly into storage areas provided by non-authorized users and still preserve system integrity.

When storing into the answer area, XCF always uses the PSW key in effect at the time that the IXCRECV receive service is called.

Target/response states

A client specifies IXCRECV with the RECEIVE=RESPONSES keyword option to receive the results associated with the message that has been sent. The following target/response states apply to any message data that you send:

- XCF considers a response to be **received** or delivered if XCF has stored the response data in a data area or has stored a response descriptor in the answer area for a response that has no data. (A response might have no data either because the responder did not provide any or because XCF no longer expects a response from the associated target).
- XCF considers a response to be **pending** if the target is still expected to send a response, but the response has not yet arrived.
- XCF considers a response to be **available** if the response has arrived but has not yet been received by the client. A response can also be "available" even if the response never arrives. If XCF determines that the target is no longer expected to respond, it considers the response to be available because the status of the response has been determined

XCF maintains status information about each target until the client message is discarded. XCF discards the client message when all of the responses associated with the message have been delivered.

If you do not expect the targets to send responses (that is, EXPECTREPLY=NO is specified on the IXCSEND request), XCF maintains status for each of the targets until the client message is discarded. When you as the client invoke IXCRECV with the RECEIVE=RESPONSES option to receive the status of the targets in the target descriptors, XCF considers the status metadata to be delivered. XCF discards the client message when all of the status metadata has been delivered.

If the unit of work that is responsible for issuing an IXCRECV (identified on the RECVBIND keyword of the IXCSEND request) to inspect the results of the IXCSEND request fails, the message status and responses are discarded and not available through IXCRECV. The token associated with the message is no longer valid.

Message completion and time out values

XCF considers a message to be "complete" when one of the following occurs:

- All of the expected responses have arrived.
- The value specified on the IXCSEND RESPTIME macro of the sender has expired.
- XCF message control service (IXCMSGC) is used to force completion of the message (REQUEST=COMPLETION).

Although there are many situations where XCF does not expect the target to respond, the following are the most common reasons:

- The message was not sent to the target system.
- The message was not delivered to the target system.
- A failure has occurred.

Consider the following cases for message completion:

- The server does not respond if XCF never sends the message to it.
- The message is not sent if XCF determines that the target system does not exist or if the target system is not running a release of z/OS that supports the XCF client/server interfaces.
- The sending system might experience resource constraints that prevent it from being able to send messages.

Even if the message arrives on the target system, it might not be delivered. There might not be an instance of the server running on the target system, or instances of the server are running, but none of them support the server level or features that are required to process the request, in which case, the server might refuse to accept the request. As a result, XCF sends back an acknowledgment to the originating system to indicate that no response should be expected from the target.

The sender of the message can specify time out values on the IXCSEND macro to determine how long to wait for a message response or how long XCF is to hold a completed message.

RESPTIME on the IXCSEND macro indicates how long the sender is willing to wait for the expected responses to arrive froma receiver. In effect, RESPTIME limits how long the client can poll for message completion or be blocked waiting for responses to arrive, or both. On the other hand, HOLDTIME on the IXCSEND macro requires the sender to retrieve the responses for a completed message in a timely fashion.

Except where work is not being dispatched in a timely fashion, the HOLDTIME value does not impact a blocked IXCRECV as the invoking unit of work is to be resumed as soon as the message completes. HOLDTIME might impact a receiver who is polling for completion, or a receiver whose receive failed for a lack of storage. If the client does not receive all the available responses within HOLDTIME, XCF discards the message and any remaining responses.

For details on time out values, see “Time out values” on page 146. For details on blocking receive occurrences, see “Blocking receives” on page 157.

Blocking receives

For the case where responses are expected, you can specify IXCRECV REQTYPE=BLOCKING that blocks the receive operation if any of the requested responses are still pending. A blocking receive suspends the calling work unit until every requested response is no longer pending. If none of the requested responses are pending when IXCRECV is invoked, the service routine returns immediately to the caller without blocking.

For the case where responses are not expected, a blocking receive blocks the receive operation if any of the send requests are still pending.

Whether responses are expected or not, the descriptors and response data (as applicable) for the requested targets are stored into the designated output areas. The return and reason code indicates

whether the message has completed (initiated send to all valid targets and received all expected responses), or is not found (message was discarded).

Message completion and IXCMSGC

The XCF message control service allows you to specify the following IXCMSGC requests to control message completion and blocked receives:

- IXCMSGC REQUEST=RELEASEMSG
- IXCMSGC REQUEST=COMPLETION
- IXCMSGC REQUEST=DISCARDMSG

IXCMSGC REQUEST=RELEASEMSG can be used by another work unit to release a blocked receiver. In such cases, the IXCRECV service routine returns to the caller indicating that the block is released. Neither the answer area nor the data area is stored. The message persists and the client can issue another IXCRECV request to receive the responses up until the time that the message is discarded (as occurs when the IXCSEND HOLDTIME value expires).

IXCMSGC REQUEST=COMPLETION allows undelivered responses that arrived before the message was forced to complete to remain available. Any responses arriving after the message is considered to be complete are discarded. Completing the message implies that the blocked work unit is released. The service routine updates the answer area and the data area for the requested targets and returns to the caller. Forcing completion in effect lets the client receive whatever responses had arrived up to that point.

IXCMSGC REQUEST=DISCARDMSG discards the message and all associated responses. Undelivered responses that arrived before the message was discarded are no longer available. Status information about the message is no longer available. Subsequently arriving responses are discarded. A blocked receiver will be released, but neither the answer area nor the data area will be stored. If IXCRECV is invoked after the message is discarded, the service routine returns to the caller indicating "not found".

Obtaining message status

You can obtain the status of the message by invoking IXCRECV with RECEIVE=STATUS. This IXCRECV request returns immediately with a return and reason code that indicates the state of the message. For example, you might use this service when polling for completion of the message.

Obtaining detailed response status

To obtain detailed response status without receiving the actual response data, use IXCRECV RECEIVE=RESPONSES and specify the NODATA keyword. For REQTYPE=BLOCKING, the calling work unit is suspended until the message is considered complete. The service routine then inspects the message and stores information about the message and any associated responses in the designated answer area.

The IXCRECV SCOPE=ALL keyword indicates that all results are to be gathered for IXCRECV RECEIVE=RESPONSES. For SCOPE=ALL, a target descriptor and a response descriptor for each response are stored in the ANSAREA. The client can then use this information, for example, to determine how much storage it needs to obtain for each response.

Delivered response

XCF discards a message when all of its associated responses have been delivered. For a response that has data, the response is considered delivered when XCF has successfully stored the response data in the indicated data area. For a response that has no data, the response is considered delivered when XCF has successfully stored a response descriptor for the response in the answer area. Furthermore, note that invoking IXCRECV RECEIVE=RESPONSES with the NODATA keyword causes the message to be considered delivered (and discarded) if none of the requested responses has any response data.

Competing receives

XCF does not allow multiple work units to be blocked for receiving the same message. XCF permits only one receive to be active at a time. If two work units invoke IXCRECV RECEIVE=RESPONSES for the same completed message at the same time (the work units will not be blocked since the message has completed), only one of them receives the response data, the status metadata, or both. The request from the other work unit is to be rejected.

Combining IXCRECV invocations

You can issue RECEIVE=STATUS and RECEIVE=RESPONSES for IXCRECV, with and without answer areas and data areas in various combinations to implement a variety of completion protocols. For example, an application might be designed with a highly variable amount of response data returned. You might use a blocking receive with an answer area to block until a response arrives, obtain a suitable data area based on the detailed response status information provided in the answer area, and then receive the response data. An application with predictable response data might poll for message completion. When the message completes, it can provide an answer area and data area large enough for all expected responses.

Response codes and the target receiver

XCF tries to provide a response code to indicate what happened to the request. The XCF response code is reported in the target descriptor stored in the answer area for the target (ANSAREA in the IXCRECV macro). If responses are expected (that is, EXPECTREPLY=YES is specified in the originating IXCSEND invocation), the response code is also stored in the corresponding response descriptor.

The IXCYSRVR macro defines the response code provided by XCF. See “Processing a request sent by a client” on page 139.

The response codes might help the sender to determine the recovery action if the target fails to respond. For example, if it can be determined that the target never received the message, it might be appropriate to resend the message. If the message is delivered to the target, the sender might need to determine whether the request was processed or not. Some requests might not be retriable.

However, note that XCF can at best indicate whether the request was presented to the target. XCF can not determine whether the request was processed correctly by the target. Even if XCF reports that the target failed while processing the request, processing of the request might have succeeded. Even if XCF reports that the request was successfully delivered to the target, processing of the request might have failed. In short, only the target can reliably indicate whether the request was successfully processed or not.

The response codes might indicate that the message was never presented to the target ("not sent", "no receiver", "not delivered"), or that the message might or might not have been presented to the target ("in progress"), or that the message was presented but not processed by the target ("refused"), or that the message was presented and possibly processed ("delivered", "failed"), or that the response sent by the target did in fact arrive ("replied"). If the response does arrive, you must check the response to determine whether the request was successful or not.

Several of the response codes have qualifiers that provide additional detail as to the specific circumstances that led to the response code. For example, a message might be "not sent" if the target system does not exist, or if the target system is not running a release of z/OS that supports the XCF client/server interfaces, or if the sending system had resource constraints. If there is "no receiver", it might mean that no instance of the server exists on the target system, or it might mean that no suitable instance of the server exists (for example, no instance of the server supports the required features).

In order to check the response when the server reply arrives you need to look for information in the response descriptor that is provided by the responding server (usually through return/reason codes, message control data, or message content). It is your responsibility to define and implement an appropriate protocol to handle the situation.

Summary of IXCRECV function

Table 12 on page 160 summarizes the functions and keywords for IXCRECV:

Table 12. Functions and keywords for IXCRECV

IXCRECV function	IXCRECV keyword
Information about the message to be processed	MSGTOKEN to identify the message token that the corresponding IXCSEND macro returns.
Information about the message data that the service routine is to provide when the receive request is for status information only.	RECEIVE=STATUS to ask for status information from the service routine.
Information about the message data that the service routine is to provide when the receive request is for response data associated with the message.	RECEIVE=RESPONSES to ask for response data of the associated message from the service routine. ANSAREA and ANSLEN to indicate where XCF is to store the status metadata and the storage buffer length of the area. NODATA to indicate that the response data is not to be stored. DATAAREA and DATALEN to indicate where XCF is to store the response data and the storage buffer length of the area. DATADESC, #DATADESC, LENDDENTRY, and BIND that identify the storage buffers where XCF is to store the response data, the number of data descriptors, the length of each entry in the data descriptor table, and the bind option that indicates how XCF is to associate the data descriptors to the response data. MSGSTGKEY to contain the storage key to use when storing the response data into areas described by DATADESC. SCOPE=ALL to indicate that all results are to be gathered. REQTYPE=BLOCKING to indicate that the caller is to be suspended until all expected results are available..

Using the XCF Server

The XCF Server is available to process server requests. You use the IXCREQ macro to format a server request message that can then be sent to the XCF Server on one or more systems in the sysplex through the IXCSEND macro. To send a request to the XCF Server and receive a response to the request, the requestor must do the following:

- Use the IXCREQ macro to construct an XCF Server request message. See “Using the IXCREQ macro” on page 161.
- Use the IXCSEND macro to send the request message to the XCF Server. See “Using the IXCSEND macro” on page 144.
- Use the IXCRECV macro to receive the response data from the systems that provided responses. See “Using the IXCRECV macro” on page 153.
- Use the mapping macro IXCYSRVR to interpret the XCF Server response data.

Using the IXCREQ macro

You use the IXCREQ macro to construct a request message that is supported by the XCF Server. A request message is constructed using the list and modify forms of the IXCREQ macro and specifying on the modify invocation the desired server function, options, and information selection criteria to be used by the XCF Server when processing the request. The area defined by the list form of the IXCREQ macro and updated by the modify form is the XCF Server request message that is to be sent to the XCF Server.

Sending a request to the XCF Server

To send a server request to the XCF Server, the client must first construct an XCF Server request message using the IXCREQ macro then use the IXCSEND macro to send the request message to the XCF Server. Specify the following on the IXCSEND macro to send a server request to the XCF Server:

1. Specify the message content to be sent to the XCF Server using the MSGDATA or MSGDESC keywords on the IXCSEND macro. The message content is the output from the list and modify form of an IXCREQ macro invocation. MSGLEN (or the dd_DataSize field of a message descriptor if you use MSGDESC) must be set to the length of the IXCREQ defined area generated by the list form of the IXCREQ macro. An EQU statement is generated by the IXCREQ list form expansion assigning the length of the IXCREQ defined area to a symbol that can be used on the IXCSEND invocation.
2. Allow the MSGCNTL keyword to default to 0. The XCF Server expects the MSGCNTL metadata for a request to be unused by the client.
3. Identify the XCF Server as the target to receive the message using the SENDTO=SERVER and SERVER keywords on the IXCSEND macro. The XCF Server Name can be found in IXCYSRVR. The XCF Server Name can be defined by specifying the following in a program:

```
IXCYSRVR_XCFSERVERNAME DC CL32'SYSXCF  IXCREQ       '  * Padded with 16 blanks

or using the EQUs provided in the IXCYSRVR macro, a constant can be defined:

XCFSERVER DC A(IXCYSRVR_SNAME1,IXCYSRVR_SNAME2,IXCYSRVR_SNAME3,IXCYSRVR*
               _SNAME4,C'    ',C'    ',C'    ',C'    ')
```

4. Identify which function is being requested from the XCF Server using the FUNCTION keyword. The functions supported by the XCF Server can be found in the IXCYSRVR macro.
5. Indicate that a reply is expected from the XCF Server by specifying EXPECTREPLY=YES.
6. Specify appropriate RESPTIME and HOLDTIME values. RESPTIME allows the sender to indicate how long the sender is willing to wait for the responses to arrive from all the XCF Servers that the request was sent to. HOLDTIME allows the sender to indicate how much time XCF is to allow for the sender to retrieve the responses using the IXCRECV macro.
7. Specify the SENDER keyword to identify the request sender and the DESCRIPTION keyword to indicate the description of the request.
8. Use the CRITERIA keyword to indicate the level of XCF Server support desired. The current XCF Server supports a server level of zero (0). The CRITERIA keyword on the IXCSEND macro is not required for currently supported XCF Server requests. You can use the XCF criteria defaults (for example, CRITERIA=DEFAULT). Specifying a non-zero value for any of the server selection criteria defined by mapping IXCYSRVR_TCRITERIA prevents the XCF Server from receiving the client request.
9. Select the target systems to send the XCF Server request to. The request can be sent to any combination of active systems in the sysplex. See the SYSTEMS keyword on the IXCSEND macro for more information on selecting systems to send the request to. IXCSEND returns a token to the requestor (RETMSGTOKEN) to represent the request. The token is used to identify the request when invoking the IXCRECV macro (keyword MSGTOKEN) to receive the response data for the request.

On each target system, the XCF Server receives the client request. The server processes the request and sends the response back to the local system for collection and return to the requestor.

Receiving responses from the XCF Server

You use the IXCRECV macro to retrieve the XCF Server responses for a request sent by IXCSEND. The request completes normally when all the expected responses arrive. If all the expected responses do not arrive within the specified RESPTIME, XCF considers the request to have completed. If the requestor does not receive the results of a completed request within HOLDTIME seconds, XCF discards the request results and responses.

To determine whether response data was returned by an XCF Server, first establish addressability to the storage provided on the IXCRECV ANSAREA and DATAAREA keywords. Consider the following:

- Each target/response descriptor record returned in the IXCRECV ANSAREA represents a target system to which the request was sent and from which a response was expected. The field aa_#Desc specifies the number of target/response descriptor entries returned in the ANSAREA.
- The rd_RespCode.RespCode_RC1 field of a response record contains the value ixcysrvr_RC1_Replied (08x) when a response was received from the target system. When it is determined that a response was received from a target XCF Server on the system named in the td_SysName field, you can use the rd_RespRetcode and rd_RespRsncode fields to determine the results of the XCF Server request. A value of zero (0) in the rd_RespRetcode field indicates that the request completed successfully. When the request completes successfully, the target XCF Server sends response data to the originating system. The field rd_MsgDesc.md_MsgAvailable indicates when response data is available in the storage area identified by the IXCRECV DATAAREA keyword.

See “Using the IXCRECV macro” on page 153.

XCF Server SERVERINFO requests

You can use the IXCREQ SERVERINFO macro keyword to construct a server request message for the XCF Server to collect information about application and system servers defined to XCF on a system in the sysplex. Information about servers defined to XCF on the local system, all systems or any combination of systems in the sysplex can be returned to the requestor in the DATAAREA provided on the IXCRECV macro.

The returned SERVERINFO response data is defined by mappings provided in the IXCYSRVR macro. See the following mappings for detailed information on the content of the response data, which is determined by the request selection criteria that was specified on the IXCREQ macro when generating the XCF Server SERVERINFO request:

ixcysrvr_tSrvrInfoAA
Server information answer area

ixcysrvr_tSrvrInfoDR
Server definition record

ixcysrvr_tSrvrInfoHR
Server data header record

ixcysrvr_tSrvrInfoWI
Server work item record

ixcysrvr_tSrvrInfoIR
Server instance record

Example of a client/server application

This example illustrates at a high level the way an application can act as a client and send a server request to the XCF Server to determine if the client's servers are active on a system in the sysplex. This example shows only mainline paths, and does not cover error conditions, serialization, or synchronization considerations.

Starting the server

The following explains what happens when a task, designated as the Server "task," is started on one or more systems in the sysplex. More than one task can be started for the same server name. When more than one task is started for a server name on the same system, the server has created multiple instances. Multiple instances of a server might be useful to manage workload or provide multiple levels of server support for clients:

1. The task issues IXCSRVR to begin the server definition process with XCF:

```
START IXCSRVR REQTYPE=START,SERVER=MYSERVER,SERVEREXIT=(R2), X
               DESCRIPTION=SRVRDESC,MINLEVEL=0,MAXLEVEL=0, X
               USERDATA=SRVRUSERDATA,INFO=(R8),            X
               RESPBIND=INSTANCE,                          X
               RETCODE=RETURN,RSNCODE=REASON
```

2. XCF calls the server exit specified on the IXCSRVR SERVEREXIT macro keyword to have the server instance perform initialization.
3. The server exit gets control with addressability to a parameter list pointed to by R1 and mapped by IXCYSRVR_TSXPL. The SXPL_SERVERCODE field contains ixcysrvr_kSC_InitServer, which indicates to the server exit that it is to perform its initialization processing.
4. Depending on the client/server design and protocols, the server might need to provide a work area to XCF to process received requests. For this example, the server does not need a work area.
5. The server exit returns to XCF after completing initialization. The server is now ready to receive requests from a client. When a server request is received from a client, XCF calls the server exit with a parameter list containing information provided by the client and needed by the server exit to process the request.

Starting the client

A client and server can devise many ways to notify the other of its existence and availability. In this example, the client is going to use the results returned from the XCF Server to determine if a server name is defined in the sysplex. If the server of interest is defined and active, the client begins sending server requests.

The client is started by a batch job or started task. The client begins by finding out which servers are defined in the sysplex. The client uses the IXCREQ macro to format a request message for the XCF Server to process, then sends the message to the XCF Server using the IXCSEND macro:

1. The client issues IXCREQ to format a server request message for the XCF Server and asks for information (REQUEST=SERVERINFO) about all servers defined to XCF in the sysplex:

```
LIST IXCREQ MF=(L,REQPL),PLISTVER=0

FORMAT IXCREQ MF=(M,REQPL),REQUEST=SERVERINFO,LISTALL
```

2. Using the output from the IXCREQ macro, the client issues IXCSEND to send the request to the XCF Server:

```
SEND     IXCSEND SENDER=ClientSender,                                  X
                 SENDTO=SERVER,                                        X
                 FUNCTION=ServFunc,                                    X
                 DESCRIPTION=ClientDesc,                               X
                 MSGID=ClientMsgID,SERVER=XcfServer,                   X
                 SYSTEMS=ALL,MSGDATA=REQPL,                            X
                 MSGLEN=REQPLL,                                        X
                 HOLDTIME=ClientHoldTime,                              X
                 RESPTIME=ClientRespTime,                              X
                 EXPECTREPLY=YES,                                      X
                 RETMSGTOKEN=ClientMsgToken,                           X
                 RETCODE=RC,RSNCODE=RSN

 XcfServer DC A(IXCYSRVR_SNAME1,IXCYSRVR_SNAME2,IXCYSRVR_SName3,IXCYSRVR*
                 _SName4,C'    ',C'    ',C'    ',C'    ')

 ServFunc  DC A(IXCYSRVR_SFunc1,IXCYSRVR_SFunc2)
```

3. To receive the responses from the XCF Server, the client issues IXCRECV. XCF on the client system will suspend the client task until all expected responses from XCF Servers are received or the RESPTIME expires:

```
RECEIVE  IXCRECV MSGTOKEN=ClientMsgToken,RECEIVE=RESPONSES,        X
         REQTYPE=BLOCKING,SCOPE=ALL,                               X
            ANSAREA=RecvAnsArea,ANSLEN=AnsAreaSize,                   X
               DATAAREA=RecvDataArea,DATALEN=DataAreaSize,               X
            RETCODE=RC,RSNCODE=RSN
```

When control returns, the client processes the results found in the ANSAREA and DATAAREA. Using the information returned by the XCF Server, the client is able to determine if the server of interest (for example, MYSERVER) is defined and available to receive requests and proceed based on its findings

Client/server compatibility

In a sysplex, software maintenance and software upgrades are typically applied in a "rolling" fashion, with new software being installed on one system at a time. It might take weeks or months for the change to be fully deployed throughout the sysplex. In the mean time, installations expect applications to remain available and operational. Thus different levels of clients, servers, or both can be running simultaneously in the sysplex, perhaps even within the same system. Clients might require their requests to be processed either by servers running at certain levels or by servers that support certain features. A given server might be able to offer different kinds of response data depending on what the client understands. Some levels of a client might be incompatible with some levels of a server.

Overview of client/server compatibility processing

XCF provides protocols intended to help clients and servers function compatibly with mixed levels of support and functionality. In particular, these protocols provide the criteria that XCF uses to determine whether a server instance is suitable for processing a given request. A request is not presented to a particular server instance if the instance is not suitable for the request.

When you start a server through IXCSRVR, you can specify the range of server levels it supports, the range of client levels whose requests it is willing to process, and the features that it supports.

When you use IXCSEND to send a request to a server, the client can either target a specific server instance, or it can have XCF select the server instance that is to process the request. If the client targets a particular server instance, XCF targets the client request to the specified server instance without comparing client specified server selection criteria to the server defined supported client levels, server levels and features.

The client is responsible for determining the suitability of the specified servers. If XCF is to select the server, the client must specify the client level, the desired range of server levels, and the features that a suitable server must support. The client and server mutually determine the content and interpretation of the levels and features.

When the request arrives on the server system, XCF compares the range of server levels requested by the client to the range of levels supported by the server. If the requested range does not intersect the supported range, the server is not eligible to process the request.

XCF also compares the client level to the range of client levels whose requests the server is willing to accept. If the client level is not in the indicated range, the server is not eligible to process the request. Finally XCF compares the features requested by the client with the features supported by the server. If the server does not support the required features, it is not eligible to process the request. If no suitable server remains, the request is cancelled with a "no receiver" response code. Otherwise, XCF selects one of the suitable server instances to process the request.

Setting up client/server compatibility

When you define the server through the IXCSRVR macro, you can specify the range of server levels that the server supports through the MINLEVEL and MAXLEVEL keywords, the range of client levels whose

requests it is willing to accept through the MINCLIENT and MAXCLIENT keywords, and the set of features that it supports through the FEATURES keywords.

When a client uses IXCSEND to send a request, it specifies its own level through the CLIENTLEVEL keyword and the routing criteria through the CRITERIA keyword that XCF is to use to determine which instances of a server are suitable for the request.

The routing criteria indicate the range of server levels that are suitable for the request and the features that the server must support in order to process the request. The desired server levels and features, as well as the client level are passed to the server exit routine when the request is presented for processing. The actual content and interpretation of the levels and features is determined by the client and server. As described below, XCF compares the requirements of the request against the support offered by the server to determine whether a given instance of the server is suitable for the request.

Using the CLIENTLEVEL and CRITERIA keywords on IXCSEND

In general, client/server requests have unique considerations because the server might not be at the same level as the client. The server might support requests, parameters, responses, or response data that the client does not support. Conversely, the server might not support requests or parameters specified by the client, or it might not provide the responses or data expected by the client. Clients and servers must take care to ensure correct operation in such cases.

XCF assumes the following when you specify the CLIENTLEVEL and CRITERIA keywords:

- Clients and servers can run at different "levels". Different levels of clients, servers, or both can be running simultaneously in the sysplex, perhaps even within the same system.
- The higher level code is responsible for ensuring compatibility with lower level code. For example, the higher level code generally continues to support all the function supported by lower levels of the code. When interacting with lower level code, higher level code adheres to the protocols and data formats expected by the lower level code. Higher level code does not present higher level data to a lower level instance unless it knows such data can be tolerated. If higher level code is incompatible with lower level code, the higher level must ensure that the different levels do not interact with each other.
- Client levels and server levels can change independently. Furthermore, any given client level has an understanding of the range of server levels it can interact with and any given server level has an understanding of the range of client levels it can interact with. A client does not specify a server level outside its range of understanding. A server does not specify a client level outside its range of understanding.
- At any given level, a server can support various features. A server can be upgraded to support new features without having to change its level. A client understands which server features must be supported to process any given request.

Server upgrade

The initial release of a client/server application most likely defaults to zero for all levels and features as the base starting point. So initially every server instance can process every request. Sometime later the installation might install a new version of the server and/or a new version of the client.

An example

For example, suppose the new level 1 release of a server supports all the old level 0 functionality. When it starts, it indicates that it supports server levels 0 to 1. The clients running in the sysplex might not yet have been upgraded to understand server level 1, so they continue to request server level 0 for their requests. Since the new level 1 server supports server level 0 functionality as well, the new level 1 server can process those requests. If there was both a level 0 and a level 1 server instance running, either one is suitable for such requests.

If the new level 1 server is not compatible with the level 0 server protocols or function, it indicates that it supports only server level 1. Suppose older versions of the clients are running in the sysplex. They are unaware of server level 1 and thus do not know that server level 1 is incompatible with their

request. However, being down level, they are still specifying server level 0 for their requests. Because the new level 1 server does not support server level 0 requests, it is not considered to be a suitable server for these requests. Any such level 0 requests need to be processed by a server that supports level 0 requests.

This usage might allow, for example, two different product releases to coexist at the same time. The server level ensures that the requests from old clients are processed by old servers, and new requests are processed by new servers even if both releases used the same server names.

You might also use the server name to ensure that requests are processed by a suitable server. If a new release changes the server names, the new server only receives requests from clients that have been upgraded to target requests to servers using the new names. There is no need to change the supported server levels in this scenario, or alternatively, you can reset them to 0.

Server and client recovery considerations

Consider the following failure and recovery situations for XCF client/server signalling processing:

- Failures to the server exit
- Server failures
- Failures with the receiver

Server exit failures

A server exit routine is considered to have failed if it fails to field an error or if it violates the rules of the XCF interface. If the server exit fails, XCF either returns to the invoker of the IXCSRVER REQTYPE=START request with a failing return code or percolates to its recovery environment. In either case, XCF cleans up its own resources and deletes the definition of the server instance. Requests for which the failed server was the last suitable instance are cancelled and acknowledged with a "no receiver" response code.

If a server exit is to establish its own recovery environment, it must establish it on each call (and delete the environment before returning to the XCF Server stub).

A server exit routine does not need to establish recovery as it might rely on the recovery established by the invoker of the IXCSRVR REQTYPE=START request. In that case, the server exit most likely uses the USERDATA keyword specified by the starter on IXCSRVR to locate control blocks for recovery resources. In the event of an error, the recovery for the XCF Server stub routine gets control and percolates to the recovery established by the invoker of the IXCSRVR REQTYPE=START request. To perform clean up of resources that recovery environment inspects the control blocks that are updated by the server exit to determine the required resources for clean up.

If the failed server exit routine is the last suitable server for a set of pending requests, the requests are cancelled and acknowledged with a response code indicating "no receiver." If you do not want these pending requests to be discarded, either ensure that the server exit routine establishes recovery so that it can handle errors and successfully retry, or ensure that there is some other suitable server instance to handle the pending requests. You might consider starting at least two server instances to reduce the risk of having a window of time where pending requests are discarded because no suitable server instance is available to process them.

The invoker of the IXCSRVR REQTYPE=START request can establish an ESTAE recovery environment but not an FRR. The server exit routine can establish either type of recovery environment.

If the server exit violates the XCF interface, the server is stopped by XCF. The XCF Server stub returns to the invoker of the IXCSRVR REQTYPE=START request with a failing return and reason code. Generally, violations of the interface occur when the server exit routine updates output fields in the Server Exit Parameter List (SXPL) in an inappropriate manner. XCF stops the server if the work area is not addressable.

Server failures

A server can assign the responsibility of sending responses to the requests presented to the server to a different entity. By default, XCF binds this responsibility to the server exit routine. If the responsible entity ends before sending an expected response, XCF sends an acknowledgment to the originating system to indicate that it should no longer expect a response from the target server.

Note that when a server system ends or fails XCF always cancels responses expected from the servers on that system. Thus if the server implementation entails third party notification that specifies another system other than the one on which the server resides to send the results, the possibility exists that XCF might cancel the expected response before the third party system responds. In effect, a race condition can occur between the third party response and the cleanup processing performed by XCF when a system is removed from the sysplex.

Receiver failures

When a response message arrives, XCF binds it to the originating send request. The originator (client) then invokes IXCRECV to receive the response. If a user error (such as inaccessible data area) occurs during the receive process, the service routine returns with a return code indicating the type of error. XCF preserves the response message until it is delivered or discarded. The client might have a chance to correct the error and reissue IXCRECV to receive the response. If the client corrects the problem and invokes IXCRECV before the HOLDTIME (specified on IXCSEND) expires, the client is able to successfully receive the responses.

When the client invokes IXCSEND to send a request, it can specify a "receive bind" (RECVBIND) to indicate the entity responsible for invoking the IXCRECV macro to receive the responses. If the indicated entity ends, XCF will discard the message and any associated responses because there will not be any work unit to invoke the IXCRECV macro. The RECVBIND specification helps XCF ensure that system resources are recovered in a timely manner.

Coding a server exit routine

A server exit is a routine that you write to process requests (messages) sent by programs that invoke the IXCSEND macro. The server exit routine is defined to XCF by the SERVEREXIT keyword when invoking the IXCSRVR macro from some task with REQTYPE=START to start a server instance. If the server instance is started successfully, XCF repeatedly calls the server exit routine to process requests that are targeted to the server.

A server exit routine can be called to perform the following functions:

- Perform initialization
- Get work area
- Process a client request.

The Server Exit Parameter List (SXPL) indicates which function the server exit is to perform and provides the parameters relevant to that function. Before calling the server exit to process a request, XCF stores the content of the request message in a work area that is provided by the server exit routine. If XCF does not have a work area large enough to hold the message content, it calls the server exit to get one.

This section presents the following information to help you code a server exit routine:

- The environment in which it receives control
- The information it receives as input
- The actions it might perform
- Programming considerations to bear in mind

Environment

The server exit routine receives control in the following environment:

The server exit routine receives control in the exact same environment that existed when the IXCSRVR REQTYPE=START request was invoked.

Minimum authorization:	Same state and PSW key of the caller of the IXCSRVR macro that defined the server instance
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN. The primary address space is equal to the primary address space of the caller of IXCSRVR, and can be swappable or non-swappable.
AMODE:	Same **AMODE:** of the caller of the IXCSRVR macro that defined the server instance.
ASC mode:	Same **ASC mode:** mode of the caller of the IXCSRVR macro that defined the server instance
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held

Entry Specifications

XCF passes information to the server exit user routine in a parameter list and in registers.

Version 0 of the server exit parameter list (SXPL). ixcysrvr_tSXPL of the IXCYSXPL mapping macro maps the parameter list that is passed to a server exit routine.

The Server Exit Parameter List (SXPL) is passed to the server exit routine by XCF. The SXPL resides in the primary address space of the server, and is defined in the IXCYSRVR macro by the mapping ixcysrvr_tSXPL. Within the SXPL, the field SXPL_ServerCode defines what function the server exit is to perform. The content of the SXPL varies according to function. The "base" portion of the SXPL is common to all functions. The content of the "function specific" portion of the SXPL is unique to the particular function. Thus, the SXPL_ServerCode determines not only what function the server exit is to perform, but also how to map the SXPL.

The field SXPL_ParameterOffset within the base portion of the SXPL indicates the offset (relative to the address of the SXPL) where the function specific parameters were stored by XCF. SXPL_ServerCode will have one of the following constant values defined in the IXCYSRVR macro:

IXCYSRVR_KSC_INITSERVER
The server exit is to perform whatever initialization is appropriate. The function specific parameters mapped by ixcysrvr_tInitServer contain copies of the keyword values that were specified on the IXCSRVR REQTYPE=START invocation that started the server. The first call of the server exit routine is always for the initialize function. XCF makes this call once.

IXCYSRVR_KSC_GETWORKAREA
The server exit is to obtain a work area for XCF to use. The function-specific parameters mapped by ixcysrvr_tGetWorkArea indicate the amount of storage that XCF requires. The server exit routine is expected to obtain the requested storage, update the SXPL_WAD field to indicate the location and size of the storage provided, and return to XCF. Note that on entry to the server exit, the SXPL_WAD field describes the work area that was last given to XCF. If there is an existing work area, the server exit will likely need to dispose of that storage before updating the SXPL_WAD with information about a new work area. If the work area provided by the server is not accessible, XCF stops the server.

IXCYSRVR_KSC_REQUEST
The server exit is to process a request. The function specific parameters mapped by ixcysrvr_tRequest contain copies of most of the keyword values specified on the IXCSEND macro invocation that was used by the client to send the request. The function specific parameters also include a "message descriptor" that identifies the size and location of the content of the request message.

Registers at Entry

When the server exit routine receives control, the GPRs contain:

Register
Contents

0
Used as a work register by the system.

1
Address of an SXPL (see ixcysrvr_tSXPL)

2-12
N/A

13
Same value as R13 when the IXCSRVR START call is invoked to define the server

14
Return address

15
Entry point address of server exit.

When the server exit routine receives control, the ARs contain:

AR0-AR1 -
Set to ALET of a primary address space (0)

AR2-R12 -
N/A

AR13
Same value as AR13 had when the IXCSRVR START call was invoked

AR14-AR15
N/A

Return Specifications

On return to XCF, the server exit routine does not have to set any return codes or place any information in the GPRs. The server exit routine returns control to the system by branching back to the address in GPR 14.

User Routine Processing

Before it calls the server exit routine to process a request, XCF determines whether a work area of sufficient size has been provided. If not, XCF calls the server exit routine with a "get work area" request ("server code" = ixcysrvr_kSC_GetWorkArea). The work area descriptor in the SXPL identifies the current work area, if any. The server exit is expected to dispose of the current work area as appropriate, obtain a new work area of sufficient size, update the SXPL_WAD to describe the new work area, and return to XCF. XCF will then prepare the relevant request, storing relevant data in the work area, and then call the server exit routine to process the request.

Initialization might not be necessary depending on the application. For example, it might be simpler to have the invoker of IXCSRVER REQTYPE=START do the initialization and then use the USERDATA keyword to pass the location of a control structure to the server exit (through SXPL_UserData).

The initialization might, for example, update SXPL_WAD to provide a work area for XCF to use when delivering requests to the server on subsequent calls. In cases where the client/server implementation is such that the request messages are of a known size, providing a work area at initialization could imply that the server exit never needs to process a "get work area" request.

Note that XCF does not call the server exit routine to perform a "shut down" or "termination" request. XCF expects the invoker of the IXCSRVR REQTYPE=START request to provide for any needed recovery of resources that might have been acquired by the server exit routine. In cases where such recovery is needed, the IXCSEND keyword USERDATA can be used to locate a control structure where the server exit records the resources for which it is responsible. When XCF stops the server, the code that started the

server (or its recovery) examines the control structure to determine the set of resources that need to be cleaned up.

When a client invokes IXCSEND SENDTO=SERVER to send a request to a server, XCF puts a copy of the request message in a server provided work area, and then calls the server to process the request. If the current work area is not large enough to hold the content of the request message, XCF first calls the server exit to get a work area of sufficient size. The SXPLGW_TotalSize field indicates how much storage is needed. The server exit is expected to obtain the requested storage and update the SXPL_WAD field to describe the storage that XCF is to use.

Upon return from the server exit, XCF inspects the refusal code (SXPL_RefusalCode) set by the server exit routine. If nonzero, XCF sends an acknowledgment (as needed) to the originator of the request to indicate that the target refused the request. The request is discarded and will not be presented to the server. The work area provided, if any, will not be used for the request that was refused, but could potentially be used for a subsequent request.

If the request is not refused, XCF inspects the SXPL_WAD to verify that the work area it describes is available for use and large enough to hold the message. If not, XCF sends an acknowledgment (as needed) to the originator of the request to indicate that the request was not delivered because the target failed to provide a work area. The request is discarded and will not be presented to the server.

The work area provided, if any, will not be used for the request that was refused but could be used for a subsequent request. If the work area is not accessible, XCF sends an acknowledgment (as needed) to the originator of the request to indicate that the request was not delivered due to a server error. XCF also stops the server.

If the provided work area is available for use, large enough to hold the request message, and accessible, XCF copies the message into the work area and calls the server exit to process the request. The SXPL_WAD is mapped by ixcysrvr_tWorkAreaDescriptor.

To provide a work area, the server exit must set theWAD_Available flag to indicate that the work area described by WAD_DataDesc is available for use by XCF. Set the WAD_StgKey field to specify the storage key that XCF should use when storing into the indicated work area. Within the WAD_DataDesc field, one must indicate the size, ALET, and address of a contiguous storage area to be used as the work.

When a server exit routine is called to process a client request, SXPL_ParameterOffset indicates the offset within the SXPL at which storage mapped by ixcysrvr_tRequest is located. These parameters reflect the keyword values specified on the IXCSEND SENDTO=SERVER invocation used by the client to send the request message to the server:

- FUNCTION
- DESCRIPTION
- CLIENTLEVEL
- Server selection criteria extracted from CRITERIA

The message descriptor (SXPLRQ_MsgDesc) contains a copy of the IXCSEND MSGID and MSGCNTL keyword values specified by the client. If the client provided actual message content (through MSGDATA or MSGDESC keywords with a nonzero MSGLEN), the message descriptor will indicate that the message content is available. If so, the data descriptor field (md_DataDesc) within the message descriptor indicates where XCF put a copy of the client message data.

The message descriptor also contains copies of other IXCSEND keyword values (SENDER, SENDERID, RESPTIME, HOLDTIME), as well as other metadata (such as ETODs indicating when the message was sent and when it arrived, and which system in the sysplex sent the message). If the client expects a reply, the md_ExpectReply flag will be set to so indicate and a copy of the RESPONSELEVEL keyword value will be provided as well.

If a reply is expected, the server (or its agent) should formulate an appropriate reply and send the response by invoking IXCSEND SENDTO=ORIGINATOR.

The md_RespToken field in the message descriptor contains the token to be specified for the RESPTOKEN keyword when sending the reply. Note that XCF will reuse the storage containing the SXPL for a subsequent request. So if the reply is not sent before the server exit routine gives up control, it must

take pains to preserve a copy of the md_RespToken value in some other storage area for later use when sending the reply.

Programming Considerations

Consider the following when writing your server exit routine:

- If the server exit disposes of a work area, understand that storage within the work area might contain data that is needed to process the request. The server exit might need to refrain from accessing such data after the work area is freed or appropriately preserve a copy of the needed data before the work area is released.
- If the server exit fails to provide a work area suitable for processing the request, XCF cancels the request with a response code indicating that the server failed to provide a suitable work area, and thus refused to process the request. (Note that when the work area is not provided, XCF does not even present the request to the server.) See “Server exit failures ” on page 166

User Routine Recovery

SRB-to-task percolation does not occur while the task's recovery routine is running. XCF waits until the task is not in recovery before abnormally ending the task.

If the server exit disposes of a work area, understand that storage within the work area might contain data that is needed to process the request. The server exit might need to refrain from accessing such data after the work area is freed or appropriately preserve a copy of the needed data before the work area is released.

Work area considerations

For some servers, it might be possible to provide one work area for XCF to use over and over for each new client request. This case might apply, for example, to a server whose request messages are of a known size and whose requests are processed synchronously by the server exit routine. If so, one might provide a work area when the server exit is called to initialize itself, and then never need to process a "get work area" request.

For some servers, a new work area must be provided for each client request to be processed. This case might apply, for example, to a server that arranges for asynchronous processing of the request by some other work unit. If the server exit leaves the work area available to XCF when it returns, the work area could be overlaid with the content of the next request message to be presented to the server exit, which in turn could corrupt the content of the work area as seen by the asynchronous work unit that is processing the previous request.

To preserve the integrity of the work area for the asynchronous work unit, the server exit needs to update the SXPL_WAD to either indicate that the work area is no longer available to XCF or to provide a new work area. Alternatively, the server exit could arrange for the asynchronous work unit to process a copy of the data in the work area, in which case the work area could be left intact for XCF to use with the next request.

Part 3. Sysplex Services for Recovery (Automatic Restart Management)

© Copyright IBM Corp. 1994, 2023

Chapter 4. Using the Automatic Restart Management Function of XCF

In a sysplex environment, a program can enhance its recovery potential by registering as an element of automatic restart management. An automatic restart management element represents a program or an application that:

- Is submitted as a job.
- Is submitted as a started task.
- Is an abstract resource. An abstract resource is a program or a set of programs that is only associated with (or has a bind to) the system on which it is running. See "Understanding Abstract Resources" on page 176.

Automatic restart management can reduce the impact of an unexpected error to an element because MVS can restart it automatically, without operator intervention. In general, MVS restarts an element when:

- The element itself fails. In this case, MVS restarts the element on the same system.
- The system on which the element was running unexpectedly fails or leaves the sysplex. In this case, MVS restarts the element on another system in the sysplex; this is called a **cross-system** restart.

In general, your installation may use automatic restart management in two ways:

1. To control the restarts of applications (such as CICS®) that already use automatic restart management as part of their recovery.
2. To write or modify installation applications to use automatic restart management as part of recovery.

To provide program recovery through automatic restart management, your installation has to activate a policy through the SETXCF START command. This can be an installation-written policy or the IBM-supplied policy defaults. Because an installation-written policy may have an effect on your program, you need to understand how your installation uses automatic restart managment for recovery in a sysplex, and code the program with these factors in mind.

Understanding How Your Installation Uses Automatic Restart Management

Your installation can use automatic restart management to provide improved availability for certain programs, and can customize automatic restart management for the sysplex in a variety of ways. Briefly, your installation can:

- Set up one or more automatic restart management policies, which can use default values for restarts or values tailored for the installation's workload.
- Enable automatic restart management restarts by issuing the SETXCF command to activate one policy. (The installation also can use SETXCF to disable automatic restart management restarts or activate a different policy.)
- Code a workload-restart installation exit to prepare a system for cross-system restarts.
- Code an element-restart installation exit to modify a restart for a particular element.

To use the functions of automatic restart management, your installation needs only to define a couple data set and start a policy. Customizing automatic restart management is optional. For more information about automatic restart management functions, see the following:

- *z/OS MVS Setting Up a Sysplex* for information about policies, and requirements for using automatic restart management.
- *z/OS MVS System Commands* for information about the SETXCF command.

© Copyright IBM Corp. 1994, 2023

- *z/OS MVS Installation Exits* for information about the workload-restart installation exits.

Requesting Automatic Restart Management Services

You can design any program to request automatic restart management functions. These work elements do not have to be members of an XCF group to use automatic restart management, but they can be. Any information about automatic restart management specifically for XCF group members is noted in the appropriate sections of Chapter 2, “Using the Cross-System Coupling Facility (XCF),” on page 7.

Understanding Abstract Resources

Starting with OS/390 Release 9, automatic restart management services support the restart of abstract resources. An abstract resource is a program or a set of programs that is only associated with or has a bind to the system on which it is running. Because an automatic restart management element representing an abstract resource is not associated with any address space, no job failure or started task failure will cause this type of element to restart.

An abstract resource identifies itself as such when registering with automatic restart management by explicitly specifying that it has a bind to the system on which it was started. If the system fails unexpectedly, automatic restart manager restarts the element on another system regardless of what the element had designated as its termination type.

Using the IXCARM Macro

Through the IXCARM macro, a program can:

- Register as an element of automatic restart management and, optionally, specify restart parameters and an event exit (REGISTER parameter).

  After the work element has issued the IXCARM macro with the REGISTER parameter, MVS can automatically restart the program when an unexpected failure occurs. You can specify restart parameters on the REGISTER request; however, in general, restart parameters in an installation-written policy override parameter values specified on the IXCARM macro.
- Indicate when it is ready to receive work (READY parameter).
- Deregister from automatic restart management when the program or application no longer needs to be restarted. If a program fails after it deregisters, MVS will not restart the program.

A program also can issue the IXCARM macro to:

- Indicate that MVS should delay the restart for this program until MVS completes the restart of a related program, which is called a **predecessor element** (WAITPRED parameter).
- Identify itself as the backup program for another element of the automatic restart manager. This identification tells MVS that the other element should not be restarted unless this backup program is deregistered (ASSOCIATE parameter).

An unauthorized application can request all functions of IXCARM. However, there are restrictions. For example, an unauthorized application cannot specify ELEMBIND=CURSYS for any of its elements.

A program can issue the IXCARM macro even if the installation has not enabled, or has disabled, automatic restart management restarts. MVS successfully processes the requests, and includes the program as part of the current information about automatic restart management, but will not attempt to restart the program until automatic restart management restarts are enabled.

MVS also will not attempt to restart (and will deregister) an element under any of these conditions:

- The element is cancelled through a CANCEL or FORCE command without the ARMRESTART parameter specified. Note that elements that are registered as abstract resources are not deregistered in this case.
- JES is down or has indicated that the element should not be restarted and the element represents a job or started task that is not started with SUB=MSTR.
- The element has reached the restart attempts threshold specified in the policy.
- The policy indicates that this element should not to be restarted.

- The event exit indicates that this element should not be restarted.
- An element-restart exit indicates that this element should not be restarted.
- Another registered element is associated with this one.
- Access to the ARM couple data set was lost.
- The override JCL dataset cannot be accessed or is bad.

Understanding How MVS Handles Restart Processing

When an **element** unexpectedly fails, automatic restart management is given control during end-of-job and end-of-memory termination after all other recovery has taken place. If the element should be restarted, then MVS:

1. Gives control to the element-restart exit.
2. Gives control to the event exit.
3. Restarts the element.
4. Issues an ENF signal when the element re-registers with the automatic restart manager.

When a **system** unexpectedly fails, automatic restart management determines if any elements were running on the system that failed. If those elements can be restarted on another system, and cross-system restarts are allowed, MVS does the following for each system on which the elements will be restarted:

1. Gives control to the workload restart exit.
2. For each element that will be restarted:
    a. Gives control to the element restart exit.
    b. Gives control to the event exit.
    c. Restarts the element.
    d. Issues an ENF signal when the element re-registers with the automatic restart manager.

Restart Considerations for Abstract Resources

An element representing a job or started task can be restarted with persistent restart text. Persistent restart text is the JCL or started task command that was used to originally start the job or started task. An element representing an abstract resource cannot be restarted with persistent restart text because automatic restart management has no knowledge of how the element was started; no persistent restart text is available. To restart a failed element that represents an abstract resource, automatic restart management must be supplied with restart text. Restart command text can be supplied in the following ways:

- By the application when it registers on the IXCARM macro with the STARTTXT keyword
- In the RESTART_METHOD statement in the ARM policy
- By the element restart installation exit in the ERESTARTTXT or EREJCLDATASET fields.

Restart text can be supplied when the element is being restarted by the element restart installation exit; therefore, automatic restart management cannot reject registrations when restart text is not provided through either the IXCARM macro or the ARM active policy. If elements representing abstract resources fail and restart text is not supplied, automatic restart management will not restart the element; the element will be deregistered.

Establishing Security for Restarted Jobs

When restarting an element, automatic restart management either can use the JCL that previously started the element (persistent JCL) or can override that JCL by specifying the installation-supplied name of a data set that contains the JCL for restarting the element. The following security considerations apply when restarting elements with either persistent or override JCL:

- When restarting an element with persistent JCL, automatic restart management establishes the same security environment as existed when the element last ran.
- When restarting an element with override JCL, automatic restart management establishes the same security environment as existed when the element most recently registered with automatic restart management. Within this security environment, automatic restart management both opens the data set containing the override JCL and submits the job.

  In most cases this has the effect of propagating the original element's job security information to the restarted element. However, there are a few special cases that you should consider:

  1. If the override JCL specifies a different user ID (with the USER= parameter on the JOB statement), then MVS does not propagate the most recent user ID to the new element and instead uses the new user ID specified. The override JCL also must specify the new user's password, unless the most recent user has the appropriate RACF® SURROGAT authority to specify the new user ID.
  2. If the override JCL does not specify a different user ID, then MVS propagates the most recent user ID to the new element. However, MVS does not propagate the most recent group ID to the new element. Instead, MVS uses the most recent user's default group as the new element's group ID unless the GROUP= parameter on the JOB statement specifies a different group ID.

     The fact that MVS does not use the most recent group ID should not normally cause any security problems for the new element. However, there are cases where access might be denied. The following examples might experience this effect:

     – When using RACF, you run with SETROPTS NOGRPLIST (disabling list-of-groups processing) specified.
     – When using RACF, you use &RACGPID (affecting the user's current connect group) in some members of the GLOBAL class.
  3. When using override JCL, the input source (port of entry, POE) for the new element will be INTRDR. This might differ from the input source of the original element, but should not normally cause any security problems. However, access might be denied in some cases, such as when using RACF and the element requires conditional access list entries that specify WHEN(JESINPUT(xxx)) where xxx is the input source of the original element.

  Note: When using override JCL, submission of the new element will fail if all the following conditions exist:

  – You use RACF
  – The JOB statement does not specify a new user ID
  – The most recent execution user is protected by RACF's PROPCNTL class.

  This applies particularly to the restart of CICS regions, where many customer installations disallow propagation of the CICS region's user ID to a submitted job using the PROPCNTL option. However, this would only pose a problem if you use override JCL during the restart.

Designing Your Application to Use Automatic Restart Management Services

The IXCARM macro may be used in several ways. The simplest way is to register a job, task, or abstract resource with the automatic restart manager during program initialization, let automatic restart management know when the job or task is ready to perform its work, and deregister when the program does its cleanup (REGISTER, READY, and DEREGISTER parameters).

A more complex use of the IXCARM macro involves controlling the sequence in which elements become ready during a restart (WAITPRED parameter — see “Waiting for Other Work to be Restarted (IXCARM REQUEST=WAITPRED)” on page 181). Another use of the IXCARM macro involves designating a backup for an element (ASSOCIATE parameter — see “Associating One Element with Another (IXCARM REQUEST=ASSOCIATE)” on page 182).

When automatic restart management restarts an element, it restarts the element from the very beginning of its code. Automatic restart management does not perform any kind of cleanup processing or recovery for elements. The event exit (see “Designing an Event Exit” on page 182), the element restart installation exit, and the workload restart installation exit have been provided to allow the element's environment to be cleaned up, recovered, and prepared for restart processing. The element can also perform some cleanup before or after it re-registers.

When an element has been restarted, it will receive a return code X'4' with a reason code of X'104' or X'108' when it re-registers with automatic restart management. This indicates that the element has been restarted and the program can do cleanup processing before continuing. Types of cleanup can be:

- To purge any partial output (depending on the type of output data set used).
- To determine the effect of using replication symbols in any dynamic allocations that may be done. Started tasks use the symbols from the system the program initially registered on before and after the re-registration. Batch jobs use the symbols from the system the program initially registered on after the re-registration; the local system's table is used before the re-registration for batch jobs.

Registering as an Element (IXCARM REQUEST=REGISTER)

A program must request that it be automatically restarted in the event of an unexpected failure by issuing the IXCARM macro with the REQUEST=REGISTER parameter. Keep in mind that:

- If your program is using checkpoint/restart, it cannot register with the automatic restart manager.
- MVS allows only one registered element per address space, unless the element represents an abstract resource.
- All IXCARM requests issued on behalf of an element must be issued from the same address space as the register request except for the following:
  - IXCARM requests for elements that represent abstract resources that can be issued from any address space.
  - IXCARM deregister requests for elements that represent jobs or started tasks can be issued from the master address space as long as the RMTOKEN keyword is specified on the request. This allows these types of elements to deregister while running under a resource manager in master's address space.
- An unauthorized application cannot specify an event exit routine or restart text when registering with automatic restart management and is not allowed to register abstract resources.

For the maximum benefit from automatic restart management, register your program as early in initialization processing as possible to avoid timeouts when the job is restarted. The only required information for a register request for a job or started task is the element name. When registering an abstract resource, you must also specify an output area in which the system will return a restart manager token. This token must be passed as input on subsequent automatic restart management requests on behalf of the abstract resource.

You may also provide:

- The event exit name and parameter list to be passed to the exit, if you design an event exit to work with this program. Programs that register as elements of the automatic restart manager should consider providing an event exit to perform any specialized processing for the restarted element.

  See “Designing an Event Exit” on page 182 for information about how to code an event exit.
- The relationship between the element and the system, which indicates whether the element has a bind to the batch job or started task under which the element is registering or has a bind to the system on which the element is registering. Elements that represent abstract resources have a bind to the system on which the element is registering.
- The answer area (mapped by IXCYARAA) for the system to return information about itself and the registration request. Some of the information returned (when the request completes successfully) includes:

ARAAREGTYPE
Indicates whether this is the initial registration of this element or the result of a restart. If this field is 0, the answer area may not be valid.

ARAAFLAGS1
Indicates whether automatic restart management has been enabled to do restarts. (The command SETXCF START,TYPE=ARM must be issued to enable the automatic restart management function.)

ARAAHOMECLONE
The replication ID of the system where the element initially registered.

ARAACURCLONE
The replication ID of the system where this registration occurred.

For more information about the parameters for the IXCARM macro, see *z/OS MVS Programming: Sysplex Services Reference*.

Specifying Restart Parameters

Restart parameters can be specified when registering the element through the IXCARM macro, in the element restart exit, or in the ARM policy. The hierarchy for parameters used during a restart is:

- The element restart exit overrides the installation-written policy.
- Installation-written policy parameters override parameters specified on the IXCARM macro.
- Parameters specified on the IXCARM macro override policy defaults.

Note: If the element restart exit or active policy alter the original method of start, the security environment the job was originally started under will not be changed.

Significant restart parameters are:

- How an element should be restarted.

  For started tasks, specify restart text to provide text that differs from the original START command text (referred to as **persistent command text**). For abstract resources, specify restart text to restart the abstract resource. If restart text is not supplied for abstract resources on registration, or in the automatic restart management policy, or by an installation-written element restart exit, no restart will be performed. For elements that represent jobs, use the RESTART_METHOD statement in the automatic restart management policy to provide restart text that differs from the original JCL (referred to as **persistent restart text**) used to submit the job.

  Start text (STARTTXT keyword on IXCARM) is overridden by RESTART_METHOD in the active ARM policy.
- The circumstances (element or system termination) under which the element is to be restarted.

  The type of termination (TERMTYPE keyword on IXCARM) is overridden by TERMTYPE in the active ARM policy.

  Note that for an element that represents an abstract resource, the element can only be restarted for a system failure, regardless of the TERMTYPE specification in the active ARM policy. Therefore, if the installation does not want the element started for a system failure, specify RESTART_ATTEMPTS(0) in the ARM policy.
- How long it will take the element to re-register.

  To ensure that the restart of a given element completed successfully, automatic restart management uses a time limit between the restart of an element and its re-registration. This limit is called the **restart timeout threshold**. If an element could take a long time to restart, code this parameter to keep the program from being de-registered.

  The restart timeout threshold (RESTARTTIMEOUT keyword on IXCARM) is overridden by RESTART_TIMEOUT in the active ARM policy.

Indicating Readiness for Work (IXCARM REQUEST=READY)

When a program has successfully registered as an element of the automatic restart manager, the program should issue the IXCARM macro with the REQUEST=READY parameter as soon as possible after initialization completes, to avoid a timeout. (When the amount of time between issuing the IXCARM macro with REQUEST=REGISTER and issuing the IXCARM macro with REQUEST=READY is greater than the ready timeout threshold, a ready timeout will occur. This interval can be specified in the automatic restart management policy on the READY_TIMEOUT parameter.) This is especially important during restart processing, when other elements might be waiting until an element is ready (or times-out) before they continue processing.

Note: For a cross-system restart, the element will not complete the READY process until all the elements in lower policy levels become ready or time-out. See "Waiting for Other Work to be Restarted (IXCARM REQUEST=WAITPRED)" on page 181 for more information on WAITPRED processing.

An application that has registered with ELEMBIND=CURSYS must specify the restart manager token that was returned on the REGISTER request when issuing the IXCARM REQUEST=READY request. The contents of the restart manager token must not have been modified. Authorized programs can also use the IXCQUERY REQINFO=ARMSTATUS service to obtain the restart manager token. This might be convenient for applications that register abstract resources in one address space and issue the IXCARM REQUEST=READY request in a different address space.

An application that has registered with ELEMBIND=CURJOB cannot specify the restart manager token that was returned. A non-zero return code will be returned if the restart manager token is specified on any IXCARM request other than REGISTER and DEREGISTER.

Deregistering the Element (IXCARM REQUEST=DEREGISTER)

When an element no longer requires automatic restarts as part of its recovery environment (usually during the program's cleanup processing), the element should issue the IXCARM macro with the REQUEST=DEREGISTER parameter.

If the element was associated with another element (IXCARM REQUEST= ASSOCIATE was issued), the system will disassociate the element as part of the DEREGISTER request. For more information on associating elements, see "Associating One Element with Another (IXCARM REQUEST=ASSOCIATE)" on page 182.

An application that has registered with ELEMBIND=CURSYS must specify the restart manager token that either was returned on the REGISTER request when issuing the IXCARM REQUEST=DEREGISTER request or through IXCQUERY REQINFO=ARMSTATUS. Optionally, an application that has registered with ELEMBIND=CURJOB can specify the restart manager token as input to the DEREGISTER request. These requests can be issued from the master address space or from the address space where the element was registered. The contents of the restart manager token must not have been modified.

Programs should be aware that an element might be deregistered by the SETXCF FORCE,ARMDEREGISTER operator command or automatically by MVS when various error conditions occur. An ENF signal is issued whenever an element is deregistered, so any unexpected situations could be handled through an ENF listener user routine (see "Monitoring Restarts through the ENFREQ Macro" on page 185 for more information).

Waiting for Other Work to be Restarted (IXCARM REQUEST=WAITPRED)

In certain cases, one program might have to wait until other programs are up and running before it can initialize successfully. During initial startup, an installation can manage such dependencies by the order in which it starts individual programs; however, this sequence is equally important when a system leaves the sysplex and MVS is to perform a cross-system restart.

For elements of the automatic restart manager, the element that must become ready first is known as a **predecessor** element. During restart processing, an installation can manage the sequence of restarting elements and their` predecessors in two ways:

- Through the assignment of elements to restart groups and to a specific level in the automatic restart management policy.

  MVS restarts all of the elements, then allows the elements in lower policy levels to become ready to do work, before allowing elements in higher levels to become ready. (For example, all elements in LEVEL 1 should indicate they are ready before elements in LEVEL 2 complete their ready processing.)
- Through the WAITPRED request on the IXCARM macro.

  By issuing IXCARM with the WAITPRED parameter, an element indicates that a predecessor element must become ready before this element can initialize successfully. During restarts, not initial starts, MVS will wait for the predecessor to issue IXCARM REQUEST=READY before allowing this element to complete ready processing. Issuing WAITPRED is most useful when an element and its predecessor are in the same restart group, by specific assignment or by default. Elements should issue WAITPRED after the register request, but before the ready request.

  If the restarted element is a predecessor of other elements (that is, other elements wait for this element to become ready before they can become ready), the element has a limited amount of time to re-register and to indicate its readiness for work. MVS provides these time limits so the other elements are not suspended — or waiting — forever, if the predecessor element fails or is waiting for a resource to become available before issuing the IXCARM macro with REQUEST=REGISTER or REQUEST=READY parameter.

  When a predecessor element exceeds the time limit for re-registering or for becoming ready, the element waiting for the predecessor receives a return code X'04' with a reason code of X'204' or X'304' from its ready request. The element should then determine if it can run without the predecessor and take whatever action is appropriate.

An application that has registered with ELEMBIND=CURSYS must specify the restart manager token that was returned either on the REGISTER request when issuing the IXCARM REQUEST=WAITPRED request or through the IXCQUERY REQINFO=ARMSTATUS service. An application that has registered with ELEMBIND=CURJOB cannot specify the restart manager token that was returned. A non-zero return code will be returned if the restart manager token is specified on any IXCARM request other than REGISTER and DEREGISTER. The contents of the restart manager token must not have been modified by the application.

Associating One Element with Another (IXCARM REQUEST=ASSOCIATE)

Another way elements can notify MVS of a dependency is through the ASSOCIATE request of the IXCARM macro. If a transaction processing application maintains a backup application for recovery purposes (such as with extended recovery facility (XRF)), the backup element should issue the IXCARM macro with the REQUEST=ASSOCIATE parameter to indicate that the system should not automatically restart the primary element. When one element is associated with another element, a restart will be done only when the backup has deregistered and the primary element fails, or is on a system that fails. In the event that the backup element fails, then the automatic restart manager will restart the backup element.

Designing an Event Exit

The event exit is available only through the IXCARM REGISTER request. This exit:

- Gets control any time the element is to be restarted, but only after the workload restart and element restart installation exits have completed processing

  Note: When this exit runs, all resource managers have completed processing and the address space the element was originally running in is no longer addressable.
- Has to be able to be loaded by every MVS system in the sysplex that is connected to the ARM couple data set
- Runs on the system on which the element is to be restarted
- Receives the address and length of a copy of the automatic restart manager event-exit parameter list, mapped by IXCYEVE. The parameter list contains the address of the event exit parameter list if EVENTEXITPL was specified on the IXCARM macro.

- Sets a return code that tells automatic restart management whether to proceed with the restart.

Exit Routine Environment

The event exit receives control, in the XCF address space, in the following environment:

Authorization:	Supervisor state and PSW key 0
Dispatchable unit mode:	Task
Cross-memory mode:	PASN = HASN = SASN.
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held

Exit Recovery

Automatic restart management does not provide any recovery for the event exit routine. Routines that require recovery must establish their own. The recovery routine must provide whatever diagnostic data is required for problem determination for the event exit routine.

If the event exit abends, passes invalid data back to automatic restart management, or cannot be invoked, the element will not be restarted.

Exit Routine Processing

The system passes control to the event exit prior to restarting the element. This exit gets control after the workload restart and element restart installation exits.

The system passes information to the event exit routine in a parameter list and in registers. The routine may release resources, clean up storage, determine if the element should be restarted, or do whatever processing is necessary to restart the element.

Processing Considerations

Consider the following when writing an event exit routine:

- The event exit routine must be a reentrant program
- The event exit should be in LPA or in an authorized LINKLIST or LINKLIB concatenation on all of the MVS systems in the sysplex that are connected to the ARM couple data set
- The event exit routine is given control on the system where the restart will occur
- If this exit is getting control because the element failed, the address space the element was running in is no longer addressable
- If this is a cross-system restart, make sure all addresses passed in the event exit parameter list, specified on the REGISTER request, are addressable from this system.

Input Register Information

On entry to the event exit routine, the general purpose registers (GPRs) contain:

Register
Contents

0
Does not contain any information for use by the event exit

1
Address of the event exit parameter list (mapped by IXCYEVE)

2-12
Do not contain any information for use by the event exit

13
Address of a 144-byte work area for use by the event exit routine. The exit routine does not have to save and restore XCF's registers in this work area. The exit routine can use this work area in any way it chooses.

14
Return address

15
Entry point address

When the event exit receives control, the access registers (ARs) contain no information for use by the event exit.

Output Register Information

When control returns to the automatic restart manager, the general purpose registers (GPRs) contain:

Register
Contents

0-14
The exit routine does not have to place any information in these registers, and does not have to restore their contents to what the contents were when the routine received control.

15
Return code

0
The automatic restart manager should proceed with the restart of the element.

4
The automatic restart manager should not restart the element.

Parameter List Contents

The parameter list that the system passes to the event exit routine is mapped by the IXCYEVE mapping macro. The parameter list is addressable from the primary address space in which the event exit routine runs, and includes the following:

- The type of termination for which the element is being restarted.
- The address of a copy of the exit parameter list specified in the EVENTEXITPL parameter of the IXCARM macro when this element registered.

  Note: This parameter list should contain only information that would be available from every system if this exit could get control for a cross-system restart.
- The length of the parameter list as specified in the EXITPLLEN parameter of the IXCARM macro when this element registered.
- The job name or address space name that this element had when it last registered with automatic restart management.
- The element name.
- The element type.
- The name of the system on which the element was running when the failure occurred.
- The name of the system where the element originally registered
- The name of the system on which the element will be restarted (that is, the name of the system on which this exit is running).

Gathering Statistical Data

The SMF type 30 and 90 records contain information about availability for automatic restart management services. See *z/OS MVS System Management Facilities (SMF)* for the contents of these records.

Monitoring Restarts through the ENFREQ Macro

To monitor automatic restart activity, use the ENFREQ macro to listen for ENF code 38, which MVS issues for the following events:

- An element was deregistered because of an MVS internal error.
- The restart of an element failed.
- An element issued the register, ready, or deregister request.
- Connectivity to the ARM couple data set is either established or re-established.
- An element is deregistered because of an ARM error.
- An element is deregistered because of a SETXCF FORCE,ARMDEREGISTER command.

For example, using the ENFREQ macro can help you design programs that have predecessor elements, or help you coordinate automation packages.

See *z/OS MVS Programming: Authorized Assembler Services Guide* for more information about using ENFREQ.

Displaying Information about Automatic Restart Management

To obtain information about elements of the automatic restart manager, you can issue:

- The IXCQUERY macro
- The D XCF,ARMSTATUS command
- The COUPLE subcommand of IPCS.

The information requested through any of the above methods should be filtered by using element name, restart group name, and so on. For more information about the above methods, see:

- *z/OS MVS Programming: Sysplex Services Reference* for more information about the IXCQUERY macro
- *z/OS MVS System Commands* for more information about the DISPLAY command
- *z/OS MVS IPCS Commands* for more information about the COUPLE subcommand.

All of the methods for displaying information about automatic restart management contain information about the state of the elements. An automatic restart management element can be in one of the following states, depending on the IXCARM requests it has issued or how the restart process has progressed: starting, available, available-to, failed, restarting, recovering. The states are defined as follows:

Starting
The element has initially registered but has not yet indicated it is ready to accept work.

Available
The element has indicated it is ready to accept work by issuing the IXCARM macro with the REQUEST=READY parameter.

Available-to
The element exceeded the ready timeout threshold before it issued the IXCARM macro with REQUEST=READY parameter. The system considers this element ready.

Note: For the IXCQUERY macro, elements in this state will be put with the elements in the available state. The available-to state is not applicable to status information available through the IXCQUERY macro.

Failed
The element has terminated and a restart has not been initiated by MVS, yet. This condition should apply only for a short amount of time if automatic restart management restarts have been enabled. (This state is not related to the failed state for an XCF member.)

Restarting
MVS has initiated a restart of this element, but it has not re-registered with the automatic restart manager yet.

For IXCQUERY requests issued on the system where the restart is occurring, the following information is also available from IXCYQUAA:

- QUAARMSRSTINGINERE — this bit indicates that the element is in a restarting state. Element restart exit processing is in progress. No exits may have been called, an exit may be in control, or all exits may have returned. It is possible that the restart will be vetoed.
- QUAARMSRSTINGINEVE — this bit indicates that the element is in a restarting state. The element's event exit is currently in control or has been processed. It is possible that the restart will be vetoed.
- QUAARMSRSTCOMMITED — this bit indicates that the element is in a restarting state. ARM has initiated the restart of the element by implementing the restart method.

Recovering
The element has been restarted and has re-registered with the automatic restart manager, but has not indicated that it is ready to accept work yet.

The following table summarizes the element state definitions.

Table 13. Automatic Restart Management Element States

Current® State	**Event**	**IXCARM Command Issued**	**Resultant State**
NOT DEFINED	Element successfully registers as an element	REQUEST =REGISTER	STARTING
STARTING	Element indicates ready to accept work	REQUEST =READY	AVAILABLE
FAILED	ARM starts the restart process		RESTARTING
RESTARTING	Element not yet restarted by ARM. Policy or exit vetoes the restart.		NOT DEFINED
RESTARTING	Element restarted, not yet reregistered		RESTARTING
RESTARTING	Element restarted, exceeds timeout threshold before reregistering (issuing REQUEST=REGISTER)		NOT DEFINED
NOT DEFINED	Element restarted, had exceeded timeout threshold before reregistering, and then registers	REQUEST =REGISTER	STARTING
RESTARTING	Element restarted, successfully reregisters	REQUEST =REGISTER	RECOVERING
RECOVERING	Element restarted, reregisters, and needs to wait for predecessors before being available.	REQUEST =WAITPRED	RECOVERING

Table 13. Automatic Restart Management Element States (continued)			
Current® State	**Event**	**IXCARM Command Issued**	**Resultant State**
RECOVERING	Exceeds timeout threshold before issuing REQUEST=READY	None (timed out)	AVAILABLE-TO
RECOVERING	Element indicates ready to accept work	REQUEST =READY	AVAILABLE

IBM-Supplied Automatic Restart Manager Policy Levels

The following element types are assigned by IBM. The elements will be restarted in the specified order unless the order is overridden by the RESTART_ORDER specified in the active automatic restart manager policy.

SYSLVL0
Elements to be restarted in level 0.

SYSIRLM
IRLM related elements to be restarted in level 0.

SYSLVL1
Elements to be restarted in level 1.

SYSDB2
DB2® related elements to be restarted in level 1.

SYSIMS
IMS relalted elements to be restarted in level 1.

SYSTCPIP
TCPIP related elements to be restarted in level 1.

SYSVTAM
VTAM® related elements to be restarted in level 1.

SYSLVL2
Elements to be restarted in level 2.

SYSCICS
CICS related elements to be restarted in level 2.

SYSMQMGR
IBM MQ queue manager related elements to be restarted in level 2.

SYSMQCH
IBM MQ channel initiator related elements to be restarted in level 2.

SYSKERB
SecureWay Security Server Network Authentication Service (Kerberos) related elements to be restarted in level 2.

SYSLVL3
Elements to be restarted in level 3.

SYSCB
z/OS Component Broker related elements to be restarted in level 3.

Example of Using the IXCARM Macro

```
         GBLC  &LEVEL;
&LEVEL;   SETC  '1.00'
IXCADEMO TITLE '-- Information and prologue for IXCADEMO v&LEVEL; (ARM +
               services sample program)'
IXCADEMO CSECT
IXCADEMO AMODE 31
IXCADEMO RMODE ANY
```

```
         SPACE ,
*/* START OF SPECIFICATIONS ***************************************
*
*
*01* MODULE-NAME = IXCADEMO
*
*02*   DESCRIPTIVE-NAME = Sample program to use ARM services.
*
*01* PROPRIETARY STATEMENT:
*
*     LICENSED MATERIALS - PROPERTY OF IBM
*     THIS MACRO IS "RESTRICTED MATERIALS OF IBM"
*     5655-068 (C) COPYRIGHT IBM CORP. 1994
*     SEE COPYRIGHT INSTRUCTIONS
*
*   STATUS = HBB5520
*
*01* FUNCTION =
*        Sample program to illustrate use of ARM services:  Register,
*        WaitPred, Ready and Deregister.
*
*02*   OPERATION =
*
*        1) Go to supervisor state.
*        2) Put out informational messages.  Wait on WTOR.
*        3) Issue IXCARM Request=Register.  Put out return/reason
*           codes.  Wait on WTOR.
*        4) If a restart, issue IXCARM Request=WaitPred.  Put out
*           return/reason codes.  Wait on WTOR.
*        5) Issue IXCARM Request=Ready.  Put out return/reason codes.
*           Wait on WTOR.
*        6) Issue IXCARM Request=Deregister.  Put out return/reason
*           codes.  Wait on WTOR.
*        7) Put out informational message.
*        8) Go to problem state.
*
* Example is written reentrantly.                                    *
*                                                                    *
**** END OF SPECIFICATIONS ***************************************/
         SPACE ,
*********************************************************************
*                                                                   *
* To link-edit this program, use statements like these:             *
*                                                                   *
*   //LINK   EXEC PGM=IEWL,                                        *
*   //            PARM='XREF,MAP,LIST,RENT,LET,NCAL'               *
*   //SYSLMOD  DD DSN=load_library,DISP=SHR                        *
*   //OBJECT   DD DSN=object_library,DISP=SHR                      *
*   //SYSUT1   DD UNIT=SYSDA,SPACE=(CYL,(1,1))                     *
*   //SYSPRINT DD SYSOUT=*                                         *
*   //SYSLIN   DD *                                                *
*    INCLUDE OBJECT(IXCADEMO)     object module                    *
*    ORDER   IXCADEMO             this csect first, for debugging  *
*    PAGE    IXCADEMO             page align, for debugging        *
*    ENTRY   IXCADEMO             this csect is entry point        *
*    MODE    AMODE(31),RMODE(ANY)                                  *
*    SETCODE AC(1)                                                 *
*    NAME    IXCADEMO(R)                                           *
*                                                                  *
* The load library must be an APF library.                         *
*                                                                  *
********************************************************************
         EJECT ,
         USING IXCADEMO,R15
         B     START              branch around constants
         SPACE
         DC    AL1(ENDCON-*-1)    length of constants
         DC    C' '
MODLNAME DC    C'IXCADEMO'        module name
         DC    C' V&LEVEL '       version
         DC    C'&SYSDATE '       date assembled
         DC    C'&SYSTIME '       time assembled
         DC    AL2(CSECTEND-IXCADEMO) length of CSECT
ENDCON   DS    0C
         SPACE
START    DS    0H
         STM   R14,R12,12(R13)    save caller's registers
         LR    R12,R15            load entry addr
         DROP  R15
         USING IXCADEMO,R12       permanent addressability
         SPACE
         STORAGE OBTAIN,          get working storage                +
```

```
               LENGTH=WORKLEN1,                                        +
               BNDRY=PAGE,                                             +
               LOC=(ANY,ANY)
         SPACE
         LR    R2,R1              save addr of area
         LR    R14,R1             load addr of work area to be zeroed
         L     R15,=A(WORKLEN1)   load length to be zeroed
         SLR   R1,R1              set padding byte & count to zero in  +
                                    source (R0 won't matter)
         MVCL  R14,R0             propagate X'00' from padding byte
         SPACE
         ST    R13,4(,R2)         save backward pointer to caller
         ST    R2,8(,R13)         save forward pointer to this CSECT
         LR    R13,R2             point to workarea
         USING WORKAREA,R13       save/work addressability
         SPACE ,
* Save information useful for ABEND recovery and ABEND debugging
         SPACE ,
         MVC   WORKID(L'MODLNAME),MODLNAME  copy module name
         MVC   WORKID+L'MODLNAME(L'SAVECN),SAVECN  copy rest
         ST    R12,BASESAVE       save base register
         ST    R13,SAVESAVE       save R13 of this routine
         TITLE '-- ARM-related front-end for IXCADEMO'
         MODESET MODE=SUP         get supervisor state for ARM requests
         SPACE ,
*********************************************************************
*                                                                   *
* Build and issue entry message.                                    *
*                                                                   *
*********************************************************************
         SPACE ,
         MVC   WTODYN(ENTRYMW),ENTRYM copy static message
         USING PSA,R0
         L     R1,PSAAOLD         point to ASCB
         USING ASCB,R1
         ICM   R2,15,ASCBJBNI     point to job name, if any
         BNZ   COPYNAME           skip if a non-zero address
         L     R2,ASCBJBNS        point to STC name
COPYNAME DS    0H
         MVC   SAVNAME(8),0(R2)   copy name
         MVC   WTODYN+(ENTNAME-ENTRYM)(8),SAVNAME copy name
         L     R1,PSATOLD         point to current task's TCB
         USING TCB,R1
         SLR   R2,R2              ensure high byte is zero
         ICM   R2,7,TCBJSCBB      get 24-bit JSCB address
         USING IEZJSCB,R2
         L     R2,JSCBACT         point to active JSCB
         L     R2,JSCBSSIB        point to life-of-job SSIB
         USING SSIB,R2
         MVC   SAVJESID,SSIBJBID  copy JESx id
         MVC   WTODYN+(ENTJESID-ENTRYM)(8),SAVJESID copy JESx id
         DROP  R0,R1,R2
         SPACE ,
         WTO   MF=(E,WTODYN)      say starting
         EJECT ,
         BAL   R14,SAYIT          wait for response to a WTOR
         SPACE ,
*********************************************************************
*                                                                   *
* Ask to be registered.                                             *
*                                                                   *
*********************************************************************
         SPACE ,
         IXCARM REQUEST=REGISTER,  get registered                    +
               ELEMENT=ELEMNAME,     element name                    +
               EVENTEXIT=EVTEXTNM,   event exit name                 +
               EVENTEXITPL=EVTEXTPR, event exit parameter list       +
               EXITPLLEN=EVTEXTPL,   event exit parameter list length +
               RESTARTTIMEOUT=NORM,  normal timeout interval         +
               ANSAREA=LCLANSWR,     answer area                     +
               RETCODE=SAVERC,       return code                     +
               RSNCODE=SAVERSN,      reason code                     +
               MF=(E,IXCARML)        parameter list area
         SPACE ,
* Put out return and reason codes.
         MVC   SAVESERV,=CL10'Register'
         BAL   R14,SAYRC          say RC/RSN codes
         SPACE ,
* See whether registered, perhaps as a restarted job or STC.
         CLC   SAVERC,=A(IXCARMRC4) rc=0 or =4?
         BNH   CHKREG             skip if registered or re-registered
* Something wrong, RC>4.
```

```
         EX    R0,*              cause 0C3 for dump
         SPACE ,
* Determine whether this is a new registration or a registration after
* being restarted, and act accordingly.
CHKREG   DS    0H
         LA    R2,LCLANSWR       point to answer area (not actually   +
                                   used)
         USING ARAA,R2           map answer area
         SPACE ,
         BAL   R14,SAYIT         wait for response to a WTOR
         SPACE ,
         DROP  R2
         CLC   SAVERC,=A(IXCARMRC0) rc=0 (not restarted)?
         BE    DOREADY           if yes, proceed
         EJECT ,
* This is a restart.
         BAL   R14,SAYIT         wait for response to a WTOR
         SPACE ,
*********************************************************************
*                                                                   *
* Wait for any restarted, predecessor elements.                     *
*                                                                   *
*********************************************************************
         SPACE ,
         IXCARM REQUEST=WAITPRED, wait for any predecessor elements    +
               RETCODE=SAVERC,      return code                        +
               RSNCODE=SAVERSN,     reason code                        +
               MF=(E,IXCARML)       parameter list area
         SPACE ,
         MVC   SAVESERV,=CL10'WaitPred'
         BAL   R14,SAYRC         say RC/RSN codes
         SPACE ,
         CLC   SAVERC,=A(IXCARMRC4) rc=0 or =4?
         BNH   DOREADY           skip if OK
* Something wrong, RC>4.
         EX    R0,*              cause 0C3 for dump
         EJECT ,
DOREADY  DS    0H
         BAL   R14,SAYIT         wait for response to a WTOR
         SPACE ,
*********************************************************************
*                                                                   *
* Say ready.                                                        *
*                                                                   *
*********************************************************************
         SPACE ,
         IXCARM REQUEST=READY,    say ready                           +
               RETCODE=SAVERC,      return code                       +
               RSNCODE=SAVERSN,     reason code                       +
               MF=(E,IXCARML)       parameter list area
         SPACE ,
         MVC   SAVESERV,=CL10'Ready'
         BAL   R14,SAYRC         say RC/RSN codes
         SPACE ,
         CLC   SAVERC,=A(IXCARMRC4) rc=0 or =4?
         BNH   MAINLINE          skip if OK
* Something wrong, RC>4.
         EX    R0,*              cause 0C3 for dump
         TITLE '-- Mainline for IXCADEMO'
*********************************************************************
*                                                                   *
* The real reason (whatever it is) why we're here.                  *
*                                                                   *
* The substantive application code would be here.                   *
*                                                                   *
*********************************************************************
         SPACE ,
MAINLINE DS    0H
         TITLE '-- ARM-related backend for IXCADEMO'
         BAL   R14,SAYIT         wait for response to a WTOR
         SPACE ,
*********************************************************************
*                                                                   *
* Now deregister and terminate.                                     *
*                                                                   *
*********************************************************************
         SPACE ,
         IXCARM REQUEST=DEREGISTER, get deregistered                 +
               RETCODE=SAVERC,      return code                      +
               RSNCODE=SAVERSN,     reason code                      +
               MF=(E,IXCARML)       parameter list area
         SPACE ,
```

```
         MVC   SAVESERV,=CL10'Deregister'
         BAL   R14,SAYRC          say RC/RSN codes
         SPACE ,
         CLC   SAVERC,=A(IXCARMRC0) rc=0?
         BNH   DONE               skip if OK
* Something wrong, RC>0.
         EX    R0,*               cause 0C3 for dump
         EJECT
*********************************************************************
*                                                                   *
* Terminate.                                                        *
*                                                                   *
*********************************************************************
         SPACE ,
DONE     DS    0H
         SPACE ,
*********************************************************************
*                                                                   *
* Build and issue exit message.                                     *
*                                                                   *
*********************************************************************
         SPACE ,
         MVC   WTODYN(EXITMW),EXITM copy static message
         MVC   WTODYN+(EXTNAME-EXITM)(8),SAVNAME copy name
         MVC   WTODYN+(EXTNAME-EXITM)(8),SAVNAME copy name
         MVC   WTODYN+(EXTJESID-EXITM)(8),SAVJESID copy JESx id
         SPACE ,
         WTO   MF=(E,WTODYN)
         SPACE ,
         MODESET MODE=PROB        back to problem state
         SPACE ,
         L     R2,SAVEAREA+4      save caller's R13
         LH    R11,RCHALF         save return code
         LR    R1,R13             point to working storage
         SPACE ,
         STORAGE RELEASE,         free working storage                +
               ADDR=(R1),           address of area to be freed       +
               LENGTH=WORKLEN1
         SPACE ,
         LR    R13,R2             restore caller's R13
         XC    8(4,R13),8(R13)    clear forward pointer of caller
         L     R14,12(,R13)       restore caller's registers
         LR    R15,R11            set return code
         LM    R0,R12,20(R13)     restore caller's registers
         BR    R14                return to caller
         TITLE '-- SAYIT, subroutine to wait'
*********************************************************************
*                                                                   *
* Subroutine to wait on a WTOR.                                     *
*                                                                   *
*********************************************************************
         SPACE
SAYIT    DS    0H
         STM   R0,R15,SAVEREGS    save entry registers
         SPACE
         MVC   WTODYN(WTORMWT),WTORM
         SPACE
* Convert R12 to hex-like EBCDIC.
         ST    R12,FULLWORK
         NC    FULLWORK,=X'7FFFFFFF' turn off any AMODE(31) bit
         UNPK  DBLWORK(9),FULLWORK(5)
         MVC   WTODYN+(WTORR12-WTORM)(8),DBLWORK
         TR    WTODYN+(WTORR12-WTORM)(8),TRTTABLE
         SPACE
* Convert R13 to hex-like EBCDIC.
         ST    R13,FULLWORK
         UNPK  DBLWORK(9),FULLWORK(5)
         MVC   WTODYN+(WTORR13-WTORM)(8),DBLWORK
         TR    WTODYN+(WTORR13-WTORM)(8),TRTTABLE
         SPACE
* Convert PASN to hex-like EBCDIC.
         EPAR  R1                 get PASN
         ST    R1,FULLWORK
         UNPK  DBLWORK(5),FULLWORK+2(3)
         MVC   WTODYN+(WTORASID-WTORM)(4),DBLWORK
         TR    WTODYN+(WTORASID-WTORM)(4),TRTTABLE
         SPACE
* Convert TCB address to hex-like EBCDIC.
         USING PSA,R0
         UNPK  DBLWORK(9),PSATOLD(5)
         MVC   WTODYN+(WTORTCB@-WTORM)(8),DBLWORK
         TR    WTODYN+(WTORTCB@-WTORM)(8),TRTTABLE
```

```
         SPACE
* Convert RB address to hex-like EBCDIC.
         L     R1,PSATOLD         get TCB address
         USING TCB,R1
         UNPK  DBLWORK(9),TCBRBP(5)
         MVC   WTODYN+(WTORRB@-WTORM)(8),DBLWORK
         TR    WTODYN+(WTORRB@-WTORM)(8),TRTTABLE
         DROP  R0,R1
         SPACE
* Convert point count to EBCDIC.
         L     R1,POINTCT         get current count
         LA    R1,1(,R1)            and add one
         ST    R1,POINTCT           and save
         L     R0,POINTCT
         CVD   R0,DBLWORK
         UNPK  WTODYN+(WTORPT#-WTORM)(3),DBLWORK(8)
         OI    WTODYN+(WTORPT#-WTORM)+2,C'0'
         SPACE
* Put out WTOR.
         XC    WTORECB,WTORECB    ensure ECB zero
         WTOR  ,WTORRPLY,         field to get reply                  +
               1,                   length of reply                   +
               WTORECB,             ECB that'll be waited on          +
               MF=(E,WTODYN)
         SPACE
* Wait on reply to WTOR.
         WAIT  ECB=WTORECB
         SPACE
         LM    R0,R15,SAVEREGS    load entry registers
         BR    R14                back to caller
         TITLE '-- SAYRC, subroutine to report RC/RSN codes'
*********************************************************************
*                                                                   *
* Subroutine to announce return and reason codes from an IXCARM ser-  *
* vice.                                                             *
*                                                                   *
*********************************************************************
         SPACE
SAYRC    DS    0H
         STM   R0,R15,SAVEREGS    save entry registers
         SPACE
         MVC   WTODYN(SERVMW),SERVM copy static message
         MVC   WTODYN+(SERVSRVC-SERVM)(10),SAVESERV copy service name
         SPACE
* Convert return code to hex-like EBCDIC
         UNPK  DBLWORK(9),SAVERC(5)
         MVC   WTODYN+(SERVRC-SERVM)(8),DBLWORK
         TR    WTODYN+(SERVRC-SERVM)(8),TRTTABLE
         SPACE
* Convert reason code to hex-like EBCDIC
         UNPK  DBLWORK(9),SAVERSN(5)
         MVC   WTODYN+(SERVRSN-SERVM)(8),DBLWORK
         TR    WTODYN+(SERVRSN-SERVM)(8),TRTTABLE
         SPACE ,
         WTO   MF=(E,WTODYN)      issue WTO
         SPACE
         LM    R0,R15,SAVEREGS    load entry registers
         BR    R14                back to caller
         TITLE '-- Constants'
*********************************************************************
*                                                                   *
* Constants                                                         *
*                                                                   *
*********************************************************************
         SPACE
SAVECN   DC    C' Savework Area'
         SPACE
ELEMNAME DC    CL16'IXCADEMO'     element name
EVTEXTNM DC    CL8'IEFBR14'       event exit name
         SPACE
EVTEXTPR DS    0F                 event exit parameter list
         DC    C'This is a parameter list'
EVTEXTLL EQU   *-EVTEXTPR         length of parameter list
EVTEXTPL DC    A(L'EVTEXTPR)      event exit parameter list length
         SPACE ,
* Entry message.
ENTRYM   DS    0X
         DC    AL1(0),AL1(ENTRYMW),AL2(0) WTO header
ENTRYMX  DC    C'ARMD1001I '      message prefix
         DC    C'IXCADEMO v&LEVEL in '
ENTNAME  DS    CL8                address space name
         DC    C'('
```

```
ENTJESID DS    CL8                JES id
         DC    C') starting'
ENTRYMT  EQU   *-ENTRYMX          length for TPUT
ENTRYMW  EQU   ENTRYMT+4          length for WTO
         SPACE ,
* Exit message.
EXITM    DS    0X
         DC    AL1(0),AL1(EXITMW),AL2(0) WTO header
EXITMX   DC    C'ARMD1002I '      message prefix
         DC    C'IXCADEMO v&LEVEL in '
EXTNAME  DS    CL8                address space name
         DC    C'('
EXTJESID DS    CL8                JES id
         DC    C') finishing'
EXITMT   EQU   *-EXITMX           length for TPUT
EXITMW   EQU   EXITMT+4           length for WTO
         SPACE ,
* WTOR message.
WTORM    DS    0X
         DS    2FL4               addr of reply and of ECB
         DC    AL1(0),AL1(WTORMW),AL2(0) WTO header
WTORMX   DC    C'ARMD1003I '      message prefix
         DC    C'R12='
WTORR12  DS    CL8
         DC    C', R13='
WTORR13  DS    CL8
         DC    C', ASN='
WTORASID DS    CL4
         DC    C', TCB at '
WTORTCB@ DS    CL8
         DC    C', RB at '
WTORRB@  DS    CL8
         DC    C', point '
WTORPT#  DS    CL3
         DC    C'; reply with anything'
WTORMT   EQU   *-WTORMX           length for TPUT
WTORMW   EQU   WTORMT+4           length for WTO
WTORMWT  EQU   WTORMT+12          length for WTOR
         SPACE ,
* Service (RC, RSN and type) message.
SERVM    DS    0X
         DC    AL1(0),AL1(SERVMW),AL2(0) WTO header
SERVMX   DC    C'ARMD1004I '      message prefix
         DC    C'Service = '
SERVSRVC DS    CL10
         DC    C', RC='
SERVRC   DS    CL8
         DC    C', RSN='
SERVRSN  DS    CL8
SERVMT   EQU   *-SERVMX           length for TPUT
SERVMW   EQU   SERVMT+4           length for WTO
         SPACE ,
* The following has to be at least 240 bytes into the CSECT
TRTTABLE EQU   *-240
         DC    C'0123456789ABCDEF'
         TITLE '-- Literals'
*********************************************************************
*                                                                   *
* Literals                                                          *
*                                                                   *
*********************************************************************
         SPACE ,
         LTORG
         TITLE '-- Save/work area'
*********************************************************************
*                                                                   *
* Save/work area DSECT                                              *
*                                                                   *
*********************************************************************
         SPACE
WORKAREA DSECT
SAVEAREA DS    18F                register save area
WORKID   DS    CL(L'MODLNAME+L'SAVECN)  EBCDIC identifier
         DS    0D                 alignment
BASESAVE DS    A                  saved base register of IXCADEMO
SAVESAVE DS    A                  saved R13 of caller
SAVEREGS DS    16F                savearea for subroutines
         SPACE
*********************************************************************
*                                                                   *
* IXCARM's parameter list area.                                     *
*                                                                   *
```

```
*************************************************************************
         SPACE
         IXCARM MF=(L,IXCARML)
         SPACE
FULLWORK DS    F
SAVERC   DS    F                 return code from IXCARM service
SAVERSN  DS    F                 reason code from IXCARM service
WTORECB  DS    F                 ECB for WTOR/WAIT
POINTCT  DS    F                 WTOR/WAIT point number
FLAGS    DS    0F
FLAG1    DS    X
FLAG2    DS    X
FLAG3    DS    X
FLAG4    DS    X
WTORRPLY DS    X                 1-byte reply area for WTOR
RCHALF   DS    H                 return code halfword
         ORG   *-1
RC       DS    X                 return code
SAVNAME  DS    CL8               job/STC name
SAVJESID DS    CL8               JESx id
         SPACE
LCLANSWR DS    XL32              answer area
SAVESERV DS    CL12              service name for message
         DS    0F                alignment for WTODYN
WTODYN   DS    CL136
         DS    0D                doubleword align
DBLWORK  DS    CL16
         DS    0D                doubleword align end of WORKAREA
         SPACE
WORKLEN1 EQU   *-WORKAREA         length of workarea
         TITLE '-- DSECTs and EQUs'
*************************************************************************
*                                                                       *
* DSECTs, EQUs & whatnot                                                *
*                                                                       *
*************************************************************************
         SPACE
         PRINT NOGEN
         YREGS ,                  register EQUs
         IHAPSA ,                 PSA mapping
         IHAASCB ,                ASCB mapping
         IKJTCB ,                 TCB mapping
         IEZJSCB ,                JSCB mapping
         IEFJSSIB ,               SSIB mapping
         IXCYARM ,                ARM return and reason codes
         IXCYARAA ,               ARM answer area mapping
         SPACE
IXCADEMO CSECT                    ensure resumed CSECT
CSECTEND DS    0D
         END
```

Part 4. Sysplex Services for Data Sharing (XES)

© Copyright IBM Corp. 1994, 2023

Chapter 5. Introduction to Sysplex Services for Data Sharing (XES)

Sysplex services for data sharing allow subsystems, system products and authorized applications running in a sysplex use a coupling facility for high-performance, high-availability data sharing. Sysplex services support data sharing while maintaining data integrity and consistency by:

- Allowing users to store and access data in a coupling facility in any of three types of structures (list, lock, or cache).
- Guaranteeing that individual operations on coupling facility data are either completed or, if necessary, backed out to their original state. Users are prevented from accessing data that is being changed.
- Providing services to help users protect data when recovering from a failure.
- Enabling users to ensure that their local copies of shared data are valid.
- Allowing users who change shared data to automatically notify other users that their local copies are no longer valid.
- Providing functions that allow users to create a customized set of locks and locking protocols including:
  - Application-defined:
    - Resource locks
    - Lock states
    - Lock state compatibility rules
  - A mechanism to allow users to resolve lock contention. When contention arises for a lock, the system passes control to the lock owner's contention exit to resolve the lock contention according to the user's defined protocols.
  - Support of failure recovery options through the retention of lock-related information that will persist across system or sysplex outages.

Coupling Facility Structures

Instead of accessing data in a coupling facility by address, you can allocate three types of objects, called **structures**, and access data in the structures as logical entities (by name, for instance). The ability to access data in this manner frees coupling facility users from having to be concerned with the physical location or address of the data.

Each type of structure, described in detail in “Types of Coupling Facility Structures” on page 199, provides a unique set functions and offers a different way of using a coupling facility. The types of structures are:

- Cache structure
- List structure
- Lock structure

A coupling facility can hold one or more structures of any type, however, each structure must reside entirely in a single coupling facility. Applications are not limited to using a single coupling facility structure. They can use multiple structures of the same type or different types.

Products and subsystems that exploit the coupling facility indicate their coupling facility structure requirements as part of their installation information. For instance, a product might require a lock structure of a certain size, with particular attributes, and a certain name. Or, a product might require that a structure be allocated in a certain level (CFLEVEL) of a coupling facility because of the functionality it provides.

When system administrators or system programmers install software that requires a coupling facility structure, they create a coupling facility resource management (CFRM) policy that specifies the name, size, and attributes of each structure to be allocated. The CFRM policy also allows the installation to limit

© Copyright IBM Corp. 1994, 2023

the amount of storage each structure can occupy and control where each structure is allocated, through a "preference list", for multiple coupling facilities.

Once the policy is defined, the operator needs to issue the SETXCF command to activate the policy. The activated policy does not cause the structures to be allocated. A structure is allocated only when the first user connects to the structure.

For More Information

To learn more about the following coupling facility topics, see *z/OS MVS Setting Up a Sysplex*:

- What is a coupling facility?
- What is the role of the coupling facility in a sysplex?
- How does an installation define coupling facility structures?
- What is a CFRM policy and how does an installation define one?
- What are the hardware requirements for the coupling facility?
- What are the software requirements for the coupling facility?
- What are the planning considerations for using a coupling facility?

For additional information about the coupling facility, see:

- Parallel Sysplex (www.ibm.com/systems/z/advantages/pso)
- *PR/SM Planning Guide*

Data Sharing Concepts and Terminology

Data sharing in a sysplex refers to the ability of concurrent subsystems (such as DB2 or IMS DB) or applications to directly access and change the same data while maintaining data integrity and consistency throughout the sysplex.

In this book, the following terms are used:

- **Data** refers to any type of information, not only information contained in a data base.
- **Application** refers to any subsystem, system product, or authorized application running on MVS in a multisystem environment or sysplex.

  Typically, multiple instances of the application, distributed across the sysplex, work together to perform a set of functions. For example, a data base product could be installed on several systems in a sysplex. On each system, an instance of the application accesses and manipulates data that it shares with the other instances of the application.
- **User** refers to an application or an instance of an application using sysplex services to access a coupling facility structure. Because users *connect* to a structure to access it, users are also referred to as **connections** or **connected users**.

Figure 17 on page 199 shows a schematic diagram of a coupling facility with connected users.

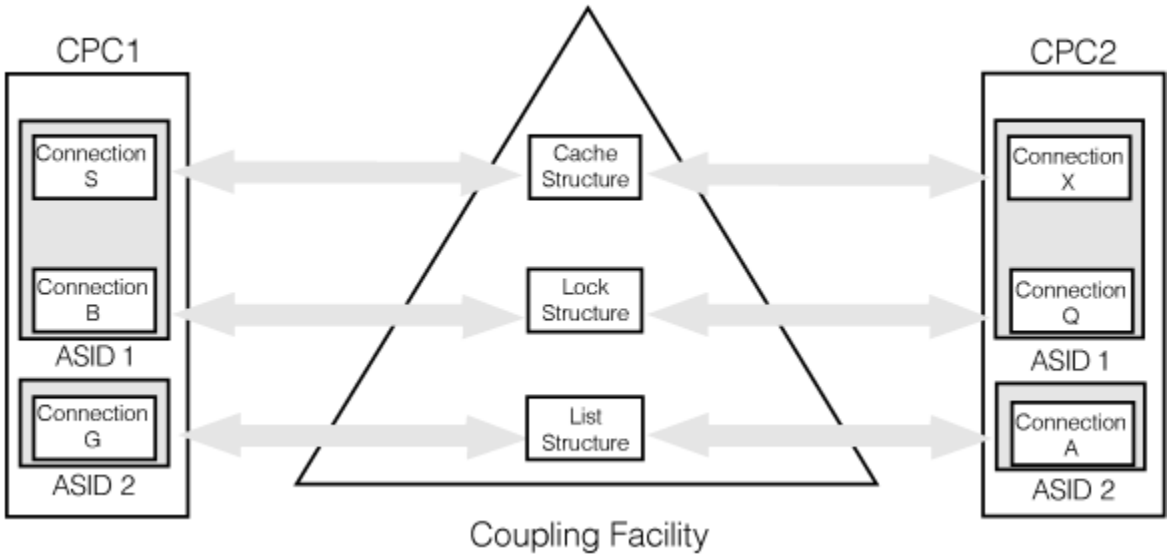


Figure 17. Multiple Systems Sharing Data Through a Coupling Facility

The Coupling Facility from the Point of View of the Programmer

To a programmer, a **coupling facility** can be viewed as shared storage that is directly accessible to connections distributed throughout the sysplex. Connections using a coupling facility must reside on systems with a direct attachment to the coupling facility.

Sysplex services for data sharing allow you to:

- Store and access data in the coupling facility.
- Choose synchronous or asynchronous structure operations, with a variety of options for handling operation completion.
- Tailor structures and services for your specific needs by customizing your structure and by coding exits to respond to events and make decisions.
- Use multiple structures, either for different purposes within the application or to implement a complex function. For instance, an application could use a lock structure to implement a serialization mechanism for use with a cache structure
- Specify whether a structure is to be deallocated if there are no active users connected to it or whether the structure is to remain allocated until it is explicitly deallocated.
- Relocate a structure elsewhere in the same coupling facility or in a different one.
- Change the size of a structure and/or reapportion the use of structure storage based on application growth or workload variations.
- Obtain diagnostic information about a coupling facility and its structures.

Types of Coupling Facility Structures

The characteristics and services associated with each structure support certain types of uses and offer certain unique functions:

Cache structure
Allows high-performance sharing of frequently-referenced data. Cache structure services, accessed through the IXLCACHE macro, allow you to:

- Store and access data in the cache structure.
- Automatically notify affected users when you change shared data in the cache system. The system keeps track of which users are using a particular piece of data and notifies those users when an update to the data makes their locally-cached version obsolete.

- Determine whether your copy of shared data is valid by checking system-maintained validity indicators for your locally-cached copies of shared data.

Certain functions provided by cache structure services depend on the level of a coupling facility in which the cache structure is allocated.

List structure
Enables users to share information organized as entries on a set of lists or queues. Connections could use a list structure, for example, to distribute work or maintain shared status information.

List structure services, accessed through the IXLLIST, IXLLSTC, IXLLSTE, and IXLLSTM macros, allow you to:

- Read, write, move, and delete list entries in a variety of ways, with and without serialization.
- Monitor list transitions from empty to non-empty without accessing the coupling facility and checking the lists directly.
- Define a lock table of exclusive locks as part of the list structure. You can use the lock table to serialize access to lists, list entries, or any other resources in the list structure.

Certain functions provided by list structure services depend on the level of a coupling facility in which the list structure is allocated.

Note: As of OS/390 Release 9, functional enhancements will be made to the IXLLSTC, IXLLSTE, and IXLLSTM macros only. The IXLLIST macro will be maintained, but will not be updated with any new support.

Lock structure
Allows users to create a customized set of locks and locking protocols for serializing user-defined resources, including list or cache structure data.

You can implement a serialization mechanism with any scope you require, thereby reducing contention for resources. For instance, rather than serializing at a data set level, you can use the lock structure to serialize access at the record or field level.

Lock structure services, accessed through the IXLLOCK macro, allow you to:

- Associate user-specified data with each lock. IXLLOCK supports shared and exclusive lock states. However, you can use the user-specified data to create additional lock states to tailor the locks to your application's needs.
- Implement customized locking protocols for your user-defined lock states.
- Resolve lock contention according to your own protocol by providing exits to handle contention resolution. The system assists in contention resolution by supplying your exit with information about the cause of the contention.
- Recover locks as part of an overall recovery mechanism to recover for the failure of another connector.

Certain functions provided by lock structure services depend on the level of a coupling facility in which the lock structure is allocated.

Using Sysplex Services for Data Sharing

This topic provides an overview of what's involved in creating or modifying an application to use a coupling facility structure. To have your application share data using a coupling facility structure, you need to:

- Create an application that shares data using sysplex services.
- Have the installation running your application create a CFRM policy that defines any coupling facility structures your application needs. You must provide the installation with information about the attributes and size of the structure your application requires.

Designing Your Application to Exploit the Coupling Facility

The process of designing your application to exploit a coupling facility involves the following tasks. If you plan to use more than one structure, you need to perform the tasks listed below for each structure.

- Select the type of coupling facility structure that fits your application.
- Study the attribute options for the structure and the functions provided by its associated sysplex services. These functions should include those provided by a particular coupling facility level (CFLEVEL).
- Determine:
  - The way your application will exploit the structure and its functions
  - How you will organize your data in the structure
  - The structure attributes you require
  - The structure size you require.
- Address issues such as serialization that relate to the shared use of the structure among multiple users. A coupling facility offers several ways to establish and maintain locking protocols for resources; these include actual locks as well as user-defined fields (such as the version number field in each list structure entry) that could be used to provide serialization.
- Understand timing issues relating to asynchronous processing of multiple, concurrent requests.
- Understand the events about which your application will be notified and decide how your application will respond to each event. When you connect to a structure, you provide the address of an event exit you have coded. Your event exit gets control from the system to receive information about an event. Events generally relate to a change in user, structure, or coupling facility status or to an error condition.
- Understand the exits you must code for the structure you are planning to use and decide what processing they should perform when they receive control. Sysplex services rely heavily on application-provided exits, to allow decision-making for critical events to be tailored to the application. Exits can also be used to receive notification of asynchronous request completion.
- Determine how your application will respond when a structure becomes full. The size of a structure in a coupling facility with CFLEVEL=1 or higher can be altered in its current location or the structure can be rebuilt with a larger size in another location.
- Plan whether your application will support system-managed processes (for example, rebuild).
- Plan how your application will monitor and control structure utilization, and how it will decide when to rebuild or change the size of the structure to increase capacity.
- Plan how your application will implement peer recovery, restart recovery, or both, in the event of:
  - User failure
  - Connectivity failure
  - Coupling facility structure failure.

  For peer recovery, XES services notify peer users when a user fails. Peer users can perform recovery, clean up resources, and decide whether the failed user will be permitted to reconnect to the structure.

  For restart recovery, XES services enable users to re-establish connection to a structure after a failure.
- Document the following requirements in your application's installation instructions:
  - The number and types of structures needed
  - The structure sizes and attributes
  - Whether the structures should be distributed across multiple coupling facilities, and if so, how they should be distributed. For optimum performance and availability, installations should spread coupling facility structures across multiple coupling facilities
  - Whether any structures cannot share the same coupling facility, for capacity, performance or availability reasons. Make the necessary coupling facility resource management (CFRM) policy exclusion list recommendations for such structures.

- Whether any structures have CFLEVEL requirements. Make the necessary CFRM policy preference list recommendations for such structures.
- Whether the structure can be rebuilt or have its size altered.

The topics following this overview of sysplex services for data sharing are intended to help you to address these design considerations. However, certain design decisions are application-specific, so it is not always possible to recommend a particular approach or protocol.

Managing recovery in a sysplex

To align with the need for recovery management in a Parallel Sysplex® environment, the IXCCFCM service is provided in z/OS V1R8 (and systems at z/OS V1R5 and higher with APAR OA11719 installed). IXCCFCM provides the framework by which coupling facility structure duplexing can provide redundancy for disaster recovery when a recovery manager is used. A recovery manager (for example, Geographically Dispersed Parallel Sysplex (GDPS®) can use the IXCCFCM interface to provide recovery status information to CFRM. The recovery manager is the piece of a disaster recovery solution that manages recovery procedures for a potential site failure. By making recovery site information available to the CFRM policy, an installation can automate its actions when disaster recovery is required. See *z/OS MVS Programming: Sysplex Services Reference*.

Guide to Sysplex Services Topics

Now that you have been introduced to sysplex services for data sharing, you can read the succeeding chapters for more information. Following this introduction are chapters devoted to each of the structures and their associated macros. These are:

- Chapter 7, “Using Cache Services (IXLCACHE),” on page 357
- Chapter 8, “Using List Services (IXLLIST),” on page 481
- Chapter 9, “Using List Services (IXLLSTE, IXLLSTM, IXLLSTC),” on page 595
- Chapter 10, “Using Lock Services (IXLLOCK),” on page 625

Once you have a basic understanding of at least one of the three structures and its services, you are ready to learn about connection services. The connection services chapter explains how to:

- Define the attributes of a structure
- Define how long a structure will persist in the coupling facility
- Connect to a structure
- Plan for collecting diagnostic information
- Delete a structure
- Rebuild a structure
- Alter the size or reapportion the storage of a structure
- Disconnect from a structure
- Respond to connection events using your event exit.

Chapter 11, “Supplementary List, Lock, and Cache Services,” on page 679, covers the macros that you use with the structure-associated macros to perform related functions.

Chapter 15, “Documenting your Coupling Facility Requirements,” on page 767 provides a checklist of the information you must provide to the users of your application or subsystem.

Chapter 6. Connection Services

MVS provides services that allow authorized programs and subsystems to use the coupling facility to share data in a sysplex. This chapter discusses coupling facility services that manage connections to coupling facility structures and includes the following information about connection services:

- Connecting to a coupling facility structure and causing allocation of the structure in a coupling facility
- Disconnecting from a coupling facility structure and causing deallocation of the structure in a coupling facility
- Participating in structure rebuild processing for coupling facility structures
- Altering the size of coupling facility structures or, if applicable, the ratio of entries to elements or the amount of storage allocated for event monitor control objects within a structure
- Communicating events about coupling facility structures to all users
- Deleting coupling facility structures.

A coupling facility structure is a named piece of storage in a coupling facility. MVS services support three types of structures — cache, list, and lock, each of which provides unique functions in a data sharing environment. You connect to a coupling facility structure in order to use the MVS services to manipulate or manage data within the structure.

The first user to successfully connect to a structure allocates the structure in the coupling facility and defines the structure attributes, including the type of structure. Other users can connect to the structure by name but cannot change the attributes of the structure as long as the structure remains allocated. Depending on the application protocol, however, it is possible through user-managed rebuild for connected users to rebuild the structure with different attributes.

Guide to the Topics

The following topics are presented to help you understand the connection services that you use to access a coupling facility.

- "Overview of Connection Services" on page 203
- "Structure Concepts" on page 205
- "Connecting to a Coupling Facility Structure" on page 222
- "Structure Rebuild Processing" on page 264
- "Responding to Connection Events" on page 333
- "Using IXLUSYNC to Coordinate Processing of Events" on page 344
- "Disconnecting from a Coupling Facility Structure" on page 346
- "Forcing the Deletion of a Coupling Facility Object" on page 350
- "Coding Exit Routines for Connection Services" on page 352

Overview of Connection Services

The coupling facility storage can contain three types of structures, each of which has its own set of services. To access these services, you must first "connect" to a structure, specifying both a name and a structure type. The name you provide for the structure when you connect is the same name that appears in the active coupling facility resource management (CFRM) policy governing the installation's use of the coupling facility. The IXLCONN macro is the service that allows you to connect to a structure.

When you no longer need access to a coupling facility structure, you can disconnect from the structure. The IXLDISC macro is the service that allows you to disconnect from a structure. In order to access the structure at some later time, you must again connect to the structure using the IXLCONN macro.

© Copyright IBM Corp. 1994, 2023

Sysplex-wide information about connectors to a structure is made known throughout the sysplex through each connector's exits. The system uses the event exit to notify you about new connections to a structure, disconnections, loss of connectivity or failure of a structure, synchronization points when a structure is being rebuilt, other user-defined synchronization points, and changes in the volatility state of the coupling facility. How you respond to these events depends on the type of event — some events require that you respond through a macro invocation, IXLEERSP, while other events require only that you set a return code in a parameter list that your event exit accesses.

Other structure-specific information is made known to connectors through additional exits, which, if applicable, you specify when you connect to a structure. The complete exit, which applies to all structure types, notifies you when a request that you submitted previously has completed. The notify exit, which applies only to serialized structures — lock and serialized list, may be used when contention for a resource occurs. The contention exit, which applies only to a lock structure, is used to manage resource contention. The list transition exit, which applies only to a list structure using list monitoring, notifies you when a list has changed from an empty to a non-empty state. Each exit type references a parameter list with which you can communicate to the system and to your peer connections. (A peer connection is another user connected to the same structure.)

For planned reconfiguration, recovery, and improved availability and usability, three additional connection services are available — IXLREBLD, IXLUSYNC, and IXLALTER.

The IXLREBLD service is for structure rebuild processing. Since its initial availability, structure rebuild processing has evolved to consist of two types, rebuild and duplexing rebuild, and two methods of achieving the specific type of structure rebuild processing, user-managed and system-managed.

- Rebuild — A procedure to construct a new instance of a named structure. The new instance can be in the same coupling facility as the old instance or in another coupling facility.
- Duplexing Rebuild — A procedure to create and maintain two instances of a named structure, referred to either as the old and new instances or the primary and secondary instances. The method used governs which structure types can be duplexed.
- User-managed — A method in which the connector(s) to the named structure must participate in the defined protocol to accomplish the type of procedure (rebuild or duplexing rebuild). The protocol includes specification of IXLCONN keywords, using IXLREBLD to participate in the processing, and understanding events associated with the processing. The connector is responsible for construction of the new instance and maintaining the duplicate data for a duplexed structure.
- System-managed — A method in which the connector(s) to the named structure must participate in the defined protocol to accomplish the type of procedure (rebuild or duplexing rebuild). The protocol includes specification of an IXLCONN keyword and understanding events associated with the processing. The system, not the connector, is responsible for propagating data from the old instance to the new instance and for a duplexed structure, maintaining duplicate data.

The IXLREBLD service can be used to start or stop either type of structure rebuild processing. It also provides the interface for the user-managed protocol. See “Structure Rebuild Processing” on page 264.

IXLUSYNC allows for defining user-defined synchronization points not only in recovery scenarios, but as your application requires.

The IXLALTER service allows you to dynamically change the size of a structure and/or the apportionment of structure storage while connectors continue to use the structure.

For cleanup processing, the IXLFORCE service allows you to delete structure resources in a coupling facility.

IXLPURGE is used to complete outstanding operations against a coupling facility structure.

Authorizing Coupling Facility Requests

The security administrator might want to protect the integrity of the data within the structure before coupling facility requests, such as IXLCONN, IXLREBLD, and IXLFORCE are issued. If the z/OS Security Server, which includes RACF, or another security product is installed, the administrator can define profiles that control the use of the structure in the coupling facility.

The following steps describe how the RACF security administrator can define RACF profiles to control the use of structures:

1. Define resource profile IXLSTR.structure-name in the FACILITY class.
2. Specify the users who have access to the structure using the RACF PERMIT command.
3. Make sure the FACILITY class is active, and generic profile checking is in effect. If in-storage profiles are maintained for the FACILITY class, refresh them.

For example, if an installation wants to permit an application with an identifier of SUBSYS1 to issue the IXLCONN macro for structure-name CACHE1, the security administrator can use the following commands:

```
RDEFINE FACILITY IXLSTR.CACHE1 UACC(NONE)

PERMIT IXLSTR.CACHE1 CLASS(FACILITY) ID(SUBSYS1) ACCESS(ALTER)

SETROPTS CLASSACT(FACILITY)
```

You can specify RACF userids or RACF groupids on the ID keyword of the PERMIT command. If RACF profiles are not defined, the default allows any authorized user or program (supervisor state and PKM allowing key 0-7) to issue coupling facility macros for the structure.

For information about RACF, see *z/OS Security Server RACF Security Administrator's Guide*.

Structure Concepts

Whether a structure is defined as a cache, list, or lock structure, certain characteristics are common to all types. The following topics provide basic information about all types of structures:

- "Defining the Structure Attributes" on page 205
- "Identifying Connection States" on page 205
- "Understanding Connection Persistence and Structure Persistence" on page 207

"Allocating a Structure in a Coupling Facility" on page 208 provides information about how the system handles a request for structure allocation. You need to understand this to provide planning information for your users when they use your coupling facility application.

Defining the Structure Attributes

When using IXLCONN to connect to a structure, you specify structure attributes that describe the structure that you need. Whether the attributes you specify are used by the system depends not only on your IXLCONN parameters, but also on resource availability in the coupling facility, what the installation has defined in its CFRM policy, and whether your IXLCONN request causes the allocation of the structure.

The structure to which you receive connectivity might or might not meet all your requirements. The system returns the actual attributes of the structure to you in the connect answer area, mapped by the macro IXLYCONA. It is your responsibility to verify that the attributes of the structure, as indicated in the answer area, are acceptable. If you decide not to accept one or more of the attributes, you can disconnect from the structure or attempt to rebuild it with different attributes.

The attributes discussed here are generic for each structure type. There are additional attributes that are specific to the type of structure. For a description of the information required on IXLCONN for each structure type, see "Connecting to a Coupling Facility Structure" on page 222.

Identifying Connection States

A connection to a coupling facility structure might be in one of four states, as defined below. You can use the IXCQUERY macro and the DISPLAY XCF operator command to determine the state of a connection.

- Undefined state — The connection does not exist.
- Active state — The connection is active.

- Failed-persistent state — The connection has abnormally terminated or disconnected with REASON=FAILURE and all event exit responses have been received. All event exit responses from peer connections indicated that the connection should not be released.
- Disconnecting or failing state — The connection has disconnected with REASON=NORMAL or REASON=DELETESTR (disconnecting state) or has been abnormally terminated or disconnected with REASON=FAILURE (failing state). All event exit responses have not yet been received for the disconnection or failure of the connection.

  If a user issues IXLCONN with the same connection name as the connection in the disconnecting or failing state, IXLCONN rejects the request with reason code IXLRSNCODERSPNOTREC. (See the IXLYCON macro for a description of all XES reason codes.)

  While the connection is in the disconnecting or failing state, you cannot force the connection with the IXLFORCE service or the SETXCF FORCE command.

  When all event exit responses are received, the connection is placed either in the undefined state (the connection does not exist) or the failed-persistent state.

  – Undefined state

    1. The connection disposition is delete, or
    2. The connection disconnected with REASON=NORMAL or REASON=DELETESTR, or
    3. The connection disposition is keep and the connector terminated abnormally or disconnected with REASON=FAILURE, and *any* peer connection indicated that the connection could be released.

  – Failed-persistent state

    The connection disposition is keep and the connector terminated abnormally or disconnected with REASON=FAILURE, and *all* peer connections indicated that the connection should not be released.

Figure 18 on page 207 shows the events that can cause a connection to change from one state to another.

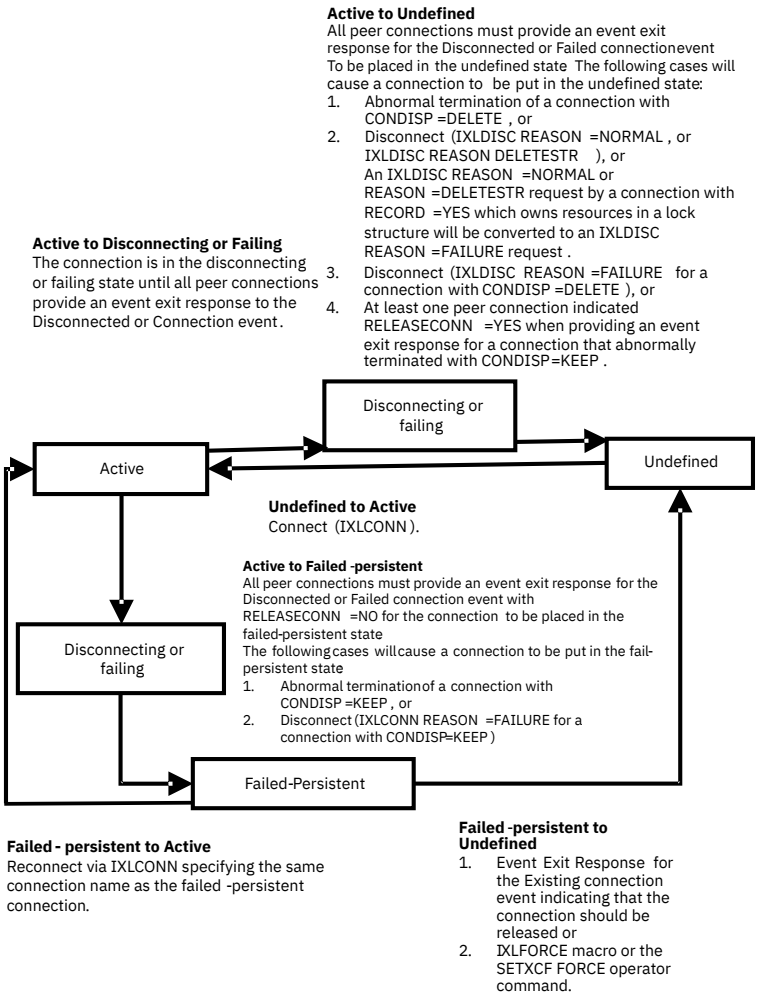


Figure 18. Connection State Transitions: Undefined, Active, Disconnecting, Failing

Understanding Connection Persistence and Structure Persistence

The attribute of persistence applies both to structures and to connections to a structure. Both structure persistence and connection persistence are specified at connect time.

Connection Persistence

The connection disposition (CONDISP) allows you to specify how to handle a connection to a coupling facility structure that ends abnormally. The connection disposition determines in the event of a failure whether or not the connection remains defined to the structure. (A failed connection can be the result of a task, address space, or system failure, or REASON=FAILURE on the IXLDISC macro. See "Disconnection Because of Failure" on page 348.)

A connection disposition of KEEP indicates that when the connection fails, the failed connection remains defined as a failed-persistent connection and the structure remains allocated. For a connection disposition of KEEP, you must specify a connect name (CONNAME) on IXLCONN. The system uses the connect name to determine when you can reestablish the connection after a failure has occurred. CONNAME uniquely identifies your connection to the structure.

If a connection with a disposition of KEEP fails, the system considers the connection to be in a failed-persistent state. The connection remains in that state until a new instance of the failed connection, a peer connection, or an operator or program take actions to change it. When the application restarts and reissues an IXLCONN request for the structure, the same CONNAME that was specified on the previous IXLCONN request must be specified. If the reconnection is successful, IXLCONN returns a return code

X'4'. The CONARECONNECTED flag in the connect answer area (IXLYCONA) is set to indicate that the connection has been reestablished.

A peer connection can indicate after recovery processing that a connection should no longer be failed-persistent by issuing IXLEERSP or by setting a return code in the event exit parameter list (IXLYEEPL). See "Deleting Failed-Persistent Connections" on page 262. Once all peer connections have completed their recovery processing for the failed connection and have responded to the Disconnect/Failed User event, the failed-persistent connection can be deleted. The operator can use the SETXCF FORCE command or an authorized program can issue the IXLFORCE macro to delete a failed-persistent connection.

A connection disposition of DELETE indicates that the connection should become undefined to MVS in the event of a failure. If the connection disposition is DELETE, then you are not required to specify CONNAME; however, if you do not provide a connection name, MVS generates one.

If the connection terminates normally (disconnect with REASON=NORMAL or REASON=DELETESTR), the persistence attribute for the connection does not apply, and so the connection becomes not defined. However, if a connector with RECORD=YES to a lock structure disconnects with REASON=NORMAL or REASON=DELETESTR while still owning resources associated with the lock structure, XES converts the reason to REASON=FAILURE.

Structure Persistence

The persistence attribute of a structure is affected both by how you define your structure disposition and the disposition of the connections to the structure.

The structure disposition (STRDISP) determines whether or not the structure remains allocated when there are no active or failed-persistent connections to the structure. A structure disposition of KEEP indicates that when there are no active or failed-persistent connections to the structure, the structure remains allocated. For example, if data in the structure needs to be kept permanently in the coupling facility, you should specify a disposition of KEEP. A structure that remains allocated when there are no active or failed-persistent connections is called a persistent structure. The operator or an authorized program can use the SETXCF FORCE command or the IXLFORCE macro to delete a persistent structure in some instances. On systems with OW33615 installed or which are at OS/390 Release 9 or higher, when connectivity to a coupling facility is lost by all systems and the system receives a request to allocate and connect to a structure in that coupling facility, the system will FORCE all failed-persistent connectors to the structure as well as the structure itself. This allows the connector to connect successfully to a new instance of the structure, so that it can perform whatever recovery actions are necessary to ensure data consistency and then begin to provide service making use of the newly allocated structure instance. See "Deleting Persistent Structures" on page 262.

A structure with a disposition of KEEP can also be deleted if the last remaining connector to the structure disconnects with REASON=DELETESTR on the IXLDISC request. See "Disconnection to Delete the Structure" on page 348.

A structure disposition of DELETE indicates that when there are no active or failed-persistent connections to the structure, the structure is deallocated. However, if there are any active or failed-persistent connections to the structure, the structure remains allocated.

Note that you can determine the persistence attribute of both a structure and a connection with the IXCQUERY macro.

Allocating a Structure in a Coupling Facility

The allocation of a structure in a coupling facility depends on several factors — application requirements, installation requirements, and availability of coupling facility storage. The application request to allocate a coupling facility structure (through the IXLCONN macro) may rely on the installation's specifications for the coupling facility's use as defined in the active coupling facility resource management (CFRM) policy. (An authorized application can query the CFRM policy by using the IXCQUERY macro.) The application's request for structure allocation, combined with the coupling facility control code's storage utilization requirements, ultimately determine if, where, and how large a structure is allocated.

Specifying the Required Coupling Facility Attributes

The application, on its IXLCONN invocation, specifies certain coupling facility attributes required for its structure. The application also must document these requirements for users of the application, so that the installation can properly configure its coupling facilities.

Attributes that the application can specify are:

- A connectivity requirement
- The level of coupling facility
- A volatility requirement
- A failure-independence requirement

Attributes that the installation controls are:

- The preference and exclusion lists

Note that both the application and the installation can specify the required size of the structure to be allocated, which can also affect the choice of coupling facility. See "Specifying the Structure Size" on page 212 and "Coupling Facility Considerations When Allocating a Structure" on page 218 for information about how the size of the for information about how the size of the structure is determined and how coupling facility resources are allocated to the structure.

Specifying a Connectivity Requirement

An application can specify its connectivity requirements with the CONNECTIVITY keyword on the IXLCONN macro when connecting to a structure. An application can specify that it requires a structure to be allocated in a coupling facility that has connectivity to all systems in the sysplex, that has connectivity to the best subset of systems in the sysplex based on SFM weights, or that most closely meets the coupling facility attributes requested.

The CONNECTIVITY keyword applies only to the IXLCONN request that causes the structure to be allocated, the first connector to the structure. The system ignores the connectivity requirement specified by subsequent connectors to the structure. See "Selecting a Coupling Facility for Structure Allocation" on page 215 for information about how the system uses the CONNECTIVITY keyword.

Specifying a coupling facility level requirement

Prior to OS/390 Release 9, an application specifies its coupling facility operational level requirements with the CFLEVEL keyword on the IXLCONN macro. MVS attempts to allocate a structure in a coupling facility of the CFLEVEL requested, that is, a coupling facility that provides at least the level of architected function that the user has requested. If necessary, the structure will be allocated in a coupling facility with a CFLEVEL lower than requested. If the structure is already allocated, the CFLEVEL is ignored. Upon successful connection to the structure, the connect answer area contains the CFLEVEL of the coupling facility in which the structure was allocated. It is the responsibility of the connector to check this field (CONACFACILITYCFLEVEL) and verify that the level is acceptable.

You should specify the lowest possible CFLEVEL on IXLCONN that will provide the required functions. This will allow the space in the higher level coupling facilities to remain available for applications that require the coupling facility functions supported only by those levels.

When exploiting a system-managed process, connectors should not specify the CFLEVEL required by the system-managed process simply because they have specified ALLOWAUTO=YES. For example, connectors should not specify CFLEVEL=8 because they support system-managed rebuild. The system will automatically attempt to allocate the structure in a coupling facility of the necessary CFLEVEL when the connector specifies ALLOWAUTO=YES.

When exploiting system-managed asynchronous structure duplexing, connectors should not specify the CFLEVEL required by system-managed asynchronous structure duplexing simply because they have specified ASYNCDUPLEX=YES. For example, connectors should not specify CFLEVEL=21 because they support system-managed asynchronous structure duplexing. The system will automatically attempt to

allocate the structure in a coupling facility of the necessary CFLEVEL when the CFRM policy requests system-managed asynchronous structure duplexing.

Identifying a minimum coupling facility level

An application can specify the minimum coupling facility level in which a structure can be allocated. The value of the IXLCONN MINCFLEVEL keyword must be equal to or less than the value specified by the CFLEVEL keyword. If MINCFLEVEL is greater than CFLEVEL, the system rejects the IXLCONN request with reason code IXLRSNCODEBADMINCFLEVEL. The specification of MINCFLEVEL also might prevent an IXLCONN request to connect to an existing structure from completing successfully. If the existing structure is allocated in a coupling facility that is at a lower coupling facility level than the value specified by MINCFLEVEL, the system rejects the IXLCONN request with reason code IXLRSNCODEINSUFFCFLEVELUSER. See "Requesting a Minimum CFLEVEL" on page 224.

The following minimum coupling facility levels support XES functions:

CFLEVEL=1

- Maximum of 255 data elements per data item
- IXLALTER requests for altering structure size and/or entry-to-element ratio
- IXLLIST request types that support entry version number comparison, automatic list key assignment, list cursor manipulation, entry key comparison, and conditional processing based on list authority.

CFLEVEL=2

- IXLCACHE REQUEST=REG_NAMELIST
- IXLCACHE REQUEST=WRITE_DATA, WHENREG=YES, with VECTORINDEX specified.
- IXLLOCK REQUEST=PROCESSMULT for batched release requests.

CFLEVEL=3

- IXLLIST request types that support event queues and their use for sublist monitoring. The request types include:
  - IXLLIST REQUEST=MONITOR_SUBLIST
  - IXLLIST REQUEST=MONITOR_SUBLISTS
  - IXLLIST REQUEST=MONITOR_EVENTQ
  - IXLLIST REQUEST=READ_EQCONTROLS
  - IXLLIST REQUEST=READ_EMCONTROLS
  - IXLLIST REQUEST=DEQ_EVENTQ

CFLEVEL=4

- IXLALTER requests for altering percentage of list structure storage that is allocated for event monitor controls.
- IXLCACHE REQUEST=UNLOCK_CO_NAME to unlock a single castout lock.
- IXLCACHE REQUEST=READ_DATA,RETURNDATA=YES|NO to register interest in an entry without returning the associated data.
- IXLCACHE REQUEST=WRITE_DATA, with no data written.
- Support for dumping structures that contain event monitor controls.
- Performance enhancements to support system cleanup of lock tables for failed connections.

CFLEVEL=5

- IXLCACHE functions that include entry version number support, delete type options to control what portions of an entry are to be deleted, suppress registration options to allow an entry to be read or written without registering interest in the entry, and information level options on READ_COCLASS and READ_COSTATS to request additional information to be returned.

- IXLCACHE REQUEST=DELETE_NAMELIST command for deleting a specific set of data entries from a structure.
- Support for cache structures containing user data field (UDF) order queues.

CFLEVEL=6

- No associated support for XES processing.

CFLEVEL=7

- Support for the use of a name class mask definition to be assigned to cache entry names. Name classes are used by the coupling facility to assign each entry to a logical group within the structure. Name classes in conjunction with the name class mask definition can be used to improve the processing efficiency of the IXLCACHE REQUEST=DELETE_NAME command.

CFLEVEL=8

- Support for the system-managed rebuild process in which the system performs all significant steps in the structure rebuild process with minimal participation by the connectors. It is not necessary to code CFLEVEL=8 if you have coded ALLOWAUTO=YES.
- The IXLCSP service, which performs the following types of computations:
  - Computes the size and ratios that are associated with a structure, given structure attributes and object counts
  - Calculates structure object counts based on the size, ratios, and other attributes associated with a structure.
- Support for user-assigned list entry IDs for list entries that are created in a list structure.

CFLEVEL=9

- Support for three new XES list structure interfaces.
  - IXLLSTE — XES List Structure Single Entry Services, which contains all requests that manipulate a single list entry.
  - IXLLSTM — XES List Structure Multiple Entry Services, which contains all requests that manipulate multiple list entries.
  - IXLLSTC — XES List Structure Control Services, which contains all requests that modify structure controls.

  The current XES List Structure Services, IXLLIST, will be maintained for compatibility, but will not be enhanced with new function after OS/390 Version 2 Release 8.
- IXLCACHE WRITE_DATA,WHENREG=NO,ASSIGN=YES|NO to allow the suppression of creating a new data entry when an existing data entry is not found.

CFLEVEL=10
No associated support for XES processing.

CFLEVEL=11
Support for the system-managed duplexing rebuild process in which the system performs all significant steps in the structure duplexing rebuild process with minimal participation by the connectors. It is not necessary to code CFLEVEL=11 if you have coded ALLOWAUTO=YES.

CFLEVEL=12

- Support for the system-managed duplexing rebuild process in which the system performs all significant steps in the structure duplexing rebuild process with minimal participation by the connectors.

CFLEVEL=21
Support for system-managed asynchronous lock structure duplexing in which updates to the primary coupling facility structure are asynchronously forwarded to the secondary structure.

For the most accurate list of CFLEVEL functions with associated hardware and software corequisites, see Coupling Facility Level (CFLEVEL) Considerations (www.ibm.com/systems/z/advantages/pso/cftable.html).

Understanding the CFLEVEL Returned by XCF and XES Services

The IXCQUERY, IXLMG, and IXLCONN services all provide options that allow you to request information about the CFLEVEL of a coupling facility. It is possible that each of these services could return a different response for the same coupling facility. This is because the response is based on the concept of the "actual", "requested", and "operational" CFLEVEL.

- IXCQUERY returns the CFLEVEL requested on the IXLCONN macro invocation. (QUASTRUSERLEVEL)
- IXLMG returns the actual CFLEVEL of the coupling facility in which the structure is allocated. (IXLYAMDCF_CFLEVEL)
- With OS/390 Release 8 and higher, IXLCONN returns the operational CFLEVEL, that is, the minimum architectural level of the coupling facility required to perform the structure operations you intend to submit. The operational level returned may differ from the actual CFLEVEL of the coupling facility, but will never be lower than what is required to perform your structure operations. (CONACFACILITYCFLEVEL)

When requesting the CFLEVEL with the DISPLAY CF command, XCF returns the actual CFLEVEL of the coupling facility.

Specifying the Structure Size

The size of a coupling facility structure is specified in the CFRM policy and also can be specified on the IXLCONN macro. Starting with SP 5.2, you also can specify an initial structure size with the INITSIZE parameter in the CFRM policy. The INITSIZE value is optional and is used only when an SP 5.2 and above system initially requests allocation of the structure in a coupling facility or subsequently requests a connection to a structure during user-managed rebuild processing.

With OS/390 Release 10 and later, the CFRM policy can also specify a minimum structure size. MINSIZE specifies the minimum bound for structure allocation.

With z/OS V2R1 and later, or z/OS V1R13 with PTFs for APAR OA40747, the CFRM policy can specify the SCMMAXSIZE keyword to indicate that a structure may use storage-class memory.

How MVS Initially Allocates the Structure

The system allocates storage for a coupling facility structure based on the level of the system requesting the allocation and the level of the coupling facility in which the structure is to be allocated. The amount of storage allocated is based not only on the connection parameters specified by the application but also on features supported by the coupling facility and the system performing the structure allocation.

Several values can affect the size of a structure — the STRSIZE value specified by the exploiter on the IXLCONN invocation and the SIZE, INITSIZE, MINSIZE, and SCMMAXSIZE values specified by the installation in the CFRM policy. If a value is too small to satisfy the required control structure space, the connection attempt will fail. IBM recommends that the policy SIZE value and IXLCONN STRSIZE value (if specified) be in a range of 1.5 - 2.0 times the INITSIZE value (if specified), to prevent situations in which the structure cannot be allocated at its initial size. As systems migrate to higher level coupling facilities that support additional features, it is possible that a value that was satisfactory for a lower level coupling facility might be unsatisfactory for a higher level.

The MINSIZE specification of the CFRM policy allows an installation to indicate the smallest size to which a structure can be altered, as well as to provide a minimum bound on all structure allocation requests. MINSIZE is an optional parameter in the CFRM policy, and its default value depends on the specification of ALLOWAUTOALT in the CFRM policy.

- If ALLOWAUTOALT(YES) is specified and MINSIZE is not specified, its default value is 75% of the INITSIZE value or 75% of the SIZE value if INITSIZE is not specified.

- If ALLOWAUTOALT(NO) is specified or defaulted to and MINSIZE is not specified, its default value is zero.

Structure size allocation uses the STRSIZE defined on the IXLCONN macro, if the application specified a value, and that value is less than the maximum size specified in the CFRM policy with the SIZE parameter and greater than the MINSIZE value. If STRSIZE is not specified on the IXLCONN macro, allocation uses the INITSIZE specified in the CFRM active policy. If INITSIZE is not specified (it is an optional parameter), then the SIZE specified in the CFRM active policy is used.

The following table shows the initial allocation size determination for a connection. The STRSIZE value specified on IXLCONN might or might not be present, depending on the application using the structure. The INITSIZE value from the CFRM policy also might or might not be present. The MINSIZE value, which is only available on an OS/390 Release 10 or higher system, is assumed to have a nonzero value in the table. In all cases, the initial allocated size of the structure will not be greater than the maximum structure size specified in the CFRM policy with the SIZE parameter nor less than the MINSIZE from the CFRM policy. (However, the structure size may be less than MINSIZE if the CFRM policy also specifies SCMMAXSIZE for the structure.)

Table 14. Initial Structure Size Allocation

	CFRM Policy	
IXLCONN	**INITSIZE specified**	**INITSIZE not specified**
IXLCONN STRSIZE specified between MINSIZE and SIZE	Target size = STRSIZE	Target size = STRSIZE
IXLCONN STRSIZE specified greater than SIZE	Target size = SIZE	Target size = SIZE
IXLCONN STRSIZE specified less than MINSIZE	Target size = MINSIZE	Target size = MINSIZE
IXLCONN STRSIZE not specified	Target size = INITSIZE	Target size = SIZE

To display the actual amount of storage allocated to a structure, issue the DISPLAY XCF,STRUCTURE command.

Structure Allocation when Rebuilding a Structure

The system allocates storage for a coupling facility structure that is undergoing rebuild processing in basically the same way as initial allocation. The INITSIZE and SIZE parameters from the CFRM policy are used, while MINSIZE, if specified, provides a minimum bound for requests at OS/390 R10 and higher.

The results of structure allocation during the rebuild process, whether user-managed or system-managed, may differ significantly from those during initial allocation, depending on what system conditions exist. The size of a structure undergoing a rebuild or duplexing rebuild process might require more coupling facility storage than specified by the SIZE parameter in order to allocate the new structure. As a general rule, SIZE specifies the maximum size to which a structure can expand. However, because the rebuild process requires that the rebuilt structure contain the objects from the original structure, and because storage algorithms for different coupling facilities may yield different results, the coupling facility configuration may dictate that additional storage is needed to accomodate the allocation of the new structure. For example, a coupling facility at CFLEVEL=8 would require more storage for a list structure than a coupling facility at CFLEVEL=7 would for the same structure. Conversely, for rebuild purposes, a structure might be allocated with a size smaller that INITSIZE or the value specified or defaulted to for MINSIZE.

When a user-managed rebuild or duplexing rebuild process is initiated, the system determines whether either of the following conditions exist:

- The installation has changed the SIZE specification in the CFRM policy, and the policy change is pending.

- The application has changed any structure attributes on its IXLCONN REBUILD request. (The application may initiate a structure rebuild for the purpose of modifying the structure attributes.)

If either of the above conditions is true, the system attempts to allocate the new structure as previously described for initial allocation.

On the other hand, if neither of the conditions listed above exist, or the rebuild process is system-managed, the system allocates the new structure based on the attributes of the initial structure. The system will allocate the structure large enough to contain all the objects that had been allocated in the old structure. The allocated size of the new structure:

- Might be larger than the maximum size indicated by SIZE.
- Might be between the INITSIZE and SIZE values. (Perhaps in order to be able to copy all data that must be copied from the old structure to the new structure.)
- Might be less than INITSIZE. (Perhaps because of a coupling facility storage constraint, but the small size still provided a sufficient number of structure objects to allow the copy process to succeed.)
- Might be less than MINSIZE. (Perhaps because of a coupling facility storage constraint, as long as there is enough space in the new structure to copy all the inuse objects from the old structure.)

Determining Maximum Structure Size

When a structure is allocated, the coupling facility sets the **maximum structure size** equal to the structure size specified in the CFRM active policy. The maximum structure size value remains constant as long as this instance of the structure remains allocated. However, the actual structure size might be less than the maximum structure size value. A smaller size could occur because:

- INITSIZE was specified in the CFRM active policy with a smaller size
- STRSIZE was specified on an IXLCONN macro with a smaller size
- Storage constraints exist in the coupling facility
- A previous structure alter reduced the structure size.

Note that during a system-managed rebuild, the new structure might be allocated with a size larger than the maximum structure size specified by SIZE, if the larger size is required to accommodate the system-managed rebuild process.

Determining Minimum Structure Size

The coupling facility also sets the **minimum structure size** when the structure is initially allocated. The minimum structure size value is the minimum coupling facility control space required to allocate the structure with the specified percentage allocation of storage for event monitor controls as well as the specified entry-to-element ratio. The minimum structure size value can change when the structure is reapportioned with an entry-to-element ratio that is different from the previous ratio.

Determining Marginal Structure Size

When allocating the structure, the coupling facility also determines the **marginal structure size** — the true minimum size at which the structure can be allocated. The marginal structure size is less than the minimum structure size and does not take into consideration the entry-to-element ratio or the percentage of storage used for event monitor controls. Thus, when allocating the storage in the structure, the percentage specified for event monitor controls is applied to the storage in the structure that is available beyond that designated as the marginal size. The entry-to-element ratio is applied to the storage that is available after the percentage for event monitor controls has been determined.

Should there not be enough space in a coupling facility to allocate the structure with the requested size, the system allocates the structure in the coupling facility in the preference list with the most available space, which satisfies the largest set of other allocation requirements. For a lock structure, the allocation fails if a large enough area in a coupling facility cannot be found to support the number of lock entries required.

The coupling facility ensures that the size of a structure is a multiple of the coupling facility storage increment (see “Coupling Facility Storage Increment” on page 221). If not, the coupling facility rounds up the size value to be a multiple of the increment. Ultimately, the actual size of the structure allocated in a coupling facility is based on storage allocation priorities with which the coupling facility control code complies and on storage constraints in the coupling facility itself. See “Coupling Facility Considerations When Allocating a Structure” on page 218.

A connected user of the structure can determine the structure's size by examining the connect answer area for both the maximum structure size and the actual structure size (fields CONAMAXSTRUCTURESIZE and CONASTRUCTURESIZE). An operator can display a structure's size by issuing the DISPLAY XCF,STRUCTURE command.

Understanding Coupling Facility Volatility

A coupling facility might support nonvolatility, that is, the ability to maintain the data stored in the coupling facility should a power outage occur. The importance of allocating a structure in a nonvolatile coupling facility is dependent on the requirements of the application. The application must document its requirement for a nonvolatile coupling facility so that the installation can properly configure its coupling facilities.

If the first connected user specifically requests with NONVOLREQ=YES that the structure be allocated in a nonvolatile coupling facility and one is available, then the request is granted. Subsequent connectors to the structure can determine whether the structure currently is in a volatile or nonvolatile coupling facility by interrogating a flag in the connect answer area (field CONAVOLATILE).

Planning for Coupling Facility Failure-Independence

An application might require that its structure be placed in a failure-independent environment. To accomplish this, the installation must ensure that the coupling facility is not in the same failure domain as the MVS systems that access it. For example, placing the coupling facility in an LPAR in a processor with one or more additional LPARs that are running MVS to access the coupling facility would not provide a failure-independent environment.

Similarly, an application designed to exploit user-managed structure duplexing requires that the structures be allocated in a failure-independent environment. To accomplish this, the installation should ensure that the coupling facility in which the old structure is allocated is failure-independent from the coupling facility in which the new structure is allocated. For example, if the old structure is allocated in a coupling facility which is in an LPAR in a processor, and the new structure is allocated in another LPAR configured as a coupling facility in the same processor, both structures would be lost should the processor fail. The installation should ensure that, when coupling facility failure-independence is required, the structure's preference list contains coupling facilities that allow XES to uphold this requirement.

Connectors needing a structure allocated in a failure-independent environment must specify NONVOLREQ=YES on their IXLCONN invocation. Connectors to the structure can determine whether the structure currently is in a failure-independent coupling facility by interrogating a flag in the connect answer area (field CONAFAILUREISOLATED).

Creating the Exclusion List

The exclusion list contains an unordered list of structures that are not to be allocated in the same coupling facility as this structure. The exclusion list of structures is defined in the CFRM policy. If the system cannot meet the exclusion list requirements but is able to allocate the structure, a flag in the connect answer area indicates that the exclusion list was ignored.

Selecting a Coupling Facility for Structure Allocation

The system allocates a structure in a coupling facility based on the requirements that the installation has specified in its active CFRM and sysplex failure management policies, and in accordance with the attributes specified on the IXLCONN macro. Prior to allocating the structure, the system may or may not reorder the preference list to reflect the coupling facility that most closely meets the allocation criteria.

- The CFRM policy not only lists structure names and sizes, but also defines preference lists and exclusion lists. A preference list is an ordered list of the coupling facilities in your installation in which you would prefer having a structure allocated. An exclusion list is an unordered list of coupling facility structures which you do not want to reside in the same coupling facility as this specific structure.
- The SFM policy assigns a weight to each system in the sysplex designating the system's relative importance in the sysplex.

How the system selects a coupling facility for structure allocation depends on whether the installation has specified that the preference list order is to be enforced. The CFRM policy statement, ENFORCEORDER(YES), available with on systems with OW33615 installed or which are running OS/390 Release 9 or higher, specifies that the system is to enforce the order of the coupling facilities in the preference list for structure allocation, and is not to reorder the list based on how well the coupling facilities meet the structure's allocation requirements. See “Preference List Order Enforced” on page 217.

Preference list order not enforced

If the CFRM active policy specifies or defaults to ENFORCEORDER(NO) (the preference list can be reordered), the system allocates the structure in the coupling facility in the preference list that meets the following allocation criteria, listed in order of relative importance from most important to least important:

1. Has connectivity to the local system trying to allocate the structure
2. Has a coupling facility operational level (CFLEVEL) equal to or greater than the requested CFLEVEL or if the connector has specified ALLOWAUTO=YES, has a CFLEVEL equal to or greater than CFLEVEL=8. Note that a CFRM policy DUPLEX specification of ASYNC or ASYNCONLY will cause a requested CFLEVEL of at least CFLEVEL=21.
3. Is a failure-independent coupling facility in relation to the coupling facility containing the old structure in user-managed duplexing rebuild processing. The system will give preference to failure-independent coupling facilities when allocating the new structure during user-managed duplexing.
4. Has space available that is greater than or equal to the requested structure size
5. For structures that are defined with an SCMMAXSIZE CFRM policy keyword, has sufficient free storage-class memory to accommodate structure objects that will reside in storage-class memory and sufficient configured storage-class memory to accommodate the specified SCMMAXSIZE value.
6. Meets the volatility requirement requested by the connector
7. Does not contain a structure in this structure's exclusion list.

Note that the system assumes certain criteria when selecting a coupling facility for new structure allocation in user-managed structure duplexing. The system always assumes LOCATION=OTHER when selecting the coupling facility. As listed above, the coupling facility chosen by the system will, if possible, be failure-independent with respect to the coupling facility containing the old structure. Lastly, if the level of connectivity to the new structure is less than that to the old structure, the action taken by the system is LESSCONNACTION=TERMINATE.

If there is no coupling facility in the preference list that meets all these allocation criteria, then the system determines the coupling facility that most closely meets the criteria. To do this, the system uses a weighting system for each of the coupling facilities in the structure's preference list. The weights correspond to the list of criteria — with system connectivity having the highest weight, CFLEVEL the next higher weight, and so on down the list. The system eliminates from the list those coupling facilities that do not meet the following connection requirements:

- CONNECTIVITY=SYSPLEX
- LOCATION=OTHER (for a structure rebuild)
- MINCFLEVEL=mincflevel

Using these weights, the system orders the coupling facilities that meet all requirements and attempts to allocate the structure. If two or more coupling facilities are assigned identical weights, then the selection is made based on the order in which the coupling facilities are defined in the CFRM policy preference list. If the attempt to allocate the structure is not successful, the system reorders the coupling facilities in the preference list, ignoring the exclusion list requirement, and again attempts to allocate the structure. The

system continues its allocation attempt in successively lower-weighted coupling facilities until allocation is successful. The system will choose the coupling facility that most closely meets the requirements of the connect request. If no coupling facility meets the allocation requirements, the IXLCONN request fails with reason code IXLRSNCODENOFAC.

An application can override the local connectivity criterion by indicating on its IXLCONN invocation that the system is to choose the "best" coupling facility for all systems in the sysplex. ("Best" can mean either the coupling facility connected to the most important systems in the sysplex or the coupling facility that meets all the allocation criteria.) See "Specifying Coupling Facility Connectivity Requirements" on page 225 for a description of the CONNECTIVITY parameter on IXLCONN.

Once it has ordered the coupling facilities in accordance with the relative importance of the allocation criteria, the system might consider the SFM weights of the systems attached to each coupling facility at the time of the IXLCONN request. If an active SFM policy is in effect in the sysplex, a system uses the SFM weights as part of the coupling facility selection criteria.

The system attempts to allocate the structure in the coupling facility that:

- Meets as many of the installation and application requirements as possible.
- Has the best available connectivity across the sysplex.

Preference List Order Enforced

If the active CFRM policy specifies ENFORCEORDER(YES), the system will not reorder the preference list based on how well the various coupling facilities meet the structure's allocation requirements. Specifically, the IXLCONN CFLEVEL specification will not be honored, which could result in the structure being allocated in a down-level coupling facility. If IXLCONN CONNECTIVITY=BESTGLOBAL has been specified, that parameter too will not be honored because this parameter, by definition, specifies that the system is to use the coupling facility allocation algorithm. However, if applicable, the system will honor the following requests:

- IXLCONN CONNECTIVITY=SYSPLEX, which specifies that the coupling facility be connected to all systems in the sysplex.
- IXLREBLD LOCATION=OTHER, which specifies that the structure is to be rebuilt in a coupling facility other than the one in which it was originally allocated.
- IXLCONN MINCFLEVEL=mincflevel, which specifies that the structure is required to be allocated in a coupling facility that supports at least the indicated minimum CFLEVEL.

Note that if the preference list order is enforced, the exclusion list of structures is not applicable because EXCLLIST is mutually exclusive with ENFORCEORDER(YES).

Using the SFM System Weights in Coupling Facility Selection

The system selects the coupling facility that is accessible from the set of systems that has the highest aggregate SFM system weight. How the system uses the SFM system weights depends on whether the CONNECTIVITY keyword is used on the IXLCONN request.

CONNECTIVITY=DEFAULT

If the CONNECTIVITY keyword is not used (or if CONNECTIVITY=DEFAULT), the system uses the default coupling facility selection algorithm described in "Selecting a Coupling Facility for Structure Allocation" on page 215. The SFM system weights, if available, are used as the lowest-weighted attribute in the selection process. The system will choose the coupling facility that **most closely** meets the requirements of the connect request. If no coupling facility meets the allocation requirements, the IXLCONN request fails with reason code IXLRSNCODENOFAC.

CONNECTIVITY=BESTGLOBAL

The system calculates the connectivity value (based on system SFM weights) of all coupling facilities in the current sysplex and uses this value as the highest-weighted attribute to select the coupling facility in which to allocate the structure. The system calculates the aggregate SFM system weights for each

coupling facility in the preference list. The system then attempts structure allocation in the one or more coupling facilities with the highest weight. If the structure allocation fails because of a local connectivity problem (that is, the system invoking the IXLCONN service did not have connectivity to the coupling facility), the IXLCONN request fails with reason code IXLRSNCODENOFAC. If, on the other hand, the reason for the allocation failure was not local connectivity but rather a reason such as insufficient storage in the coupling facility, the system continues to attempt to select a coupling facility by considering the coupling facilities in the preference list with the next highest aggregate SFM system weights. Using the same procedure as for the coupling facilities with the highest weights, the system will continue its attempt to allocate the structure until all coupling facilities in the preference list have been considered.

CONNECTIVITY=SYSPLEX

The system does not use the SFM system weights, as all systems in the sysplex must be connected to the same coupling facility. If no coupling facility meets this requirement, the IXLCONN request fails with reason code IXLRSNCODENOFAC.

Understanding Connectivity in a Mixed Sysplex Environment

In a mixed sysplex environment made up of systems at MVS SP Version 5 and OS/390 Release 1, each of those systems must have APAR OW19718 installed in order to coexist with one or more OS/390 Release 2 systems. The APAR allows the systems to use the SFM weights in a consistent manner. With this support, the system selects a coupling facility for structure allocation based on the level of the system invoking the IXLCONN service:

- A request from an OS/390 Release 2 and higher system uses the SFM weights as part of the coupling facility ordering process when selecting a coupling facility. If an SFM policy is not in effect in the sysplex, all system are considered to have equal weight.
- A request from an MVS SP 5.1 through OS/390 Release 1 system uses the default selection algorithm for coupling facility selection and does not factor in the SFM weights.

In a mixed sysplex environment in which any system is at the MVS SP Version 4 level, that system will cause a request from another system to connect to a structure with a specification of IXLCONN CONNECTIVITY=SYSPLEX to fail.

Coupling Facility Considerations When Allocating a Structure

Coupling facility structure size includes both control areas required by the coupling facility control code and data areas used by the application. The size is also affected by coupling facility allocation rules and the coupling facility allocation increment size, which is a function of the level of the coupling facility.

The actual allocation of coupling facility resources for a given structure depends on:

- CFRM policy specification
- Authorized application specification when using the XES services
- Coupling facility storage constraints
- Coupling facility storage increment
- Coupling facility level.

You must take all of these factors into account when determining how to define your CFRM policy and how to configure your coupling facility.

Understanding Coupling Facility Storage

The storage for a coupling facility LPAR is defined in the same way as a noncoupling facility partition. However, the storage in a coupling facility LPAR cannot be dynamically reconfigured. If another partition on the same processor fails, its storage cannot be taken over by the coupling facility partition. Or, if the coupling facility partition fails, its storage cannot be taken over by another partition.

Coupling facility storage is used to contain both coupling facility control information and application data. In early processors, storage was logically segregated between these two functions. The installation was

required to configure storage as either *control storage* (which might contain both control information and data) or *noncontrol storage* (which might contain only data). This distinction is now obsolete, and in current processors all storage that is configured to the coupling facility is available for either purpose. Effectively, all storage is control storage. However, some APIs (such as IXLMG) return information about the amount of coupling facility storage required for control information.

Beginning with CFLEVEL 19, you can also configure storage-class (*flash*) memory to a coupling facility LPAR. Storage-class memory provides an overflow capability to minimize the probability of structure-full conditions. As with coupling facility real storage, you cannot dynamically reconfigure storage-class memory, nor can one partition reclaim storage-class memory that is configured to another partition. Use of storage-class memory increases the amount of coupling facility real storage that is required by the affected structures.

Some processors also support storage-class memory, or flash memory. You can configure storage-class memory in large amounts that are relative to the actual structure size. It is used to provide an overflow capability to avoid structure-full conditions. The installation assigns storage-class memory to a structure by specifying SCMMAXSIZE in the CFRM policy. Storage-class memory can contain both structure controls and data. It is not directly accessible to the application; the coupling facility migrates controls and data between real storage and storage-class memory as necessary to satisfy requests that are initiated in the normal manner. In contrast to coupling facility real storage, storage-class memory is not allocated to a structure until it is required, and it is returned to a free pool when no longer in use. When used, storage-class memory also requires more coupling facility real storage (*augmented space*) to allow the coupling facility to track the location of the stored data and controls. The augmented space is not included in the structure size but is instead allocated from the coupling facility's empty space on an on-demand basis. Like storage-class memory itself, it is returned to the free pool when no longer required. Since access to storage-class memory is slower than access to coupling facility real storage, requests that must retrieve from or write to storage-class memory might incur a performance penalty. See *z/OS MVS Setting Up a Sysplex* for more discussion about the implications of storage-class memory exploitation.

The size of the control structures is affected by the CFLEVEL of the coupling facility in which the structure is to be allocated. Different CFLEVELs have different control structure requirements, with the result that the same structure might have significantly different sizes depending on the CFLEVEL of the coupling facility.

The DISPLAY CF command displays information about coupling facility storage including the total amount, total in-use, total free control and noncontrol storage, and storage-class memory. The DISPLAY XCF,STRUCTURE command displays information about coupling facility real storage and storage-class memory that is allocated to a structure..

Coupling Facility Resource Allocation “Rules”

A coupling facility structure is located in a particular coupling facility and allocated at a certain size based on values specified by the installation in a CFRM policy, by the authorized application in its request for XES services, and by characteristics of the coupling facility itself, such as storage constraints, storage increment, and structure ID limit.

CFRM Policy Specification

The CFRM policy contains the maximum structure size, as well as the ordered preference list of coupling facilities and unordered list of structures for allocation of the structure. The structure size defined in the CFRM policy is used as the attempted allocation size unless it is overridden by a structure size specified on the IXLCONN macro.

Authorized Application Specification

When requesting an XES service to connect to a structure, the authorized application optionally can specify a size for the structure. The system uses the smaller of the two sizes (as specified in the CFRM policy or by the authorized application), as the target allocation size for the structure.

The authorized application also is required to specify certain structure attributes when connecting to a structure. These structure attributes are used by the coupling facility control code when determining how

to most efficiently allocate the various parts of the structure in the coupling facility. Some examples of structure attributes are data element size, whether or not locks are used, the number of list headers, and whether an adjunct area is required. The coupling facility control code evaluates each attribute in the following sequence:

- **Available space**

  The structure is allocated as large as possible based on the available storage in the requested coupling facility. The target size is derived from either the CFRM policy or the authorized application's request to connect to the structure.

- **Entry/element ratio**

  Within the total available space allocated to the structure, the coupling facility control code attempts to allocate entries and elements in a way that most accurately approximates the requested entry/element ratio. If the structure has been allocated with a size less than the minimum structure size required by the specified structure attributes, then the entry/element ratio may deviate from the requested value. If the structure has been allocated with a size greater than or equal to this minimum structure size, then the entry/element ratio should be satisfied.

- **Entry and element counts**

  Within the total available space allocated to the structure, the coupling facility control code attempts to maximize the actual number of entries and elements (as opposed to the ratio) that can be placed in the structure.

Coupling Facility Storage Constraints

If the CFRM policy specifies the SCMMAXSIZE keyword to indicate that a structure is eligible to use storage-class memory, a lack of storage-class memory might affect structure allocation in a coupling facility at or above CFLEVEL 19.

- The maximum amount of storage-class memory that a structure can use is the smaller of the CFRM policy SCMMAXSIZE value and the total amount of storage-class memory that is configured to the coupling facility. Other factors might further limit the amount.
- The use of storage-class memory increases the amount of control storage that is required to support a given number of entries and elements. It therefore reduces the number of entries and elements that can be accommodated in coupling facility real storage by a structure of a specified size. If the amount of storage-class memory that is configured to the coupling facility is less than the SCMMAXSIZE specification and the structure is therefore allocated to support a smaller maximum amount of storage-class memory than intended by the CFRM policy, it would be capable of containing more entries and elements than it would if allocated in a coupling facility with more configured storage-class memory. That would cause a problem if the structure were to be subsequently rebuilt into a coupling facility with more storage-class memory, because the rebuild new structure instance would not be able to accommodate the same object counts as when the structure was originally allocated. To prevent this conflict, the system limits the number of entries and elements when allocating in a coupling facility with less storage-class memory than specified by the policy.

  Note: This behavior depends on the amount of storage-class memory that is configured to the coupling facility, not the amount of free storage-class memory.

- Storage-class memory is not allocated to the structure until required for use. It is therefore possible to overcommit the storage-class memory that is configured to the coupling facility. It is possible to define the CFRM policy such that the sum of the SCMMAXSIZE values for allocated structures exceeds the total amount of storage-class memory that is available to the coupling facility. To ensure that you have the desired amount of storage available in the event of an application failure that causes the structure to fill up, do not overcommit storage class memory.

The use of storage-class memory affects the structure's resource usage in a complex way. See *z/OS MVS Setting Up a Sysplex* for additional discussion about its implications.

Coupling Facility Storage Increment

Coupling facility storage is allocated in multiples of the coupling facility model-dependent storage increment size. For coupling facility levels 0 through 14, all structure allocations are rounded up to a multiple of 256K. At coupling facility level 15, allocations are rounded up to a multiple of 512K. For coupling facility levels 16 and higher, allocations are rounded up to a multiple of 1M. See the "PR/SM Planning Guide" for current storage increment sizes.

Storage-class memory (CFLEVEL 19 and above) is allocated in multiples of the coupling facility model-dependent storage-class memory increment size. The increment size is 1M.

Coupling Facility Structure ID Limit

The coupling facility control code imposes a limit on the number of structures that can reside in any one coupling facility. See *PR/SM Planning Guide* for the structure ID limit for the level of coupling facility that you are using.

Successful Completion of Structure Allocation

Each time you successfully invoke IXLCONN for a structure, the system places a connect token (CONTOKEN) in the connect answer area. CONTOKEN identifies each connection to the structure and is unique for each connection within the sysplex. You can issue IXLCONN from any system in the sysplex that is connected to the coupling facility.

Figure 19 on page 221 shows task 1 allocating a structure for the first time:

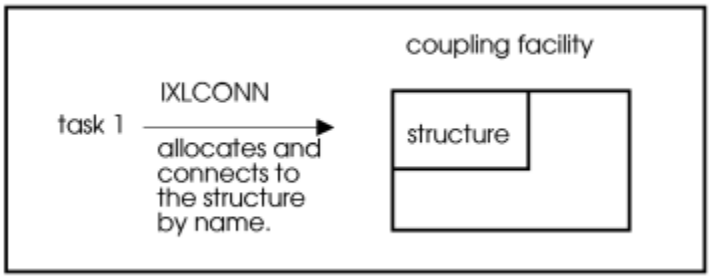


Figure 19. Allocating a Structure

In Figure 20 on page 221 task 2 connects to the same structure:

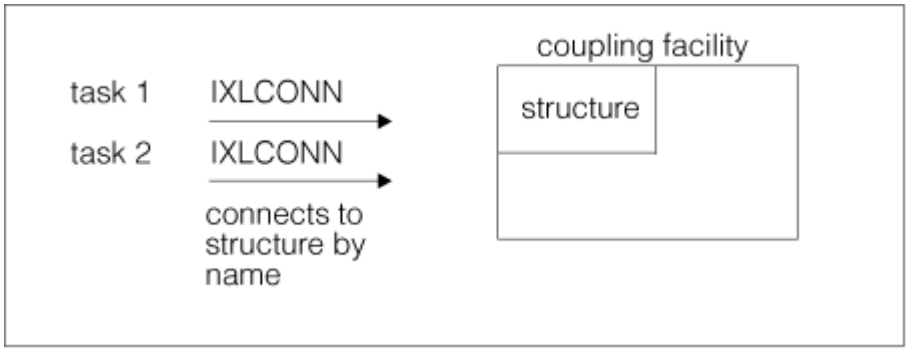


Figure 20. Connecting to an Allocated Structure

Whether the first connector or a subsequent connector to a structure, all connectors must verify that the structure attributes are acceptable.

- For the first connector, even though the return code might be IXLRETCODEOK, IXLCONN might not have satisfied all attributes requested. The CONACONNALLOC flag is set to indicate that this connection allocated the structure in the coupling facility.

- Subsequent connectors to an already allocated structure must verify that the attributes received by the first connector to the structure are acceptable.

If you find that the structure attributes are not acceptable, you can do one of the following:

- Disconnect from the structure.
- Rebuild the structure.

For information on rebuilding a structure, see "Structure Rebuild Processing" on page 264.

Once you are connected to a structure, you can use specific coupling facility structure services (IXLCACHE, IXLLIST, or IXLLOCK) to manipulate the cache, list, or lock structure.

Note about designing structure connections

XES associates a connection to a coupling facility structure with the task that issues the IXLCONN macro. When that task is ended, either normally or abnormally, the connection to the structure fails. The task that issued the IXLCONN macro is responsible for termination/cleanup and is accountable for all resources associated with the task.

This task association has several implications which require that you evaluate carefully your application design:

- Should a task that has issued one or more IXLCONN requests terminate abnormally for some reason, then all the associated XES connection(s) will be terminated as a result of the termination. For example, if your task connects to several structures, each of which is required to support the task's function, and your task fails, all connections fail. This might be acceptable because your task would not have been able to provide any function without connectivity to all structures.
- Should a task that performs both XES-related and non-XES-related functions fail, then both types of processing are disrupted. The failure of the single task causes a loss of capability with a greater scope than might be necessary.

IBM, therefore, recommends that you evaluate your design with the following considerations:

- Do not aggregate unrelated XES connections under the same task.
- Do not aggregate XES connections under the same task with non-XES-related functions.

Connecting to a Coupling Facility Structure

As an authorized user, you connect to a coupling facility structure to manipulate data using sysplex services. The data within the structure depends on the type of structure - cache, list, or lock.

Overview of Connect Processing

You connect to a coupling facility structure to use XES services to manipulate structure data. The system administrator must define the characteristics of the structure in an administrative CFRM policy and an operator must activate that policy before you can connect to the structure.

The first user to successfully connect to a structure causes the structure to be allocated in a coupling facility, using the attributes from both the policy definition and the IXLCONN parameters. Subsequent connectors to the structure cannot change these initial attributes, unless all connectors agree to rebuild (ALLOWREBLD=YES) or alter (ALLOWALTER=YES) the structure with new attributes. The number of users that are allowed to connect to a structure is a function of the coupling facility model, the CFRM couple data set format statements, and for a lock structure, a user-supplied limit when the lock structure is allocated.

If you wish your IXLCONN request to perform only a connection but not an allocation of the structure, consider specifying ALLOC=NO on the request. For example, if you connect to a structure only to work with existing structure data, and the structure was potentially not allocated, then coding ALLOC=NO could avoid the unnecessary allocation of a new structure.

A connector to a structure is aware of other connectors to the same structure, (called peer connections), through its event exit.

Upon successful completion of your IXLCONN request, you

- Receive data in the connect answer area (mapped by IXLYCONA)
- Are connected to the coupling facility structure you requested
- Can request structure services that are valid for this type of structure
- Will be notified about other connections to this structure through your event exit. (Any other active connections to the structure also are notified of your connection through their event exits.)

Note: If you are connecting to a lock or a serialized list structure, the system joins an XCF group for your connection. You might need, therefore, to allow for an increase in the number of XCF groups and reformat the sysplex couple data set accordingly. Be aware that this XCF group is strictly for the system's use. If you wish to use XCF services, then you must join your own group using IXCJOIN.

If your IXLCONN request does not complete successfully, you might decide to use the ENF notification of events to determine whether to retry the request. Users waiting to connect to a structure can use ENF event code 35 to be notified when coupling facility resources become available. Whether a subsequent IXLCONN request will be successful depends on the then current set of factors, such as whether the structure dumping or structure rebuild processing is in progress.

For planned reconfiguration or recovery, connected users to a structure can rebuild the structure. The new structure has the same name as the old structure, but can be placed in a different coupling facility and can have some changed attributes, such as size. All connected users must participate in the rebuilding process or else must disconnect from the structure. Rebuilding requires stringent coordination among the participating systems; checkpoints in the form of event notifications require responses from all participants.

For improved availability and usability, connected users to a cache structure can duplex the structure. By default, the new instance of the structure is placed in a different coupling facility and has the same or better connectivity as the old structure. As with rebuilding, all connected users must participate in the user-managed duplexing process, all participating systems are required to coordinate their actions, and all participating systems are required to respond to event notifications.

Naming the Structure

Use the STRNAME parameter to identify the name of the structure. This is a required parameter. The name you choose is also the name that is to be specified in an installation's CFRM active policy. You must supply this name to users of your application.

Use the TYPE parameter to specify whether the structure is to be allocated as a cache, list, or lock structure.

Naming the Connection

Use the CONNAME parameter to identify your connection to the structure. CONNAME is required if the connection is to be persistent (CONDISP=KEEP) and optional if the connection is to be non-persistent (CONDISP=DELETE). However, if you do not specify a name when CONDISP=DELETE, the system generates a unique name for the connection. This field cannot be changed if the structure is rebuilt.

Note also that IXLCONN REBUILD will not be successful unless you specify the same connection name as for the original connect (either user-specified or system-generated).

Specifying Connector Data

Use the CONDATA parameter to provide eight bytes of connection data. The system passes this data to your exits when invoked, and is for your use only. A possible use for CONDATA is as a pointer to a control block that represents the connector. This field cannot be changed if the structure is rebuilt.

Providing a Connection Level

Use the CONLEVEL parameter to provide eight bytes of connector data that specifies any non-local information, such as connection or version level. The system passes this data to the structure's peer connections through the event exit. (See the EEPLSUBJCONLEVEL field in IXLYEEPL.)

A possible use for CONLEVEL is to provide a way for different levels of connected users to share the same structure. Depending on the migration protocol employed, peer connectors might not allow a lower-level connection to the structure and the lower-level connector would disconnect immediately upon determining the connect level of peer connections. Alternately, the protocol might require that the uplevel connectors limit their functionality to that of a lower level connector.

This field cannot be changed if the structure is rebuilt.

Requesting a Coupling Facility Level

Use the CFLEVEL parameter to specify the level of the coupling facility in which you want the structure to be allocated. The CFLEVEL requested should identify the level of architected function that the user requires. (The coupling facility levels and the associated functionality are defined in *PR/SM Planning Guide*.) The CFLEVEL parameter is ignored if the structure is already allocated.

If a coupling facility of the requested CFLEVEL is not available, the system might allocate the structure in a lower level coupling facility. The field CONACFACILITYCFLEVEL in the connect answer area contains the CFLEVEL of the coupling facility in which the structure is allocated.

If, on the IXLCONN request, you specify a CFLEVEL higher than that supported by the system on which you are running, the IXLCONN request fails with reason code IXLRSNCODECFLEVEL. The following information is returned in the connect answer area:

- CONAMVSRELEASEMAXCFLEVEL — the maximum CFLEVEL value supported by the system.
- CONACFACILITYCFLEVEL — the level of operations supported by the coupling facility.

The CFLEVEL requested by each connector to a structure is saved in the CFRM active policy. Other connectors are informed of this level through their event exits — for new connection, existing connection, rebuild new connection, and rebuild existing connection events.

The CFLEVEL can be specified only through IXLCONN. To change to a different coupling facility level, you must disconnect and connect to the structure again with a different value. You also cannot change your requested CFLEVEL when rebuilding the structure. The structure *might* be rebuilt in a coupling facility with a different CFLEVEL, but that is dependent on the first connector to issue the IXLCONN REBUILD request and what coupling facility resources are available for allocating the new structure.

Systems at OS/390 Release 9 and higher can specify MINCFLEVEL in conjunction with CFLEVEL to ensure that if the structure is allocated, it is allocated in a coupling facility that provides the required level of functionality. See “Requesting a Minimum CFLEVEL” on page 224.

Note that the CFLEVEL can affect the size of a structure. Different CFLEVELs will have different control structure requirements, which in some cases may cause a structure to become unallocatable or unusable.

Requesting a Minimum CFLEVEL

Use the MINCFLEVEL parameter to specify the minimum coupling facility level in which to allocate the structure. The value of MINCFLEVEL must be equal to or less than the value of CFLEVEL.

Specifying MINCFLEVEL prevents the system from allocating the structure in a coupling facility from the structure's preference list that is at a lower coupling facility level than MINCFLEVEL. This applies to initial allocation, user-managed rebuild allocation, and user-managed duplexing allocation. If the structure's preference list does not contain a suitable coupling facility in which to allocate the structure, the system rejects the IXLCONN request with reason code IXLRSNCODENOFAC. The CONAFACILITYARRAY, which contains information about each coupling facility in which allocation was attempted, will contain the reason code CONARSNINSUFFCFLEVELUSER for each coupling facility that was not chosen because it did not meet the specified MINCFLEVEL requirement.

If a structure is already allocated, the system will prevent connectors that specify MINCFLEVEL to connect to the structure if the MINCFLEVEL specified is higher than the coupling facility level in which the structure is allocated. This applies to both initial connect and rebuild connect. The system rejects the IXLCONN request to connect with reason code IXLRSNCODEINSUFFCFLEVELUSER when the structure is allocated in a lower level coupling facility than MINCFLEVEL. The CFLEVEL at which the structure is currently allocated is returned in CONAFACILITYCFLEVEL.

Connectors must be aware when specifying the MINCFLEVEL keyword that if connectors specify different values for MINCFLEVEL it is possible for some connectors to be unable to connect to the structure. It might also be true that the installation does not have a coupling facility at the requested MINCFLEVEL, in which case the connect request would never be successful.

System-managed Processing Considerations

The ability to support system-managed processes is identified by the IXLCONN ALLOWAUTO=YES keyword. By definition, a system-managed process will not allocate a new structure in a coupling facility whose CFLEVEL is less than the CFLEVEL reported to any connector. Coupling facilities that are at a CFLEVEL lower than the specified MINCFLEVEL will also not be considered eligible for allocation.

The allocation algorithm when ALLOWAUTO=YES is specified is modified slightly in OS/390 Release 9 so that coupling facilities at CFLEVEL=8 or higher are sorted to the front of the preference list. For example, assume a preference list contains three coupling facilities, all considered equal except for their CFLEVELs.

```
CFLEVEL=7     CFLEVEL=8     CFLEVEL=9
```

If a connector specifies ALLOWAUTO=YES,CFLEVEL=9, the preference list would be reordered to:

```
CFLEVEL=9     CFLEVEL=8     CFLEVEL=7
```

The allocation algorithm change enables those coupling facilities with a CFLEVEL higher than CFLEVEL=8 to be sorted to the front of the preference list, followed by coupling facilities with a CFLEVEL=8, and finally those coupling facilities with CFLEVELs below CFLEVEL=8. Prior to this change in the allocation algorithm, only those coupling facilities with a CFLEVEL higher than CFLEVEL=8 were reordered to the front of the preference list and the remaining order of the preference list was unchanged.

Specifying Coupling Facility Connectivity Requirements

Use the CONNECTIVITY keyword to define the application's connectivity requirements to the structure. The system uses this requirement to select the coupling facility in which to allocate the structure. CONNECTIVITY values are:

- SYSPLEX — Requests that the system allocate the structure in a coupling facility that has connectivity to **all** systems currently in the sysplex. If no coupling facility meets this requirement, the IXLCONN request fails with reason code IXLRSNCODENOFAC. In the connect answer area, CONAFACILITYARRAY, which contains an entry for each coupling facility in which allocation was attempted, indicates the reason why the allocation failed for that coupling facility. The reason code for a coupling facility that did not meet the connectivity requirement is CONARSNINSUFFCONNECTIVITY.

  Specifying SYSPLEX implies that all systems currently in the sysplex have an active CFRM policy, are capable of attaching to a coupling facility, and have operating links to a coupling facility.
- BESTGLOBAL — Requests that the system allocate the structure in the coupling facility that provides the best global connectivity to systems in the sysplex, if possible. The system calculates the connectivity value of all coupling facilities in the current sysplex and uses this value as the highest attribute to select the coupling facility in which to allocate the structure. See “Selecting a Coupling Facility for Structure Allocation” on page 215. The system attempts to allocate the structure in a coupling facility with the highest connectivity value that has the best local and global connectivity. (The coupling facility selected must have connectivity to the local system that issued the IXLCONN request.) If the structure allocation fails, the system selects successively lower-weighted coupling facilities in which to attempt the allocation. If no allocation is possible, the IXLCONN request fails with reason code IXLRSNCODENOFAC.

In the connect answer area, CONAFACILITYARRAY contains reason code CONARSNPREFERREDCFSELECTED for any coupling facility that was not selected because another coupling facility was a preferable choice.

- DEFAULT — Requests that the system use the coupling facility selection algorithm to select the coupling facility in which to allocate the structure. If there is an active SFM policy, the system calculates the connectivity value for a coupling facility only when selecting a coupling facility that is equal in regard to all coupling facility attributes other than connectivity. If there is no active SFM policy, the algorithm does not include the calculation of the coupling facility connectivity value.

  If the system cannot find a coupling facility that meets all requirements, the system attempts structure allocation in successively lower-weighted coupling facilities until allocation is successful. The system chooses the coupling facility that most closely meets the requirements of the IXLCONN request.

Allowing User-Managed Rebuild for a Structure

Connectors specify whether or not they will allow the structure to be rebuilt through user-managed rebuild processing (ALLOWREBLD). If user-managed rebuild is allowed, the connectors can allocate another structure of the same name and rebuild data into the new structure. ALLOWREBLD=YES is the default, so if you do not allow the structure to be rebuilt through user-managed rebuild processing, you must provide your own interfaces for planned shutdown before reconfiguring a coupling facility. You also must specifically code ALLOWREBLD=NO, which will prevent a user-managed rebuild from being started.

Allowing the Structure to be Duplexed

User-managed duplexing rebuild, a variation of the structure rebuild process, is available only for cache structures. For duplexing to occur, all connectors to the structure must specify not only ALLOWDUPREBLD=YES but also ALLOWREBLD=YES when connecting to the structure.

Comparing User-Managed Rebuild and Duplexing Rebuild

Structure rebuild and duplexing rebuild provide the framework by which an application can ensure that there is a viable and accurate version of a structure being used by the application.

Structure rebuild allows you to reconstruct the data in a structure when necessary, for example, after a failure. Duplexing rebuild allows you to maintain the data in duplexed structures on an ongoing basis, so that in the event of a failure, the duplexed structure can be switched to easily. Duplexing rebuild is the solution for those applications that are unable or find it difficult to reconstruct their structure data after a failure occurs.

Enabling Support of System-Managed Processes

There are two types of system-managed processes: system-managed rebuild and system-managed duplexing rebuild.

- System-managed rebuild, which is intended for use in planned reconfiguration scenarios, provides a means for rebuilding a structure with minimal participation from connectors to the structure. Connectors use the ALLOWAUTO parameter to indicate whether they support the system-managed rebuild process.
- System-managed duplexing rebuild, which is intended to provide a failure recovery capability, provides a means for duplexing the structure with minimal participation from connectors to the structure(s). The system allocates a duplexed instance of the structure, attaches connectors to the structure, copies data to the duplexed instance of the structure, and then maintains the structures in a synchronized state. In the event of a failure, such as a loss of connectivity, structure failure, or coupling facility failure affecting one of the two duplexed structure instances, the system can failover to the unaffected structure.

  When the system is running on a z15™ server that supports recovery process boosts, system failover processing requests System Recovery Boost's recovery process boost support to complete failover actions and resume availability to the unaffected structure instance.

For more information on System Recovery Boost, see: IBM Z System Recovery Boost Content Solution (www.ibm.com/support/z-content-solutions/system-recovery-boost/) or IBM Documentation (www.ibm.com/docs/en/zos).

Connectors use the ALLOWAUTO parameter to indicate whether they support the system-managed duplexing rebuild process.

In order to use the system-managed processes, an application must be written to handle the following:

- Processing of the Structure Temporarily Unavailable and Structure Available events.
- The use of extended restart tokens.
- Evaluation of the information presented with the Structure State Change event and reacting appropriately to changes in structure characteristics.
- The use of both physical structure version number to uniquely identify an instance of a structure. (In IXLYCONA, the physical version numbers are identified by CONAPHYSICALSTRUCTUREVERSION and CONAPHYSICALSTRUCTUREVERSION2; in IXLYEEPL, the physical version numbers are identified by EEPLSSCSTRPHYSICALVERSION and EEPLSSCSTRPHYSICALVERSION2.

Enabling support of system-managed asynchronous duplexing

Two types of system-managed duplexing are available: system-managed synchronous duplexing and system-managed asynchronous duplexing.

- *System-managed synchronous duplexing* came first and is often implied by references to system-managed duplexing. Updates to the structure are typically sent to both instances in parallel and synchronized to keep both instances consistent.
- *System-managed asynchronous duplexing* is available only for lock structures. For better performance than system-managed synchronous duplexing, updates are made in the primary structure instance before asynchronously forwarding the updates to the secondary structure instance. Connectors use the IXLCONN ASYNCDUPLEX parameter to indicate whether they support system-managed asynchronous duplexing.

To support system-managed asynchronous duplexing, an application must be written to handle the following situations:

- All processing requirements that are associated with system-managed process support (AllowAuto=YES) as described earlier.
- The use of asynchronous duplexing request sequence numbers (ADUPREQSEQNUM). For more information, see “Working with structures in the Async Duplex Established phase” on page 285.

Allowing the Structure to be Altered

Use the ALLOWALTER parameter to indicate whether you permit the structure to be altered. If you specify ALLOWALTER=YES, you also must specify CFLEVEL=1 or higher because the structure must be allocated in a coupling facility that supports structure alter processing. **For structure alter to occur, all connectors to the structure must specify ALLOWALTER=YES and CFLEVEL=1 or higher.**

When you specify ALLOWALTER=YES, you can also specify:

- Whether the entry-to-element ratio can be changed (RATIO)
- Whether the percentage of event monitor controls (EMC) storage can be changed (RATIO)
- The minimum number (as a percent value) of both entries and elements you want to be available at the conclusion of the structure alter process.

  For list structures, this is a percentage of currently “in-use” entries and elements; for cache structures, this is a percentage of “in-use and changed” entries and elements.

  – MINENTRY specifies the minimum level of available entries.
  – MINELEMENT specifies the minimum level of available elements.

- The minimum amount of storage (as a percent value) of storage allocated for event monitor controls that you want available at the conclusion of the structure alter process.

  For keyed list structures, this is a percentage of “currently-in-use” EMCs.

  – MINEMC specifies the minimum level of available EMCs.

With OS/390 Release 10 and higher, if ALLOWALTER=YES has been specified, an installation can specify that eligible structures are to be automatically altered by the system. See *z/OS MVS Setting Up a Sysplex* for a description of the automatic alter function.

The application must support alter for the structure to be allocated with support for storage-class memory. Consider the following conditions:

- If the CFRM policy specifies SCMMAXSIZE and all other environmental requirements are met, and if the first connector specifies ALLOWALTER=YES, then the system allocates the structure with support for storage-class memory.
- After the structure is allocated with support for storage-class memory, the system does not permit any connector that does not specify ALLOWALTER=YES to connect to the structure.

Comparing Structure Rebuild and Structure Alter

Structure alter and structure rebuild are complementary functions, each with its own purpose in a coupling facility environment. The structure rebuild function, introduced in SP 5.1, allows a connector to a structure to change many of the structure attributes, but requires the other connectors to participate in the rebuild process. The structure alter function, available in SP 5.2, allows an authorized user, not necessarily a connector to a structure, to change the structure's size, entry-to-element ratio, and percentage of storage for event monitor controls, without disrupting the structure's current connectors. The structure rebuild function physically relocates the structure, either in the same or a different coupling facility, thus requiring the installation to plan its CFRM policy to allow for coupling facility space to be left available for possible later rebuild use. The structure alter function does not relocate the structure, but changes it “in place”. Structure alter does not require additional coupling facility space to be reserved for a “new” structure, and does not disrupt the processing of connectors to the structure while it is being altered.

Handling Dump Serialization

You can specify the amount of time (if any) that SVC Dump can hold serialization on the structure for dumping purposes. SVC Dump supports dumping of list, serialized list, and cache structures; it does not support dumping of lock structures.

Use the ACCESSTIME keyword to can indicate the following:

- The structure is not permitted to be dumped (dump serialization may not be held)

  ```
  ACCESSTIME=MAXIMUM,MAXTIME=0
  ```

- Dump serialization can be held up to a maximum specified time

  ```
  ACCESSTIME=MAXIMUM,MAXTIME=n
  ```

  where *n* is tenths of seconds.

- Dump serialization can be held for as long as it takes to dump all data that was requested to be dumped.

  ```
  ACCESSTIME=NOLIMIT
  ```

The operator can override the ACCESSTIME parameter that was specified on the IXLCONN macro with the DUMP command.

Monitoring structure repopulation

Use the MONITOR keyword to specify how the system is to monitor progress toward completion of structure repopulation during user-managed rebuild or duplexing rebuild.

The system requires that connectors complete structure repopulation in a timely manner to avoid degrading overall sysplex performance. If a connector does not make satisfactory progress, as defined by the specified or defaulted MONITOR keyword, the system reports a hang and may take automatic action to resolve the situation, depending on the installation setting of the SFM policy CFSTRHANGTIME keyword.

The system monitors each connector individually, and the MONITOR specification applies only to the connector initiating the IXLCONN request, not to the structure as a whole.

Specifying structure attributes for all structures

The following IXLCONN parameters define the common requirements of the cache, list, and lock structures. Parameters specific to each structure type are explained in other topics.

STRNAME
Specifies the name of the structure to which you want to connect.

STRSIZE
Specifies the size of the structure in 4K blocks. The size specified in the CFRM policy is the maximum size for allocation of this structure. To allocate a smaller size structure, use this IXLCONN keyword.

CONDATA
Specifies connector data to be passed to your exit routines.

STRDISP
Specifies the disposition of the structure when all connections are released.

CONDISP
Specifies the disposition of this connection in case of the connection's abnormal termination.

CONNAME
Specifies the name of this connection.

ALLOWREBLD
Specifies whether this connection allows user-managed structure rebuild to be initiated for the structure.

ALLOWALTER
Specifies whether this connection allows structure alter to be initiated for the structure.

ALLOWAUTO
Specifies whether this connection allows system-managed processes to be initiated for the structure.

SUSPEND
Specifies whether this connection can tolerate suspension of work units during system-managed processing for a structure.

RATIO
Specifies whether this connection allows the ratio of entries-to-elements to be changed if the structure is altered.

MINENTRY
Specifies the minimum number of “in-use” (list) and “in-use and changed” (cache) entries that are to be available at the completion of structure alter processing.

MINELEMENT
Specifies the number of “in-use” (list) and “in-use and changed” (cache) elements that are to be available at the completion of structure alter processing.

NONVOLREQ
Specifies whether the connector to the structure requires that the data in the structure be both nonvolatile and failure-independent.

CONLEVEL
Specifies a connector's level to be passed to peer connections in the event exit.

CFLEVEL
Specifies the requested level of the coupling facility in which the structure is to be allocated.

CONNECTIVITY
Specifies the scope of system connectivity to a coupling facility in which the structure is to be allocated.

EVENTEXIT
Specifies the address of your event exit.

COMPLETEEXIT
Specifies the address of your complete exit.

ACCESSTIME
Specifies the length of time that you can tolerate not having access to the structure while SVC Dump holds serialization on the structure.

MAXTIME
Specifies the maximum amount of time that you can tolerate not having access to the structure.

MONITOR
Specifies how the system is to monitor progress toward completion of structure repopulation during user-managed rebuild or duplexing rebuild.

The IXLCSP service can be used to assist you when defining certain IXLCONN parameters. See "Using the IXLCSP Service to Determine Structure Size or Attributes" on page 242.

Connecting to a Cache Structure

This section describes the IXLCONN parameters that you code to connect to a cache structure. To help you code the IXLCONN macro, use the general IXLCONN guidance information in "Connecting to a Coupling Facility Structure" on page 222 together with the information provided here.

The first application that connects to a cache structure allocates the structure and defines its attributes. Subsequent connectors to the structure use the structure as it has been allocated by the first connector. The following IXLCONN parameters define the attributes of the cache structure or its connector:

ALLOWDUPREBLD
Specifies whether this connection allows user-managed duplexing rebuild to be initiated for the cache structure.

ELEMCHAR or ELEMINCRNUM
Specifies the data element size for the cache structure.

MAXELEMNUM
Specifies the maximum number of data elements per data entry. For a coupling facility of CFLEVEL=0, the maximum number can be from 1 to 16. For a coupling facility of CFLEVEL=1 or higher, the maximum number can be from 1 to 255.

DIRRATIO
Specifies the directory component of the directory-to-element ratio.

ELEMENTRATIO
Specifies the element component of the directory-to-element ratio.

ADJUNCT
Specifies whether the cache structure is to contain adjunct areas.

VECTORLEN
Specifies the maximum number of data items for which the connection can have concurrent registration.

NUMCOCLASS
Specifies the maximum number of cast-out classes that can be used by the connection.

NUMSTGCLASS
Specifies the maximum number of storage classes that can be used by the connection.

UDFORDER
Specifies whether a user data field (UDF) order queue should be maintained for each cast-out class for the structure. Applicable only to cache structures allocated in a coupling facility with CFLEVEL=5 or higher.

NAMECLASSMASK
Specifies the name class mask pattern definition to be applied to entry names at connect time. Name classes are used by the coupling facility to assign each entry to a name class within the structure. Name classes can be used to improve the processing efficiency of IXLCACHE REQUEST=DELETE_NAME command. Applicable only to cache structures allocated in a coupling facility with CFLEVEL=7 or higher.

SUPPRESSEVENTS
Specifies whether the origination of certain connection and disconnection events should be suppressed for this connector. The following events can be suppressed for their originator: New Connection, Existing Connection, Rebuild New Connection, Rebuild Existing Connection, and Discontinued or Failed Connection. Suppression of these events may provide a significant performance benefit at connect or disconnect time to connectors who do not need the information presented. See "Suppressing Certain Events for a Connector" on page 233.

ASYNCXI
Specifies whether this connection prefers cross-invalidation operations against local caches to be processed synchronously or asynchronously to the completion of cache operation requests initiated by this connection.

The use of the ASYNCXI keyword with the value of IxlConnAsyncXiYes (1) is meaningful only when a cache structure is allocated in a CFLEVEL=23 or higher coupling facility and the z/OS system where the IXLCONN is issued from supports asynchronous cross-invalidation.

To determine whether the support for asynchronous cross-invalidation processing is available on the system from which you are connecting to a cache structure, issue IXCQUERY REQINFO=FEATURES. QuReqRfAsyncXi, if returned, indicates whether the support is available. If the support is not available and you connect with ASYNCXI(IxlConnAsyncXiYes), the parameter will be ignored.

Selecting the Number of Data Elements and Their Size

To select the data element size for the cache structure, you need to understand the approximate sizes of the smallest and largest pieces of data to be stored in the cache entries. If the data can fit into adjunct areas, you could avoid using data entries altogether. **Code a value of 0 for ELEMENTRATIO to define a cache structure without data entries.** The system ignores the MAXELEMNUM parameter if you specify it with an ELEMENTRATIO of 0.

The system allows a maximum of 16 data elements per data entry (with CFLEVEL=0) or 255 data elements per data entry (with CFLEVEL=1 or higher), but you can use the MAXELEMNUM parameter to specify a smaller maximum number if you want to further restrict the size of the largest data entries.

The value you specify for MAXELEMNUM must be greater than or equal to the value specified for ELEMENTRATIO divided by the value specified for DIRRATIO:

MAXELEMNUM >= (ELEMENTRATIO / DIRRATIO)

The data element size multiplied by the maximum number of data elements must be sufficient to accommodate the largest piece of data that you need to manipulate as a single entry. For a list of possible data element sizes, see Table 21 on page 360.

Effect of CFLEVEL on MAXELEMNUM

Even if you request that a structure be allocated in a CFLEVEL=1 or higher coupling facility (thus allowing up to 255 data elements per data entry), the system might need to allocate the structure in a CFLEVEL=0 coupling facility instead. The system will attempt to allocate the structure with an entry size as great as

that specified on the IXLCONN invocation and then adjust the number of data elements to fit into the entry size. You can examine the resulting data element and data entry values in the connect answer area.

For example: You request to connect to a structure with a data element size of 256 bytes and a MAXELEMNUM of 128 (thus implying a CFLEVEL=1 or higher coupling facility). The maximum entry size is 32K (256 bytes x 128). If the system is forced to allocate the structure in a CFLEVEL=0 coupling facility, it will allocate the structure with a data element size of 2K and a MAXELEMNUM of 16. The maximum entry size is still 32K, but the MAXELEMNUM is changed to conform to the maximum allowed in a CFLEVEL=0 coupling facility. (The system increases the data element size by the same power of 2 by which the MAXELEMNUM value was decreased.)

Note that a change to MAXELEMNUM will have a corresponding effect on the directory-to-element ratio you specify. If the system changes the element size, it also must change the directory-to-element ratio to suit the maximum entry size. (The ratio is adjusted by the same power of 2 calculation described above.) You can examine the resulting directory-to-element ratio information in the connect answer area.

Selecting the Directory-to-Element Ratio

You cannot control directly the number of directory entries or data elements the cache structure will hold. The installation uses the CFRM policy to specify the amount of storage a particular cache structure will occupy. When the cache structure is allocated, its storage is subdivided to reserve space for cache structure components such as data elements and directory entries. The value you specify for the directory-to-element ratio is used by the system to determine the proportion of the cache structure storage to allocate to each component. The ratio, expressed as a pair of whole numbers, such as 1:4, is passed to IXLCONN using the DIRRATIO and ELEMENTRATIO parameters as follows:

- The DIRRATIO parameter specifies the part of the ratio for the directory entries (for instance, the 1 in the 1:4 ratio)
- The ELEMENTRATIO parameter specifies the part of the ratio for the data elements (for instance, the 4 in the 1:4 ratio).

In general, the directory-to-element ratio should reflect the **average** number of data elements per cache entry. For example, if your data element size is 4096 bytes, and you estimate that about half of the cache entries will require 1 data element and about half of the cache entries will require 8 data elements, then you would want a ratio of 1:4.5 which you would express in whole numbers as 2:9.

Although you request a particular directory-to-element ratio, the system might use a slightly different ratio. The actual number of entries and elements in the structure, rather than the ratio, is returned to you in the IXLCONN answer area mapped by the IXLYCONA macro. Note that these values in IXLYCONA are not exact values.

If the directory-to-element ratio is incorrect for your use of the structure, you will encounter frequent rejections of IXLCACHE requests because either the cache or cache structure is full.

See “Using the IXLCSP Service to Determine Structure Size or Attributes” on page 242.

Determining Whether to Have Adjunct Areas

The adjunct area can contain 64 bytes of user-specified data, such as information about the status of the data entry or a time stamp. The adjunct area is maintained separately from the data entry so you can change the contents of the data entry or the adjunct area independently.

Selecting the Number of Cast-Out classes

The maximum number of cast-out classes that you select is dependent upon the needs of your application. For information that can help you make this selection, see “Casting out Data Items and Reclaim Processing” on page 385.

Selecting the Number of Storage Classes

The maximum number of storage classes that you select is dependent upon the needs of your application. For information that can help you make this selection, see “Assigning and Using Storage Classes” on page 381.

Determining Whether to Have User Data Field (UDF) Order Queues

UDF order queues, available only in a cache structure allocated in a coupling facility of CFLEVEL=5 or higher, provide the ability to have a queue associated with each cast-out class, which the coupling facility maintains in order by user-data field. You can use IXLCACHE REQUEST=READ_COSTATS to determine the lowest user-data field for any entry in the cast-out class when UDF order queues are present.

Determining Whether to Use Name Class Masks

A name class mask can be used to enhance the performance of the IXLCACHE REQUEST=DELETE_NAME command in a coupling facility of CFLEVEL=7 or higher. By establishing a naming convention for entries in a cache structure, the name class mask can be used when deleting entries that adhere to that naming convention. For an example of the use of a name class mask in conjunction with the name class specified when deleting entries from a cache structure, see “Using Name Classes in a Coupling Facility” on page 444.

Suppressing Certain Events for a Connector

On systems with OW38840 installed or which are at OS/390 Release 9 or higher, connectors to a cache structure may request that the system suppress certain connection and disconnection events that the connector might otherwise generate. Suppressing these events (New Connection, Existing Connection, Rebuild New Connection, Rebuild Existing Connection, and Disconnected or Failed Connection) may provide a significant performance benefit at connect time to connectors who do not need the information presented in these events.

Note that suppression of these events is on a per-connection basis, where it is the ORIGINATION of the event that is suppressed, not the RECEIPT of the event. A review of the events and how they originate follows:

- A New Connection event notifies existing connectors of a new connection to the structure. The event originates from the new connector.
- An Existing Connection event notifies a new connector to the structure of all the current existing connections to the structure. The event originates from the new connector.
- A Rebuild New Connection event notifies existing connectors of a new connection to the new structure. The event originates from the new connector.
- A Rebuild Existing Connection event notifies a new connector of all the current existing connectors to the new structure. The event originates from the new connector.
- A Disconnected or Failed Connection event notifies all active connectors to the structure that a connector has either disconnected or failed. The event originates from the connector that either disconnected or failed.

To understand the effect of using SUPPRESSEVENTS, consider the following:

- A connection that is suppressing the origination of events may nevertheless receive such events for another connection to the structure that is NOT suppressing the origination of events. In the case of response-required events, this connection is still required to provide responses to any such event that is presented to it, regardless of whether the connection is suppressing events itself.
- When a connection is suppressing the origination of events, other connections will not receive the suppressed events originating from that connection, even if the other connections themselves are NOT suppressing events.

Also note that depending on whether OW38840 is present on the system where the connection is running (or whether the system is at OS/390 Release 9 or higher), a request to suppress events may or may not be honored for the current connection.

Specifying Asynchronous or Synchronous Cross-Invalidate Processing

A cache structure and its related services provide sysplex users with data consistency of their shared data.

One such cache structure mechanism that is used to facilitate cache data consistency is called "cross-invali-dation." Cross-invalidation processing involves setting an indicator in a local cache vector for each of the users of a cache structure data item to indicate whether the locally cached copy of the data is valid or not valid. Cross-invalidates are performed as part of cache requests and may be performed synchronously or asynchronously to the completion of a cache request.

Since users and /or coupling facilities may be remotely located, synchronous cross invalidation signal processing associated with transactions encompassing multiple cache requests may cause a delay in transaction completion. The connector may desire to have cross-invalidations performed asynchronously to a cache request and synchronize the completion of all cross-invalidates for multiple cache requests with a single transactional commit point. The preference for asynchronously completing cross-invalidations is best suited for performance sensitive transactional processing made up of multiple cache requests that can benefit from reduced coupling facility overhead per cache request and support follow-on transaction commit processing (that is, using the IXLAXISN service) to ensure the completion of cross-invalidations.

On systems with APAR OA54688 installed or which are at z/OS V2R4 or higher, using the ASYNCXI parameter on the IXLCONN invocation, a cache structure connector may specify their preference as to whether cross-invalidation operations associated with cache requests issued by the connector are to be performed synchronously or asynchronously to the completion of the cache request in the coupling facility. For more information on cross-invalidations and maintaining data consistency, see "Maintaining Data Consistency" on page 373

When requesting asynchronous cross-invalidations considerable thought must be given to the serialization protocols you implement. Please see section "Serialization considerations when requesting asynchronous cross-invalidations" on page 380

Connecting to a List Structure

This section describes the IXLCONN parameters that you code to connect to a list structure. To connect to a list structure, code the IXLCONN macro using the general IXLCONN guidance information in "Connecting to a Coupling Facility Structure" on page 222 together with the information provided here.

The first application that connects to a list structure allocates it and defines its characteristics. Subsequent connectors to the list structure use the list structure as it has been allocated by the first connector. The following IXLCONN parameters define the attributes of the list structure:

ADJUNCT
Specifies whether the list structure is to contain adjunct area. Adjunct area is required when secondary keys are specified for a list structure allocated in a coupling facililty of CFLEVEL=9 or higher.

ELEMCHAR or ELEMINCRNUM
Specifies the data element size to be used.

ELEMENTRATIO
Specifies the element component of the entry-to-element ratio.

EMCSTGPCT
Specifies the percentage of available storage that is to be set aside for event monitor controls used for sublist monitoring. The sublist monitoring function

- Is available only with a coupling facility of CFLEVEL=3 or higher.
- Requires a list structure defined as having keyed list entries (REFOPTION=KEY).

ENTRYIDTYPE
Specifies whether the system or the user will assign the list entry IDs for list entries created in the structure. A request for user-assigned list entry IDs requires that the list structure be allocated in a coupling facility of CFLEVEL=8 or higher.

ENTRYRATIO
Specifies the entry component of the entry-to-element ratio.

KEYTYPE
Specifies whether only entry keys, or both entry keys and secondary keys may be used when creating, moving, or locating list entries, or when comparing the entry keys of list entries.

LISTCNTLTYPE
Specifies whether the amount of coupling facility storage which may reside on a given list header is to be controlled by limiting the maximum number of entries or the maximum number of data elements.

LISTHEADERS
Specifies the number of lists to be allocated in the list structure.

LISTTRANEXIT
If you are planning to use list monitoring or event queue monitoring, specifies the address of your list transition exit.

LOCKENTRIES
For a serialized list structure, specifies the number of lock entries in the lock table.

MAXCONN
Specifies the maximum number of users allowed to connect to the list structure.

MAXELEMNUM
Specifies the maximum number of data elements per data entry. For a coupling facility of CFLEVEL=0, the maximum number can be from 1 to 16. For a coupling facility of CFLEVEL=1 or higher, the maximum number can be from 1 to 255.

NOTIFYEXIT
For a serialized list structure, specifies the address of your notify exit.

REFOPTION
Specifies whether list entries are to be referenced by entry name, entry key, or neither. List entries can always be referenced by entry ID or unkeyed position.

VECTORLEN
If you are planning to use list monitoring, specifies the maximum number of list headers that you can monitor for transitions between empty and non-empty states.

If you are planning to monitor your event queue, specify a VECTORLEN that includes a vector index to assign for event queue monitoring.

- If you are using event queue monitoring without also using list monitoring, specify a vector with a single vector index.
- If you are using event queue monitoring in conjunction with list monitoring, specify a vector index whose length equals the number of list headers that are to be concurrently monitored plus one for the event queue.

Selecting the Data Element Size

To select the data element size for the list structure, you need to understand the approximate sizes of the smallest and largest pieces of data to be stored in the list entries. If the data can fit into adjunct areas, you could avoid using data entries altogether. **Code a value of 0 for ELEMENTRATIO to define a list structure without data entries.** The system ignores the MAXELEMNUM parameter if you specify it with an ELEMENTRATIO of 0.

The system allows a maximum of 16 data elements per data entry (with CFLEVEL=0) or 255 data elements per data entry (with CFLEVEL=1 or higher), but you can use the MAXELEMNUM parameter to specify a smaller maximum number if you want to further restrict the size of the largest data entries. In all cases, whatever value you choose for MAXELEMNUM, the maximum size of a data entry is 64K.

The value you specify for MAXELEMNUM must be greater than or equal to the value specified for ELEMENTRATIO divided by the value specified for ENTRYRATIO:

MAXELEMNUM >= (ELEMENTRATIO / ENTRYRATIO)

The data element size multiplied by the maximum number of data elements must be sufficient to accommodate the largest piece of data that you need to manipulate as a single entry. See Table 33 on page 485 for a list of data element sizes.

Effect of CFLEVEL on MAXELEMNUM

Even if you request that a structure be allocated in a CFLEVEL=1 or higher coupling facility (thus allowing up to 255 data elements per data entry), the system might need to allocate the structure in a CFLEVEL=0 coupling facility instead. The system will attempt to allocate the structure with an entry size as great as that specified on the IXLCONN invocation and then adjust the number of data elements to fit into the entry size. You can examine the resulting data element and data entry values in the connect answer area.

For example: You request to connect to a structure with a data element size of 256 bytes and a MAXELEMNUM of 128 (thus implying a CFLEVEL=1 or higher coupling facility). The maximum entry size is 32K (256 bytes x 128). If the system is forced to allocate the structure in a CFLEVEL=0 coupling facility, it will allocate the structure with a data element size of 2K and a MAXELEMNUM of 16. The maximum entry size is still 32K, but the MAXELEMNUM is changed to conform to the maximum allowed in a CFLEVEL=0 coupling facility. (The system increases the data element size by the same power of 2 by which the MAXELEMNUM value was decreased.)

Note that a change to MAXELEMNUM will have a corresponding effect on the entry-to-element ratio you specify. If the system changes the element size, it also must change the entry-to-element ratio to suit the maximum entry size. The ratio is adjusted by the same power of 2 calculation described above.) You can examine the resulting entry-to-element ratio information in the connect answer area.

Requesting Storage for Event Monitor Controls

The EMCSTGPCT parameter allows you to specify the percentage of available storage that is to be set aside for event monitor controls. **Available storage** is defined as that storage that remains in the allocated structure after the storage required for the marginal structure size has been assigned. (The marginal structure size is the true minimum size at which the structure can be allocated. It consists of structure controls and overhead, and under certain conditions, *might* contain a small number of entries and elements.) Figure 21 on page 237 shows a structure with an amount of its space used as the marginal structure size. The remainder of the space in the structure is available for event monitor controls and entries and elements.

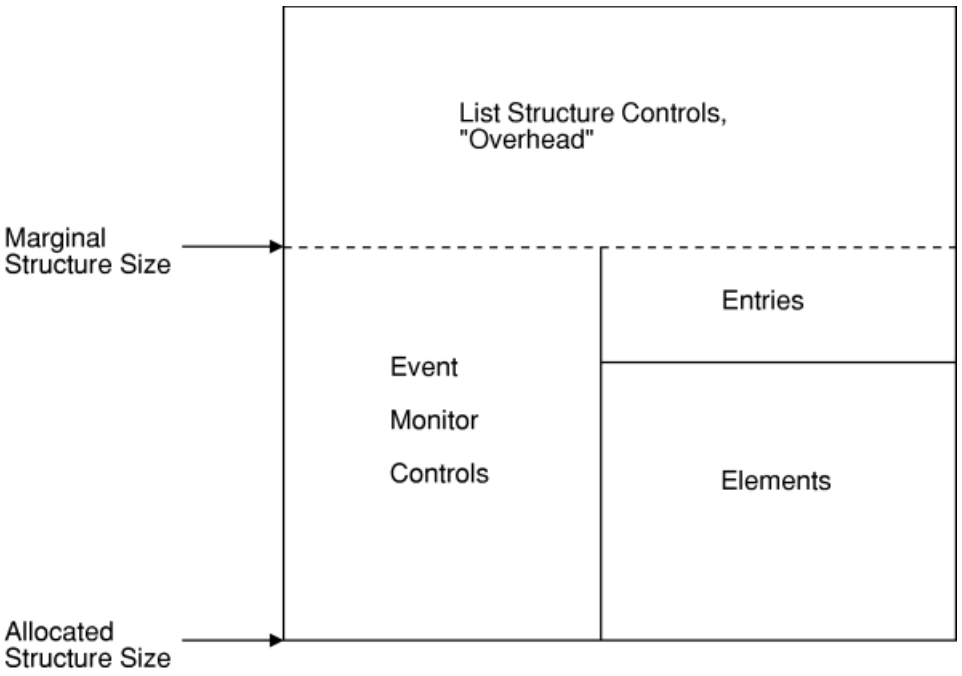


Figure 21. List Structure Space Allocation

Note that the EMCSTGPCT parameter is applied first to the available storage to set aside a percentage of the storage for event monitor controls (EMCs). EMCs are used when monitoring sublists, an IXLLIST function available with a coupling facility of CFLEVEL=3 or higher when the list structure has been allocated as having keyed list entries (REFOPTION=KEY). The sublist monitoring function also requires that the IXLLIST user have a local vector, which is requested by specifying a nonzero VECTORLEN. The figure shows that as the EMCSTGPCT percentage value increases, there will be less storage available for entries and elements.

After the storage for the EMCs is assigned, the remaining storage is available for entries and elements. The ENTRYRATIO and the ELEMENTRATIO keywords determine how many entries and elements can be defined in that storage area.

Selecting the Entry-To-Element Ratio

You cannot control directly the number of list entries or data elements the list structure will hold. The installation uses the CFRM policy to specify the amount of storage a particular list structure will occupy. When the list structure is allocated and, if applicable, the percentage of list structure storage has been set aside for event monitor controls objects, the list structure storage is subdivided to reserve space for list structure components such as data elements and list entry controls. The value you specify for the entry-to-element ratio is used by the system to determine the proportion of the list structure storage to allocate to each component. The ratio, expressed as a pair of whole numbers, such as 1:4, is passed to IXLCONN using the following ENTRYRATIO and ELEMENTRATIO parameters.

- The ENTRYRATIO parameter specifies the part of the ratio for the list entry controls (for instance, the 1 in the 1:4 ratio)
- The ELEMENTRATIO parameter specifies the part of the ratio for the data elements (for instance, the 4 in the 1:4 ratio).

In general, the entry-to-element ratio should reflect the **average** number of data elements per list entry. For example, if your data element size is 4096 byes, and you estimate that about half of the list entries will require 1 data element and about half of the list entries will require 8 data elements, then you would want a ratio of 1:4.5 which you would express in whole numbers as 2:9.

Although you request a particular entry-to-element through the IXLCONN macro, the system might use a slightly different ratio. The actual number of entries and elements in the structure, rather than the ratio, is returned to you in the IXLCONN answer area mapped by the IXLYCONA macro. Note that these values in

IXLYCONA are not exact values because the coupling facility might reserve some entries and elements for its own use. The reserved entries and elements are not available for your use, but are accounted for in the IXLYCONA counts.

If the entry-to-element ratio is incorrect for your use of the list structure, you will encounter frequent rejections of IXLLIST requests because the list structure is full. If you are monitoring the entry and element counts to avoid a structure full condition, take into account the reserved entries and elements used by the coupling facility.

Deciding How to Limit the Storage Used by Each List

The LISTCNTLTYPE parameter allows you to choose how storage use is to be managed for individual lists. You can limit either the number of list entries per list or the number of data elements per list. A limit on storage use per list may be needed to prevent the excessive use of storage by certain lists.

The flexibility offered by the choice of limits allows you to select the type of limit that best suits your use of the list structure. For instance, if your main concern is to limit the number of entries that might build up on a list, you should limit the number of list entries per list. If your main concern is to prevent the entries on a given list from consuming too much of the storage in the structure, you should limit the number of data elements per list.

Determining Whether to Have Adjunct Areas

The adjunct area can contain 64 bytes of user-specified data, such as information about the status of the data entry or a time stamp. The adjunct area is maintained separately from the data entry so you can change the contents of the data entry or the adjunct area independently.

For a list structure allocated with secondary keys (available only with CFLEVEL=9 or higher), the first 32 bytes of the adjunct area is used to store the secondary key for a list entry.

Determining Whether to Have Named or Keyed List Entries

Named entries let users reference list entries by a user-specified name. Keyed entries let users maintain list entries in a keyed order. The choice of named or keyed entries, or the use of neither, depends on how the list structure is being used. For instance, if the list entries represent units of work ordered by priority, you might choose keyed entries. If the list entries represent customer records in a particular category, you might choose named entries. If the lists represent units of work to be processed on a FIFO basis, there might be no need for names or keys.

List structures allocated in a coupling facility of CFLEVEL=9 or higher optionally can be allocated with secondary keys as a further means of referencing a list entry by keyed order.

Note that sublist monitoring and event queue monitoring are functions that require that the list structure have keyed entries.

Connecting to a Lock Structure

This section describes the IXLCONN parameters that you code to connect to a lock structure. To help you code the IXLCONN macro, use the general IXLCONN guidance information in “Connecting to a Coupling Facility Structure” together with the information provided here.

The first application that connects to a lock structure allocates the structure and defines its characteristics. Subsequent connectors to the structure use the structure as it has been allocated by the first connector. The following IXLCONN parameters define the attributes of the lock structure:

RECORD
Specifies whether the lock structure is to include record data.

PCTENTRYRSV
Specifies the percentage of the total number of record data entries in the structure that must remain free after completion of a request that attempts to create a record data entry. Use this keyword to ensure that enough free entries remain available to support application and connector recovery

scenarios. The PCTENTRYRSV parameter is meaningful only when the lock structure provides recording capabilities (RECORD=YES), the structure is allocated in a CFLEVEL=25 or higher coupling facility and the z/OS system from which the IXLCONN is issued supports the PCTENTRYRSV keyword.

RNAMELEN
Specifies whether resource names are a fixed or a variable length for the lock structure.

LOCKENTRIES
Specifies the number of lock entries in the lock structure.

NUMUSERS|MAXCONN
Specifies the maximum number of users allowed to connect to the lock structure.

MONITORSTORAGE
Specifies whether internal storage for lock structures is to be monitored.

CONTEXIT
Specifies the address of your contention exit.

NOTIFYEXIT
Specifies the address of your notify exit.

Determining Whether to Specify Record Data

Record data allows you to maintain information about a resource that you own so that, if you should lose connectivity or fail, your peer connections can initiate recovery processing for the resource. The data that you include in the 64-byte record data entry is entirely determined by your protocol, as are any recovery procedures that you may implement using that data.

An application can better manage the availability of record data entries and can thereby create a record data entry during application and connector recovery scenarios when deemed necessary (as long as there is a free record data entry in the structure to satisfy the request) by using the IXLCONN PCTENTRYRSV keyword to specify the percentage of the total number of record data entries that are to remain free (reserved) after completion of a request that attempts to create a record data entry. The value for the percentage of entries that must remain free is a composite of the PCTENTRYRSV value specified by all active connectors to the structure. The composite percentage for the structure is the most restrictive (highest) PCTENTRYRSV value specified by an active connector. When needed, an application may indicate on a lock request that attempts to create a record data entry to proceed with creating a record data entry even if the resulting percentage of free entries at the completion of the request would be less than the established percent entry reserved threshold (if any) for the structure. For more information on how to use the “reserved” record data entries, see information related to IXLLOCK, IXLRT and IXLYNEPL / IXLSYNCH in *z/OS MVS Programming: Sysplex Services Reference*.

The maximum number of record data entries that a structure can support is returned in the IXLYCONA answer area (CONALOCKMAXRECORDELEMENTS). If the structure is already allocated, the number of record elements in use at the time of the connect is returned in CONALOCKRECORDELEMENTS.

Understanding the Resource Name Length Attribute

Use the RNAMELEN parameter to specify whether the length of the resource name (RNAME) is a fixed or a variable length. Prior to OS/390 Release 2, the resource name always had a length of 64 bytes. With OS/390 Release 2 and higher, you can specify as a structure attribute for the lock structure whether you want to use resource names that have a fixed length of 64 bytes, or that have a variable length of from 1 to 300 bytes. The default, if you do not specify the RNAMELEN parameter, is that resource names will have a fixed length of 64 bytes.

The first connector to the structure establishes the resource name length attribute. Subsequent connectors to the structure must specify the same value for the attribute or the attempt to connect fails with reason code IXLRSNCODESTRTYPE. When rebuilding a lock structure, you must specify an RNAMELEN parameter on your IXLCONN REBUILD request that is consistent with the RNAMELEN specified for the original structure. The IXLCONN REBUILD invocation fails with reason code IXLRSNCODESTRTYPE if you specify the RNAMELEN parameter.

Determining the Number of List Structure Users

Use the MAXCONN parameter to limit the number of users of a list structure. The limit can be one that your application imposes or a limit based on the level of coupling facility that you are using. The limit is returned in the IXLYCONA answer area field `CONACFACILITYUSERLIMIT`.

When the requested MAXCONN attribute is in the range 0 to 32, the MAXCONN value is 32.

To allocate a structure that can support more than 32 connectors, you must use the MAXCONN keyword on the initial connect request to allocate the structure, and you must ensure that the structure is allocated in a coupling facility of CFLEVEL=17 or higher. Specifying the MAXCONN keyword with any value indicates that the connection can support a user-id limit change that results from a system-managed process (for example, rebuild). XCF communicates the user-id limit change through the structure state change event. Specifying the MAXCONN keyword with a value greater than 32 indicates that the connector can understand events for connections. Events have a *target connection* (the connector to which the event is being delivered to) and a *subject connection* (the connector that is the subject of the event) with *connection identifiers* up to the specified value (for example, `EeplExistingConnection` or `EeplNewConnection`).

The total number of list structure users is the minimum of one of the following:

- MAXCONN parameter on IXLCONN.
- Number of CONNECT records that the CFRM policy accepts. (These records limit the number of connections per structure.)
- User-id limit based on the coupling facility level.

For the rebuild of a list structure, you can specify a MAXCONN value that is greater than the number of in-use connections to the old structure instance. If you specify a MAXCONN value that is less than the number of in-use connections to the old structure instance, the request fails with reason code IXLRSNCODEINCOMPATNUMUSER.

Installations should not use more than 32 instances of the application until the following recommendations are met.

- You have upgraded all the relevant application instances to a level that supports greater than 32 connectors.
- The sysplex contains at least two coupling facilities that are CFLEVEL=17 or higher.

Failure to implement these recommendations can result in an unsafe migration path to greater-than-32 connectors to a structure and can lead to failed connection attempts, failure to rebuild the structure, or failure to duplex the structure.

Determining the Number of Lock Entries

Use the LOCKENTRIES parameter to specify the number of entries in the lock structure. This value determines the number of available ‘slots’ in a structure's lock table, to which a specific resource is mapped by a hashing algorithm. The value of LOCKENTRIES is rounded up to a power of 2, if it is not already specified as such. If you define the lock structure to have no record data associated with it (RECORD=NO), you can request that the system is to attempt to obtain the largest possible number of locks for the allocated size of the structure by specifying LOCKENTRIES=0. The system returns the number of lock entries actually allocated in the IXLYCONA answer area, field CONALOCKENTRIES. A value of 0 for a lock structure with record data is not valid and the request fails with reason code IXLRSNCODENOLENTRIES.

Determining the Number of Lock Structure Users

Use the NUMUSERS or MAXCONN parameters to limit the number of users of a lock structure. The limit can be one that your application imposes or may be a limit based on the level of coupling facility you are using. The limit is returned in the IXLYCONA answer area, field CONAFACILITYMAXLOCKUSERS.

When the NUMUSERS keyword is used by any connection to the structure, the actual number of connections that is supported by the structure does not exceed 32. This rule also applies when the

IXLCONN is issued for the rebuild process. To allocate a structure that supports more than 32 connectors, you must use the MAXCONN keyword on the initial connect request to allocate the structure, you must ensure that the structure is allocated in a coupling facility of CFLEVEL=17 or higher, and all subsequent IXLCONN requests must specify the MAXCONN keyword with a value greater than 32. Specifying the MAXCONN keyword with any value indicates that the connection can support a user-id limit change that results from a system-managed process (for example, rebuild). XCF communicates the user-id limit change through the structure state change event.

Specifying the MAXCONN keyword with a value greater than 32 indicates that the connector can understand events for subject connections with connection identifiers up to the specified value specified (for example, `EeplExistingConnection` or `EeplNewConnection`).

The total number of lock structure users is the minimum of one of the following:

- NUMUSERS or MAXCONN parameter on IXLCONN
- Number of CONNECT records supported by the CFRM policy. (These records limit the number of connections per structure.)
- User-id limit based on the coupling facility level.

The number of lock structure users is returned in field CONALOCKNUMUSERS in IXLYCONA.

For the rebuild of a lock structure, you can specify a NUMUSER or MAXCONN value that is greater than the number of in-use connections to the old structure instance. If you specify a NUMUSER or MAXCONN value that is less than the number of in-use connections to the old structure instance, the request fails with reason code IXLRSNCODEINCOMPATNUMUSER.

If you do not specify the same keyword (either MAXCONN or NUMUSERS) on the initial connect request and the rebuild connect request, the rebuild connect request fails with reason code IXLRSNCODEBADLOCKNUMUSER.

Installations should not use more than 32 instances of the application until the following recommendations are met.

- You have upgraded all the relevant application instances to a level that supports greater than 32 connectors.
- The sysplex contains at least two coupling facilities that are CFLEVEL=17 or higher.

Failure to implement these recommendations can result in an unsafe migration path to greater-than-32 connectors to a structure and can lead to failed connection attempts, failure to rebuild the structure, or failure to duplex the structure.

Determining whether to monitor internal storage for lock structures

Whenever internal storage for lock structures is exhausted, the system issues an X'026' abend causing connector termination. Monitoring lock structure internal storage provides a way to avoid encountering this abend by instead rejecting IXLLOCK OBTAIN and ALTER requests that would allow the amount of inuse storage to exceed a pre-established threshold. These requests are rejected with return code IxlRetCodeEnvError and reason code IxlRsnCodeResourcesConstrained until the amount of inuse storages goes below the threshold.

The IXLCONN MONITORSTORAGE parameter allows the connector to indicate whether IXLLOCK requestors can tolerate receiving the IxlRetCodeEnvError return code and IxlRsnCodeResourcesConstrained reason code before XES uses it to reject IXLLOCK OBTAIN and ALTER requests. Specifying that storage is to be monitored (IxlconnMonitorStorageYes) indicates that requestors can tolerate receiving the return and reason code. It is the responsibility of individual exploiters to determine what action to take when requestors receive the return and reason code. Possible actions include:

- Failing the application.
- Waiting for a sufficient number of IXLLOCK RELEASE requests to be performed, allowing the amount of insuse storage to go below the threshold that is causing IXLLOCK OBTAIN and ALTER requests to be rejected. The IXLLOCK OBTAIN or ALTER request can then be reissued.

- Reissuing the IXLLOCK OBTAIN or ALTER request specifying the CRITICALREQUEST parameter to indicate IxllockCriticalRequestYes. This allows the storage over and above the internal threshold to be available for critical processing. This includes storage used by:
  - XES for all asynchronous processing associated with IXLLOCK requests that have already been accepted.
  - IXLLOCK OBTAIN and ALTER requests that specify CRITICALREQUEST(IxllockCriticalRequestYes).

Note that even when monitoring of storage is requested, it could still be possible for XES storage to become exhausted and the resulting X'026' abend to occur. (For example, if too many IXLLOCK OBTAIN or ALTER requests were specified with CRITICALREQUEST(IxllockCriticalRequestYes) once the internal threshold was reached.

IXLLOCK RELEASE and PROCESSMULT requests are not affected by monitoring of storage and are always allowed to be processed. This allows inuse storage associated with the locks that are unlocked to be freed. IXLSYNCH and IXLRT requests are also not affected by monitoring of storage because they are considered to be XES critical requests that must always complete.

To determine whether the support for monitoring internal storage for lock structures is available on the system from which you are connecting to a structure, issue IXCQUERY REQINFO=FEATURES. QuReqRfIxlconnMonitorStorage, if returned, indicates whether the support is available. If the support is not available and you connect with MonitorStorage(IxlconnMonitorStorageYes), the parameter will be ignored.

Using the IXLCSP Service to Determine Structure Size or Attributes

The XES Structure Computation Service (IXLCSP), with a coupling facility of CFLEVEL=8 or higher, provides a means for obtaining both coupling facility capacity planning and structure size optimization information. Potential uses for the IXLCSP service are:

- Planning coupling facility storage utilization and the contents of CFRM policies
- Planning structure size required by an application
- Optimization of connect parameters.

The XES Structure Computation Service (IXLCSP) can be used in any of the following ways:

- To compute the size and ratios that are associated with a structure, given structure attributes and object counts.
- To calculate structure object counts based on the size, ratios, and other attributes associated with a structure.
- To compute the amount of storage-class memory (SCM, or flash memory) that is required to provide the wanted overflow storage capacity, which is given structure attributes and SCM object counts.

IXLCSP directs a request to compute either a structure's size or object counts to a coupling facility of CFLEVEL=8 or higher. The coupling facility performs the requested calculation as if it were allocating the structure. However, no structure allocation occurs and the contents of the target coupling facility are unchanged at the conclusion of the IXLCSP calculation.

The following considerations apply to the target coupling facility:

- The target coupling facility must be described in the CFRM active policy and accessible from the system invoking IXLCSP.
- The calculations performed by the coupling facility are idealized in the sense that they do not account for any constraints or conditions (such as storage shortages) that might prevent a structure from being allocated in the coupling facility.
- The calculations performed by the coupling facility are appropriate to the CFLEVEL of that coupling facility. Coupling facilities at different CFLEVELs will, in all likelihood, return different answers.
- Storage-class memory calculations require a coupling facility with a CFLEVEL greater than or equal to 19.

Determining Structure Size and Ratios Given Structure Attributes

Use the appropriate IXLCONN values as input to the IXLCSP service to arrive at a structure size. This size can then be input to the CFRM policy definitions. Chapter 15, “Documenting your Coupling Facility Requirements,” on page 767 describes the process by which you can use the parameters specified on the IXLCONN macro as input to the IXLCSP service.

Determining Structure Counts Given Structure Size and Ratios

To use IXLCSP to determine structure counts, you must know the values of the INITSIZE and SIZE parameters that were used when defining the structure in the CFRM policy, and the ratios to be used when connecting (IXLCONN DIRRATIO, ENTRURATIO, ELEMENTRATIO, and EMCSTGPCT parameters, as appropriate). The values returned by IXLCSP can then be used as input to the IXLCONN service. Structure counts available from IXLCSP are:

- Cache structure
  - Number of directory entries that can be contained in the target structure
  - Total number of elements that can be contained in the target structure
- List structure
  - Number of event monitor controls that can be contained in the target structure
  - Number of list entries that can be contained in the target structure
  - Total number of elements that can be contained in the target structure
- Lock structure
  - Number of record data entries that can be contained in the target structure
  - Number of lock entries that can be obtained in the target structure.

Determining Storage-Class Memory Size Given Counts and Attributes

Determine the amount of storage-class memory to associate with the structure by first determining the counts of entries and data elements that might need to spill to SCM to provide the desired amount of overflow protection. Using those SCM counts as input to IXLCSP, compute the required amount of SCM. You can then use the result to specify the SCMMAXSIZE parameter of the CFRM policy definition.

Defining the Required Exit Routines

XES uses exit routines to communicate some information to connected coupling facility users. Depending on the structure type, you will need to supply one or more of these exit routines, which are identified on the IXLCONN macro.

Event Exit

XES invokes your event exit to report error and status information, such as a new connection or a failed structure. “Events Reported to the Event Exit” on page 335 lists the events that are reported to the event exit. All connected users of a coupling facility structure must provide an event exit. The EVENTEXIT keyword of IXLCONN identifies the address of your routine.

Note that the Event exit might receive control before the system returns to the next sequential instruction following the IXLCONN request.

Complete Exit

XES invokes your complete exit to inform you that a previous IXLCACHE, IXLLIST, or IXLLOCK request that you submitted was processed asynchronously and has completed.

For IXLCACHE and IXLLIST requests, the complete exit is invoked when you specify either:

- MODE=ASYNCEXIT

- MODE=SYNCEXIT, which then received a return code IXLRETCODEWARNING and a reason code IXLRSNCODEASYNCH.

For IXLLOCK requests, the complete exit is invoked when you specify MODE=SYNCEXIT. However, if the lock request can be processed synchronously, the MODE keyword is ignored and the request is processed synchronously.

All connected users of a coupling facility structure must provide a complete exit. The COMPLETEXIT keyword of IXLCONN identifies the address of your routine.

Notify Exit

XES invokes your notify exit to inform you that another connected user of a structure has requested use of the resource associated with the structure.

- For a serialized IXLLIST request, the notify exit is used to inform a connected user that other connected users have requested a lock currently owned by this user.
- For an IXLLOCK request, the notify exit is used by the connected user managing contention for a resource to communicate with other owners of the resource.

Connected users of lock and serialized list structures must provide a notify exit. The NOTIFYEXIT keyword of IXLCONN identifies the address of your routine.

Contention Exit

XES invokes your contention exit to allow a connector to assume resource management responsibilities when contention for a resource is recognized. This process of presenting a request for a resource is called percolation. Connected users of a lock structure must provide a contention exit. The CONTEXIT keyword of IXLCONN identifies the address of your routine.

List Transition Exit

XES invokes your list transition exit to inform you that a list header that you are monitoring has changed from an empty to a non-empty state. Connected users of a list structure that are using the list monitoring function of IXLLIST can provide a list transition exit, depending on the type of monitoring being done. The LISTTRANEXIT keyword of IXLCONN identifies the address of your routine.

Summary of Required Exit Routines

For a **cache structure**, the event exit and the complete exit are required.

For a **list structure**, the event exit and the complete exit are required. The list transition exit is optional.

For a **serialized list structure**, the event exit, complete exit, and notify exit are required. The list transition exit is optional.

For a **lock structure**, the event exit, complete exit, notify exit, and contention exit are required.

See “Coding Exit Routines for Connection Services” on page 352 for information about writing exit routines.

Determining the Success of a Connection

When you invoke IXLCONN, you identify the storage area where the system is to return information about the success or failure of your connect request.

RETCODE
Contains the return code.

RSNCODE
Contains the reason code.

If your request to connect to a structure is successful, RETCODE contains one of the following:

IXLRETCODEOK
Your connection is successful. The system has returned data to you in the answer area. See "Receiving Answer Area Information" on page 245.

IXLRETCODEWARNING
Your connection is successful, but you might need to do additional processing based on the information returned to you in the answer area. See "Receiving Answer Area Information" on page 245.

If RSNCODE is IXLRSNCODESPECIALCONN, check the CONAFLAGS field in the answer area.

When your connection is successful, it is your responsibility to verify that the structure attributes, which may differ from those which you requested, are acceptable.

If your request to connect to a structure is unsuccessful, RETCODE contains one of the following:

IXLRETCODEPARMERROR
You have incorrectly specified a parameter on the IXLCONN request.

IXLRETCODEENVERROR
There is an environmental error.

IXLRETCODECOMPERROR
A system failure occurred. Provide IBM with the diagnostic data available in the answer area.

The reason codes for each of the unsuccessful return codes are defined in IXLYCON, Cross-System Extended Services Constants.

Receiving Answer Area Information

When you invoke IXLCONN, you identify the storage area where the system is to return information about the status of your request. Use the following IXLCONN parameters to specify this area:

ANSAREA
Contains the address of the answer area. Use the IXLYCONA macro to map this area.

ANSLEN
Contains the length of the answer area. It must be large enough to hold the answer area mapped by IXLYCONA.

At the completion of IXLCONN processing, the answer area contains the following information depending on the outcome of the request to connect to a structure.

Successful Completion of a Connection

IXLCONN returns the following information in the ANSAREA area:

CONACONTOKEN
Token that uniquely identifies the connection within the sysplex. You must specify the CONACONTOKEN value returned by IXLCONN as input to other structure requests such as IXLCACHE, IXLLIST, or IXLLOCK.

Whenever the following events occur, the system invalidates your CONACONTOKEN:

- If the structure is in certain phases of the user-managed rebuild or duplexing processes. (Note that the CONTOKEN is not invalidated during system-managed processes.)
- If your connection disconnects or fails.
- If a structure fails, or a failure of the coupling facility occurs.
- If you lose connectivity to a structure.

You cannot access the structure when the CONACONTOKEN is invalidated.

CONACONNAME
Name that uniquely identifies the connection to the structure. If you do not specify a name on IXLCONN, the system generates a unique name.

CONACONID
A connection identifier to identify this active connection. If the active connection becomes failed-persistent, the connection retains the same connection identifier. If the failed persistent connection is able to reconnect, the CONACONID remains the same.

CONASTRUCTUREATTRIBUTES
Structure specific attributes. You must verify that the attributes for the structure are acceptable. If the attributes are not acceptable, you can release your connection by issuing IXLDISC or you can attempt to rebuild the structure.

If the connector caused the structure to be allocated, the CONACONALLOC bit will be on in CONASTRUCTUREATTRFLAGS.

See "Verifying Structure Attributes" on page 247 for the type of attribute information the system returns for a cache, list, and lock structure.

Byte 2 of CONASTRUCTUREATTRIBUTES contains system-managed duplexing information, indicating whether the structure is being duplexed by system-managed duplexing rebuild, and if so, whether the primary structure is failure-isolated from the secondary structure.

CONAFLAGS
Connection status flags that indicate whether the structure is in a special state. The special states include:

- Rebuild (CONAREBUILD)
- Rebuild stop (CONAREBUILDSTOP)
- User sync point event (CONAUSYNCEVENTSET)
- Alter in progress (CONAALTERINPROGRESS)
- Whether the connection is new or has been reconnected (CONARECONNECTED).

When connecting during a user-managed structure rebuild process or when a user sync point is set, you are expected to participate in the process indicated by the CONAFLAGS and respond to the event. The connect answer area contains the information that you would have received in the event exit if you had been connected to the structure at the time of the event.

If you connect to a structure that is in the process of being altered, the CONAALTERINFO area contains information about changes being made to the structure.

CONAREBUILDFLAGS
Flags for a connection that occurs during user-managed duplexing rebuild processing. Information includes:

- Duplexing rebuild in progress (CONAREBUILDDUPLEX)
- Duplexing rebuild switch in progress (CONAREBUILDDUPLEXSWITCH)

CONACONNECTIONVERSION
Connection version number. Each time you connect to a version of the structure, your connection version number increases. For example, if a failed-persistent connection reconnects to a structure, the connection version number is incremented and is greater than the connection version number of the original connection. However, if you connect with the REBUILD option, the connection version number is the same as the original connection version number. The rebuild connect request does not define a new connection; at rebuild connect time the original connection must be active.

CONASTRUCTUREVERSION or CONAPHYSICALSTRUCTUREVERSION
Physical structure version number. Connectors that specified or defaulted to IXLCONN ALLOWAUTO=NO use this field to uniquely identify a physical instance of a structure. Connectors that specified IXLCONN ALLOWAUTO=YES must use this field, along with CONAPHYSICALSTRUCTUREVERSION2, to identify a physical instance of the structure. Each time a structure is allocated for the same structure name, the version number for the structure increases. For example, when a new structure is allocated during rebuild, the structure version number of the new structure is greater than the structure version number of the original structure. See "Understanding the Structure Version Numbers" on page 252.

CONAPHYSICALSTRUCTUREVERSION2
Second physical structure version number. Applicable only for connectors that specified IXLCONN ALLOWAUTO=YES. This field, along with CONAPHYSICALSTRUCTUREVERSION, uniquely identifies a physical instance of the structure. See "Understanding the Structure Version Numbers" on page 252.

CONALOGICALSTRUCTUREVERSION
Used for diagnostic purposes.

CONAFPCONNSNOTINPOLICY
Information about failed-persistent connections, specifically the number of failed-persistent connections that are defined in the structure, but which could not be reconciled into the policy because the number of CONNECT records in the CFRM active policy is too small. This situation occurs only when all systems fail and the first system is re-IPLed into the sysplex with a CFRM policy that supports a smaller number of CONNECT records. The system issues warning message IXC502I.

CONAUSERSYNCPOINTEVENT
A user sync point event if one was defined by an existing connector using the IXLUSYNC macro. You are expected to perform the processing required for the event and then provide a confirmation using the IXLUSYNC macro. See "Using IXLUSYNC to Coordinate Processing of Events" on page 344.

CONAREBUILDINFO
Information for a connection that connects during user-managed structure rebuild process. You are expected to participate in the processing by responding to events. Further, for user-managed duplexing rebuild, once the structure is in the Duplex Established phase, you are expected to maintain the synchronization of the data in the duplexed structure. of duplexed structure data. See "Structure Rebuild Processing" on page 264.

CONAALTERINFO
Information for a connection that connects during structure alter. The information is valid when the alter flag in CONAFLAGS (CONAALTERINPROGRESS) is set.

CONAFACILITYARRAY
If this connection allocated the structure (CONACONNALLOC is set), the connect answer area contains information about each coupling facility in which the system attempted to allocate the structure in order to explain why the structure was allocated in the coupling facility that it was. If the connect request failed because no suitable coupling facility was found in the preference list (reason code IXLRSNCODENOFAC and CONACONNALLOC not set), this array indicates which coupling facilities were attempted and describes why each was not suitable. A reason code (CONAFACILITYRSNCODE) is provided for each coupling facility in which the system could not allocate the structure, which explains why the structure was not allocated in that coupling facility.

CONACFACILITYINFO
Current information about the coupling facility in which the structure is allocated. The information includes the operational level of the coupling facility, space utilization, and model-dependent limits.

CONAESTIMATEDMAXENTRIES
Estimated maximum number of entries supported by the structure. Using both real storage and storage-class memory, you can allocate at most this number of entries to the structure. This count is only an estimate and therefore only substantially accurate. Connectors must not rely on the availability of exactly this number of entries for use. The number is zero when storage-class memory is not associated with the structure.

CONAESTIMATEDMAXELEMENTS
Estimated maximum number of elements supported by the structure. Using both real storage and storage-class memory, you can allocate at most this number of elements to the structure. This count is only an estimate and therefore only substantially accurate. Connectors must not rely on the availability of exactly this number of elements for use. The number is zero when storage-class memory is not associated with the structure.

Verifying Structure Attributes

The system returns the following information for the allocated structure. If the attributes with which the structure has been allocated are not acceptable, you can release your connection.

Cache Structure

IXLCONN returns the following structure attributes for a cache structure:

CONACACHEDIRENTRYCOUNT
Approximate number of directory entries supported in the structure. This count is only substantially accurate.

CONACACHEMAXELEMENTCOUNT
Approximate maximum number of data elements supported by the structure. This count is only substantially accurate.

CONACACHEADJUNCT
Flag to indicate whether the structure supports adjunct data.

CONACACHEUDFORDER
Flag to indicate whether the structure supports a queue ordered by user data field for each cast-out class. Only applicable to cache structures allocated in a coupling facility with CFLEVEL=5 or higher.

CONACACHENAMECLASSMASK
Value of the name class mask in effect for the structure. Applicable only to cache structures allocated in a coupling facility of CFLEVEL=7 or higher.

CONACACHEMAXSTGCLASS
Maximum storage class value.

CONACACHEMAXCOCLASS
Maximum castout class value.

CONACACHEELEMCHAR
Data element characteristic, if applicable.

CONACACHEELEMINCRNUM
Data element increment number, if applicable.

CONACACHEMAXELEMNUM
Maximum number of data elements per entry, if applicable.

CONACACHECHGDIRENTRYCOUNT
Approximate count of changed directory entries. Applies only to cache structures allocated in a coupling facility of CFLEVEL=1 or higher.

CONACACHECHGDIRELEMENTCOUNT
Approximate count of changed data elements. Applies only to cache structures allocated in a coupling facility of CFLEVEL=1 or higher.

List Structure

IXLCONN returns the following structure attributes for a list structure:

CONALISTFLAGS
Flags to indicate whether list counts are kept on an entry or an element basis, whether the structure supports lock entries, data elements, and adjunct data, and whether the structure supports named or keyed entries.

CONALISTELEMINCRNUM
Data element increment number, if applicable.

CONALISTELEMCHAR
Data element characteristic, if applicable.

CONALISTMAXELEMNUM
Maximum number of data elements per entry, if applicable.

CONALISTHEADERS
Number of list headers.

CONALISTLOCKENTRIES
Number of lock entries.

CONALISTELEMENTCOUNT
Number of data elements in use at the time of the connect. This count includes the number of list elements for the structure that currently reside in coupling facility real and storage-class memory. The number is valid only if data elements are supported by the structure.

CONALISTMAXELEMENTCOUNT
Approximate maximum number of data elements supported by the real storage that is allocated to the structure. See ConaEstimatedMaxElements for the total that is available for allocation to the structure (including both elements in real storage and storage-class memory). The number is valid only if data elements are supported by the structure. This count is only an approximation and therefore only substantially accurate. Connectors must not rely on the availability of exactly this number of elements for use.

CONALISTENTRYCOUNT
Number of entries in use at the time of the connect. This count includes the number of list entries for the structure that currently reside in coupling facility real and storage-class memory.

CONALISTMAXENTRYCOUNT
Approximate maximum number of entries supported by the real storage allocated to the structure. See ConaEstimatedMaxEntries for the total that is available for allocation to the structure (including both entries in real storage and storage-class memory). This count is only an approximation and therefore only substantially accurate. Connectors must not rely on the availability of exactly this number of entries for use.

CONALISTEMCCOUNT
Number of event monitor controls in use at the time of the connect, if applicable. Applies only to keyed list structures allocated in a coupling facility with CFLEVEL=3 or higher.

CONALISTMAXEMCCOUNT
Approximate maximum number of event monitor controls in the structure, if applicable. Applies only to keyed list structures allocated in a coupling facility with CFLEVEL=3 or higher.

Lock Structure

IXLCONN returns the following structure attributes for a lock structure:

CONALOCKFLAGS
Flag to indicate whether record data elements are allocated.

CONALOCKNUMUSERS
Number of users supported.

CONALOCKENTRIES
Number of lock entries in the structure.

CONALOCKRECORDELEMENTS
Actual number of record elements in use at the time of the connect, if applicable.

CONALOCKMAXRECORDELEMENTS
Maximum number of record data elements supported by the structure, if applicable.

Handling Failed Attempts to Connect to a Structure

When IXLCONN is not successful (the system rejects a connect request), you must consider the situations that might have caused the rejection. In a short term situation, you probably want to reissue the connect request in a timely manner. Examples of short term situations that cause a connect request to be rejected are:

- The requested structure is in structure rebuild processing.
- The requested structure is being altered and the connection either does not support the structure alter function or cannot tolerate the target values specified for the alter request.
- The requested structure is being dumped.
- A failed-persistent connection is attempting to reconnect before all other connections have provided an event exit response for the connector's failure.

For these cases, you should listen for event notification facility (ENF) event code 35 to determine when to reissue the connect request. ENF Event code 35 signals listeners about a change in the state of coupling facility resources. See "Using ENF Event Code 35" on page 250.

Another type of situation might require system administrator or operator intervention and therefore take a significantly greater amount of time to resolve. It might be necessary to activate a new policy or reconfigure connectivity to a coupling facility. Examples of longer term situations that cause a connect request to be rejected are:

- All connections to the specified structure are in use. (The maximum number of connections for which the CFRM policy was formatted has been reached.)
- A request to join an XCF group failed. (The maximum number of groups and/or members for which the sysplex couple data set was formatted has been reached.)
- The requested structure name is not defined in the active policy.
- The requesting system does not have connectivity to the coupling facility containing the specified structure.
- The structure allocation failed because there was no suitable facility to allocate the structure based on the preference list in the policy.
- The connection failed because information about the previous instance of this connection (for reconnect) could not be reconciled into the policy.
- The coupling facility function is not active. (There might be no CFRM couple data set available or a CFRM policy might not be active.)
- The coupling facility has insufficient connectivity to systems in the sysplex.

On systems with OW33615 installed or which are at OS/390 Release 9 or higher, the system provides additional processing when an attempt to connect to a structure fails. The system first checks the state of the connectors to the structure.

- If connectors are in an active or failing state, for certain system rejections of a connect request, the system will retry the IXLCONN invocation internally for a period of time before returning to the caller. The return code that is finally returned will indicate the results of only the most recent retry attempt.

  The system will retry a connection attempt when the following transient conditions prevent the connect request from succeeding:

  – IXLRSNCODENOMORECONNS
  – IXLRSNCODECONNPREVENTED
  – IXLRSNCODERSPNOTREC
  – IXLRSNCODEDUMPINPROGRESS
- If the system is unable to complete the allocate and connect request because there are no active connectors to the structure and no system in the sysplex has connectivity to the coupling facility containing the structure, the system forces the structure and any failed-persistent connectors. Once the structure has been deallocated, the system will attempt to allocate a new instance of the structure.

  In a duplexing environment, if the structure is duplexed and the primary structure fails, as long as there is connectivity to the secondary structure, a duplex switch is done and the requesting connector is connected to the surviving structure.

Using ENF Event Code 35

ENF Event code 35 is available to inform interested subsystems or applications of changes in the state of sysplex resources. Use the ENF Event code 35 to monitor both the availability of coupling facility resources and the changes to system membership in the sysplex. For example, use the ENF Event code 35 to be notified when coupling facility resources are now available so that you can reissue a previously rejected connection request. Or, use the ENF Event code 35 to be notified when a system is joining or has been partitioned from the sysplex.

When using ENF Event code 35 to monitor the availability of coupling facility resources, define an ENFREQ LISTEN service to specify a listen exit for event code 35 *prior to* attempting a connect request. If the system rejects the connect request, the listen exit receives control when there is a change in the state of coupling facility resources. The system issues the ENF signal on all active systems in the sysplex with an established ENF event code listen routine. You can then retry the rejected connect request. Delete the listen exit once the connect request is successful.

When a system either is joining or has been removed from the sysplex, the system issues the ENF signal on all active systems in the sysplex, except on the system that is joining or has been partitioned from the sysplex.

On entry to the application's or subsystem's ENF listen exit, GPR 1 contains the address of a fullword that contains the address of the ENF parameter list. The XCF ENF Signal parameter list, mapped by IXCYENF, contains:

- The particular function code for the event being signalled.
- If applicable, the coupling facility structure name that has been affected by some action (such as a change in coupling facility policy or the completion of a rebuild request).
- If applicable, the system name and system ID (slot number) of a system that has either entered the sysplex or been removed from the sysplex.

The function code returned in IXCYENF indicates to the ENFREQ LISTEN subscriber that the system has determined that one of the following has occurred:

- **IXCYENFFUNCTIONRESAVAIL** — A new coupling facility resource is available. The system, however, cannot limit the scope to a particular coupling facility structure becoming available. Some reasons for these changes in availability might be:
  - Introduction of a new coupling facility
    - Due to a change in coupling facility policy
    - Due to one or more additional coupling facilities being made available
    - Due to a change in coupling facility connectivity.
  - Deallocation of a coupling facility structure.
  - Completion of a structure alter that reduced the size of a structure.
  - Deallocation or decrease in the amount of coupling facility dump space.
  - Change in coupling facility policy.
  - Change in coupling facility volatility state where the coupling facility has become non-volatile.
- **IXCYENFFUNCTIONSTRAVAIL** — A particular coupling facility structure has been affected by some change. The system provides the name of the coupling facility structure in the parameter list. Some changes that might trigger this function code are:
  - The SETXCF MODIFY command is used to permit CF structure alter processing.
  - Disconnection of a user of a coupling facility structure.
  - Completion of structure rebuild processing for a coupling facility structure.
  - Completion of alter processing for a coupling facility structure.
  - Release of dump serialization for a coupling facility structure.
  - Information concerning a coupling facility structure or a connection to a coupling facility structure reconstructed into the active CFRM policy from the coupling facility.
- **IXCYENFFUNCTIONSYSJOINEDSYSPLEX** — A system has joined the sysplex.
- **IXCYENFFUNCTIONSYSLEFTSYSPLEX** — A system has been partitioned from the sysplex.
- **IXCYENFFUNCTIONSITEUPDATE** — A CF definition with a SITE specified has been added or an existing CF SITE specification has changed.

When to Use ENF Event Code 35

For monitoring coupling facility resources, you can use either the ENF event code 35 interface or the event exit interface.

The ENF Event code 35 interface is intended for the application or subsystem that is attempting to connect to a structure in a coupling facility, but has not been successful. The information returned by this interface indicates that new coupling facility resources are now available and that the user should, if appropriate, reissue the connect request.

The event exit interface is intended for the application or subsystem that has successfully connected to a connector to a structure in a coupling facility. The information returned by the event exit pertains to structure availability and connection specific status.

For monitoring sysplex membership, you can use either the ENF event code 35 interface or the group user routine interface.

- The ENF Event code 35 interface is intended for the application or subsystem that is not a member of an XCF group (and therefore does not have a group user routine established), but that needs to be aware when a system is joining or has been partitioned from the sysplex. The information returned by this interface identifies the system by name and system ID.

  To avoid the system overhead of joining an XCF group, an ENF listen exit might be more appropriate for your application.

For information about ENFREQ, see *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG*.

Understanding the Structure Version Numbers

The structure version number (CONAPHYSICALSTRUCTUREVERSION) is used to identify the instance of a structure with a given name. The structure version number changes when a new instance of the structure is allocated, as in a user-managed or system-managed rebuild, when there is at least one active connector to observe the allocation. For example, in a user-managed rebuild, the value of the physical structure version number that is returned on an initial connect to a structure might be “A”. When the IXLCONN REBUILDs are performed during the user-managed rebuild process, the physical structure version number returned for the new structure might be “B”. Keeping track of a structure's physical structure version number allows you to uniquely identify the instance of the structure with which you are working.

In OS/390 Release 8, a second physical structure version number (CONAPHYSICALSTRUCTUREVERSION2) is introduced. This version number is used only in system-managed protocols. Its purpose, in combination with CONAPHYSICALSTRUCTUREVERSION, is to uniquely identify an instance of a structure. For example, if a connector supports system-managed protocols (that is, specifies ALLOWAUTO=YES), the version numbers received on initial connect might be:

CONAPHYSICALSTRUCTUREVERSION
A

CONAPHYSICALSTRUCTUREVERSION2
0

During a system-managed rebuild, the version numbers provided with the Structure State Change event might be:

EEPLSSCSTRPHYSICALVERSION
B

EEPLSSCSTRPHYSICALVERSION2
0

During a system-managed duplexing rebuild, it is likely that both version numbers will contain non-zero values.

Because the two pairs of values are not identical, the connector can recognize that a new instance of the structure has been allocated. The following describes how these fields are intended to be used.

- When presented to the connector, whether via the Structure State Change Notification event or as the result of a connect request, the connector should “harden” **both** physical structure version numbers on whatever external media are used for recording such persistent information about structures in use.
- On any subsequent connect to the structure, the connector should compare **both** returned physical structure version numbers against **both** of the saved physical structure version numbers. If either nonzero structure version number returned on the connect request matches either of the two nonzero saved physical structure version numbers, then the user is connected to a viable structure instance and no special structure data recovery processing is required. Otherwise, such data recovery is required.

Reconnecting to a Structure

You can use the IXLCONN macro to reestablish a failed-persistent connection to a structure. Reconnection to a structure might be necessary when a connection terminates abnormally and peer recovery is not possible, or when the protocol is to use restart recovery instead of peer recovery.

To reconnect to a structure, specify the same connect name on the IXLCONN macro as was used for the prior connection to the structure. When the reconnection is complete, the version number of the structure is the same as it was for the prior connection to the structure. The system does not increment the version number of the structure because the structure is not new and has not been reallocated. However, the connection version will be different from the previous connection version.

When the connection is reconnected, IXLCONN sets a return and a reason code (IxlRsnCodeSpecialConn) to indicate that additional status information is available about the connection and possibly also the structure. A bit in the CONAFLAGS field on IXLYCONA indicates whether the connection has been reconnected. The reconnected user might need to do additional clean-up or recovery, such as for locks held or work in progress by the previous instance of the connection, depending on the application's protocol.

Note that you can use the IXCQUERY macro to determine the names of the connections that are in a failed-persistent state.

Figure 22 on page 253 illustrates structure B with two active connections, A and C. Connection A is an existing connection. Connection C has just connected as a new connection and has specified a connection disposition of KEEP and a connection name of CNAME:

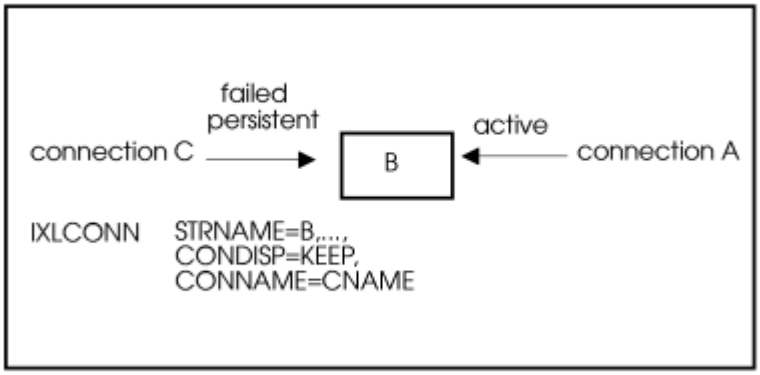


Figure 22. Active Connections

Figure 23 on page 254 illustrates what happens when connection C fails (in this case connection C issues IXLDISC for the structure with REASON=FAILURE as part of an error routine). Connection C enters a failed-persistent state:

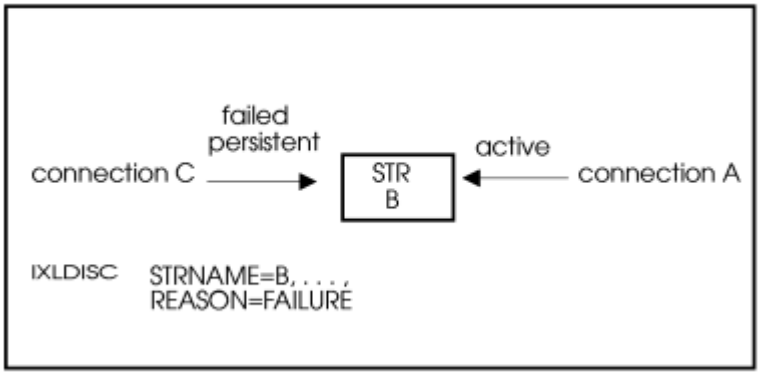


Figure 23. A Failed-Persistent Connection

Figure 24 on page 254 illustrates connection A acknowledging connection C's failure.

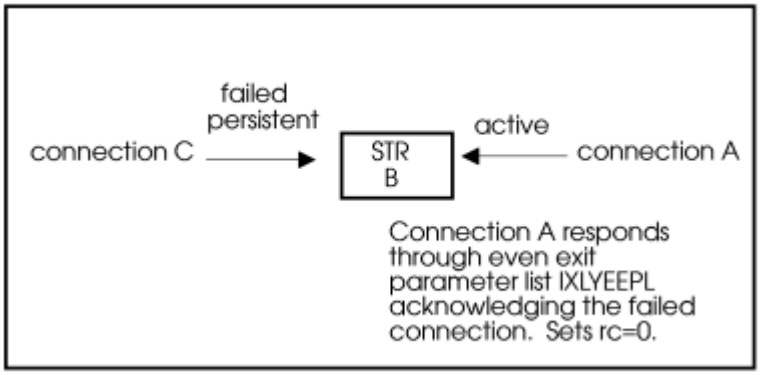


Figure 24. Acknowledging a Failed-Persistent Connection

If the connection in a failed-persistent state can restart and perform recovery for itself other active connections have acknowledged the failed connection through the event exit parameter list, the connection can reconnect. Figure 25 on page 254 illustrates what happens when connection C reconnects to structure B:

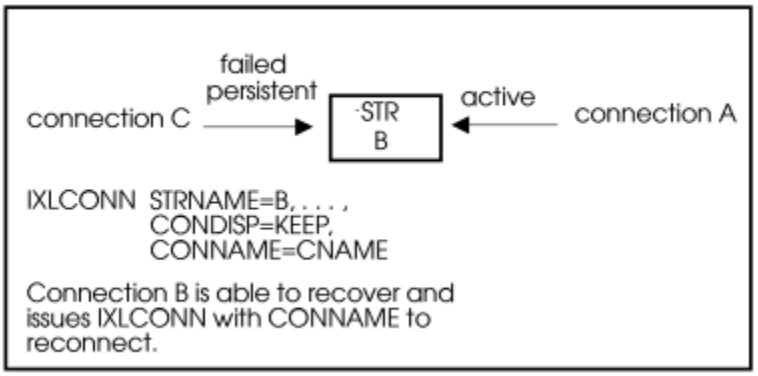


Figure 25. Reconnection of a Failed-Persistent Connection

For an example of how active connections can delete a failed-persistent connection, see "Deleting Failed-Persistent Connections" on page 262.

Connecting to a Structure During User-Managed Rebuild

The user-managed rebuild process causes the structure to be reallocated in another location, but with the same structure name. The rebuild process also allows you to change the attributes of the original structure. Connectors to the structure being rebuilt can move data from the old structure to the new structure. When the rebuild process is complete, the system deallocates the old structure and connectors continue normal processing using the new structure. See “Structure Rebuild Processing” on page 264 for a description of the rebuild process.

Depending on the phase of the rebuild process, you might or might not be allowed to connect to the structure that is being rebuilt. IXLCONN sets the following return codes when you request a connection to a structure that is being rebuilt:

- IXLCONN sets a return and a reason code (IXLRSNCODESPECIALCONN) to indicate that additional status information is available. Two bits in the CONAFLAGS field in IXLYCONA indicate whether the structure is in rebuild or rebuild stop processing. Both of these states require that the connector must participate in the user-managed rebuild process. If you do not want to participate in the process, you should issue IXLDISC to disconnect from the structure. You also have the option of causing the structure to stop being rebuilt unless rebuild stop is already in progress. See “Handling New Connections During a User-Managed Rebuild Process” on page 289 for additional information about the IXLRSNCODESPECIALCONN reason code.
- IXLCONN also might set a return and a reason code (IXLRSNCODECONNPREVENTED) to indicate that a new connection is not permitted at this time because rebuild is in progress. In this situation, you should use ENF event code 35 to determine when the rebuild process is complete. The ENF signal parameter list contains the name of the structure that has been rebuilt. See “Using ENF Event Code 35” on page 250 for information about using ENF event code 35.

Connecting to a Structure During User-Managed Duplexing Rebuild

User-managed duplexing rebuild allows connectors to request that a second instance of the structure be allocated in another coupling facility for the purpose of duplexing the data in each structure to achieve increased availability and usability. The duplexing process also allows you to change the attributes of the original structure. Connectors to the structure being duplexed can copy data from the old structure to the new structure. Once the structure is duplexed (Duplex Established phase), connectors synchronize their use of both structures. At any time it is possible to discontinue the duplex process and either fall back to using the original structure or switch (forward complete) to use the secondary structure. See “Overview of user-managed rebuild processing” on page 268 for a description of the structure duplexing process.

Depending on the phase of the duplexing process, you might or might not be allowed to connect to the structure that is being duplexed. IXLCONN sets the following return codes when you request a connection to a structure that is being duplexed:

- IXLCONN sets a return and reason code (IXLRSNCODESPECIALCONN) to indicate that additional status information is available. The same two bits in the CONAFLAGS field (CONAREBUILD and CONAREBUILDSTOP) in IXLYCONA that are used by structure rebuild also report the status of a duplexed structure. Additionally, a bit in the CONAREBUILDFLAGS field indicates whether the rebuild in progress is a duplexing rebuild. See “Handling New Connections During a User-Managed Rebuild Process” on page 289 for additional information about the IXLRSNCODESPECIALCONN reason code.
- IXLCONN also might set a return and reason code (IXLRSNCODECONNPREVENTED) to indicate that a new connection is not permitted at this time because all active connectors have confirmed the Duplex Rebuild Complete event and must complete their cleanup of one instance of the structure. New connections are not permitted until all active connections have provided an event exit response to the Rebuild Cleanup event. New connections also are not permitted if a request to stop the duplexing rebuild to fall back to using the old structure is received.

Connections to duplexed structures are allowed throughout most phases of the duplexing operation, and the connectors are expected to participate in the duplexing process. The connect answer area identifies the phase of duplexing the structure is in and whether switch processing is in progress. See “Handling New Connections During a User-Managed Rebuild Process” on page 289.

Connections to structures in the Duplex Established phase while structure alter is being processed for the duplexed structure are also allowed. See "Altering a duplexed structure" on page 326 for a description of structure alter for a duplexed structure.

Connecting to a structure during a system-managed process

Connections to a structure are not permitted in most phases of system-managed processing. During the processing, the system might be transferring data from one instance of a structure to another, deallocating an instance of a structure, or performing other operations on the affected structure that would be disrupted by new connections to the structure. IXLCONN sets reason code IXLRSNCODECONNPREVENTED to indicate that a new connection is not permitted at this time because a system-managed process is in progress. Use ENF event code 35 to determine when the rebuild process is complete. The ENF signal parameter list contains the name of the structure that was affected by the system-managed process.

During system-managed duplexing rebuild, an attempt to connect to a structure that is in the Duplex Established phase is allowed, assuming that the connector has specified ALLOWAUTO=YES on IXLCONN. Connect processing transparently attaches the connector to both structures of the duplex pair and initializes the connection so that all coupling facility operations that are subsequently performed by the connector are duplexed appropriately.

Unsuccessful attempts to connect to a structure during a duplex established phase can occur for the following reasons:

- The new or failed-persistent connector does not have connectivity to one or the other of the duplexed pair.
- One of the structure instances does not have an available CONID for the new connector.
- The structure instances of the duplexed pair have been allocated with different user–id limits, and the new connector does not support user-id limit changes (that is, the MAXCONN keyword was not specified by the new connection).
- The structure is in the Async Duplex Established phase and IXLCONN AsyncDuplex=NO was specified or defaulted to. The attempt by XES to stop duplexing to allow the connect was not successful.

In these situations, XCF handles the connect request as follows:

- The system automatically initiates duplexing rebuild stop processing to revert to simplex mode. When a connectivity issue occurs, the system keeps the structure instance to which the connector has connectivity. When the structure instances have different user-id limits, the system keeps the structure instance with the largest user-id limit. Any attempt to automatically reduplex the structure is suppressed until the duplexing stop/switch processing has completed.
- The unsuccessful connect request will be retried for a period of time to permit it to successfully connect once the structure has reverted to simplex mode. If the retried connect request is successful, an attempt to reduplex the structure will be triggered by the successful connect request. If none of the retried connect requests is successful, then the connect request fails with an appropriate return and reason code that indicates that the connector cannot connect to the structure while stop/switch processing is in progress for the structure.
- When the duplexing rebuild stop/switch processing completes, the system issues ENF 35 for structure availability. If the connector listens for ENF 35 signals, it might then retry the connect request in response to that signal, and connect to the structure in simplex mode. If the connector does not reattempt to connect in response to the ENF 35 signal, then the structure might remain unduplexed until such time as duplex enabled monitoring attempts to reestablish duplexing for the structure.

Connecting to a Structure That Is Being Altered

Structure alter dynamically changes the size and/or entry-to-element ratio of a structure without requiring connectors to quiesce their use of the structure. Connectors must be at the SP 5.2 or higher level, and must have specified ALLOWALTER=YES and CFLEVEL=1 or higher on their IXLCONN invocation. All connectors also must be consistent in their specification for RATIO — if all existing connectors indicated that the entry-to-element ratio can be changed (RATIO=YES) then any new connector must specify

RATIO=YES. Assuming that those prerequisites are met, whether a connector is allowed to connect to a structure that is in the process of being altered is determined by the entry and element minimum levels specified on IXLCONN.

- IXLCONN accepts a request to connect when the connector specifies:
  - ALLOWALTER=YES
  - Entry and element minimum levels that are the same or less restrictive than the current composite established for the structure.

  The connector must examine the connect answer area to determine the state of the structure.

  Note that if the connector determines that the structure is in the duplexing rebuild process as well as a structure alter process, the connector must be able to support the duplexing protocol.
- IXLCONN rejects a request to connect with one of the following reason codes:
  - IXLRSNCODEALTERNOTALLOW — the connection specified ALLOWALTER=NO.
  - IXLRSNCODEALTERRESTRICT
    - The connection specifies more restrictive limits for the entry and element minimum levels than are currently in effect for the structure.
    - The connection specifies RATIO=NO and the current composite established for the structure indicates that the ratio can change during structure alter.
  - IXLRSNCODECONNPREVENTED — the connection is not at the SP 5.2 level.

See "Altering a Coupling Facility Structure" on page 324 for a description of the alter process.

Connecting to a Structure when a Synchronization Point Is Set

User synchronization points are used to provide synchronization of processing among connectors to a structure. When a user attempts to connect to a structure after one of these synchronization points has been set, IXLCONN sets a return and a reason code (IXLRSNCODESPECIALCONN) to indicate that additional information is available about the connection. A bit in the CONAFLAGS field of IXLYCONA indicates that a user sync point has been set. IXLYCONA also contains the next user sync point event and the user state associated with the event. The user must do whatever processing is required by the event and respond to confirm the event by using the IXLUSYNC service. See "Using IXLUSYNC to Coordinate Processing of Events" on page 344 for information about using IXLUSYNC.

Dumping Considerations

IXLCONN rejects new connections for a structure that currently is serialized for dumping. IXLCONN sets return code IXLRETCODEENVERROR and reason code IXLRSNCODEDUMPINPROGRESS to indicate that the serialization is in effect. In this situation, the user should use ENF event code 35 to determine when the dumping serialization is released and new connections are allowed. The ENF signal parameter list contains the name of the structure for which dumping serialization has been released.

Handling a Connection's Abnormal Termination

The topics below describe how the system handles the following types of connection termination.

1. Connector's system terminates
2. Connector's address space terminates
3. Connector's task terminates
4. An address space other than the connector's address space terminates with outstanding IXLCACHE, IXLLIST, or IXLRT operations. (The connector remains active.)
5. A task other than the connector's task terminates with outstanding IXLCACHE, IXLLIST, or IXLRT operations. (The connector remains active.)

Note:

1. The connector that requests XES services must provide abnormal termination processing for a connection by establishing end-of-task (EOT)/end-of-memory (EOM) resource managers. XES assumes that the connector is the owner of any storage passed to an XES service, specifically IXLCACHE, IXLLIST, and IXLRT.

   If the connector is not the owner of the storage passed to XES, then the connector must provide an address space termination resource manager for handling cases where the owner of the storage terminates. The address space termination resource manager must invoke IXLPURGE to break any XES-established storage binds before allowing the storage to be cleaned up.
2. When a recovery exit receives control while its subsystem or system component is suspended by an IXLLIST, IXLCACHE, IXLRT, or IXLFCOMP request, the recovery exit must issue IXLPURGE to complete or purge the request. The recovery exit must do this prior to deleting any storage passed as input to XES and prior to looking at the answer area to determine the status of the request.
3. In certain instances, XES must quiesce the activity of user exits in order to perform cleanup processing. For example, when a user disconnects or abnormally terminates, XES will force to completion any user exits executing on behalf of that user by issuing a PURGEDQ against the appropriate units of work. Note that if a connector terminates while a rebuild is in progress, any exits pertaining to both the original and the new structures will be forced to completion. In addition to forcing the currently executing user exits to completion, XES will also prevent any new invocations of these exits by cancelling any events that are pending presentation.

   A user exit must be sensitive to conditions that can occur as a result of actions taken by XES and must be able to handle these as appropriate. For example, if a user exit has suspended itself, when the PURGEDQ is issued the system abends the user exit's unit of work with a retryable X'47B' abend and gives control to the user exit's recovery routine. (Note that although the recovery routine can retry, the user exit can not re-suspend itself because the system will fail any request to suspend a unit of work that has been the target of a PURGEDQ.) If the recovery routine percolates back to the system, its associated connection is terminated.

Case 1. Connector's System Terminates

When a connector's system terminates, another system in the sysplex performs the clean-up processing.

- The system notifies all peer connections through the Disconnected or Failed connection event that is presented to each peer connector's event exit.
- All peer connections must respond to the Disconnected or Failed Connection event. When the system has received all event exit responses, the connection is placed in either the undefined state or the failed-persistent state.
  - Undefined state

    1) The failed connection specified CONDISP=DELETE on the connection, or

    2) The failed connection specified CONDISP=KEEP on the connection and at least one peer connection responded that the connection should be released.
  - Failed-persistent state

    The failed connection specified CONDISP=KEEP on the connection and all peer connections responded that the connection should not be released.
- The system disconnects all connections owned by the terminated system when all responses are received. For each connection, the system must clean up all structure-specific resources, such as castout locks and registered interest for a cache structure. See “Handling Resources for a Disconnection” on page 348 for a list of resources that are cleaned up when the failed connection is detached from the coupling facility structure.
- At this point, the structure might be deallocated if the structure has a STRDISP of DELETE and there are no more defined connections.

Case 2. Connector's Address Space Terminates

When a connector's address space terminates, the connector's end-of-memory resource manager receives control in the master address space. The resource manager must perform storage clean-up before turning control over to the XES resource manager for additional processing.

- Connector Resource Manager Processing

  The EOM resource manager must clean up all storage associated with outstanding coupling facility requests, specifically the storage buffers associated with IXLCACHE, IXLLIST, and IXLRT. Note that no input storage buffers are provided for IXLLOCK. Use the IXLPURGE service to purge the outstanding requests and ensure that there are no XES-established binds to the storage associated with the request. At the completion of IXLPURGE processing, control returns to the end-of-memory resource manager with all storage binds broken. Processing after invoking IXLPURGE differs according to whether the request was asynchronous or synchronous.

  – Asynchronous request

    Request-related storage can be released without waiting for notification of request completion. Because the connector's address space is terminated, request completion notification cannot be scheduled.

  – Suspended Synchronous request

    The following processing normally occurs for an IXLLIST, IXLCACHE, or IXLRT request:

    - The answer area is initialized with IxlRsnCodeUnknown reason code prior to performing the request.
    - The answer area is updated with the request results when the request is completed. The answer area is updated while running under the requestor's unit of work with addressability from the connector's and requestor's address spaces.

    Request-related storage for requests initiated with the home address space equal to the connector's address space can be released without waiting for notification of request completion. The requestor can no longer run.

    Request-related storage for requests initiated with the home address space not equal to the connector's address space are handled by the recovery routine of the requestor. If the connector's address space has terminated, the requestor can observe the IxlRsnCodeUnknown reason code in the answer area. However, if the answer area storage is in the connector's address space, the answer area will not be addressable. For the answer area to be addressable during termination processing when a connector's address space terminates, the answer area storage must be in common storage.

- XES Resource Manager Processing

  At the completion of processing by the connector's resource manager, the XES resource manager continues the clean-up processing.

  – XES invokes IXLPURGE to release any additional storage binds. XES uses the STOKEN of the terminating address space as input to IXLPURGE.

  – XES notifies all peer connections about the termination by invoking their event exits with the Disconnection or Failed connection event.

  – All peer connections must respond to the Disconnected or Failed Connection event. When the system has received all event exit responses, the connection is placed in either the undefined state or the failed-persistent state.

    - Undefined state

      1) The failed connection specified CONDISP=DELETE on the connection, or 2) The failed connection specified CONDISP=KEEP on the connection and at least one peer connection responded that the connection should be released.

    - Failed-persistent state

      The failed connection specified CONDISP=KEEP on the connection and all peer connections responded that the connection should not be released.

– XES disconnects a connection owned by the terminating address space from the structure when all responses are received. For each connection, all structure-specific resources such as local vectors, castout locks, etc. are cleaned up. See “Handling Resources for a Disconnection” on page 348 for resources associated with each structure type.
– At this point, the structure might be deallocated if the structure has a STRDISP of DELETE and there are no more defined connections.

Case 3. Connector's Task Terminates

When a connector's task terminates, the connector's end-of-task resource manager receives control running under the failing task. The resource manager must perform storage clean-up before turning control over to the XES resource manager for additional processing.

- Connector Resource Manager Processing

  The connector's end-of-task resource manager must clean up all storage associated with outstanding coupling facility requests, specifically the storage buffers associated with IXLCACHE, IXLLIST, and IXLRT. Note that no input storage buffers are provided for IXLLOCK. Use the IXLPURGE service to purge the outstanding requests and ensure that there are no XES-established binds to the storage associated with the request. At the completion of IXLPURGE processing, control returns to the end-of-task resource manager with all storage binds broken. IXLPURGE processing differs according to whether the request was asynchronous or synchronous.

  – Asynchronous request

    Request-related storage cannot be deleted until the connector is notified about each request's completion. The connection is still active and therefore, request completion notification will be scheduled normally (either through posting an ECB or driving the complete exit). Issue IXLFCOMP to obtain the results of asynchronous request tokens. If necessary, invoke IXLFCOMP before invoking IXLPURGE.

  – Suspended Synchronous request

    The requestor's recovery routine receives control for a suspended request running under the connector's task. Prior to this, the XES recovery routine received control and attempted to complete the request. The request recovery routine must issue IXLPURGE to ensure that the request is complete.

    The system resumes a suspended request associated with a task other than the connector's task and returns a return code that indicates whether the request was purged or completed. The system resumes the suspended task whether the suspended task's home address space is equal to the connector's address space or not.

- XES Resource Manager Processing

  At the completion of processing by the connector's resource manager, the XES resource manager continues the clean-up processing.

  – XES invokes IXLPURGE to release any additional storage binds. XES uses the TTOKEN of the terminating task as input to IXLPURGE.
  – XES notifies all peer connections about the termination by invoking their event exits with the Disconnection or Failed connection event.
  – All peer connections must respond to the Disconnected or Failed connection event. When the system has received all event exit responses, the connection is placed in either the undefined state or the failed-persistent state.

    - Undefined state

      1) The failed connection specified CONDISP=DELETE on the connection, or 2) The failed connection specified CONDISP=KEEP on the connection and at least one peer connection responded that the connection should be released.

    - Failed-persistent state

The failed connection specified CONDISP=KEEP on the connection and all peer connections responded that the connection should not be released.

- XES disconnects a connection owned by the terminating task from the structure when all responses are received. For each connection, all structure-specific resources such as local vectors, castout locks, etc. are cleaned up. See “Handling Resources for a Disconnection” on page 348 for resources associated with each structure type.
- At this point, the structure might be deallocated if the structure has a STRDISP of DELETE and there are no more defined connections.

Case 4. An Address Space Other Than the Connector's Address Space Terminates with Outstanding IXLCACHE, IXLLIST, or IXLRT Operations. The connection remains active.

When an address space other than the connector's terminates, the connector's end-of-memory resource manager receives control in the master address space. The resource manager must perform storage clean-up before turning control over to the XES resource manager for additional processing.

- Connector Resource Manager Processing

  The connector's end-of-memory resource manager must clean up all storage associated with outstanding coupling facility requests, specifically the storage buffers associated with IXLCACHE, IXLLIST, and IXLRT. Note that no input storage buffers are provided for IXLLOCK. Use the IXLPURGE service to purge the outstanding requests and ensure that there are no XES-established binds to the storage associated with the request. At the completion of IXLPURGE processing, control returns to the end-of-memory resource manager with all storage binds broken. IXLPURGE processing differs according to whether the request was asynchronous or synchronous.

  - Asynchronous request

    When IXLPURGE completes, the complete exit and ECB notifications complete normally for asynchronous requests. Issue IXLFCOMP to obtain the results of asynchronous token requests. Once IXLPURGE completes, the system does not suspend an IXLFCOMP request because the outstanding request has already been purged and therefore is complete. Request-related storage cannot be deleted until all processing for the request has been completed.

    Note: In order to issue IXLFCOMP, the requestor must be running with the primary address space equal to the connector's primary address space and have the same addressability as when the asynchronous request was initially issued.

  - Suspended synchronous request

    For IXLLIST, IXLCACHE, and IXLRT requests, XES initializes the answer area mapped by the appropriate macro, IXLYLAA, IXLYCAA, or IXLYRTAA, with the IxlRsnCodeUnknown reason code prior to performing the request. When the request completes, XES updates the answer area with the request results, while running under the requestor's unit of work and with addressability to the connector's and the requestor's address spaces. If the requestor's address space has terminated, the requestor observes the IxlRsnCodeUnknown reason code in the answer area.

- XES Resource Manager Processing

  At the completion of processing by the connector's resource manager, the XES resource manager continues the clean-up processing.

  - XES invokes IXLPURGE to release any additional storage binds. XES uses the STOKEN of the terminating address space as input to IXLPURGE.

Case 5. A Task Other Than the Connector's Task Terminates with Outstanding IXLCACHE, IXLLIST, or IXLRT Operations. The connection remains active.

When a task other than the connector's task terminates, the connector's end-of-task resource manager receives control running under the failing task. The resource manager must perform storage clean-up before turning control over to the XES resource manager for additional processing.

- Connector Resource Manager Processing

The connector's end-of-task resource manager must clean up all storage associated with outstanding coupling facility requests, specifically the storage buffers associated with IXLCACHE, IXLLIST, and IXLRT. Note that no input storage buffers are provided for IXLLOCK. Use the IXLPURGE service to purge the outstanding requests and ensure that there are no XES-established binds to the storage associated with the request. At the completion of IXLPURGE processing, control returns to the end-of-task resource manager with all storage binds broken. IXLPURGE processing differs according to whether the request was asynchronous or synchronous.

- Asynchronous request

  When the IXLPURGE request completes, if the connector is still active, request completion notification is scheduled normally (either through posting an ECB or driving the complete exit). Issue IXLFCOMP to obtain the results of asynchronous token requests. Once IXLPURGE completes, the system does not suspend an IXLFCOMP request because the outstanding request has already been purged and therefore is complete. Request-related storage cannot be deleted until all processing for the request has been deleted.

  Note: In order to issue IXLFCOMP, the requestor must be running with the primary address space equal to the connector's primary address space and have the same addressability as when the asynchronous request was issued initially.

- Suspended synchronous request

  For IXLLIST, IXLCACHE, and IXLRT requests suspended at the time of the failure, the XES recovery routine receives control and attempts to complete the request. When the requestor's recovery routine receives control, the connector must issue IXLPURGE in order to ensure that the request is complete.

- XES Resource Manager Processing

  At the completion of processing by the connector's resource manager, the XES resource manager continues the clean-up processing.

  - XES invokes IXLPURGE to release any additional storage binds. XES uses the TTOKEN of the terminating task as input to IXLPURGE.

Deleting Persistent Structures

When there are no defined (active or failed-persistent) connections to a structure with a disposition of KEEP, the structure is persistent and remains allocated. In most cases, to delete a persistent structure after there are no defined connections, you can do the following:

- Issue the IXLFORCE macro
- Instruct the operator or use an extended MCS console interface to issue the SETXCF FORCE command.

On systems with OW33615 installed or which are at OS/390 Release 9 or higher, the system will automatically FORCE a structure and all failed-persistent connectors to the structure to which there is no connectivity if:

- An attempt is made to connect to the structure, and
- The structure has no active connectors, and
- No system in the sysplex has connectivity to the coupling facility containing the existing structure.

See *z/OS MVS Programming: Sysplex Services Reference* for information about IXLFORCE, and *z/OS MVS System Commands* for information about SETXCF FORCE.

With system-managed duplexing rebuild processing, if all active connectors disconnect or fail, there are no failed-persistent connectors, and the structure is non-persistent, both structure instances are deallocated and the structure is no longer in system-managed duplexing.

Deleting Failed-Persistent Connections

Failed-persistent connections result when a connection with a disposition of KEEP fails as the result of a task, address space, or system failure, or when IXLDISC REASON=FAILURE is issued. Failure of

the connection is reported to the event exit of all connected users. When all connectors acknowledge the event, the failing connection with a disposition of KEEP becomes failed-persistent. Users develop protocols on how to handle failed-persistent connections. If the failed-persistent connection cannot reconnect to the structure, you can delete the failed-persistent connection.

Connections can delete a failed-persistent connection in the following ways:

- Through the event exit or IXLEERSP macro
- IXLFORCE macro

The following steps summarize the process by which an active connection uses the event exit or IXLEERSP to eliminate the failing connection:

1. All active connections are informed of the failing connection through their event exits.
2. If one or more active connections can perform recovery for the failing connection, they do so.
3. The active connection indicates to MVS that recovery for the failing connection has completed by doing one of the following:
   - Setting return code = X'01' in IXLYEEPL event exit parameter list before returning from the event exit
   - Setting a return code = X'08' in IXLYEEPL before returning from the event exit. When the recovery processing is complete, the active connector issues the IXLEERSP macro with EVENT=DISCFAILCONN,RELEASECONN=YES.
4. Active connections must respond to the Disconnected or Failed Connection event through their event exits. The failing connection remains in the failing state until all acknowledgments are received.

See “Responding to Connection Events” on page 333 and “Using IXLEERSP” on page 355.

Figure 26 on page 263 illustrates what happens when an active connection to structure B performs recovery for failed-persistent connection C and sets return code 1 in IXLYEEPL to release the connection:

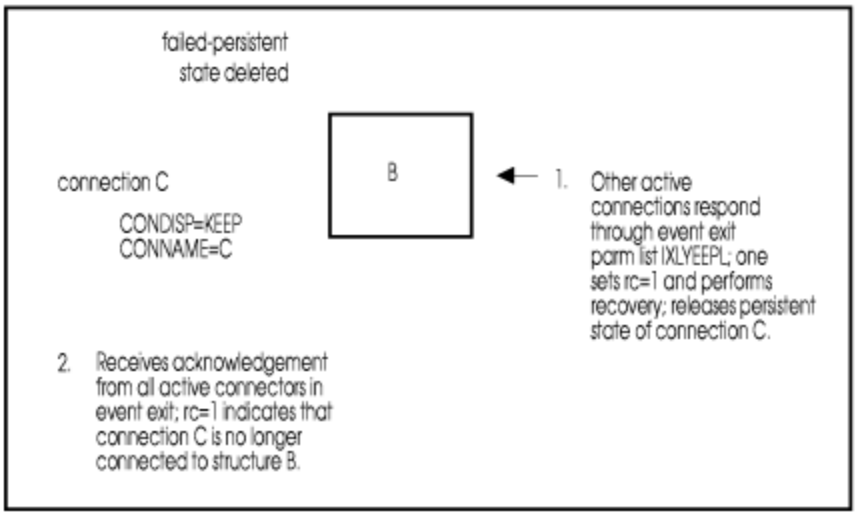


Figure 26. Deleting a Failed-Persistent Connection using IXLYEEPL

On systems with OW33615 installed or which are at OS/390 Release 9 or higher, the system will automatically FORCE all failed-persistent connections and the associated structure when the IXLCONN macro is invoked to connect to the inaccessible structure, there are no active connectors, and there is no connectivity to the coupling facility containing the inaccessible structure.

Using IXLFORCE or the SETXCF FORCE Command

The IXLFORCE macro or SETXCF FORCE command deletes a persistent structure or a failed-persistent connection. Users can invoke the macro or command to perform resource cleanup on structures. In order

to delete the structure as a result of the macro or command, active connections must disconnect normally and persistent-connections must be released. Active connections must acknowledge the failing state of a connection before the macro or command can delete the failed-persistent connection.

You may use RACF or another security product to protect structures. See "Authorizing Coupling Facility Requests" on page 204. For information on protecting the use of MVS commands like SETXCF, see *z/OS MVS Planning: Operations*.

See "Forcing the Deletion of a Coupling Facility Object" on page 350 for additional information about the IXLFORCE macro.

Structure Rebuild Processing

There are two types of structure rebuild processing. Rebuild involves the construction of a new instance of a structure. Duplexing rebuild involves the construction of a secondary instance of a structure and the subsequent maintenance of both instances of the structure in a synchronized manner.

There are two methods by which rebuild can be accomplished: user-managed and system-managed. The primary difference between the two methods is in the amount of direct participation by the connectors.

- The user-managed method requires that connectors to the coupling facility containing the structure develop protocols to coordinate the rebuilding or duplexing of the structure and the propagation of data within the structure. User-managed rebuild processing requires that there be at least one active connector to the coupling facility containing the structure.
- The system-managed method requires that connectors recognize that the structure will become temporarily unavailable for requests but does not require them to develop protocols to coordinate rebuild processing. The system provides the support necessary for rebuild or duplexing rebuild and therefore does not require that there be any active connectors to the coupling facility structure being rebuilt.

The following table outlines the basic differences between user-managed rebuild and system-managed rebuild.

Table 15. User-Managed vs. System-Managed Rebuild Processing

Feature	User-Managed Rebuild	User-Managed Duplexing Rebuild	System-Managed Rebuild	System-Managed Duplexing Rebuild
Purpose	Planned reconfiguration and recovery.	Improved availability and usability for cache structures.	Primarily planned reconfiguration . Cannot recover for loss of structure connectivity, structure failure, or coupling facility failure.	Robust failover capability.
Minimum CFLEVEL	None.	None.	CFLEVEL=8.	CFLEVEL=11.
Minimum release	MVS SP5.1.0.	OS/390 V2 R6	OS/390 V2 R8	z/OS V1 R2
Active CFRM couple data set requirements	Any.	Any.	Formatted to support system-managed rebuild.	Formatted to support system-managed duplexing rebuild.

Table 15. User-Managed vs. System-Managed Rebuild Processing (continued)

Feature	User-Managed Rebuild	User-Managed Duplexing Rebuild	System-Managed Rebuild	System-Managed Duplexing Rebuild
Connector requirements	At least one active connector required.	At least one active connector required.	No active connector requirements.	No active connector requirements.
Treatment of failed-persistent connectors	Not preserved across the rebuild.		Preserved across the rebuild.	Preserved across the rebuild.
Connector support	Controlled by connector specification of IXLCONN ALLOWREBLD keyword.	Controlled by connector specification of IXLCONN ALLOWDUPREBLD and ALLOWREBLD keywords.	Controlled by connector specification of IXLCONN ALLOWAUTO keyword.	Controlled by connector specification of IXLCONN ALLOWAUTO keyword.
Initiation	SETXCF START,REBUILD command or IXLREBLD macro	SETXCF START,REBUILD, DUPLEX command or IXLREBLD macro, or internally by the system.	SETXCF START,REBUILD command or IXLREBLD macro	SETXCF START,REBUILD, DUPLEX command, IXLREBLD macro, or internally by the system.
Quiescing access to the structure	Responsibility of the connected users.	Responsibility of the connected users.	Responsibility of the system on behalf of the users. Events are presented that allow the connectors to optionally do some quiescing of requests at the connection level.	Responsibility of the system on behalf of the users. Events are presented that allow the connectors to optionally do some quiescing of requests at the connection level.

Table 15. User-Managed vs. System-Managed Rebuild Processing (continued)

Feature	User-Managed Rebuild	User-Managed Duplexing Rebuild	System-Managed Rebuild	System-Managed Duplexing Rebuild
Events received by active connection(s)	May include: • Rebuild Quiesce • Rebuild Connect • Rebuild Connects Complete • Rebuild New Connection • Rebuild Existing Connection • Rebuild Connect Failure • Rebuild Cleanup • Rebuild Complete • Rebuild Stop • Rebuild Stop Complete	May include, in addition to the events listed for user-managed rebuld: • Rebuild Duplex Established • Rebuild Switch	May include: • Structure Temporarily Unavailable • Structure State Change • Structure Available • Alter Begin • Alter End Alter events are presented only if the connector supports alter processing.	May include: • Structure Temporarily Unavailable • Structure State Change • Structure Available
Creation of new structure	Allocated by first connector to perform rebuild connect.	Allocated by first connector to perform rebuild connect.	Allocated by the system without connector participation.	Allocated by the system without connector participation.
CONTOKEN use	Two CONTOKENs for the user to manage, both of which the system might invalidate during the user-managed rebuild process.	Two CONTOKENS.	One CONTOKEN that is neither changed nor invalidated.	One CONTOKEN.

Table 15. User-Managed vs. System-Managed Rebuild Processing (continued)

Feature	User-Managed Rebuild	User-Managed Duplexing Rebuild	System-Managed Rebuild	System-Managed Duplexing Rebuild
Structure attribute changes	Changes can be requested at rebuild connect.	Changes can be requested at rebuild connect.	Original structure attributes are preserved. The structure size might differ from the old structure, and if the connectors allow alter, the structure object counts might differ as well. The user-id limit for list and lock structures can change if all the connectors support user-id limit changes by specifying MAXCONN on the IXLCONN request.	Original structure attributes are preserved. The user-id limit for list and lock structures can change if all the connectors support user-id limit changes by specifying MAXCONN on the IXLCONN request.
Connection to new structure	All connectors must reconnect via IXLCONN REBUILD.	All connectors must reconnect.	Reconnected by the system without connector participation.	Reconnected by the system without connector participation.
Population of new structure	Connectors coordinate to populate new structure with all pertinent data.	Connectors coordinate to populate new (secondary) structure with all pertinent data.	The system copies data from old structure to new without connector participation.	The system copies data from primary structure to secondary without connector participation.
New connections during rebuild processing	Permitted only during the Rebuild Quiesce phase.	Permitted only during the Rebuild Quiesce, Rebuild Connect, and Duplex Established phases.	Not permitted.	Permitted only during a Duplex Established phase.

Initiating a structure rebuild process

Rebuild can be initiated by an authorized program, by an operator, or by MVS. For each structure affected by the rebuild request, the system determines whether to start a rebuild or duplexing rebuild and, if started, which method (user-managed or system-managed) to use.

- A **user-managed rebuild** is initiated if there is at least one active connector to the structure and all active connectors have specified or defaulted to IXLCONN ALLOWREBLD=YES. The specification of IXLCONN ALLOWAUTO is not considered.
- A **user-managed duplexing rebuild** is initiated if there is at least one active connector and all active connectors have specified IXLCONN ALLOWDUPREBLD=YES.
- A **system-managed rebuild** is initiated if a user-managed rebuild is not initiated and the following conditions are met:
  - There is at least one active connector and all connectors have specified IXLCONN ALLOWAUTO=YES.
  - There are no active connectors to the structure. The specification of IXLCONN ALLOWAUTO is not considered.
  - The CFRM couple data set supports SMREBLD.
- A **system-managed duplexing rebuild** is initiated if the following conditions are met:
  - There are no connectors (active or failed-persistent) or all connectors have specified IXLCONN ALLOWAUTO=YES.
  - There are connectors, or the CFRM active policy indicates that the structure had previously been duplexed.
  - All active connectors have connectivity to the structure.
  - There are no connectors that could not be reconciled into the active policy for the structure.
  - The structure is not marked as failed.
  - The structure is system-managed duplexing-capable (allocated in a coupling facility of CFLEVEL=11 or later and allocated by a system at OS/390 Release 8 or later).
  - There is at least one other coupling facility of CFLEVEL=11 or higher in the preference list.
  - The CFRM couple data set supports SMDUPLEX.
  - There are no pending policy changes for the structure.
  - There is at least one other coupling facility in the preference list that allows system-managed synchronous duplexing when paired with the current coupling facility (using DUPLEX parameters SYNCONLY or ASYNC), or the CFRM couple data set supports ASYNCDUPLEX and the structure is capable of system-managed asynchronous duplexing:
    - The structure is a lock structure.
    - The structure is allocated in a CFLEVEL=21 or higher coupling facility with asynchronous duplexing controls.
    - All active connectors that are specified or defaulted to IXLCONN AsyncDuplex=YES.
  - Duplexing is feasible because there is another coupling facility that is eligible for duplexing with the active structure instance

"Overview of user-managed rebuild processing" on page 268 describes the user-managed rebuild process. The system-managed rebuild process is described in "Overview of System-Managed Rebuild Processing" on page 301.

Overview of user-managed rebuild processing

The two types of user-managed rebuild processing are *rebuild* and *duplexing rebuild*.

- *Rebuild* is intended for planned reconfiguration and recovery scenarios. An installation might initiate rebuild because of loss of connectivity to a coupling facility or a structure failure.
- *Duplexing rebuild* is intended for improved availability and usability for cache structures. An installation might initiate duplexing rebuild for continuous use of the structure by an application in the event of a structure failure or a loss of connectivity by one system to a coupling facility, especially when reconstructing the structure's data might be difficult or impossible. Duplexing rebuild allows a connector to a structure to allocate another structure for the purpose of duplexing the data in the structure. This type of rebuild process allows users to use two instances of a cache structure in two

different coupling facilities and, if necessary, to revert to using only one of the structure instances. Duplexing rebuild is available only for cache structures.

As user-managed processes, both rebuild and duplexing rebuild require that connectors develop protocols among themselves to control the rebuilding process. The system reports certain rebuilding events to the event exits so that connectors can coordinate rebuilding.

- For rebuild, once the original structure has been rebuilt (rebuilding is complete), the system deallocates the original structure and connectors can use the new structure.
- When the data in the original structure has been copied to the duplexed structure (a Duplex Established phase has been reached), connectors can duplex cache operations to both structures. Once the connectors have decided to stop duplexing rebuild and use only one of the structures, the system deallocates the structure that is no longer required. Otherwise, both instances of the cache structure exist so that the connectors can duplex the data in the structure.
- When user-managed rebuild processing is initiated to recover from failures such as a loss of connectivity, structure failure, or coupling facility failure, and the system is running on a z15™ server that supports recovery process boosts, the system will request System Recovery Boost's recovery process boost support to provide additional capacity to accelerate the user rebuild processing performed by the system and connector event exits.

  For more information on System Recovery Boost, see: IBM Z System Recovery Boost Content Solution (www.ibm.com/support/z-content-solutions/system-recovery-boost/) or IBM Documentation (www.ibm.com/docs/en/zos).

IXLCONN Support for User-Managed Processes

To support user-managed rebuild processes, the following must be specified on the IXLCONN invocation:

- For user-managed rebuild, IXLCONN ALLOWREBLD=YES must be specified or defaulted to by all connectors to the structure.
- For user-managed duplexing rebuild, IXLCONN ALLOWREBLD=YES must be specified or defaulted to **and** ALLOWDUPREBLD=YES must be specified.

Phases for User-Managed Processes

The rebuild and duplexing rebuild processes involve a series of established phases, during which all active connectors to the structure coordinate their activities through MVS. The responsibility for managing the structure and its contents during these phases is that of the connector to the structure. The phases are:

- Rebuild Quiesce Phase
- Rebuild Connect Phase
- Rebuild Duplex Established Phase
- Rebuild Cleanup Phase

Connectors enter and leave each of these phases based on event notification through their event exits. The events indicate to the connected user the start or completion of a specific phase of the rebuild process. Connected users must respond to some, but not all, of these events.

A brief description of each of the rebuild phases follows:

Rebuild Quiesce Phase

This phase applies to the rebuild and duplexing rebuild processes. During the Rebuild Quiesce Phase, connectors to the structure are notified of a request to rebuild the structure through the Rebuild Quiesce event and are given the opportunity to decide whether to participate in the rebuild process. Connectors quiesce their activity to the structure and if necessary, might purge outstanding requests to the structure. Each connector participating in the rebuild process must respond to the Rebuild Quiesce event, at which point its connect token is invalidated (to ensure the continuance of the structure's quiesced state).

Note: If the activity to the structure is based on a restart token, the connector participating in the rebuild process should process the request to completion before responding to the Rebuild Quiesce event. See "Completing Outstanding Structure Requests" on page 275.

When the system has received a response to the Rebuild Quiesce event from each active connector to the structure, the Rebuild Quiesce Sync Point is reached and the Rebuild Quiesce Phase ends.

Rebuild Connect Phase

This phase applies to rebuild and duplexing rebuild processes. When the Rebuild Quiesce phase ends, each connector receives a Rebuild Connect event to indicate that the connector should issue a connect request for the new or duplexed structure. The system allocates the new structure upon receipt of the first connect request; subsequent connect requests allow the issuer access to the newly-allocated structure. Connectors to the structure are notified of other connected users through their event exit. When a connected user has issued its connect request to the new structure, the user can begin reconstructing or propagating its data to the new structure. As each connector completes its processing to transfer data to the new structure, the connector issues a rebuild complete request (IXLREBLD REQUEST=COMPLETE) to indicate the completion to the system.

When the system has received the complete request from all active connectors to the structure, the Rebuild Connect Phase ends and one of two sync points is reached:

- If this is rebuild, the Rebuild Complete Sync Point is reached and processing continues with the Rebuild Cleanup Phase.
- If this is duplexing rebuild, the Rebuild Duplex Established Sync Point is reached and processing continues with the Rebuild Duplex Established Phase.

Rebuild Duplex Established Phase

This phase applies to the duplexing rebuild process, not to the rebuild process. When the Rebuild Connect Phase ends, the system notifies each connector with a Rebuild Duplex Established event. During a Rebuild Duplex Established phase, connectors operate in duplex mode accessing both the old and new structures to ensure that both structures are fully synchronized. New connectors can request access to both structures, first to the old structure, and if successfully connected, then to the new structure. If the attempt to connect to the old structure is not successful, the system does not allow the user to connect to the new structure, and both structures remain in a Duplex Established phase. If the attempt to connect to the old structure is successful, but the connect to the new structure is not, MVS stops the duplexing process for the new structure.

A Rebuild Duplex Established phase is open-ended, that is, connectors can continue in duplex mode for as long as is required. At some point, MVS might determine that duplexing should be discontinued, or a connector or an operator might send a request to the system to end the Duplex Established phase. The request will specify which of the duplexed structures is to be kept — either by falling back to use the old structure or by switching forward to complete the rebuild process using the new structure.

- If the request is to stop the duplexing and fall back to the old structure (KEEP=OLD), processing occurs as for stopping a non-duplexed structure rebuild. See "Stopping a User-Managed Rebuild Process" on page 288.
- If the request is to stop the duplexing and switch to the new structure (KEEP=NEW), MVS marks the structure as "switch in progress" and delivers the Rebuild Switch event to all connectors. Before responding to this event, all connectors, including those who have connected while the switch is in progress, must quiesce duplexing rebuild and complete operations to the new structure. To confirm the completion of these activities, each connector issues IXLREBLD REQUEST=DUPLEXCOMPLETE. When the system has received the REQUEST=DUPLEXCOMPLETE request from all connectors, the Rebuild Duplex Complete Sync Point is reached and the Rebuild Duplex Established phase ends. Processing continues with the Rebuild Cleanup phase.

Rebuild Cleanup Phase

This phase applies to rebuild and duplexing rebuild processes. During the Rebuild Cleanup Phase, each connector receives a Rebuild Cleanup event to specify that the connector is to clean up any information

that pertains to the old structure being discarded. As each connector completes its cleanup processing, it notifies the system through its response to the Rebuild Cleanup event. When the system has received all cleanup confirmations, the Rebuild Cleanup Sync Point is reached. The system notifies all connectors through the Rebuild Process Complete event, deallocates the old structure, and allows connectors to access the new structure again.

Role of CFRM Policy in the Rebuild Process

A change in the active CFRM policy might be required when an installation uses a rebuild process to move a structure to another coupling facility or to create a duplexed structure in another coupling facility. The active CFRM policy defines the coupling facility preference list and the structure exclusion list of the structure that is to be rebuilt or duplexed. The CFRM policy also specifies for each structure whether duplexing rebuild is to be manually initiated or is able to be automatically initiated by MVS.

Options for Initiating Duplexing Rebuild

The DUPLEX option in the CFRM policy allows you to specify for each structure how the initiation of duplexing rebuild is to be handled:

- DUPLEX(DISABLED) — Duplexing rebuild is not allowed. If a duplexing rebuild is in progress for the structure at the time the CFRM policy changes to DUPLEX(DISABLED), MVS will automatically attempt to stop the duplexing rebuild and fall back to the old structure.

  DUPLEX(DISABLED) is the default.
- DUPLEX(ALLOWED) — Duplexing rebuild is allowed to be manually established through the SETXCF operator command or the IXLREBLD macro, but will not be automatically initiated by MVS.
- DUPLEX(ENABLED) — Duplexing rebuild is allowed for both manual initiation and automatic initiation by MVS. If duplexing rebuild is not in progress at the time the CFRM policy changes to DUPLEX(ENABLED), MVS may start a duplexing rebuild.

Note that changes to the CFRM policy that affect the DUPLEX option will always take effect immediately. If the structure is allocated at the time of the CFRM policy change, the DUPLEX option will not be made pending. Any duplexing rebuild actions that are required because of the CFRM policy change will take effect immediately.

Automatic Duplexing

When DUPLEX(ENABLED) is specified for a structure in the active CFRM policy, the system will attempt to start a duplexing rebuild for a structure that is not currently duplexed when certain triggering events occur. Such duplexing triggers are shown in the following examples:

- The structure becomes allocated with at least one active connector.
- The CFRM policy changes to DUPLEX(ENABLED) for an allocated structure with at least one active connector.
- A new coupling facility resource becomes available.
- The last connector that did not support duplexing rebuild (ALLOWDUPREBLD=NO) disconnects or is forced from the structure.

The system will not attempt to start a duplexing rebuild for the structure under the following conditions:

- The new structure in duplexing rebuild could not be allocated with the same connectivity as the old structure.
- Stop processing completes, and IGNOREDUPLEX=YES was specified on the IXLREBLD STOPDUPLEX request.

The system might be unable to duplex a structure again after an operator-initiated stop of the duplexing rebuild process.

- When an operator initiates a stop of the duplexing rebuild, the system will stop it as requested. There might be a delay before reduplexing when only two coupling facilities are available for duplexing the structure. Reduplexing will occur immediately in configurations with three or more coupling facilities

available for duplexing the structure. When reduplexing the structure for this trigger, the system ensures that the structure is not duplexed back into the same coupling facility from which a duplex instance of the structure was just deallocated. Note, however, that if the duplexing rebuild fails or is stopped again, the coupling facility for which the duplexing was stopped may be selected for the next duplexing attempt.

MVS will stop the duplexing rebuild for the structure under the following conditions:

- The CFRM policy changes to DUPLEX(DISABLED).
- A coupling facility containing one of the structures is removed from the structure's preference list.

Rebuilding with a New CFRM Policy

The system administrator might need to redefine the CFRM policy to remove the current coupling facility from the preference list in the CFRM policy and make sure that the preference list contains another coupling facility that has both enough space for the new structure and connectivity to all systems currently connected to the structure. Once the CFRM policy is defined, the system administrator activates the new CFRM policy and issues the SETXCF command to start rebuild processing.

Rebuilding without a New CFRM Policy

The rebuild option, LOCATION=OTHER, specifies that the structure is to be rebuilt in any coupling facility listed in the active CFRM policy's preference list "OTHER than" the coupling facility in which the structure exists now. This option allows you to rebuild the structure without having to change your active CFRM policy if the currently active CFRM policy contains other suitable coupling facilities in the structure's preference list. See "MVS-Initiated Rebuild Processing" on page 294 for information about using the REBUILDPERCENT mechanism to rebuild a structure in another coupling facility.

Rebuild Connectivity Requirements

The system by default allows a structure to be rebuilt only when it can ensure the following connectivity levels at the time the rebuild is initiated:

- If the rebuild was initiated because of a loss of connectivity, the rebuilt structure will have **better** connectivity than the connectivity of the set of connectors to the old structure that did not lose connectivity to that structure.
- If the rebuild was initiated for any other reason, the rebuilt structure will have **equivalent or better** connectivity than the connectivity of the set of connectors to the old structure that did not lose connectivity to that structure.

When the reason for the rebuild is other than loss of connectivity, the application or the operator can override this system default by including a keyword, LESSCONNACTION=CONTINUE, on the macro invocation or command. If the new structure cannot be allocated with equivalent or better connectivity than the old structure, and the application or the operator did not specify LESSCONNACTION=CONTINUE, the system does not initiate the rebuild process. For a duplexing rebuild, LESCONNACTION=TERMINATE is assumed.

User-Managed Rebuild Events and the Event Exit

All active connectors to a structure are required to participate in the user-managed rebuild process for that structure. During the course of the rebuild, events are presented to the event exits of all connectors to the structure. The events notify the connected users of the start or completion of specific phases of the rebuild process. The connected user must respond to some, but not all, of these events.

The following list summarizes the events that the system reports about the rebuild process to the event exit and the responses expected by the event exits:

Rebuild Quiesce
Request to start structure rebuild processing. The IXLYEEPL will indicate the type of rebuild (rebuild or duplexing rebuild). (Response is required via IXLEERSP.)

Rebuild Connect
Request to issue IXLCONN REBUILD for the structure after all connectors have quiesced the use of the structure. When the connector has propagated all required data to the new structure, it must confirm that this processing is complete with IXLREBLD REQUEST=COMPLETE. (Response is required, first with IXLCONN REBUILD, and then via IXLREBLD REQUEST=COMPLETE.)

Rebuild Connects Complete
Confirmation that all connected users have issued IXLCONN REBUILD for the structure. This event is not presented to connectors during the duplexing rebuild process.

Rebuild New Connection
New connection to the new structure.

Rebuild Existing Connection
Existing connection to the new structure.

Rebuild Connect Failure
IXLCONN REBUILD failure for a connector because of abnormal task or address space termination. (Response is required via IXLEERSP or in IXLYEEPL).

Rebuild Duplex Established
Duplexing has been established by each connector. Connectors can begin duplexed structure operations. This event pertains only to duplexing rebuild.

Rebuild Switch
Request to switch to using only the new instance of a duplexed structure. Request to issue REQUEST=DUPLEXCOMPLETE after quiescing use of both instances of the structure. This event pertains only to duplexing rebuild. (Response is required via IXLREBLD.)

Rebuild Cleanup
Confirmation that all connected users have completed structure processing and should clean up information related to the old structure which will be deallocated. (Response is required via IXLEERSP.)

Rebuild Process Complete
Confirmation that the structure has been rebuilt.

Rebuild Stop
Request to stop a structure rebuild process. If this is a duplexing rebuild, the request is to stop the duplexing and use the old structure. Connected users must clean up information about the new structure, which will be deallocated. (Response is required via IXLEERSP.)

Rebuild Stop Process Complete
Confirmation that the rebuilding process has been stopped.

Note that user-defined synchronization points can also be used if additional coordination is required for rebuilding a structure. See “Using IXLUSYNC to Coordinate Processing of Events” on page 344.

Some rebuild events require a response from all connected users that are participating in the process. You can confirm the following rebuilding events through the IXLEERSP macro:

- Rebuild Quiesce. You must respond to a request for rebuilding the structure after you have quiesced your use of the existing structure. To continue the rebuilding process, issue IXLEERSP with EVENT=REBLDQUIESCE.
- Rebuild Cleanup. You must ensure that resources associated with the original structure have been released. To confirm the event, issue IXLEERSP with EVENT=REBLDCLEANUP.
- Rebuild Connect Failure. You must respond to the rebuild connect failure after cleaning up any control information. To confirm the event, issue IXLEERSP with EVENT=REBLDCONNFAIL or respond with IXLYEEPL.
- Rebuild Stop. You must respond to the stop rebuild request. To confirm the event, issue IXLEERSP with EVENT=REBLDSTOP.

Some rebuilding events can be superseded by a Rebuild Stop event. In these cases, the connector must respond to the Rebuild Stop event rather than to the event that was previously expected. The timing of an event being superseded by a Rebuild Stop event might result in some connectors seeing the superseded

event and other connectors not seeing it. Therefore, some connectors might see the prior event and then the Rebuild Stop event; other connectors might never see the prior event and only see the Rebuild Stop event. Note that if a connector responds to an event that has been superseded, the system returns a failing return code to the connector.

The following rebuilding events can be superseded by a Rebuild Stop event:

- Rebuild Quiesce
- Rebuild Connect
- Rebuild Connects Complete
- Rebuild Duplex Established

See "Delivery of Rebuild Stop Event" on page 289.

XES Monitoring of Rebuild Event Responses

XES monitors certain events to ensure that required responses are received from connected users in a timely manner. The rebuild events that XES monitors are:

- Rebuild Quiesce
- Rebuild Connect
- Rebuild Connect Failure
- Rebuild Switch
- Rebuild Cleanup
- Rebuild Stop

It is possible to connect to a structure during rebuild processing, in which case the connector is expected to return an explicit response depending on the rebuild phase into which the user connected. See "Handling New Connections During a User-Managed Rebuild Process" on page 289 for detailed information. XES monitors the following required responses:

- After connecting during the Rebuild Quiesce phase
- After connecting during the Rebuild Connect phase
- After connecting during the Duplex Established phase
- After connecting during the Rebuild Switch process
- After connecting during the Rebuild Stop process

If responses are not received in a timely manner, XES issues a message for each connector owing an expected response that is overdue. These messages can then be analyzed by the system programmer, operator, or automation package for the appropriate action to be taken so that processing can continue. See "XES Monitoring of Event Responses" on page 341.

Starting the User-Managed Rebuild Process

The following are required for starting user-managed rebuild processing:

- Rebuild and duplexing rebuild require that there be at least one active connector to the structure at the time of the request.
- Duplexing rebuild requires that there be another eligible coupling facility with connectivity to all connectors to the old structure. The coupling facility should provide a failure-independent (failure-isolation) environment. This ensures that the duplexed structures will not both be subject to loss because of a single hardware failure in which both coupling facilities reside. See "Planning for Coupling Facility Failure-Independence" on page 215.

Understanding the Rebuild Quiesce Phase

During the Rebuild Quiesce phase:

1. The Rebuild Quiesce event is delivered.
2. Connectors decide whether to participate in the rebuild or duplexing rebuild process.
3. Connectors quiesce their coupling facility accesses to the structure and their use of restart tokens.
4. Connectors respond to the Rebuild Quiesce event.

Delivery of the Rebuild Quiesce Event

As soon as the rebuild start request is successfully completed, all connected users are notified of the rebuild through the Rebuild Quiesce event. Note that the Rebuild Stop event may supersede the Rebuild Quiesce event. The system presents the Rebuild Quiesce event to the event exit along with the reason for rebuilding the structure, and an indication of failed-persistent connections to the original structure. (Note that failed-persistent connections to the old structure will not exist in the new structure after rebuild. It is the connection's responsibility to ensure that any necessary recovery for the failed-persistent connection is complete before proceeding with the rebuild process.)

If MVS has initiated the rebuild based on a policy-specified parameter concerning loss of connectivity (REBUILDPERCENT), the system also presents to the event exit the percentage of lost connectivity which caused MVS to initiate the rebuild. See "MVS-Initiated Rebuild Processing" on page 294 for a description of how MVS decides to initiate rebuild processing.

Rebuild Quiesce Information Returned in IXLYEEPL

On systems with OW36894 installed or which are at OS/390 Release 9 or higher, the following flags specific to the Rebuild Quiesce event are returned in IXLYEEPL:

- EEPLREBUILDQUIESCELCCONT — Specifies the LESSCONNACTION attribute for the rebuild (either LESSCONNACTION=TERMINATE or LESSCONNACTION=CONTINUE).
- EEPLREBUILDQUIESCELOCOTHER — Specifies the LOCATION attribute for the rebuild (either LOCATION=NORMAL or LOCATION=OTHER).

Responding to the Rebuild Quiesce Event

Users can respond to the Rebuild Quiesce event in one of the following ways:

- Decide to participate in the rebuild process.
- Disconnect from the structure and allow other connected users to participate in the rebuild process.
- Stop the rebuild process by issuing IXLREBLD REQUEST=STOP or IXLREBLD REQUEST=STOPDUPLEX. See "Stopping a User-Managed Rebuild Process" on page 288.

Note that if users choose to stop the rebuild process, the system will generate a Rebuild Stop event to be delivered to the event exits of the structure's connectors. The Rebuild Stop event will supersede any Rebuild Quiesce event that has not yet been delivered to a connector.

If connectors decide to participate in rebuilding, they must

1. Wait for outstanding requests to the structure to complete.
2. Stop making any new structure requests like IXLCACHE, IXLLIST, IXLLOCK, or IXLRT.
3. Quiesce the use of restart tokens.
4. Issue IXLEERSP EVENT=REBLDQUIESCE to respond to the event.

Completing Outstanding Structure Requests

Before responding to the Rebuild Quiesce event, users should complete any request that needs to be restarted because it either exceeded the time-out criteria for the coupling facility or requires more buffer space to return all requested information. In these situations, the system returns either a restart token (from certain IXLCACHE, IXLLIST, and IXLRT invocations) or an entry identifier (from certain IXLLIST and IXLRT invocations).

It is important to remember that after a structure is rebuilt, the new structure may not be an identical copy of the old structure. Specifically,

- Information in the restart token used to access the old structure will not be valid for continued operations on the new structure.
- Entry ID values will differ between the old structure and the new structure.
- Depending on the exploiter's protocol, not all entries or data present in the old structure will necessarily be present in the new structure.
- Depending on the exploiter's protocol, the order of lists that have been rebuilt in the new structure (for example, lists of record data within a lock structure) may differ between the old structure and the new structure.

For these reasons, users should not have any outstanding restart tokens (RESTOKENs or EXTRESTOKENs) or entry identifiers (ENTRYIDs) that are to be used to redrive processes after replying to the Rebuild Quiesce event. Users should always fully complete these types of requests before replying to the Rebuild Quiesce event.

Once a user has responded to the Rebuild Quiesce event, the user's connect token is temporarily invalidated to prevent any new accesses to the structure. Therefore, it might be necessary to purge outstanding requests before responding to the Rebuild Quiesce event.

- **List, Serialized List, and Cache Structures**

  Use IXLPURGE to purge outstanding IXLLIST or IXLCACHE operations on these structures. Do not respond to the Rebuild Quiesce event until receipt of a confirmation of the completion of each of these outstanding IXLLIST and IXLCACHE events.

- **Lock Structures**

  IXLPURGE cannot be used to purge outstanding IXLLOCK requests. Use IXLUSYNC to ensure that all users have recognized the Rebuild Quiesce event and that no connector has issued IXLREBLD REQUEST=STOP. Use IXLPURGE to purge outstanding IXLRT operations. Do not respond to the Rebuild Quiesce event until all outstanding IXLRT operations are complete.

After the Rebuild Quiesce event has been provided, the system handles exit routines as follows:

- The system does not prevent the contention, complete, and notify exits from being driven for events related to the original structure. However, the connector can optionally defer the contention exit during the rebuild processing. See “Contention Exit Processing” on page 662 for a description of how the contention exit can be deferred.
- The system disables the list transition exit for a structure that is being rebuilt when the connected user responds to the Rebuild Quiesce event. The list transition exit remains disabled until either a Rebuild Complete or Rebuild Stop Process Complete event is presented to the user's event exit.

Note that the invalidation of the connect token after the Rebuild Quiesce event is temporary. XES revalidates the original CONTOKEN later in the process when either the structure rebuild is complete or the rebuild is stopped (and all events have been acknowledged). Users will use the original CONTOKEN to access the new structure when it is rebuilt. The user is unable to access the original structure.

Completing the Rebuild Quiesce Phase

When all users that are to participate in rebuilding have issued IXLEERSP EVENT=REBLDQUIESCE, the system reports the Rebuild Connect event to the event exits of all connected users.

Connecting to the New Structure

The REBUILD option of IXLCONN allows a new version of the structure to be allocated and permits the requesting connector access to the new structure. See “Using the IXLCONN macro for rebuilds” on page 277.

Understanding the Rebuild Connect Phase

During the Rebuild Connect phase:

1. The Rebuild Connect event is delivered.
2. Connectors issue IXLCONN REBUILD to connect to the new structure.
3. Connectors reconstruct the new structure.
4. Connectors issue IXLREBLD REQUEST=COMPLETE to indicate that the structure is reconstructed.

Delivery of Rebuild Connect Event

As soon as all connectors participating in the rebuilding or duplexing have confirmed that their use of the structure has been quiesced, the connectors are notified through the Rebuild Connect event. Note that the Rebuild Stop event and the Rebuild Connects Complete event may supersede the Rebuild Connect event. The users do not need to respond to this event, but instead should issue IXLCONN with the REBUILD option. During the Rebuild Connect phase:

- For rebuild, the system allows only rebuild connect requests to the structure from this point until all rebuild processing is complete. The system rejects all new connections with reason code IXLRSNCODECONNPREVENTED.
- For duplexing rebuild, the system allows new connections to the old structure and to the new structure if a connection to the old structure already exists for that connector. New connectors use IXLCONN to connect to the old structure and IXLCONN REBUILD to connect to the new structure. The CONAREBUILDPHASE field in IXLYCONA indicates in which phase the connection occurred (Rebuild Quiesce phase, Rebuild Connect phase, or Duplex Established phase).

Using the IXLCONN macro for rebuilds

The first connector to issue IXLCONN with REBUILD or REQTYPE=REBUILDCONNECT allocates the new structure. The first connected user also defines the attributes for the new structure. When the new structure is allocated, pending policy changes to structure size or location also apply. Other users issue IXLCONN REBUILD to connect to the new structure but cannot change the structure attributes. The connect answer area contains information about the rules used to allocate the structure. It is the connector's responsibility to verify that the attributes of the structure are acceptable.

To issue IXLCONN REBUILD or use the default the user must be connected to the original structure. The user must issue IXLCONN REBUILD with the same structure name and CONNAME as the original structure. Table 16 on page 277 lists the structure attributes that users can change when rebuilding a structure.

Table 16. Structure attributes that can be changed with rebuild connect

Cache	List	Lock
STRSIZE NONVOLREQ ACCESSTIME ELEMCHAR* ELEMINCRNUM MAXELEMNUM* DIRRATIO* ELEMENTRATIO* ADJUNCT VECTORLEN NUMCOCLASS NUMSTGCLASS UDFORDER NAMECLASSMASK	STRSIZE NONVOLREQ ACCESSTIME ELEMCHAR* ELEMINCRNUM MAXELEMNUM* ENTRYRATIO* ELEMENTRATIO* ADJUNCTP* VECTORLEN LISTCNTLTYPE* REFOPTION* LISTHEADERS EMCSTGPCT* MAXCONN	STRSIZE NONVOLREQ ACCESSTIME LOCKENTRIES MONITORSTORAGE NUMUSERS MAXCONN PCTENTRYRSV

Note: An asterisk indicates that the parameter is ignored by a system with APAR OA33448 installed when KEEPRATIOS=YES is specified on IXLCONN REBLD.

The following restrictions also apply:

- IXLCONN REBUILD must be issued from the same system and address space as the original IXLCONN request. You can issue IXLCONN REBUILD from a task other than the task that issued the original IXLCONN request or from the same task that issued the original IXLCONN request.
- Users of IXLCONN REBUILD cannot change the following attributes of the structure:
  - TYPE (structure type)
  - RECORD (record data for lock structure)
  - RNAMELEN (resource name length for lock structure)
- If users specified VECTORLEN on the IXLCONN request for the original structure, then they must also specify it on the IXLCONN REBUILD request for the rebuild structure. A user can, however, change the size of VECTORLEN on the IXLCONN REBUILD request. If users did not specify VECTORLEN on the IXLCONN request for the original structure, then they must not specify it on the IXLCONN REBUILD request.
- If users specified LOCKENTRIES on the IXLCONN request for the original structure, then they must also specify it on the IXLCONN REBUILD request for the rebuild structure. A user can, however, change the value for the number of lockentries on the IXLCONN REBUILD request.
- The value for NUMUSERS (specified for lock structures on IXLCONN to define the maximum number of connected users) or MAXCONN (specified for list or lock structures on IXLCONN) cannot be less than the value specified for the original structure.
- When changing the size (STRSIZE) of a structure, the maximum structure size is determined by the SIZE parameter in the CFRM active policy. The system rejects a request specifying a STRSIZE larger than the current maximum structure size in the CFRM active policy.

  Note: If the size of the structure has been altered to a value different from the SIZE parameter in the CFRM active policy, it is the responsibility of the installation to change that value, if appropriate.
- When allocating a percentage of available structure storage for event monitor controls, it IS possible on IXLCONN REBUILD requests for the connector to specify a percentage value (EMCSTGPCT) that is different from what was specified on the initial IXLCONN request. Thus, a user could specify that the rebuilt structure was to provide for EMCs even if the original structure did not. However, note that it is NOT possible to request the allocation of a local vector on an IXLCONN REBUILD request unless the initial IXLCONN request had requested a local vector.

  Note also that a user cannot change the specification of a list transition exit name when invoking IXLCONN REBUILD. LISTTRANEXIT is relevant to both sublist monitoring and event queue monitoring.
- The following keywords are IGNORED for rebuild connect requests (IXLCONN REBUILD or IXLCONN REQTYPE=REBUILDCONNECT):
  - ASYNCXI
  - CFLEVEL
  - CONTEXIT
  - NOTIFYEXIT
  - LISTTRANEXIT.

  Note that some of these keywords are required, and you must therefore specify them for rebuild connect requests.
- The following keywords have no meaning when specified on IXLCONN REBUILD requests because the attributes are propagated from the original IXLCONN request. However, you might need to specify them on the IXLCONN REBUILD request or the request fails.
  - STRDISP (structure disposition)
  - CONDISP (connection disposition)
  - CONDATA (connect data)
  - EVENTEXIT (event exit name)

  - COMPLETEEXIT (complete exit name)
  - CONLEVEL (connection level)
  - ALLOWREBLD (whether rebuild is allowed)
  - ALLOWDUPREBLD (whether duplexing rebuild is allowed)
  - ALLOWAUTO (whether system-managed processes are supported).
- The value for NUMUSERS (specified for lock structures on IXLCONN to define the maximum number of connected users) or MAXCONN (specified for list or lock structures on IXLCONN to define the maximum number of connected users) cannot be less than the number of currently in-use connections to the old structure instance.

The location of the new structure depends on the following:

- If LOCATION=OTHER was specified when the rebuild was initiated, XES will not allocate the structure in the same coupling facility as the original structure. (LOCATION=OTHER is assumed for duplexing rebuild.)
- For structure rebuild, XES allocates the structure in the first coupling facility in the preference list that meets the standard allocation requirements. See "Allocating a Structure in a Coupling Facility" on page 208.
- For duplexing rebuild, XES attempts to allocate the structure in a coupling facility in the preference list that not only meets the standard allocation requirements but also provides failure-independence with respect to the coupling facility in which the old structure is allocated. If such a coupling facility is not available, the installation should consider changing the active CFRM policy so that the structure can be duplexed in a failure-independent environment. See "Planning for Coupling Facility Failure-Independence" on page 215 for a description of the failure-independent coupling facility attribute.

Specifying Coupling Facility Connectivity Requirements for Rebuild Processing

The following information about structure connectivity and the rebuild process applies to systems at the OS/390 Release 2 level and higher. Systems running on a lower level of the system that allocate a structure cannot use the CONNECTIVITY function of IXLCONN or the LESSCONNACTION function of IXLREBLD.

You can specify a CONNECTIVITY value on the IXLCONN REBUILD invocation to request a coupling facility with the same requirements that existed at initial connect time. The system selects a coupling facility that meets the requested CONNECTIVITY specification, if possible.

By default, the system does not allow a structure to be rebuilt if the new structure will have poorer connectivity than the original structure. The system evaluates the current connectivity of connectors to both the old and the new structures and allows the rebuild to proceed only if the connectivity of connectors to the new structure will be better than or equal to that of connectors to the old structure.

The LESSCONNACTION keyword of IXLREBLD allows you to override this default action and to specify whether you want the system to rebuild the structure in spite of a resulting degradation of connectivity. With LESSCONNACTION, you can specify that the system is to stop rebuild processing (TERMINATE) or to continue rebuild processing (CONTINUE) if the new structure would have poorer connectivity than the original structure.

When a structure is in a duplexing rebuild process, the system assumes LESSCONNACTION=TERMINATE and does not allow the new structure to be allocated in a coupling facility that does not provide equivalent or better connectivity.

Evaluating Current Connectivity Status

The system determines whether the new structure will have equivalent or better connectivity than the old structure by evaluating the current connectivity of both. For both the old structure and the new structure, the system calculates the aggregate SFM system weight of all systems that:

- Have connectivity to the coupling facility in which the structure resides, and

- Have one or more active connectors to the old structure.

If the system determines that connectivity to the new structure will be better (for a rebuild reason of loss of connectivity) or equivalent or better (for any other rebuild-initiation reason), the rebuild is allowed to proceed. Those systems that have connectivity to the new coupling facility and have one or more active connectors to the old structure will participate in the rebuild. For those systems that might have had connectivity to the old structure in the original coupling facility but do not have connectivity to the new coupling facility, the IXLCONN REBUILD request will fail.

If the system calculates that connectivity to the new structure will be poorer than to the original structure, then the LESSCONNACTION parameter is used. Note that if the reason for the rebuild is a loss of connectivity, the system ignores the LESSCONNACTION specification and stops the rebuild.

Handling Failed Attempts to Rebuild a Structure

If you have specified a CONNECTIVITY value of SYSPLEX on your IXLCONN REBUILD invocation, XCF attempts to select a coupling facility with full connectivity to all systems in the sysplex. If there is no coupling facility with sysplex connectivity at the time of the rebuild, the application will be unable to rebuild the structure and continue processing. There are two options that the application might consider:

- Document for the installation that the rebuild protocol for the application requires a coupling facility with sysplex connectivity. Neither the IXLREBLD macro invocation nor the SETXCF START,REBUILD operator command will be successful until the installation makes such a coupling facility available.
- Design a protocol by which the application reissues the IXLCONN REBUILD request, but this time with a CONNECTIVITY=BESTGLOBAL. The application would then have the responsibility of causing any systems that are not connected to the selected coupling facility to be removed from the sysplex.

  The application should consider carefully the use of this option, as it does require some degree of effort.

Sample Protocol

1. The connector issues IXLCONN REBUILD CONNECTIVITY=SYSPLEX. When this fails, the connector can issue IXLCONN CONNECTIVITY=BESTGLOBAL.
2. XCF keeps track of the connectors' rebuild attempts, and when all connectors have issued at least one IXLCONN REBUILD request, the system reports the Rebuild Connects Complete (EEPLREBUILDCONNECTSCOMPLETE) event. (This event notifies all connectors of the number of successful and unsuccessful connections to the new structure.)

   Note that a connector can issue its second IXLCONN REBUILD request only until that point at which the all active connectors have issued IXLREBLD REQUEST=COMPLETE.
3. The systems then can either:

   a. Disconnect from the old structure, let the rebuild continue and complete, and attempt to connect to the rebuilt structure when notified by the ENF 35 event that additional coupling facility resources are available, or

   b. Stop the rebuild and somehow notify all connectors to retry the IXLCONN REBUILD with CONNECTIVITY=BESTGLOBAL.

Successful Completion of IXLCONN REBUILD

When IXLCONN REBUILD is successful, the system returns return code IXLRETCODEWARNING and reason code IXLRSNCODESPECIALCONN. The CONAFLAGS field in the connect answer area indicates REBUILD=YES. The connected user can expect the following:

- Connection to the new structure.
- Ability to make other coupling facility requests to the new structure through the temporary CONTOKEN returned. The original CONTOKEN is used to access the old structure.
- Notification of structure and connection events through the event exit.

To understand how the system maintains connect tokens during this phase, see the description of CONACONTOKEN in "Receiving Answer Area Information from IXLCONN REBUILD" on page 281.

The system reports the following connection events to the event exit of each connected user that is in either the structure rebuild or the structure duplexing process:

- Rebuild New Connection. Existing connections to the new structure receive notification of each new user that connects to the new structure through IXLCONN REBUILD.
- Rebuild Existing Connection. Each new connection to the new structure receives notification of each existing connection to the new structure.

Handling a Failed IXLCONN REBUILD Request

When an IXLCONN REBUILD request for structure rebuild is not successful, the connector has three options:

- Disconnect from the old structure and let the rebuild continue.
- Reissue the IXLCONN REBUILD request with one or more changed parameters, based on the return and reason code returned by the failed attempt. Note, however, that the more times you reissue the IXLCONN REBUILD request, the longer you are holding up the entire rebuild cycle for all connectors involved.
- Stop the rebuild process. Note that if you choose to stop the rebuild process, the system will generate a Rebuild Stop event to be delivered to the event exits of the structure's connectors. The Rebuild Stop event will supersede any Rebuild Connect event that has not yet been delivered and may occur either before or after the connector has issued IXLCONN REBUILD to connect to the new structure.

When an IXLCONN REBUILD request for duplexing rebuild is not successful, the following occurs:

- If the connector is connected to the old structure but is unable to connect to the new structure because of lack of connectivity to that coupling facility, the system initiates a fall back to the old structure for all connectors. (Duplexing rebuild assumes LESSCONNACTION=TERMINATE.) If the IXLCONN REBUILD request to connect to the new structure fails for any other reason, it is the responsibility of the user to either stop the rebuild or disconnect.
- The system will attempt to duplex the structure in a different coupling facility if the active CFRM policy specifies DUPLEX(ENABLED) for the structure.

Receiving Answer Area Information from IXLCONN REBUILD

At the completion of its processing, IXLCONN REBUILD returns the following information in the connect answer area, mapped by IXLYCONA.

CONACONTOKEN
Connect token that uniquely identifies the connection to a new structure within the sysplex. This CONTOKEN is temporary and is not the same CONTOKEN value that IXLCONN returned for the original structure.

During the rebuilding process, use the temporary CONTOKEN only when using mainline services IXLCACHE, IXLLIST, IXLLOCK, IXLRT, IXLSYNCH, or IXLFCOMP to the new structure.

For all other coupling facility requests (IXLDISC, IXLEERSP, and IXLREBLD), use the CONTOKEN returned from IXLCONN for the original structure. When the system reports that the rebuilding process is complete (Rebuild Complete event), discard the temporary connect token and use the CONTOKEN returned from IXLCONN for the original structure to access the new structure.

For successful IXLCONN REBUILD requests for cache and list structures, the system revalidates the CONTOKEN returned from IXLCONN for the original structure. At this stage, users can make IXLCACHE or IXLLIST structure requests. Accessing the original cache or list structure allows users to move data between the original and new structures. Lock users cannot use the CONTOKEN for the original structure to access the original structure during rebuild.

CONACONID
A connection identifier. The connection identifier is the same as that for the original structure.

CONAFLAGS
Connection status flags.

CONASTRUCTUREATTRFLAGS
Structure type attributes. Users must verify that the attributes for the structure are acceptable. Otherwise, they should disconnect or stop the structure rebuild.

CONASTRUCTUREVERSION
Structure version number. The structure version number will be greater than the structure version number of the old structure.

CONACONNECTIONVERSION
Connection version number. The connection version number will be equivalent to the connection version number of the original connection.

CONAVECTORTOKEN and CONAVECTORLEN
For TYPE=CACHE or TYPE=LIST with list monitoring structures, a vector token and vector length used to identify the user's local vector. Use the new vector token from IXLCONN REBUILD after the rebuild process is complete. However, if the rebuild process is stopped, use the vector token returned on the original IXLCONN request.

See the IXLYCONA macro in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

The connector that allocates the new structure receives an indication in the IXLCONN answer area (CONACONNALLOC field). Subsequent connectors receive an indication that they are connected to the new structure through the CONAREBUILD field of the answer area.

The event exit for the connector might receive Rebuild Existing Connection events before the IXLCONN REBUILD request for the new connection completes.

For cache or unserialized list structures, the event exit for existing connectors might receive Rebuild New Connection events after the IXLCONN REBUILD request for the new connection completes. Users of cache and unserialized list structures can rebuild information into the new structures as soon as the IXLCONN REBUILD request completes. Thus, list and cache users are able to access the original and new structure before they receive the Rebuild Connects Complete event in their event exits. (The Rebuild Connects Complete event indicates that all users have issued IXLCONN REBUILD for a structure rebuild. The Rebuild Connects Complete event is not presented to users who have issued IXLCONN REBUILD for a duplexing rebuild.)

List and cache users can perform copy or read operations for cache or list data in the original structure to help with rebuilding; however, IBM recommends that you do not change data in the original structure during the rebuilding process. In the event of a REBUILD STOP request, you will need to use the original structure once again.

Reconstructing the New Coupling Facility Structure

As soon as users have successfully issued IXLCONN REBUILD, they can begin to reconstruct the data in the new coupling facility structure. There are several ways to reconstruct the data, depending on the application's protocol.

- Do no reconstruction, but allow the data to repopulate the structure strictly through normal use of the structure after the rebuild completes.
- Reconstruct the data in the structure from in-storage control blocks or data buffers.
- Explicitly move the data from the old structure to the new structure.

Explicitly moving data from the old structure to the new structure is entirely the responsibility of the user. Not only must data be moved to the new structure, but also such components as:

- Adjunct data.
- List control information that you have set for list headers, such as list descriptions or list limits on entries/elements.

Note that if your new structure contains different numbers of entries/elements than your old structure did because of different structure attributes that took effect on IXLCONN REBUILD, you may want to set your list limits differently in the new structure to take these changes into account.

- List monitoring interest. (You must re-register.)
- Locks. (You must re-obtain.)

In most applications, all connectors participate in the repopulation of the new structure. However, it is also possible to design an application such that one "master" connector is responsible for all the work associated with reconstructing the data in the new structure, while the other connectors wait passively for the repopulation to complete. In this case, the application protocol must provide for takeover of the master connector's responsibilities by another connector if the original master should fail.

If there is a large amount of data to initialize, repopulation of the new structure may be quite time-consuming, particularly if only one connector is doing the work. System constraints may also slow processing. System hang-detect processing monitors the time required to complete the repopulation, and will declare individual connectors to be hung if they fail to provide the expected IXLREBLD REQUEST=COMPLETE or DUPLEXCOMPLETE response within the required time. A hang declaration will trigger diagnostic dumps and possibly automatic hang resolution actions, depending on the installation's use of the SFM policy CFSTRHANGTIME keyword. To avoid a premature hang declaration when the application is actually progressing, consider using the IXLCONN MONITOR keyword to specify how the system is to track repopulation progress. You can specify that the system is to monitor the rate of coupling facility requests directed to the new structure, or you can make explicit status reports using the IXLREBLD REQUEST=POPULATING or WAITING interface.

When the connector specifies IXLCONN REQTYPE=REBUILDCONNECT MONITOR=IXLREBLD:

- Each connector that is actively repopulating the structure should invoke IXLREBLD REQUEST=POPULATING at intervals not to exceed 2 minutes as long as repopulation is making satisfactory progress, as defined by the application.

  Note: Connectors responsible for repopulating the structure must not simply initiate periodic POPULATING requests to avoid declaration of a hang by z/OS hang detection processing. Bypassing hang detection in this manner can expose the entire sysplex to delays and loss of function.
- A connector that is waiting passively for a master connector to complete repopulation need only invoke IXLREBLD REQUEST=WAITING once within 2 minutes of the start of the repopulation phase, or if a new master connector is assigned.

  The connector designated as the master must have completed its rebuild connect and been reported to the issuing connector using an EeplRebuildNewConnection or EeplRebuildExistingConnection event.

Test the QuReqRfRepopulateProgress flag in the feature string returned by an IXCQUERY REQINFO=FEATURES invocation to determine whether the progress monitoring support is available on the current system.

Delivery of the Rebuild Connects Complete Event

When all connected users have issued IXLCONN REBUILD, the system reports the Rebuild Connects Complete event to the event exits of all connected users. Note that the Rebuild Stop event may supersede the Rebuild Connects Complete event. The system indicates the number of active connections at the time all connections attempted to do a rebuild connect and the number of connections that successfully did a rebuild connect to the structure. You are not required to respond to this event. Depending on user protocol, connected users can stop the rebuilding process if they determine that the number of connected users to the new structure is not sufficient. If all connected users issue the IXLCONN REBUILD before the Rebuild Connect event is delivered to all connectors, the Rebuild Connects Complete event will supersede any Rebuild Connect event that has not yet been delivered.

If a connector or the operator has stopped the rebuild process during this phase, the Rebuild Stop event will supersede any Rebuild Connects Complete event that has not yet been delivered.

The Rebuild Connects Complete event is presented only to connectors rebuilding a structure and not to connectors duplexing a structure.

Completing the Rebuild Connect Phase

When connected users have performed the necessary processing to propagate (or reconstruct) data into the new structure, they must:

1. Complete all outstanding requests to the structure
2. Prevent new structure requests like IXLCACHE, IXLLIST, IXLLOCK, or IXLRT
3. Issue IXLREBLD REQUEST=COMPLETE.

If this is a structure rebuild, as each connector issues IXLREBLD REQUEST=COMPLETE, the system invalidates both the temporary and the original connect tokens to prevent access to either structure. The reason you are not allowed access to both the old and the new structure between the time you issue the IXLREBLD REQUEST=COMPLETE and the time you receive the Rebuild Process Complete event is because there is still the possibility of the rebuild processing being stopped. If that occurs, the new structure would be deallocated and normal processing would continue using the old structure. When the system has received IXLREBLD REQUEST=COMPLETE from all connectors, the Rebuild Complete sync point is reached and processing continues with the Rebuild Cleanup phase. See “Completing the User-Managed Rebuild Process” on page 286.

If this is a duplexing rebuild, when the system has received IXLREBLD REQUEST=COMPLETE from all connectors, the Rebuild Duplex Established sync point is reached and processing continues with the Duplex Established phase.

Working with structures in the Duplex Established Phase

While a structure is in the Duplex Established phase, connectors will continue to receive notification through their event exits of new or existing connections and connection failures. Connectors receive event notification for both structure instances while both structures are allocated. Fields in IXLYEEPL identify the duplexed state of the structure.

New connectors to the old and new structures are allowed while in a Duplex Established phase. See “Handling New Connections During a User-Managed Rebuild Process” on page 289 for a description of the actions a new connector must take when connecting to a duplexed structure.

The synchronization of duplexed structures is the responsibility of the connectors using them. In general, whatever can be done to a structure in simplex mode can be done to a structure in duplex mode. This includes altering the structures while they are in a Duplex Established phase. (See “Altering a duplexed structure” on page 326.)

The propagation of data to the new structure and the subsequent synchronization of that data through duplexing mainline operations to both structures is a matter of user protocol and solely the responsibility of the user. It is also the user's responsibility to handle failure scenarios, such as one of the duplexed structures reaching a “structure full” condition. MVS will handle failures such as loss of connectivity or failure of one of the structure instances, but the user should be prepared to handle other situations.

Understanding the Duplex Established Phase

During the Duplex Established phase:

1. The Rebuild Duplex Established event is delivered.
2. Connectors operate in duplex mode accessing both old and new structures. It is the connector's responsibility to keep the duplexed structure synchronized and to handle any failure conditions that occur while attempting to maintain this synchronization.
3. The connector or operator can decide to stop the Duplex Established phase and fall back to the old structure or forward complete (switch) to the new structure.
   - When connectors indicate completion of their switch to the new structure, the Rebuild Cleanup phase is entered.
   - When connectors indicate completion of their stop processing to fall back to using the old structure, they return to simplex mode through Rebuild Stop processing. Once the system has accepted the

request to stop structure duplexing processing in a particular direction, a request to stop it in the opposite direction will be rejected.

Delivery of Rebuild Duplex Established Event

As soon as all connectors participating in the duplexing process have issued IXLREBLD REQUEST=COMPLETE to confirm that the rebuild of the duplexed structure is complete, the connectors are notified through the Rebuild Duplex Established event. Note that the Rebuild Stop event may supersede the Rebuild Duplex Established event. The users do not need to respond to this event, but continue with their mainline use of both structures in a duplexed fashion.

If a connector or the operator has stopped the duplexing process prior to this phase, the Rebuild Stop event will supersede any Rebuild Duplex Established event that has not yet been delivered.

The Duplex Established phase can last indefinitely, or at least until either an operator command or a macro invocation is received requesting that the duplexing be stopped or until a failure condition affecting one of the structure instances causes MVS to stop duplexing. At that point, all connectors to the structure must quiesce their use of both structures in preparation for either falling back to use the old structure or switching to use the new structure. The system waits for all connectors to confirm that they have completed their duplexing operations by issuing either IXLREBLD REQUEST=DUPLEXCOMPLETE to switch to the new structure or IXLEERSP EVENT=REBLDSTOP to fall back to the old structure before returning to simplex mode. See “Stopping a User-Managed Rebuild Process” on page 288 for a description of how the duplexing process is stopped to fall back to using the old structure.

Working with structures in the Async Duplex Established phase

In the Async Duplex Established phase, a structure is being asynchronously duplexed by the system. This is a specialized case of the Duplex Established phase that is used for system-managed synchronous duplexing. Most processing for the Async Duplex Established phase is the same as for the duplex established phase. However, when a coupling facility lock structure is in the async duplex established phase, special consideration must be made when using the structure record data. Connectors that specify IXLCONN ASYNCDUPLEX=YES must be prepared to work with structures in the Async Duplex Established phase.

When the structure has established asynchronous duplexing, XES reports the completion of requests when the update to the primary structure instance completes. Each update is assigned a unique ADUPREQSEQNUM (asynchronous duplexing request sequence number). The update to the secondary structure instance occurs asynchronously. It is this asynchronous background processing to update the secondary structure instance that can allow processing to achieve performance characteristics more similar to simplex than synchronous duplexing.

The system, on behalf of the connector, keeps track of pending or uncommitted updates to the asynchronously duplexed secondary structure. When duplexing failover to the secondary structure instance occurs, uncommitted updates that are made by active connections are applied to the remaining structure by the system. Uncommitted updates that are made by disconnecting/terminating or failed-persistent connections might not be applied.

The user must be prepared to lose uncommitted updates in the event of a simultaneous duplexing failover and connector failure. To prevent this loss of record data updates, the user must wait for the necessary updates to be committed (applied to the secondary structure instance). The IXLADUPX service can be used to achieve this using the ADUPREQSEQNUM provided by the connector's latest request. The ADUPREQSEQNUM is an ever-increasing value and using IXLADUPX for one ADUPREQSEQNUM value covers that value as well as all smaller ADUPREQSEQNUM values. Waiting for uncommitted updates to commit might be needed, for example, before a transaction is committed.

It is the user's responsibility to request the ADUPREQSEQNUM when it might be needed. Any request that might involve a record data operation (with the exception of just a read) is a candidate for providing an ADUPREQSEQNUM. It might be an IXLLOCK request that specifies or defaults to RDATA=WRITE, RDATA=REACQUIRE, or RDATA=DELETE, or an IXLLOCK PROCESSMULT that uses LRB_XRDATA, or an IXLRT request, or an IXLSYNCH request that uses NEPLORTACTION. Requesting an ADUPREQSEQNUM from an IXLLOCK invocation by specifying the ADUPREQSEQNUM or REQVERSION

requires that the system support system-managed asynchronous duplexing. Macro IXCYQUAA defines the QuReqRfAsyncDuplex bit in the QuReqFeatures string that can be used to test for system-managed asynchronous duplexing support. Use IXCQUERY REQINFO=FEATURES to get the QuReqFeatures string. It can be assumed that systems at a z/OS level higher than V2R2 support system-managed asynchronous duplexing.

To achieve good asynchronous duplexing performance, a user should only use IXLADUPX to wait for updates to be applied to the secondary structure instance when necessary. For instance, it might not be necessary to wait for an IXLLOCK OBTAIN,RDATA=WRITE to complete in both structure instances when it will be followed by other similar requests before the transaction is committed. Not until the primary has all updates that are required to commit the transaction is it necessary to wait for those updates to be applied to the secondary structure. When record data is needed to recover updates to a shared resource, ensure that the record data updates are made to both structure instances before proceeding.

Stopping a Duplexing Rebuild to Forward Complete

Once a stop to switch to the new structure has been accepted, a stop to fall back to the old structure will be rejected.

Understanding Rebuild Stopduplex Processing to Forward Complete

The following list summarizes the events for a stop duplexing request to complete processing and use the new structure:

1. Stop duplexing initiated through SETXCF STOP,REBUILD,DUPLEX or IXLREBLD REQUEST=STOPDUPLEX with KEEP=NEW.
2. The system reports Rebuild Switch event to the event exit.
3. Connector stops duplexing, performs cleanup, and issues IXLREBLD REQUEST=DUPLEXCOMPLETE to respond to the Rebuild Switch event.
4. When all responses are received, the system reports the Rebuild Cleanup event to event exit.

Delivery of Rebuild Switch Event

Once a request to stop duplexing and forward complete (switch) to the new structure is received, XES presents the Rebuild Switch event to each event exit. This event requires that when connectors have quiesced their use of the old structure and completed their switch to the new structure, they must issue IXLREBLD REQUEST=DUPLEXCOMPLETE.

Responding to the Rebuild Switch Event

Before providing a response to the Rebuild Switch event, connectors must quiesce their use of the old structure. See “Completing Outstanding Structure Requests” on page 275 for information about quiescing the use of a structure. New connectors who connect while the switch is in progress are notified through the Connect answer area and are expected to participate by connecting to the new structure. See “Handling New Connections During a User-Managed Rebuild Process” on page 289. As connectors respond to the Rebuild Switch event with the IXLREBLD REQUEST=DUPLEXCOMPLETE request, the system invalidates both their old and new tokens used to access the structure. When all connectors have responded to the Rebuild Switch event, the system enters the Rebuild Cleanup phase to complete the rebuild process.

Completing the User-Managed Rebuild Process

For rebuild and duplexing rebuild, the completion of the process results in connectors using the new structure, which is accessed through the old (original) connect token.

Understanding the Rebuild Cleanup Phase

During the Rebuild Cleanup phase:

1. The Rebuild Cleanup event is delivered.
2. Connectors notify the system when cleanup is completed by responding to the Rebuild Cleanup event.
3. The Rebuild Process Complete event is delivered.
4. Connectors continue processing with the remaining structure.

Delivery of Rebuild Cleanup Event

The Rebuild Cleanup phase is entered as the result of the connector's indication that rebuild processing is complete:

- IXLREBLD REQUEST=COMPLETE while in the Rebuild Connect phase for structure rebuild.
- IXLREBLD REQUEST=DUPLEXCOMPLETE while in the Duplex Established phase for duplexing rebuild.

Once all connected users have indicated that the rebuild process is complete, MVS presents the Rebuild Cleanup event to each event exit. This event requires a confirmation using IXLEERSP with EVENT=REBLDCLEANUP.

Responding to the Rebuild Cleanup Event

Before providing a response to the Rebuild Cleanup event, all connectors should clean up information related to the structure that will be deallocated. Connectors discard the temporary connect token and the old vector token (if applicable). Note that the vector token returned on the IXLCONN REBUILD is not a temporary token like the connect token. The vector token returned must be used to access the structure after the rebuild has completed. (Users do not have the option to stop the rebuild process at this point. See “Stopping a User-Managed Rebuild Process” on page 288.)

In certain instances, XES must quiesce the activity of user exits in order to perform cleanup processing. For example, when a connector provides an event exit response for the Rebuild Cleanup event, XES will force to completion any user exits that are executing on behalf of that user's connection to BOTH the original and the new structures issuing a PURGEDQ against the appropriate units of work. No new events will be presented to the user exits on behalf of the original structure (as it is being discarded). Normal user exit processing will resume for the rebuilt structure upon completion of the rebuild process.

A user exit must be sensitive to conditions that can occur as a result of actions taken by XES and must be able to handle these as appropriate. For example, if a user exit has suspended itself, when the PURGEDQ is issued the system abends the user exit's unit of work with a retryable X'47B' abend and gives control to the user exit's recovery routine. (Note that although the recovery routine can retry, the user exit can not re-suspend itself because the system will fail any request to suspend a unit of work that has been the target of a PURGEDQ.) If the recovery routine percolates back to the system, its associated connection is terminated.

The rebuild process is actually complete when the Rebuild Process Complete event has been presented to the event exit. From the time that all connectors have issued the IXLREBLD REQUEST=COMPLETE or REQUEST=DUPLEXCOMPLETE and XES begins notifying each connector of the Rebuild Cleanup event until the Rebuild Process Complete event is presented, the rebuild process cannot be stopped and a new rebuild request for the structure cannot be started.

- If a REBUILD REQUEST=STOP request is initiated, the request is rejected with reason code IXLRSNCODEINCLEANUP indicating that the rebuild process cannot be stopped during the cleanup phase.
- If a REBUILD REQUEST=START request is initiated, the request is rejected with reason code IXLRSNCODEALREADYREBUILDING indicating that a rebuild request already is in progress.

XES notifies connectors of structure failure and/or loss of connectivity to the new structure that occur during the time between when the IXLREBLD REQUEST=COMPLETE is issued and the Rebuild Process Complete event is issued after the event is presented.

Completing the Rebuild Cleanup Phase

After cleaning up information about the old structure, each connector confirms the completion of its cleanup with IXLEERSP EVENT=REBLDCLEANUP. When the system has received the IXLEERSP confirmations from all connectors, the Rebuild Cleanup sync point is reached, the original contoken has been revalidated for each connector, and the Rebuild Process Complete event is delivered to all connectors.

Delivery of the Rebuild Process Complete Event

After each user receives the Rebuild Process Complete event, the user can access the new structure. The users do not need to respond to this event. When the rebuild process is complete, the system deletes the original structure.

When the rebuild process is complete, the system issues an ENF event code 35 so that connectors who were denied access to the structure during the rebuild can retry their connect request.

Stopping a User-Managed Rebuild Process

The user-managed rebuild process can be stopped through either the SETXCF STOP command or the IXLREBLD macro. Stopping a rebuild implies that the new structure (the one in rebuild processing) is to be discarded and that processing is to continue with the old (or original) structure. For duplexing rebuild however, a request to stop the rebuild processing requires the identification of which structure should remain — the old structure or the new structure. Depending on which is selected, duplexing rebuild processing will either fall back to use the old structure or switch to use the new structure. For duplexing rebuild, a request to stop the rebuild processing can be made when there are no active connections to the structure.

Note that you cannot stop a rebuild that was initiated as a rebuild with a macro invocation or operator command that specifies a duplexing rebuild; nor can you stop a rebuild that was initiated as a duplexing rebuild with a macro invocation or operator command that specifies a rebuild.

Rebuild stop is initiated through the SETXCF STOP,REBUILD command or IXLREBLD REQUEST=STOP. Stopping a duplexing rebuild to fall back to the old structure can be initiated through the SETXCF STOP,REBUILD,DUPLEX command or IXLREBLD REQUEST=STOPDUPLEX with KEEP=OLD. See “Stopping a Duplexing Rebuild to Forward Complete” on page 286 for information about stopping a duplexing rebuild process to use the new structure. Note that you cannot issue a STOPDUPLEX request for a structure that is already being stopped for a switch to the new structure.

Users can stop the rebuild process for a structure up until the time the rebuild enters the Cleanup Phase. At that point all connectors have issued IXLREBLD REQUEST=COMPLETE and the system passes the Rebuild Cleanup event to event exits of the users. After that event, rebuild stop requests fail.

Reasons for stopping a rebuild process can include:

- Loss of connectivity to the either the original structure or the new structure
- Failure of either original structure
- User-specified reason code
- An operator-initiated command.

MVS stops the rebuild process for the new structure:

- If there are no active connections to the structure being rebuilt. The system releases resources used during the rebuilding process. (This does not apply to a structure in the Duplex Established phase, which is allowed to exist with no active connectors.)
- If the structure being rebuilt fails. The system indicates the reason in the event exit.

The system also issues ENF event code 35 when rebuild stop is complete.

Understanding Rebuild Stop Processing

The following list summarizes the events for a stop rebuilding request:

1. Stop rebuilding initiated through a SETXCF operator command or an IXLREBLD macro invocation.
2. The system reports Rebuild Stop event to the event exit. Connection must issue IXLEERSP to respond to the event.
3. Connection stops activity to the new structure, performs cleanup, and issues IXLEERSP EVENT=REBLDSTOP.
4. When all IXLEERSP responses are received, the system reports the Rebuild Stop Process Complete event to the event exit.

Delivery of Rebuild Stop Event

Once a request to stop a rebuild is received, XES presents the Rebuild Stop event to each event exit. This event requires a confirmation using IXLEERSP with EVENT=REBLDSTOP.

A Rebuild Stop event may supersede some other rebuild events. For example, if a connector has quiesced his use of the old structure and is waiting for the system to report the Rebuild Connect event, the system might instead report a Rebuild Stop event indicating that another connector or the operator has stopped the rebuild process. Similarly, a connector might receive a Rebuild Stop event instead of a Rebuild Quiesce event if another connector or the operator stopped a rebuild before all connectors have been notified about the pending rebuild.

The following rebuilding events can be superseded by a Rebuild Stop event:

- Rebuild Quiesce
- Rebuild Connect
- Rebuild Connects Complete
- Rebuild Duplex Established

Responding to a Rebuild Stop Event

The system reports the Rebuild Stop event and the reason to the event exit of all the connections. When connections receive the Rebuild Stop event, they must:

- Complete any outstanding requests to both the old and new structure. See “Completing Outstanding Structure Requests” on page 275 for complete information about handling outstanding requests.
- Before providing a response to the event, all connectors should clean up information related to the new structure, stop using the temporary connect token and the new vector token, and be prepared to resume using the old structure.
- Issue IXLEERSP with EVENT=REBLDSTOP to respond to the event.

When all connections have confirmed the Rebuild Stop event, the system reports that rebuilding has stopped (Rebuild Stop Process Complete event) to the event exit. If the original structure is not in a failed state, users can access the original structure using the original contoken and vector token. Otherwise, users might have to disconnect from the structure, or initiate another rebuild.

Handling New Connections During a User-Managed Rebuild Process

How new connections are handled differs substantially between rebuild and duplexing rebuild.

- **Rebuild**

  The system permits new connections to the original structure up until all responses for the Rebuild Quiesce event have been received. (The system must receive IXLEERSP responses from all connected users that are participating in rebuilding before it reports a Rebuild Connect event.)

  The system informs the new connection that rebuild is in progress by returning reason code IXLRSNCODESPECIALCONN from the IXLCONN invocation. The new connector can find information

about the rebuild in the IXLCONN answer area. CONAREBUILDINFO contains information about the reason for the rebuild, failed-persistent connectors, the percent loss of connectivity associated with an MVS-initiated loss of connectivity rebuild, and flags to indicate whether rebuild is in progress (CONAREBUILD) or rebuild stop is in progress (CONAREBUILDSTOP).

- If rebuild is in progress, the new connection can participate by first stopping activity to the original structure and then providing an IXLEERSP response with EVENT=REBLDQUIESCE. XES will monitor this required response.
- If rebuild stop is in progress, the new connection must provide an IXLEERSP response with EVENT=REBLDSTOP. XES will monitor this required response. See "Stopping a User-Managed Rebuild Process" on page 288.

Connections can listen for ENF event code 35 to determine when rebuilding is complete.

Note that for structure rebuild, a new connector can connect to the structure only up until the Rebuild Quiesce sync point is reached.

- **Duplexing Rebuild**

  New connectors to the old structure when the structure is in the Rebuild Quiesce, Rebuild Connect, or Duplex Established phases, are allowed with the following qualifications:

  - Rebuild Quiesce Phase

    A new connector to the old structure who requests to connect to a structure during the Rebuild Quiesce phase receives a valid CONTOKEN from IXLCONN for accessing the old structure. The connector's event exit does not receive a Rebuild Quiesce event, but the connector should examine the connect answer area to determine the state of the rebuild (such as CONAREBLDFLAGS to determine whether duplexing is in progress and CONAREBLDPHASE to determine the phase in which the connect occurred). The connector is expected to provide an IXLEERSP EVENT=REBLDQUIESCE confirmation, at which time the original CONTOKEN is invalidated. XES will monitor the required response to this event.

  - Rebuild Connect Phase

    A new connector to the old structure who requests to connect to a structure during the Rebuild Connect phase receives a CONTOKEN from IXLCONN that is not valid yet for accessing the old structure. The connector's event exit does not receive a Rebuild Quiesce event, and the connector is neither expected to return an IXLEERSP EVENT=REBLDQUIESCE confirmation, nor will it receive a Rebuild Connect event. However, the connector is expected to issue an IXLCONN REBUILD to connect to the new structure, at which time the original CONTOKEN will be validated and a new CONTOKEN will be returned from IXLCONN REBUILD. From that point on, the new connector is expected to participate in the duplexing process by propagating data to the new structure and comfirming its completion with IXLREBLD REQUEST=COMPLETE. XES will monitor the required responses to these events.

  - Duplex Established Phase

    A new connector to the old structure who requests to connect to a structure during the Duplex Established phase receives a valid CONTOKEN from IXLCONN for accessing the old structure. The connector's event exit does not receive a Rebuild Quiesce event, and the connector is neither expected to return an IXLEERSP EVENT=REBLDQUIESCE confirmation nor will it receive a Rebuild Connect event. However, the connector is expected to issue an IXLCONN REBUILD that will return a valid CONTOKEN with which to access the new structure. XES will monitor for the required IXLCONN REBUILD invocation. From that point on, the new connector is expected to participate in the duplexing rebuild process as are the other connectors.

    If a switch to the new structure is in progress when the connection completes (CONAREBUILDSWITCHINPROGRESS indicator), the connector is expected to participate in the switch by first issuing IXLCONN REBUILD to connect to the new structure and then IXLREBLD REQUEST=DUPLEXCOMPLETE when appropriate. XES will monitor the required responses to this event.

Handling Disconnections During Rebuilding

Users can normally disconnect from the structure during any stage of the rebuilding process. The system frees both original and new structure resources for the disconnected user. Existing connections receive a Disconnected or Failed Connection event in their event exits. This event reports whether the subject connection is connected to both the old and the new structures.

Handling Failed Connections During Rebuilding

If a connection fails or disconnects abnormally (REASON=FAILURE) during rebuilding, the system frees any resources for the user and reports the failed event to the event exit of all connected users. Existing connections must develop protocols to determine if they should continue to rebuild the structure.

If the failed user specified CONDISP=KEEP at connect time, the connection becomes failed-persistent. Existing peer connections might also need to develop special processing to handle this situation.

In some instances, the system allows a peer connector to respond on behalf of a failed connector before all responses have been received. See “Providing a Response for a Failed Connector” on page 291. After all existing connections have responded to the failed event through the event exit, the system also handles any outstanding event response that the failed connection needed to provide.

When connections rely on each other to coordinate rebuilding, they must coordinate how to respond when one of them fails. For example, connection A and B are each responsible for completing the rebuilding of a structure. Connection A rebuilds its share of the data into the new structure, but connection B fails before it can rebuild its data into the structure. The following occurs:

- Connection A responds to the failed event of connection B by issuing IXLEERSP with EVENT=DISCFAILCONN. Connection B has CONDISP=DELETE and is deleted.
- The system reports a Rebuild Cleanup event to the event exit of connection A.

At this point, connection A cannot stop the rebuilding process, and the new structure does not contain data updates from connection B.

To avoid this scenario, connection A can

- Stop the rebuild process prior to responding to the disconnect/failed event
- Issue the IXLEERSP response to delete connection B
- Perform recovery for connection B
- Initiate another rebuilding operation

If connection A is able to perform processing for connection B, connection A could also complete rebuilding the structure and then issue IXLEERSP to respond to the failed event. Thus, the structure can be rebuilt with the necessary data.

If all connections to the structure fail prior to the Rebuild Cleanup phase, the rebuild is stopped and the new structure is deallocated. If all connections fail during the Rebuild Cleanup phase, the rebuild is completed and the old structure is deallocated. ENF event code 35 is issued in either case when the structure is deallocated.

Providing a Response for a Failed Connector

The system permits a connector to respond on behalf of a failed peer connector in two instances:

- If the connector failed with an outstanding response to EVENT=REBLDCLEANUP
- If the connector failed with an outstanding response to EVENT=REBLDSTOP

and not all event responses have been received.

With the capability to provide a response on behalf of a failed connector, the previous rebuilding scenario could result as follows:

- Connection B fails with an outstanding response for a Rebuild Cleanup event.

- Connection A is notified of the Disconnected or Failed Connection event.
- Connection A responds to the Rebuild Cleanup event for Connection B using the PROXYRESPONSE=YES parameter.
- The rebuild process completes.
- Connection A responds to the Disconnected or Failed Connection event.

Note that the connector issuing IXLEERSP with the PROXYRESPONSE=YES keyword is responsible for following any protocols that the application uses during its rebuild cleanup or rebuild stop processing. For example, suppose an application uses two structures — a cache structure and a lock structure. When the rebuild of the lock structure enters the rebuild complete phase, the application updates the cache structure. If a connector fails at this point, and a peer connector decides to respond for the failed connector with the PROXYRESPONSE keyword, that connector has to ensure that the updates to the corresponding cache structure are performed. The updates could be performed immediately by the active connector that issued the PROXYRESPONSE confirmation, or could be done during the processing of the DISCFAILCONN event.

Handling Rebuild Connect Failures

When an IXLCONN REBUILD is issued from a task different from the original connecting task and the task fails before the IXLCONN REBUILD completes, all peer connections are notified of the REBUILD connect failure in their event exits. The peer connections must respond to the event with IXLEERSP EVENT=REBLDCONNFAIL or with an IXLYEEPL response. The IXLCONN REBUILD may be attempted again after all rebuild connect failure responses have been received, provided that rebuild is still in the phase where REBUILD connects are permitted. If REBUILD connects are not permitted, the original connection should disconnect or stop the rebuild.

Handling Failures during Duplexing Rebuild

This section summarizes how MVS handles certain failures during phases of the duplexing rebuild process. The failures discussed are:

- Loss of connectivity to one or more structures
- Failure of a structure
- Failure of a connection

Handling Loss of Connectivity during Duplexing Rebuild

The way in which the system handles loss of connectivity to a structure that occurs while duplexing rebuild is in progress depends on:

- The rebuild phase in which the loss of connectivity occurs, and
- Which of the structures experienced the loss of connectivity.

Before the Duplex Established Phase

Before duplexing is established, there is the possibility of both a new and an old structure existing, but not all connectors have issued IXLREBLD REQUEST=COMPLETE.

- If a loss of connectivity to the old structure occurs, MVS presents the Lossconn Percentage Notification (LOSSCONNPCTNOTIFY) event to all active connectors to the structure. The event indicates the percentage loss of connectivity. It is the connectors' responsibility to devise a protocol for respondiing to the percentage value. Depending on the amount of lost connectivity as specified by the lossconn percentage, the connectors might or might not be able to continue their processing to establish duplexing.
- If a loss of connectivity to the new structure occurs, MVS presents the Loss of Connectivity (LOSSCONN) event to all active connectors to the structure and immediately initiates a fallback to the old structure. Connectors can decide to attempt duplexing again if appropriate.

During the Duplex Established Phase

- If a loss of connectivity to the old structure occurs, MVS presents the LOSSCONN event to all active connectors and automatically initiates a switch to the new structure.
- If a loss of connectivity to the new structure occurs, MVS presents the LOSSCONN event to all active connectors and initiates a fallback to the old structure.

After the Duplex Established Phase

- If a loss of connectivity occurs to the old structure, MVS does not present the LOSSCONN event. Once the switch has completed, the old structure will be deallocated and the former new structure will not have experienced a loss of connectivity. If appropriate, another duplexing rebuild might occur.
- If a loss of connectivity occurs to the new structure after a switch has been requested, MVS defers presenting the LOSSCONN event until after the switch to the new structure is complete. At that time, the policy will determine whether another duplexing rebuild should be attempted.

  If another duplexing rebuild is not automatically initiated, the deferred LOSSCONN event might indicate to delay action, and if so, MVS will later present either an XES Recommended Action event or, for an MVS-initiated structure rebuild based on REBUILDPERCENT, a Rebuild Quiesce event.

  If another duplexing rebuild is automatically initiated, the deferred LOSSCONN event will be presented after the Rebuild Process Complete event, followed by a Rebuild Quiesce event indicating that MVS is initiating a duplexing rebuild. Those connectors who had lost connectivity to the former new structure are not able to participate in the duplexing and will receive a LOSSCONN event. The system delivers the Lossconn Percentage Notification event indicating the percentage loss of connectivity, to all active connectors. The connectors' protocol determines how the percentage is handled.

Handling Structure Failure

How the system handles the failure of a structure during the structure duplexing process again depends on the rebuild phase in which the structure failed, and which of the structure instances failed.

Before the Duplex Established Phase

- If the old structure fails before the Duplex Established phase, MVS presents the STRFAIL event for the old structure to all connectors and then stops the duplexing rebuild to fall back to the old structure. Connectors might need to disconnect, or can attempt to rebuild the structure, if possible.
- If the new structure fails before the Duplex Established phase, MVS stops the duplexing rebuild to fall back to the old structure and notifies connectors through the STOPDUPLEX reason code that they should attempt to duplex the structure. MVS does not present the STRFAIL event for the failure of the new structure.

During the Duplex Established Phase

- If the old structure fails during the Duplex Established phase, MVS presents the STRFAIL event for the old structure to all connectors and then initiates a switch to the new structure. At the completion of switch processing, connectors or MVS can attempt to duplex the structure again.
- If the new structure fails during the Duplex Established phase, MVS initiates a fallback to the old structure, at the completion of which, connectors or MVS can attempt duplexing again. MVS does not present the STRFAIL event.

After the Duplex Established Phase

- If the old structure fails after a switch to the new structure has been requested, the failure is ignored. MVS does not present the STRFAIL event for the old structure because it is in the process of being deallocated. At the completion of switch processing, connectors or MVS can attempt to duplex the structure again.

- If the new structure fails after a switch to the new structure has been requested, MVS allows the switch to complete before presenting the STRFAIL event to the connectors. It is not possible to duplex the structure because the single instance of the structure has failed.

Handling Connection Failure

When all active connections to a structure that is in the duplexing process fail or disconnect, the actions taken by MVS depend on the duplexing phase in which the last connector disconnects.

Before the Duplex Established Phase

Only the old structure is viable at this point, so MVS stops the duplexing to fall back to the old structure.

During the Duplex Established Phase

The following applies to the duplexed structure before a request to switch to the new structure is made:

- If the failure involves all connections to the structure and all systems using the CFRM active policy, MVS stops the duplexing rebuild to switch to the new structure.
- If the failure involves all connections to the structure, but does not include the failure of all systems using the CFRM active policy, MVS allows the structure to remain in its duplexed state with no active connections.

During Switch Processing and the Cleanup Phase

MVS completes the switch to the new structure.

During Stop Processing

If the structure was in the Duplex Established phase, and the failure involves all connections to the structure and all systems using the CFRM active policy, MVS stops the duplexing rebuild to switch to the new structure.

In all other cases, MVS stops the duplexing rebuild to fall back to the old structure.

MVS-Initiated Rebuild Processing

MVS provides the support that allows the installation to specify through its policy information whether or not a coupling facility structure should be rebuilt when a loss of connectivity to the coupling facility occurs. Loss of connectivity to a coupling facility can occur because of a failure of a coupling facility attachment or because of certain types of failures of the coupling facility itself. Depending on the scope of the failure, the appropriate action for MVS to take might be to initiate rebuild of a structure.

When a loss of connectivity from a system to a coupling facility occurs, MVS detects the failure on one or more systems in the sysplex. Each system on which the loss of connectivity is detected will execute an algorithm to determine what action should be taken. The algorithm is executed for each coupling facility structure affected by the loss of connectivity. Based on the results of the algorithm, MVS determines if policy action should be taken and notifies each connected user of the loss of connectivity event.

If the determination is to initiate a structure rebuild as a result of a loss of connectivity, any system that participates in the structure rebuild process that is running on a z15™ server that supports recovery process boosts will request System Recovery Boost's recovery process boost support to expedite the rebuild process running on that system to return connectivity to the structure and its content.

For more information on System Recovery Boost, see: IBM Z System Recovery Boost Content Solution (www.ibm.com/support/z-content-solutions/system-recovery-boost/) or IBM Documentation (www.ibm.com/docs/en/zos).

To allow MVS to initiate this structure rebuild, the installation must do the following:

1. Have a structure for which all active connections support structure rebuild.
2. Specify a REBUILDPERCENT value in your CFRM policy for each structure that MVS is to evaluate for rebuild, or allow it to default to 1. Note that the default rebuild percent value is 1 for systems at

OS/390 Release 10 and higher and for lower-level systems with OW41959 applied. Otherwise, the default rebuild percent value is 100.

3. Optionally, have in place an active SFM policy that supports the use of system weight values for performing recovery actions in the event of loss of connectivity between systems. (An active SFM policy is required for a sysplex made up of systems at OS/390 Release 2 or lower or a sysplex without OW30814 installed, if you want MVS to initiate a structure rebuild.)

How MVS Determines Whether to Initiate Structure Rebuild Processing

When MVS detects a loss of connectivity, MVS determines the viability of rebuilding each structure affected by the connectivity loss.

If the determination is to initiate a structure rebuild, MVS defers that action until one of the following occurs:

- The percentage of lost connectivity reaches 100%.
- The internal time value used by MVS expires.

The decision to initiate a structure rebuild is affected by the sysplex configuration (for example, an environment that is not failure-independent is recognized) and whether there is an active SFM policy in the sysplex.

- Non Failure-Independent Sysplex Environment

  A configuration that is not vulnerable to a single point of failure is failure-independent. Having a coupling facility reside on the same physical system as sysplex members that access it is not a failure-independent configuration. To handle loss of connectivity in situations where one or more systems in the sysplex reside on the same physical system as the coupling facility, systems with OW33615 installed or which are at OS/390 Release 9 or higher provide support to ensure that structure rebuild is initiated in a timely manner.

  Whether there is an active SFM policy or not, the system declares a loss of connectivity of 100% when:

  – All sysplex members that are failure independent report a loss of connectivity, and
  – Sysplex member(s) that are not failure independent do not report a similar loss of connectivity.

  The assumption is that all members in the same physical system as the coupling facility have lost connectivity when all other members have also lost connectivity. Setting the loss of connectivity percentage to 100% ensures that MVS will attempt a rebuild of the structures to which connectivity has been lost in a timely manner.

- Failure-Independent Sysplex Environment

  In a failure-independent sysplex environment, processing is dependent on whether there is an active SFM policy.

  – If MVS determines that there is an active SFM policy in the sysplex:
    - MVS verifies that the SFM policy data is at the same level on all systems.
    - MVS checks the active CFRM policy to see if a rebuild percent value has been specified for affected structures, or takes the default rebuild percent value of 1.
    - MVS calculates the percentage of lost connectivity using the system weights specified in the active SFM policy:
      - A = the total value of systems on which there exists a user of a coupling facility structure that resides in the coupling facility to which connectivity has been lost.
      - B = the total value of systems that have lost connectivity to the coupling facility and on which there exists a user of a structure in that coupling facility.

      Note that if there are multiple users of a coupling facility structure on one MVS system, that system weight is added to each total only once.
    - MVS calculates the total system weight of (A) all systems containing at least one active connection to the structure in the coupling facility that have lost connectivity, and (B) all systems containing

at least one active connection to a structure in the coupling facility for which lost connectivity has been recognized. Note that if there are multiple users of a structure on one system, that system weight is counted only once.

For example, if a structure has one connection per system and all systems are of equal weight 10, then in an eight-system sysplex if one system lost connectivity, the value of A (total system weight of all systems containing an active connection that have lost connectivity) is 10 and the value of B (total system weight of all systems containing an active connection) is 80.

- MVS determines what action is to be taken and informs connected users through event exit processing.

  The determination is arrived at by dividing A by B, multiplying by 100, and then comparing the result with the rebuild percent value for the structure in the active CFRM policy.

  - If the result is greater than or equal to REBUILDPERCENT, then MVS initiates a structure rebuild.
  - If the result is less than REBUILDPERCENT, MVS does not initiate a rebuild.

  In the example above, (10/80)*100 would be the value compared to the REBUILDPERCENT value. If the value of REBUILDPERCENT was 13 or higher, a rebuild would not be initiated.

– If MVS determines that there is not an active SFM policy in the sysplex:

  - MVS verifies that rebuild is supported for the structure.
  - MVS initiates the structure rebuild for any loss of connectivity affecting the structure, regardless of the REBUILDPERCENT specification, if structure rebuild is supported.

If the determination was to initiate a structure rebuild, MVS defers that action until one of the following occurs:

- The percentage of lost connectivity reaches 100%.
- The internal time value used by MVS expires.

Reporting the Percentage of Lost Connectivity

Once it initiates the rebuild processing, MVS notifies all connected users of the percentage of loss of connectivity through the event exit parameter list (EEPLREBUILDPCTLOSSCONN). This field is passed on all rebuilding events except Rebuild Complete and Rebuild Stop Complete. See Table 17 on page 335 for more information about data passed to the event exit. Based on the percentage of lost connectivity, users can decide whether to allow the rebuild process to continue.

The system determines the percentage of lost connectivity as follows:

- If there is an active SFM policy in the sysplex, the system uses the system weights defined in the policy using the calculations described above.
- On systems with OW33615 installed or which are at OS/390 Release 9 or higher, if there is no active SFM policy in the sysplex, then if all sysplex members that are failure-independent report a loss of connectivity and sysplex members that are not failure-independent do not report a similar loss of connectivity, the system reports the percentage loss of connectivity as 100%. Otherwise, the percentage loss of connectivity is 0.

Reporting Policy-Based Actions to Connectors

Connectors that are informed of the loss of connectivity event can examine the EEPLLOSSCONNDELAYACTION field in the IXLYEEPL to determine if MVS is initiating policy-based actions. EEPLLOSSCONNDELAYACTION is a bit that indicates the following:

- ON — MVS is taking policy-based actions, and will subsequently be reporting one of two actions to the event exit.

  – A Rebuild Quiesce event will be presented if MVS determines that rebuild processing is to be initiated.

  - A XES Recommended Action event will be presented at a later time to trigger action by the connection to disconnect from the structure.
- OFF — MVS could not process a policy action. This condition might occur for a variety of reasons including:
  - There is not an SFM policy that is active on ALL the systems in the sysplex, or there is a change that is being processed for the SFM policy across systems in the sysplex and the change has not yet been observed by all systems in the sysplex.
  - Rebuild is already in progress for the coupling facility structure.

Responding to the XES Recommended Action Event

The action that XES recommends to those connectors who have lost connectivity to a coupling facility structure is that the connection should disconnect from the structure. The recommendation is based on the percentage scope of lost connectivity calculated from the weights specified in the SFM policy. The percentage value in EEPLXESRECOMMENDACTIONPCTLOSSCONN indicates the percentage scope of lost connectivity calculated from the weights specified in the SFM policy, as seen by the system receiving the event. This percentage value is valid only when EEPLXESRECOMMENDACTIONPOLICY is equal to B'1'.

Dumping Considerations

If an SVC dump of the structure occurs during user-managed rebuild or duplexing rebuild, the rebuilding phase determines whether the system returns dump information for the original structure or the new structure:

- If SVC dump is requested during structure rebuild or duplexing, but before any IXLCONN REBUILD request has allocated the new structure, dump information is provided for the original structure only.
- If SVC dump is requested during structure rebuild or duplexing after an IXLCONN REBUILD request has allocated the new structure but before the Rebuild Cleanup sync point is reached, dump information is provided for both the original and the new structure.
- If SVC dump is requested during structure rebuild or duplexing after the Rebuild Cleanup sync point has been reached, dump information is provided for the new structure only.
- If SVC dump is requested during structure rebuild or duplexing up until the Rebuild Stop sync point is reached, dump information is provided for the old structure only.

Summary of User-Managed Structure Rebuild Processing

"User-Managed Rebuild Timeline" on page 298 summarizes the phases associated with the user-managed structure rebuild process.

The following list summarizes that process:

1. Rebuild for a structure is initiated through SETXCF START,REBUILD or IXLREBLD REQUEST=START or internally by MVS.
2. System reports Rebuild Quiesce event to each connector's event exit.
3. Connector stops activity to original structure and issues IXLEERSP EVENT=REBLDQUIESCE to respond to the event.
4. When all IXLEERSP responses are received, the system reports Rebuild Connect event to each connector's event exit.
5. Connector issues IXLCONN REBUILD for the structure. If the first to issue IXLCONN, the connector allocates the new structure; otherwise, the connector connects to the new structure.
6. At any time after successfully connecting to the new structure, the connector issues IXLCACHE, IXLLIST, IXLLOCK, IXLRT and other coupling facility macros to rebuild data for the structure.
7. When all connectors issue IXLCONN REBUILD, the system reports the Rebuild Connects Complete event to the connectors' event exits.
8. When the rebuild is complete, each connector issues IXLREBLD REQUEST=COMPLETE.

9. When all connectors have issued IXLREBLD REQUEST=COMPLETE, the system reports Rebuild Cleanup event to event exit.
10. Each connector cleans up references to original structure and issues IXLEERSP EVENT=REBLDCLEANUP.
11. When all IXLEERSP responses are received, the system reports the Rebuild Process Complete event to event exits.
12. Connector resumes normal processing with the new structure.

User-Managed Rebuild Timeline

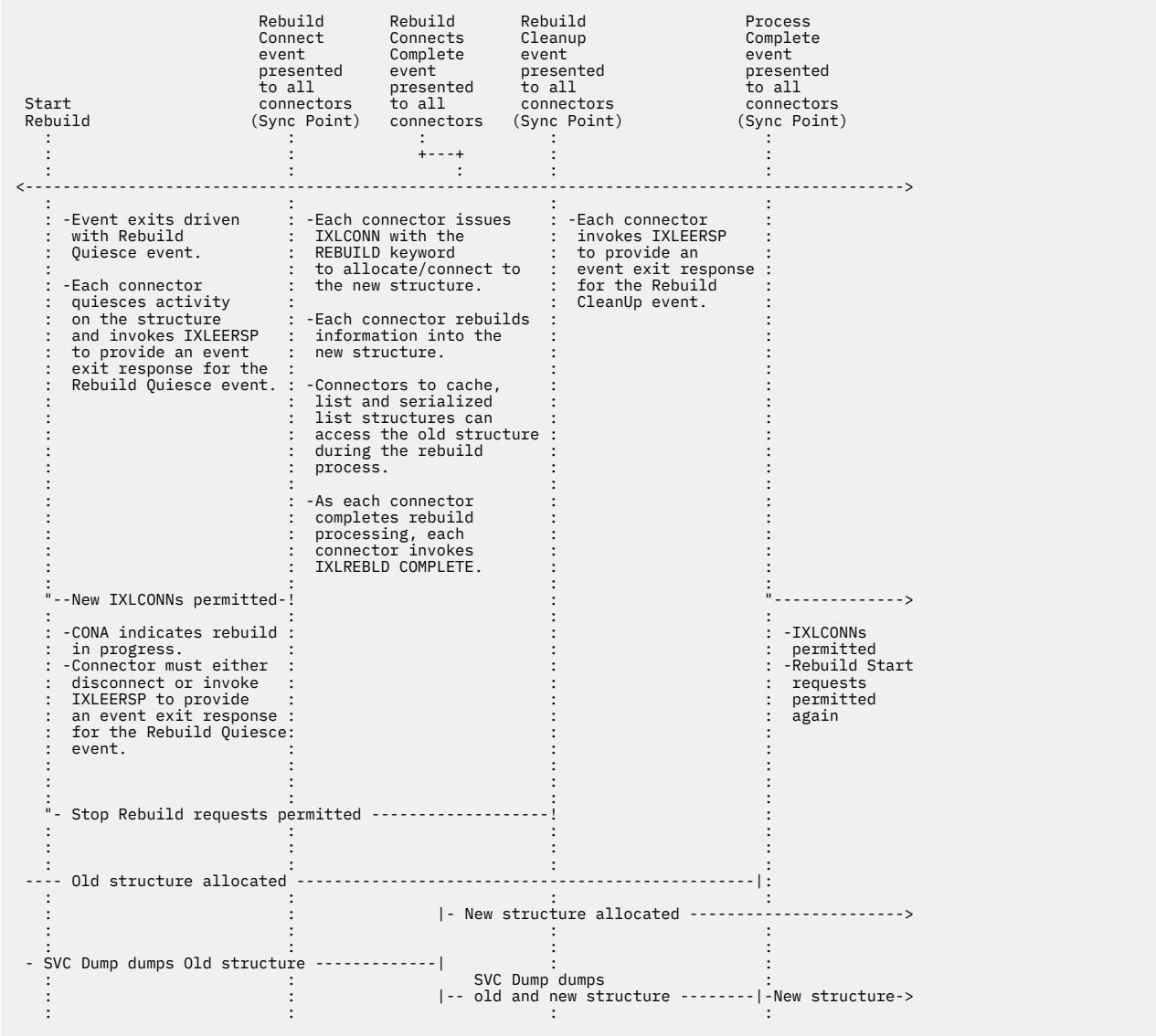


Figure 27. Rebuild Timeline

Summary of User-Managed Duplexing Rebuild Process

User-managed duplexing rebuild is available only for cache structures. “User-Managed Duplexing Rebuild Timeline” on page 300 summarizes the phases associated with the user-managed duplexing rebuild process.

The following list summarizes that process:

1. Duplexing rebuild for a structure is initiated through SETXCF START,REBUILD,DUPLEX or IXLREBLD REQUEST=STARTDUPLEX. or internally by MVS.

2. The system reports the Rebuild Quiesce event to connector's event exit.
3. Connector stops activity to the original structure and issues IXLEERSP EVENT=REBLDQUIESCE to respond to the event.
4. When all IXLEERSP responses are received, the system reports the Rebuild Connect event to the connector's event exit.
5. Connector issues IXLCONN REBUILD for the structure. If the first to issue IXLCONN, the connector allocates the new structure; otherwise, the connector connects to the new structure. The system revalidates the token to access the old structure and provides a new token to access the new structure.
6. Connectors propagate data to the new structure to synchronize both structures and issue IXLREBLD REQUEST=COMPLETE when finished.
7. When all IXLREBLD REQUEST=COMPLETE requests are received, the system reports the Duplex Established event to each connector's event exit.
8. Connectors continue in duplexed mode until a request is received to stop the duplexing and either fall back to the old structure or forward complete (switch) to the new structure.
9. If a fall back to the old structure is requested, the system reports the Rebuild Stop event to connector's event exit. See "Summary of Rebuild and Duplexing Rebuild Stop Processing" on page 300.
   - Connector quiesces use of both structures, completes any necessary processing for the new structure, and issues IXLREBLD REQUEST=DUPLEXCOMPLETE.
   - When all connectors have issued IXLREBLD REQUEST=DUPLEXCOMPLETE, the system reports a Rebuild Cleanup event to each connector's event exit. Connector must issue IXLEERSP to respond to the event.
10. If a switch to the new structure is requested, the system reports a Rebuild Switch event to connector's event exit.
    - Connector cleans up references to the old structure. The original token is used to access the new structure.
    - When all IXLEERSP responses are received, the system reports a Rebuild Process Complete event to each connector's event exit.
11. Connector resumes processing with the remaining structure.

User-Managed Duplexing Rebuild Timeline

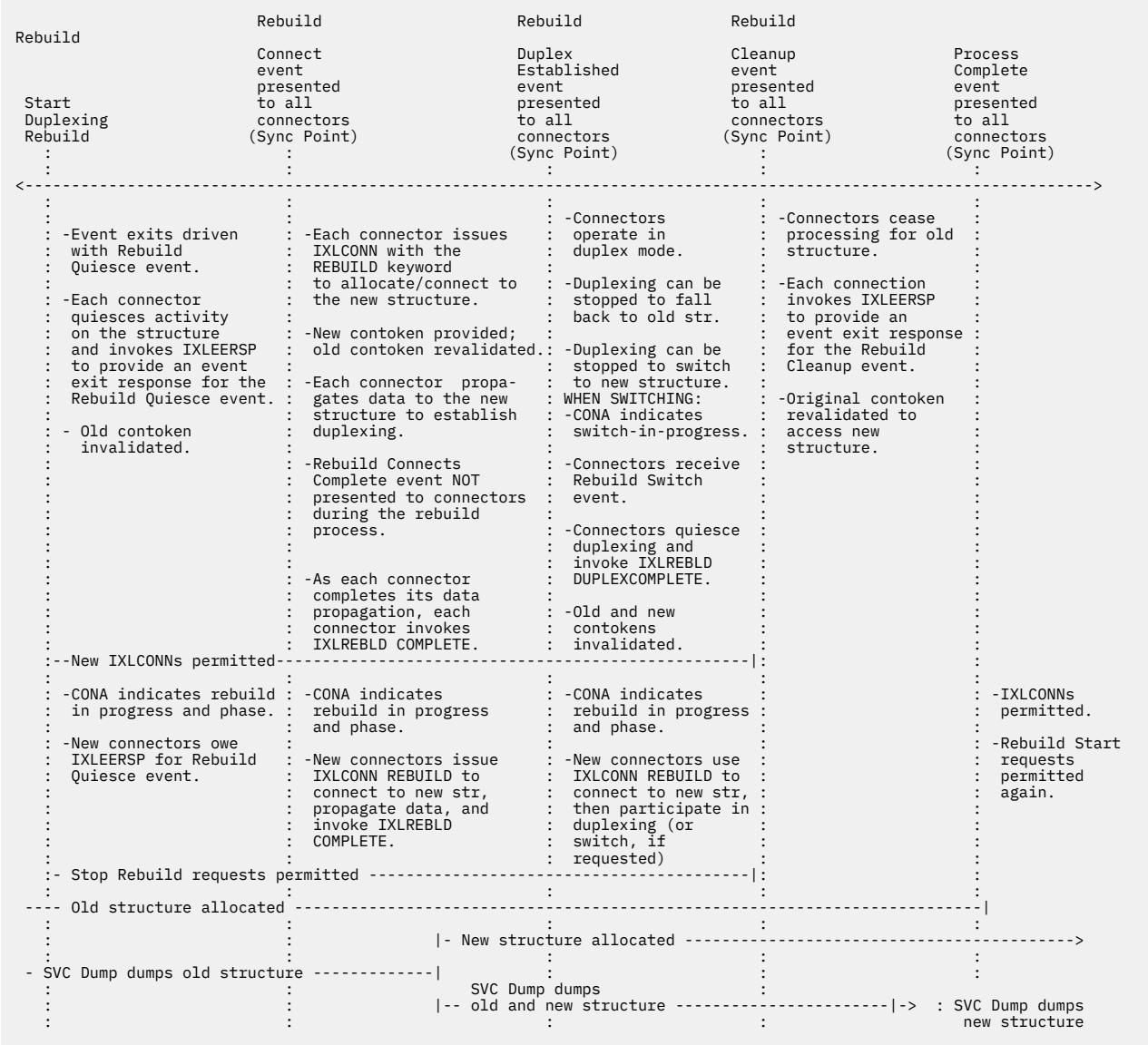


Figure 28. User-Managed Duplexing Rebuild Timeline

Summary of Rebuild and Duplexing Rebuild Stop Processing

The steps for stopping a structure rebuild to continue processing with the old structure are identical to the steps for stopping a duplexing rebuild to continue processing with the old structure.

1. Connector or operator requests that the rebuild or duplexing rebuild process be stopped and processing continues with the old structure.
2. The system reports a Rebuild Stop event to connector's event exit. Connector must issue IXLEERSP to respond to the event.
3. Connectors quiesce use of the new structure, perform cleanup, and respond to the Rebuild Stop event with IXLEERSP EVENT=REBLDSTOP.
4. When all IXLEERSP responses are received, the system reports a Rebuild Stop Process Complete event to event exit.
5. Connector resumes processing with the old structure.

Overview of System-Managed Rebuild Processing

System-managed processing provides a means for rebuilding or duplexing a structure with minimal participation from connectors to the structure. During a system-managed process, connectors receive events that delineate the period during which the structure is unavailable for requests. During that time, the system defers accesses to the structure, manages the old and new structure instances that exist during the process and propagates data to the new structure. At the conclusion of the system-managed process, connectors receive one or more events notifying them of any changes to the structure.

The two types of system-managed processing are rebuild and duplexing rebuild.

- Rebuild is intended for planned reconfiguration scenarios. The system allocates the new structure, propagates the necessary structure data to the new structure, and then switches over to using the new structure instance.
- Duplexing rebuild is intended to provide a failure recovery capability through failover to the unaffected structure instance. The system performs the significant steps in the duplexing rebuild process, including allocating the secondary (new) instance of the structure, attaching users to the new structure instance, copying all necessary data from the primary (old) instance to the secondary instance, and then transparently duplexing coupling facility operations to both instances of the structure.

System-managed processing can function as long as the old and new structures remain viable and there is at least one system in the sysplex capable of performing the required system-managed processing. Connectors can specify that they support system-managed processing even if they do not support user-managed rebuild or duplexing rebuild.

During a system-managed process, the system defers any requests that are submitted while the structure is unavailable. The requests will be processed after the rebuild process has completed, been terminated, or for duplexing rebuild, reached the duplexed established phase. In a system-managed process, connectors are not required to cease their operations against the structure before responding to the event signifying that the structure is unavailable. However, IBM recommends that they do so to minimize the system resources required to quiesce activity against the structure.

System-managed rebuild is supported only for planned reconfiguration. When the coupling facility or the structure has failed, or when any active connectors have lost connectivity, system-managed rebuild will not be used to rebuild the structure.

System-managed rebuild has the following requirements:

- The structure must be allocated in a coupling facility of CFLEVEL=8 or higher.
- The CFRM couple data set must have been formatted with the ITEM NAME(SMREBLD) NUMBER(1) statement and be active as the primary CFRM couple data set. In order to activate the CFRM couple data set, all systems using the CFRM couple data set must be at OS/390 Release 8 or higher.
- A list structure or lock structure with record data must have been allocated by a system at OS/390 Release 8 or higher in order for system-managed rebuild to occur.

System-managed duplexing rebuild is intended to provide a robust failover capability through the creation and maintenance of a duplex copy of a structure in advance of any failure. However, when the coupling facility or the structure has failed, or when any active connectors have lost connectivity, system-managed duplexing rebuild will not be used to rebuild the structure.

System-managed duplexing rebuild has the following requirements:

- The structure must be allocated in a coupling facility of CFLEVEL=11 or higher.
- CF-to-CF links must be available.
- The CFRM couple data set must have been formatted with the ITEM NAME(SMDUPLEX) NUMBER(1) statement and be active as the primary CFRM couple data set. In order to activate the CFRM couple data set, all systems using the CFRM couple data set must be at z/OS Release 2 with OW41617 installed.
- A list structure or lock structure with record data must have been allocated by a system at OS/390 Release 8 or higher in order for system-managed duplexing rebuild to occur.

IXLCONN support for system-managed processing

To support system-managed rebuild processes, the following must be specified on the IXLCONN invocation:

- For both system-managed rebuild and duplexing rebuild, IXLCONN ALLOWAUTO=YES must be specified by all connectors to the structure.

Phases for system-managed processing

The system-managed processes involve a series of phases, during which the system coordinates all activities required to rebuild or duplex the structure. The system is responsible for managing the structure and its contents. While the system is managing the rebuild or duplexing rebuild process, it will perform actions on behalf of the connector while running in the connector's address space and perform system-based processing from the XCF address space to reconstruct the new structure from the old structure.

The connector is responsible for recognizing three events — Structure Temporarily Unavailable, Structure State Change, and Structure Available — and must respond to the Structure Temporarily Unavailable event before the system assumes responsibility for managing the subsequent rebuild or duplexing rebuild process.

The system-managed phases are:

- Startup
- Quiesce
- Allocate
- Attach
- Copy
- Duplex Established (duplexing rebuild only, excluding asynchronous duplexing)
- Async Duplex Established (asynchronous duplexing only)
- Cleanup

If there are no active connectors to the structure, the Startup, Quiesce, Attach, and Cleanup phases is not driven.

A system-managed rebuild and duplexing rebuild process transitions through these phases but a response from the connector is only required during the Startup phase for the Structure Temporarily Unavailable event. The other system-managed phases are not externalized to the connector through event exit events, but are handled internally by MVS. The IXCQUERY macro and DISPLAY XCF messages provide information about the structure during the entire rebuild or duplexing rebuild process.

A brief description of each of the system-managed phases follows.

Startup Phase

During the Startup phase, the system will notify connectors of the impending system-managed rebuild or duplexing rebuild through the Structure Temporarily Unavailable event. Connectors are required to respond to this event.

Quiesce

When all responses to the Structure Temporarily Unavailable event have been received from active connectors, the system will notify XES on behalf of all connectors of the request to rebuild or duplex the structure through the Rebuild Quiesce event. XES responds to this event on behalf of the connector.

All activity to the structure will be quiesced once all responses to the Rebuild Quiesce event have been received.

Allocate

During this system-based process, one system is responsible for allocating a new instance of the structure.

Attach

Systems with active connections will perform system-based attach to connect the active connections to the new structure.

Copy

Systems with connectivity to both the old and the new structure will perform system-based copy. This phase is further divided into subphases based on the type of structure being rebuilt or duplexed.

When copy processing is complete, for rebuild processing, the Cleanup phase is entered. For duplexing rebuild, the Duplex Established or Async Duplex Established phase is entered.

Duplex Established or Async Duplex Established

The system unquiesces the structure and redrives user operations that have been delayed while the structure was quiesced. Operations which were originally intended to be sent to a single coupling facility structure in simplex mode are now converted into the appropriate duplexed operations. The structure remains in this phase until either a request is issued to stop duplexing or a failure of the duplexing protocol occurs.

During this phase, the system notifies all connectors of the Structure State Change event, which is used to report the new “composite” attributes of the duplexed pair of structures. The Structure Available event also is issued since the structure is no longer quiesced, and the structure-specific ENF 35 signal is issued so that users can request connections to the structure that had been prevented during the period that the structure was quiesced. Connectors do not need to respond to these events.

Cleanup

During the system-managed Cleanup phase, the system will notify all connectors of the Structure State Change event, deallocate the old instance of the structure, and resume access to the structure so that the queued requests can be driven against the new structure. Connectors do not need to respond to the Structure State Change event.

At the conclusion of the Cleanup phase, the system delivers the Structure Available event to all connectors and may also deliver the Alter Begin and Alter End events as well.

Role of CFRM in system-managed processing

For system-managed rebuild, the system uses the values from the CFRM active policy with the following exceptions:

- The system does not automatically initiate system-managed rebuild based on REBUILDPERCENT calculations. The specification of REBUILDPERCENT in a CFRM policy structure description applies to the percentage of connections that lost connectivity to the structure. System-managed rebuild is not supported for lost connectivity.
- If there is a pending CFRM policy change that modifies the structure SIZE or INITSIZE, and all connectors specified IXLCONN ALLOWALTER=YES, the system will allocate the new structure using the sizes from the pending policy, subject to the requirement that the resulting size be large enough to contain the data to be copied from the old structure. In some cases, the resultant structure size might be larger than the maximum structure SIZE specified in the CFRM policy.
- If there is a pending CFRM policy change that modifies the structure SIZE or INITSIZE, and not all connectors specified IXLCONN ALLOWALTER=YES, the policy changes will remain pending after the system-managed rebuild.

For system-managed duplexing rebuild, the system's use of the CFRM policy values is as follows:

- The DUPLEX(ENABLED[,dupops]) or DUPLEX(ALLOWED[,dupops]) specification determines whether the installation intends a structure to be eligible for duplexing along with any CF site and duplexing mode preference.
- A change in the CFRM policy to DUPLEX(DISABLED) causes system-managed duplexing rebuild to stop.
- System-managed duplexing rebuild cannot be started when CFRM policy changes are pending for the structure.

MVS-Initiated Duplexing Rebuild

When DUPLEX(ENABLED) is specified for a structure in the CFRM active policy and the structure is not duplexed, MVS will attempt to start system-managed or user-managed duplexing rebuild when certain triggering events occur in the system:

- Connect
- Disconnect
- Change policy
- Force
- Gain connectivity to a coupling facility
- Gain ownership of a coupling facility (first system in the sysplex gains access to a coupling facility
- Reconciliation (comparison of coupling facility structure contents to CFRM policy contents
- Rebuild or Duplexing Rebuild process completion
- REALLOCATE processing

These are the same trigger conditions that apply to user-managed duplexing rebuild. In addition, the system establishes a monitor to initiate duplexing for certain triggering events. The duplex enabled monitor will initiate either a user-managed or system-managed duplexing rebuild. It is only established when the CFRM couple data set for SMDUPLEX is in use. This periodic monitoring identifies those structures which previously could not be duplexed due to lack of resources, but now might be able to be duplexed. The duplexing attempt is an **eventual** action as opposed to an immediate action. For example, if coupling facility resources become available as a result of one of the following events, the system will eventually attempt to duplex one or more structures using the coupling facility resources. The events for which periodic monitoring applies are:

- Release of structure dump serialization
- Structure deallocation
- Structure alter/contraction completion
- Reduction in coupling facility dump space size

In addition to these, the following conditions apply only to system-managed duplexing rebuild:

- Gain of CF-to-CF link connectivity
- PSWITCH to a CFRM couple data set that supports SMDUPLEX

Events and the Event Exit for System-Managed Processing

If the system has determined that system-managed processing is to occur, during its course the system will present events to the event exits of all active connectors to the structure. The events notify the connected users of the progress of the system-managed process and of changes to the structure that might occur as a result.

The following list summarizes the events that the system reports about the system-managed process to the event exit and the responses expected by the event exits:

Structure Temporarily Unavailable
Indicates the start of the system-managed process, during which the structure is unavailable for processing coupling facility requests. Response is required via IXLYEEPL or IXLEERSP.

Structure State Change
Describes changes to the structure or the coupling facility in which the structure resides. These changes might have occurred as a result of the system-managed process. Response is not required.

Structure Available
The structure is available for coupling facility requests. Response is not required.

In addition, if the structure connectors allow structure alter, the following events may be presented to inform the connectors about structure object count changes that occurred as a result of the system-managed rebuild.

Alter Begin
Indicates the start of alter processing associated with the system-managed rebuild process. If all connectors had specified ALLOWALTER=YES, both the Alter Begin and Alter End events are delivered. Response is not required.

Alter End
Indicates the end of alter processing associated with the system-managed rebuild process. Response is not required.

XES Monitoring of Active Connector Event Responses

XES monitors the Structure Temporarily Unavailable event to ensure that connected users respond in a timely manner. If a response is not received in a timely manner, XES issues a message for each connector owing an expected response that is overdue. These messages can then be analyzed by the system programmer, operator, or automation package for the appropriate action to be taken so that processing can continue. See "XES Monitoring of Event Responses" on page 341.

Starting the System-Managed Process

The system presents the Structure Temporarily Unavailable event to the event exits of active connectors to communicate the start of the system-managed rebuild or duplexing rebuild. To determine why the structure is temporarily unavailable, examine the IXLYEEPL. If EEPLSTRAVAILABILITYPROCESS=EEPLSYSMANAGEDREBUILD, then the structure is unavailable because of a system-managed rebuild request. If EEPLSTRAVAILABILITYPROCESS=EEPLSYSMANAGEDDUPLEXINGREBUILD, then the structure is unavailable because of a system-managed duplexing rebuild request.

- To allow the rebuild or duplexing rebuild to continue, the connector must respond to the event.
- If you do not wish to be connected to the new instance of the structure that will be created by the rebuild or duplexing rebuild, you can either stop the process by issuing IXLREBLD REQUEST=STOP or IXLREBLD REQUEST=STOPDUPLEX or by disconnecting from the structure. However, connectors that are supporting system-managed processes are unlikely to disallow the rebuild or duplexing rebuild from continuing.

Responding to the Structure Temporarily Unavailable Event

Before responding to the Structure Temporarily Unavailable event, connections should consider quiescing their use of the structure. This is not required for system-managed processes, because the system will defer any incoming requests until the structure is again available. However, IBM does recommend that once the Structure Temporarily Unavailable event is received, connectors refrain from issuing coupling facility requests against the affected structure. This minimizes system resources required to quiesce operations during the system-managed process and improves overall system performance.

To respond to the Structure Temporarily Unavailable event, either set the return code in IXLYEEPL (EEPLRETCODE=IXLRCEVENTEXITRESPONSE) or issue IXLEERSP EVENT=STRTEMPUNAVAIL.

In a system-managed process, the system does not invalidate the connector's connect token as it does in a user-managed process.

Note that once a response has been provided for the Structure Temporarily Unavailable event and before the Structure Available event has been presented to signify the completion of the system-managed

process, XES event exits should neither issue any coupling facility requests nor suspend processing, as either of these actions could cause a deadlock with the system-managed processing.

After the system receives all responses to the Structure Temporarily Unavailable event, the system quiesces activity against the structure. While structure activity is quiesced, the system handles exit routines as follows:

- The system does not drive the contention, complete, or notify exits.
- The system disables the list transition exit for a structure that is undergoing a system-managed process.

Suspending Work Units during System-Managed Processing

At connect time, connectors must specify the IXLCONN SUSPEND parameter to indicate whether the connection wants the system to suspend work units that issue coupling facility requests against a structure while the structure is quiesced during a system-managed process, regardless of the MODE specified on the request.

- Specifying SUSPEND=YES directs the system to override a request's MODE specification when possible, and suspend the requestor. This permits the system to limit the number of incoming requests by suspending the work units that otherwise would be submitting them, and thus minimize the system resources required to quiesce activity against the structure that is undergoing the system-managed process. When the system overrides the MODE parameter and suspends the requestor, upon unquiesce of the system-managed process, the system:
  - Resumes the requestor
  - Notifies the requestor of request completion as specified by the original MODE value.
    - If the MODE requested that the sytem attempt to complete the request synchronously, only returning once the request completes (MODE=SYNCSUSPEND), the requestor will receive a return code indicating final disposition of the request when it completes. For IXLLOCK and serialized list (IXLLIST, IXLLSTC, IXLLSTE) requests encountering contention, this means after contention has been resolved.
    - If the MODE requested that the system attempt to complete the request synchronously and can tolerate asynchronous completion (for example, MODE=SYNCEXIT), the requestor will receive a return code indicating synchronous completion. Notification of any asynchronous request completion occurs once the request actually completes using the mechanism specified by the MODE parameter (for example, the Complete exit). For IXLLOCK and serialized list (IXLLIST, IXLLSTC, IXLLSTE) requests encountering contention, notification of request completion occurs after contention has been resolved.
    - If the MODE requested asynchronous processing (for example, MODE=ASYNCEXIT), the requestor will receive a return code indicating asynchronous completion and will be notified of the results of the request through the mechanism specified by the MODE parameter (for example, the Complete exit). For serialized list (IXLLIST, IXLLSTC, IXLLSTE) requests that encounter contention, notification of request completion occurs after contention has been resolved.
- Specifying SUSPEND=NO indicates to the system that the connector cannot tolerate suspension of work units that have submitted coupling facility requests against a structure, except as noted on the list (IXLLIST, IXLLSTC, IXLLSTE, IXLLSTM) or cache (IXLCACHE) or IXLLOCK MODE parameter. The system will honor the requests' MODE specification in completing the request and will use suspend/resume processing only when MODE=SYNCSUSPEND is specified by the work unit. The system will quiesce all other activity to the structure undergoing system-managed processing by deferring requests internally until the quiesce process completes.
- Specifying SUSPEND=FAIL indicates that the connector cannot tolerate the potentially long-term suspension or delay of units of work submitting coupling facility requests while the structure is quiesced for system-managed processing. Requests which cannot be immediately processed due to the structure being quiesced for system-managed processing will be failed. Such requests will neither be deferred internally for asynchronous processing, nor will the requesting unit of work be suspended. SUSPEND=FAIL is not applicable to lock or serialized list structures. Connect attempts to these types of structures will fail with reason code IXLRSNCODEBADSUSPENDOPTION if SUSPEND=FAIL is specified.

Support for connectors to a coupling facility structure that allows them to specify that the system is to fail a request rather than have it suspended or processed asynchronously during a system-managed process is only available when:

- The system is running OS/390 V2R8 through z/OS V1R1 with APAR OW39892 or z/OS V1R2 and higher.
- The connector specifies the IXLCONN request with the ALLOWAUTO=YES and SUSPEND=FAIL keywords. These keywords require an IXLCONN macro version of 7 or higher to be specified or defaulted by the IXLCONN request's PLISTVER keyword.

To determine whether the support is available on the system from which you are connecting to a structure, issue IXCQUERY REQINFO=FEATURES. QUREQRFIXLCONNSUSPENDFAIL, if returned, indicates whether SUSPEND=FAIL support is available. If the support is not available and you connect with SUSPEND=FAIL, the results will be unpredictable.

Note that the SUSPEND=YES and SUSPEND=NO options do not affect IXLLOCK requests that specify MODE=SYNCFAIL. If the system receives a MODE=SYNCFAIL request while the target structure is unavailable because of system-managed rebuild processing, the request is not deferred. Instead, the system fails the request with the IXLRSNCODENODELAY reason code, regardless of the value specified by the IXLCONN SUSPEND keyword.

Establishing the New Structure in System-Managed Process

Both system-managed processes (rebuild and duplexing rebuild) follow the same basic approach to establishing the new structure, that is, the system creates a new instance of the structure, connects users to the structure, and populates the new instance of the structure with data from the old structure. The sequence of events is:

1. One of the systems in the sysplex allocates a new instance of the structure.
2. Each system in the sysplex connects users from that system to the new instance of the structure.
3. One or more of the systems in the sysplex copies data from the old instance of the structure to the new instance.

Allocating the New Structure

The process of allocating the new structure differs significantly between system-managed rebuild and duplexing rebuild, primarily because of the stringent coupling facility requirements needed by duplexing rebuild.

- System-managed Rebuild

  The system determines the location of the new structure using the following guidelines:

  - If the request to start the rebuild specified POPULATECF, only the specified coupling facility is a valid rebuild target.
  - If the request to start the rebuild specified LOCATION=OTHER or there is no pending policy change affecting the relevant structure, the new structure will not be allocated in the same coupling facility as the original structure.
  - The new structure will not be allocated in the same coupling facility as the original structure unless one of the following is true:
    - There is a pending policy change that does not involve a change to the structure SIZE or INITSIZE.
    - There is a pending policy change that affects SIZE or INITSIZE, and all active or failed-persistent connectors specified IXLCONN ALLOWALTER=YES.
  - The allocating system allocates the structure in the first coupling facility in the preference list that meets the standard allocation requirements, with the following additional requirements.
    - The coupling facility must have sufficient available storage to allocate a new structure that will be large enough to contain all the data to be copied from the old structure.

- The coupling facility must be at a CFLEVEL sufficient to support the system-managed rebuild process. System-managed rebuild requires a coupling facility of CFLEVEL=8 or higher.
- The CFLEVEL must be as least as high as the CFLEVEL reported to connectors when they connected to the original structure.
- All systems in the sysplex with active connectors to the structure undergoing system-managed rebuild must have connectivity to the coupling facility.

The system stops the rebuild process if there is no coupling facility that meets the allocation requirements.

Connectors do not have the option of changing structure attributes during a system-managed rebuild. In the new structure, the attributes that can be specified on IXLCONN will be identical to those of the old structure, with the following possible exceptions:

– If there is a pending CFRM policy change that modifies the structure SIZE or INITSIZE, and all active and failed-persistent connectors specified IXLCONN ALLOWALTER=YES, the system will allocate the new structure using the sizes from the pending policy, subject to the requirement that the resulting size must be large enough to contain the data to be copied from the old structure. In this case, resulting structure attributes such as entry-to-element ratios may differ from the values originally specified by connectors.

  Whether or not structure attributes change during a system-managed rebuild, if all connectors specified IXLCONN ALLOWALTER=YES the system will present Alter Begin and Alter End events to the event exits of active connectors at the conclusion of system-managed rebuild processing (after the Structure Available event has been delivered).

– For list and lock structures, if the coupling facility model-dependent limit for the maximum number of connectors to a structure of a given type is different for the new structure instance and all the connectors to the old structure instance support user-id limit changes by specifying MAXCONN on the IXLCONN, the resulting structure attribute, user-id limit,for the new structure instance can differ from the value of the old structure instance. XCF communicates the user-id limit change through the structure state change event.

• System-managed Duplexing Rebuild

  The system will always assume a rebuild attribute of LOCATION=OTHER when allocating the secondary structure, so that the primary and secondary instances are allocated in two different coupling facilities. When there are active connectors to the structure, the system will also assume a rebuild attribute of LESSCONNACTION=TERMINATE, so that the attempt to start duplexing for the structure will be automatically stopped if it would cause a loss of coupling facility connectivity for any active connector to the structure. Lastly, the system will give strong preference to placing the structures in two different coupling facilities that are failure-isolated with respect to one another.

  Determining the eligibility of a coupling facility in which to allocate the secondary structure takes the following considerations into account.

  – ENFORCEORDER

    When the installation has specified in the CFRM policy that the preference list order is to be strictly enforced for the structure, the system will only apply those eligibility list considerations that involve dropping ineligible coupling facilities from the preference list. MVS will not apply any of the considerations that involve weighting the coupling facilities and reordering the preference list based on these attribute weights (considerations for volatility, failure-isolation from connectors, failure-isolation from primary coupling facility, and exclusion list).

  – CF-to-CF Link Connectivity

    At the time of the secondary structure allocation, there must be CF-to-CF connectivity between the coupling facility in which the primary structure is allocated and any coupling facility in which the secondary structure is to be allocated. Any coupling facility in the preference list that does not have CF-to-CF connectivity to the coupling facility where the primary structure resides is dropped from the eligibility list. Message IXC574I and CONAFACILITYARRAY indicate the reason why the secondary structure could not be allocated in a particular coupling facility.

– LOCATION(OTHER)

  The secondary structure cannot be allocated in the same coupling facility as the primary structure under any conditions. The coupling facility that contains the primary structure is dropped from the eligibility list.

– LESSCONNACTION(TERMINATE)

  All active connectors to the primary structure must also have connectivity to the coupling facility in which the secondary structure is to be allocated. Therefore, any coupling facility that does not provide connectivity for all current active connectors to the structure is dropped from the eligibility list.

  When there are no active connectors to the structure, MVS may allow the allocation of the secondary structure in a coupling facility that has less connectivity to systems than does the coupling facility in which the primary structure is allocated. If, at a later time, a connector attempts to connect to the now-duplexed structure and MVS observes that the connector is running on a system that does not have connectivity to both structure instances, MVS will drop the structure out of duplexing. The structure instance that will be kept is that which is accessible to the connector. Note that after the connector connects, MVS may subsequently reduplex the structure into another coupling facility that does provide full connectivity for the set of active connectors to the structure.

– POPULATECF

  The concept of “POPULATECF” is not applicable to a duplexing rebuild, and thus does not affect the allocation of a secondary structure. Rebuild processes of any kind cannot be initiated against a duplexed structure.

– Available Space

  In order to create a secondary structure that is an exact copy of the primary structure (exact structure attributes and same total and maximum structure counts for all structure objects), it might be necessary for MVS to allocate the secondary structure with a size that significantly differs from that of the primary structure. This is because of different coupling facility storage allocation mechanisms that may exist at different CFLEVELs. Therefore, MVS will process each coupling facility in the preference list as follows:

  - Determine the target size (and minimum required control space) of a structure allocated in this particular coupling facility that has the same structure attributes and total and maximum object counts as the primary structure. Note that the maximum structure size from the CFRM policy is not used as an upper bound to the determined target structure; rather, the maximum structure size is allowed to “float” to whatever size is necessary to accomodate the number of structure objects that exist in the primary structure.
  - Compare the determined target size and minimum required control space results to the actual free space and free control space in this coupling facility.
    - If the coupling facility has sufficient free space and free control space to accomodate the allocation of the structure, include this coupling facility in the eligibility list.
    - If not, drop this coupling facility from the eligibility list.

– CFLEVEL

  System-managed duplexing rebuild requires a coupling facility of CFLEVEL=11 or higher, so any coupling facility in the preference list which is at a lower CFLEVEL is dropped from the eligibility list.

  In addition, the secondary structure must be allocated in a coupling facility that is at least at a CFLEVEL as high as, or higher than, the highest CFLEVEL that has been reported back to any connector to the structure via IXLYCONA at any time since the primary structure was allocated. This CFLEVEL will be the lower of either (a) the actual CFLEVEL of the coupling facility in which the primary structure is allocated, or (b) the highest requested CFLEVEL value requested by any past or present connector to the primary structure. Any coupling facilities that are not at this minimum required CFLEVEL or higher are dropped from the eligibility list.

– Volatility

If any active or failed-persistent connectors to the structure requested nonvolatility, the system will give preference in the eligibility list to allocating the structure in a nonvolatile coupling facility, using the normal eligibility list weighting for nonvolatility.

– Failure-isolation from Connectors

  If any active or failed-persistent connectors to the structure requested nonvolatility (and thus implicitly requested failure-isolation), MVS will give preference to allocating the secondary structure in a coupling facility that is standalone, that is, failure-isolated with respect to all active connectors to the structure.

  Non-standalone coupling facilities, which do not provide failure-isolation from all active connectors, will be allowed to remain in the eligibility list, but behind those that do provide full failure-isolation, using the normal eligibility list weighting for failure-isolation.

– Failure-isolation from Primary Coupling Facility

  MVS will give preference to allocating the secondary structure in a coupling facility that is duplex failure-isolated (that is, in a different processor) from the coupling facility in which the primary structure is allocated. Coupling facilities that do not provide failure-isolation from the primary coupling facility will be allowed to remain in the eligibility list, behind all coupling facilities that do provide this failure-isolation. This failure-isolation from the primary coupling facility is a very high-weighted attribute in determining the eligibility list order.

  If the secondary structure is allocated in a coupling facility that is not duplex failure-isolated from the primary, MVS issues a highlighted eventual action warning message, IXC553E, to warn about the lack of CF-to-CF failure-isolation.

– Exclusion List

  The system prefers to allocate the secondary structure in a coupling facility that does not contain any allocated structure instances (primary or secondary) for any structure listed in this structure's exclusion list. However, coupling facilities containing such structure instances will be allowed to remain on the eligibility list, subsequent to those that do not contain such structures, given the normal exclusion list eligibility list weighting.

– Application of Pending Policy

  The system will not initiate system-managed duplexing rebuild processing while a CFRM policy change is pending against a structure. When such a policy change is pending, the installation must take whatever action against the structure is required to cause the pending policy change to take effect. This generally involves rebuilding the structure through either a user-managed or system-managed rebuild process, or causing the structure to be deallocated or reallocated.

– Model-dependent Limit on Number of Connectors

  It is necessary that all connectors to the primary (or old) structure can be attached to the secondary (or new) structure with the same Connection ID number. The maximum number of connectors to a structure of a given type is a model-dependent coupling facility attribute, and therefore it is possible that some of the coupling facilities in the preference list have a limit that is too low to accommodate all of the attachments that are present in the old structure at this time. Any coupling facility whose model-dependent limit on the number of connectors is insufficient will be dropped from the eligibility list.

Connectors do not have the option of changing structure attributes during a system-managed duplexing rebuild. In the new structure, the attributes that can be specified on IXLCONN will be identical to those of the old structure, with the following possible exception:

– For list and lock structures, if the coupling facility model-dependent limit for the maximum number of connectors to a structure of a given type is different for the new structure instance and all the connectors to the old structure instance support user-id limit changes by specifying MAXCONN on the IXLCONN, the resulting structure attribute and user-id limit, for the new structure instance might differ from the value of the old structure instance. XCF communicates the user id-limit change through the structure state change event when the duplexing rebuild process is stopped to switch to the new (secondary) instance.

Considerations for Cache Structures during System-Managed Processing

When the system is attempting a system-managed process for a cache structure, additional coupling facility considerations apply. If there is no coupling facility with sufficient storage to copy all data that must be copied from the old structure to the new structure, the rebuild process will attempt to allocate the new structure big enough to copy all appropriate data other than registration data. See "Populating the New Structure" on page 311 for additional information about the implications of not copying registration data.

Connecting Users to the New Structure

After the new structure is allocated, for both rebuild and duplexing rebuild, the system attempts to connect (Attach phase) active connectors to the new structure. If there are no active connectors to the structure, this phase is skipped.

For each connector, the attachment information is the same in the old and new structure, and the local vector token associated with each user designates the same local vector. There is only one allocated vector for each connection to a duplexed structure. The local vector token is used to physically attach each active connector to the new structure.

If the system is unable to connect an active connector to the new structure, the rebuild or duplexing rebuild process stops. See "Handling Loss of Connectivity during System-Managed Processing" on page 318.

Failed-persistent connectors are attached to the new structure at the beginning of the Copy phase.

Populating the New Structure

After the system has connected all active users to the new structure, one or more systems in the sysplex cooperate to populate the new structure by copying data to it from the old structure. Both the old and the new structure must remain viable and accessible to the systems copying the data during this Copy Phase of the process.

The copy process is similar for both system-managed rebuild and duplexing rebuild. Major differences are noted in the following description of the data copied. The data copied includes:

- Cache structures
  - Registration of interest in cache data, with the following exceptions.

    Cache structures contain information about users' interest in data items stored in the structure. Users track the validity of their local copies of the cached data items by registering interest in particular data items. Under most circumstances, the rebuild process copies this registration for all entries in the structure, preserving the validity of all entries that are in the users' local caches. However, the rebuild process will not attempt to copy registration data, for any entries, if there is no suitable coupling facility with sufficient storage to contain both the registration data and all other structure data that must be copied, but at least one coupling facility has sufficient storage to copy all the non-registration data.

    Not having the registration data copied can have a short-term impact on application performance after the rebuild completes. In this case, the system indicates in the connectors' local cache vectors that all local copies of cached data items are not valid. Users must therefore refresh their local buffers, possibly by reading from the cache structure. Registrations will gradually be reestablished through normal cache reference.

    For **system-managed duplexing rebuild**, the following difference exists: Copying of cache structure registrations is never performed during system-managed duplexing rebuild. Therefore, when switching forward to simplex mode using the secondary cache structure, MVS will ensure that all users' local cache vectors are overindicated as not valid (thus all local cache buffers are invalidated), since there will be no valid registration information in the surviving simplex structure when the switchover occurs.

- All directory entries, if registrations are being copied. If registrations are not being copied, then only the directory entries accompanying changed or castout locked entries are copied.
- All changed data (if applicable), with adjunct data (if applicable). Changed data includes entries that are locked for castout. Unchanged data is not copied.
- Castout class and storage class definitions, including the assignment of entries to castout classes and storage classes, and including the storage class statistics for all storage classes.

  For **system-managed duplexing rebuild**, the following difference exists: Cache structure storage class statistics are not copied from the old structure to the new structure, nor are they duplexed on an ongoing basis between the two structure instances. Rather, each structure instance maintains its own storage class statistics independently. An IXLCACHE request against a duplexed structure to return the storage class statistics will always return only one set of storage class information, that of the primary (or old) structure. An IXLMG request against a duplexed structure, on the other hand, will return two sets of storage class information, one from each of the allocated instances of the structure. This allows monitor programs such as RMF to display actual storage class information showing how each of the structures in the duplex pair is being used over time.

  When the structure switches from duplex mode to simplex mode, keeping the secondary instance of the structure, there will be a discontinuity in the information returned to the connector by the IXLCACHE request to return storage class statistics. The request at that time will return the set of storage class information from the secondary structure, which is now a simplex structure. The Structure State Change Notification event presented when switching out of Duplex Established can serve as an external notification of this discontinuity.

- List structures
  - All list entries and associated data, with adjunct data (if applicable). All list entry attributes, such as names, keys, entry IDs, and version numbers, are preserved, as is the ordering of entries on all lists in the structure.
  - Lock table entries (if applicable (serialized lists))
  - Registered monitoring interest in lists, sublists, and event queues (if applicable), as well as the event queues themselves.
- Lock structures
  - Lock table entries. Resource status (contention status, global management, resource queues, for example) remains unchanged across the rebuild.
  - Record data (if applicable), including the entry IDs associated with the record data.

Once the new structure has been populated with the data from the old structure, and the system determines that the structure is viable, either the old structure can be deallocated and connected users can be notified of the new instance of the structure (for rebuild) or the old and new structure instances remain allocated and connected users enter the Duplex Established phase.

Completing or Continuing the System-Managed Process

When the system-managed process commits to using the new structure, connectors receive the Structure State Change event. The purpose of this event is to alert connected users to any structure characteristics that might have changed during the course of the system-managed process.

- For system-managed rebuild, EEPLSTRSTATECHANGEINFO contains information about the coupling facility in which the new structure has been rebuilt and specifies the structure's physical version numbers.
- For system-managed duplexing rebuild, the new “composite” attributes of the duplexed pair of structures are returned in IXLYEEPL.
  - VOLATILITY

    If either structure is nonvolatile, the composite state is nonvolatile.
  - FAILURE-ISOLATION

If the connector is failure-isolated from either structure instance, the composite state is failure-isolated.

– PHYSICAL STRUCTURE VERSION NUMBERS

  There is no composite state of this attribute. The physical structure version numbers for both structure instances (primary and secondary) will be returned.

  See “Understanding the Structure Version Numbers” on page 252 for additional information about structure version numbers.

– LOGICAL STRUCTURE VERSION NUMBER

  There is no composite state of this attribute. The logical version number will be the same for the primary and secondary structure in a duplex pair.

– CFNAME

  There is no composite state of this attribute. The CFNAME is for the coupling facility where the primary instance of the structure is allocated.

– CFLEVEL

  When establishing duplexing, the secondary structure must be allocated in at least as high a CFLEVEL as that which has been previously reported back to any connector as the primary structure's CFLEVEL. However, the secondary structure's actual CFLEVEL may validly be lower than the actual CFLEVEL of the primary structure.

  Therefore, when a new connector connects to the duplexed structure, the composite CFLEVEL reported is the lower of the connector's requested CFLEVEL or the lower of the actual CFLEVELs for the primary or secondary structure.

– EXCLUSION LIST

  The exclusion list indicator is not applicable to a connect during the Duplex Established phase.

– DUPLEXING STATE INDICATORS

  There are two attributes that pertain to duplexed structures. The indicators are:

  - An indicator of whether the structure is simplex or duplexed.
  - If duplexed, an indication of whether the primary structure instance is duplex failure-isolated from the secondary.

  When connecting to a structure during the Duplex Established phase, the structure will be presented as duplexed, with the structures either being duplex failure-isolated from one another or not.

No response is required for the Structure State Change event. It simply provides an opportunity for connected users to evaluate the new structure's attributes based on the coupling facility containing the structure and take any action deemed appropriate.

To communicate the end of a system-managed rebuild or the beginning of the Duplex Established phase of system-managed duplexing rebuild, the system presents the Structure Available event to the event exits of the active connectors to the structure. No response is required for this event. Its purpose is to inform connectors that had previously quiesced their activity against the structure that they may now resume their coupling facility requests. For duplexing rebuild, operations which were originally intended to be sent to a single coupling facility structure while in simplex mode are now converted into duplex operations and sent to both structures.

Whether or not the structure's size or object counts were modified during the system-managed rebuild, Structure Alter Begin and Structure Alter End events are presented to the event exits of the active connectors to the structure if all connectors specified ALLOWALTER=YES.

Working with Structures in a Duplex Established Phase

Once a Duplex Established phase has been reached, a subset of all subsequent structure update operations will be transparently duplexed by the system. A Duplex Established phase will exist indefinitely until either a request is issued to stop duplexing (through the SETXCF command or IXLREBLD macro)

or as a result of a failure of the duplexing protocol. When duplexing is stopped, an indication of which structure instance should remain allocated is indicated.

During the Duplex Established phase, new users are allowed to connect to the structure, assuming that the connector allows system-managed processes. See "Handling Connection Changes During System-Managed Processing" on page 315.

Working with structures in the Async Duplex Established phase

In the Async Duplex Established phase, a structure is being asynchronously duplexed by the system. This is a specialized case of the Duplex Established phase that is used for system-managed synchronous duplexing. Most processing for the Async Duplex Established phase is the same as for the duplex established phase. However, when a coupling facility lock structure is in the async duplex established phase, special consideration must be made when using the structure record data. Connectors that specify IXLCONN ASYNCDUPLEX=YES must be prepared to work with structures in the Async Duplex Established phase.

When the structure has established asynchronous duplexing, XES reports the completion of requests when the update to the primary structure instance completes. Each update is assigned a unique ADUPREQSEQNUM (asynchronous duplexing request sequence number). The update to the secondary structure instance occurs asynchronously. It is this asynchronous background processing to update the secondary structure instance that can allow processing to achieve performance characteristics more similar to simplex than synchronous duplexing.

The system, on behalf of the connector, keeps track of pending or uncommitted updates to the asynchronously duplexed secondary structure. When duplexing failover to the secondary structure instance occurs, uncommitted updates that are made by active connections are applied to the remaining structure by the system. Uncommitted updates that are made by disconnecting/terminating or failed-persistent connections might not be applied.

The user must be prepared to lose uncommitted updates in the event of a simultaneous duplexing failover and connector failure. To prevent this loss of record data updates, the user must wait for the necessary updates to be committed (applied to the secondary structure instance). The IXLADUPX service can be used to achieve this using the ADUPREQSEQNUM provided by the connector's latest request. The ADUPREQSEQNUM is an ever-increasing value and using IXLADUPX for one ADUPREQSEQNUM value covers that value as well as all smaller ADUPREQSEQNUM values. Waiting for uncommitted updates to commit might be needed, for example, before a transaction is committed.

It is the user's responsibility to request the ADUPREQSEQNUM when it might be needed. Any request that might involve a record data operation (with the exception of just a read) is a candidate for providing an ADUPREQSEQNUM. It might be an IXLLOCK request that specifies or defaults to RDATA=WRITE, RDATA=REACQUIRE, or RDATA=DELETE, or an IXLLOCK PROCESSMULT that uses LRB_XRDATA, or an IXLRT request, or an IXLSYNCH request that uses NEPLORTACTION. Requesting an ADUPREQSEQNUM from an IXLLOCK invocation by specifying the ADUPREQSEQNUM or REQVERSION requires that the system support system-managed asynchronous duplexing. Macro IXCYQUAA defines the QuReqRfAsyncDuplex bit in the QuReqFeatures string that can be used to test for system-managed asynchronous duplexing support. Use IXCQUERY REQINFO=FEATURES to get the QuReqFeatures string. It can be assumed that systems at a z/OS level higher than V2R2 support system-managed asynchronous duplexing.

To achieve good asynchronous duplexing performance, a user should only use IXLADUPX to wait for updates to be applied to the secondary structure instance when necessary. For instance, it might not be necessary to wait for an IXLLOCK OBTAIN,RDATA=WRITE to complete in both structure instances when it will be followed by other similar requests before the transaction is committed. Not until the primary has all updates that are required to commit the transaction is it necessary to wait for those updates to be applied to the secondary structure. When record data is needed to recover updates to a shared resource, ensure that the record data updates are made to both structure instances before proceeding.

Stopping the System-Managed Rebuild Process

To stop a system-managed rebuild, use either the SETXCF STOP,REBUILD command or the IXLREBLD macro. Users can stop the rebuild process for a structure up until the time the Cleanup phase is entered. After that point, requests to stop the rebuild will fail.

When the stopping of a system-managed rebuild is complete, the system presents the Structure Available event to the event exits of all active connectors. Upon receipt of the Structure Available event, connectors who had quiesced activity against the structure can resume submitting coupling facility requests, and requests that were deferred during the system-managed process are redriven.

Stopping a System-Managed Duplexing Rebuild

To stop a system-managed duplexing rebuild, use either the SETXCF STOP,REBUILD,DUPLEX command or the IXLREBLD macro. The system might also stop the duplexing rebuild, for example, in response to a failure affecting one of the structure instances, or a change in the CFRM policy DUPLEX specification to DISABLED.

Users can stop the duplexing rebuild process assuming that it has not already been stopped. When a duplexing rebuild process is stopped to fall back to the old (primary) instance, the phases can be Copy Stop, Stop, or Quiesce for Stop. When a duplexing rebuild process is stopped to switch to the new (secondary) instance, the phases are Switch and Cleanup. Stopping a duplexing rebuild to switch to the new structure is only allowed from the Duplex Established phase. The system rejects requests to switch to the new structure during all other phases of the process with appropriate return and reason codes.

Handling Connection Changes During System-Managed Processing

During a system-managed process, the system does not allow new connections to the structure except during the Duplex Established phase of duplexing rebuild. However, existing connectors are allowed to disconnect from the structure. Failed connectors are handled as having disconnected from the structure.

New Connections

- Rebuild

  The system will fail an IXLCONN invocation for a new connection during system-managed rebuild with return code (X'0C'), reason code IXLRSNCODECONNPREVENTED (X'xxxx0C09').

- Duplexing Rebuild

  New connections (or reconnections) to a duplexed structure during the Duplex Established phase are permitted assuming the connector allows system-managed processes by specifying ALLOWAUTO=YES on IXLCONN. When this criterion is not met, the connect request while the structure is Duplex Established is rejected with reason code IXLRSNCODECONNPREVENTED (X'0C09').

  When the connect is permitted, connect processing will transparently attach the connector to both structure instances of the duplex pair, and initialize the connection so that all coupling facility operations that are subsequently performed by the new connector are duplexed appropriately.

  The connect request, when successful, will complete either with return code 0 when none of the “special conditions” for connect processing apply, or with reason code X'0407' (IXLRSNCODESPECIALCONN) when one or more of the defined “special conditions” do apply at connect time. These “special conditions” include:

  – Reconnected
  – User sync point in progress
  – Alter in progress

  Note that the “rebuild in progress” condition is used to inform the connection that it needs to participate in the process and since system-managed processing does not require connection participation, the condition does not apply.

If at the time of the connect request, the duplexed structure has no active connectors, and the primary and secondary structures are both inaccessible from all systems in the sysplex, the system will force the deallocation of both of the duplexed structure instances. The system will then retry the connect request to cause a new structure instance to be allocated in some accessible coupling facility. Once a new instance of the structure is allocated, it may become duplexed.

If the new connector does not have connectivity to both instances of the duplexed structure, the connect request will be processed as follows:

- The system will automatically initiate duplexing rebuild stop processing to revert to simplex mode, keeping either the primary structure instance or the secondary structure instance, whichever the connector has connectivity to. The duplexing stop processing will proceed asynchronously with respect to the additional connect request retry processing described below, and it will internally suppress any attempt to automatically reduplex the structure at the time that the duplexing rebuild stop/switch processing completes.
- The unsuccessful connect request will be internally deferred and retried for a period of time, trying to catch the structure after it has reverted to simplex mode.
- When the duplexing rebuild stop/switch processing completes, duplexing will not be automatically re-initiated at that time. This allows the structure to remain in simplex mode to accomodate the retried connect request.
- If the retried connect request to the structure in simplex mode is successful, and the connector still has connectivity to the surviving simplex structure at that time, the connect request succeeds. An attempt to reduplex the structure will be triggered by the successful connect request to try to reestablish duplexing in some other coupling facility to which all of the current connectors have connectivity.
- While stop/switch processing is in progress for the structure, if none of the retried connect requests is successful connecting to the structure in simplex mode, then the connect request will fail with an indicative return and reason code indicating that the connector cannot connect to the structure.

  However, subsequently when the structure eventually completes its stop or switch processing, an ENF 35 for structure availability will be issued. If the connector listens for ENF 35 signals, the connect request can be retried in response to that signal, and the connector can connect to the structure in simplex mode (and again, this successful connect will serve as a trigger to reduplex the structure). If the connector does not reattempt the connect in response to the ENF 35 signal, then the structure may remain unduplexed until such time as duplex enabled monitoring attempts to reestablish duplexing for the structure.

IXLYCONA Contents when Connecting during the Duplex Established Phase

The structure attribute information returned in IXLYCONA when connecting to a structure that is duplexed represents a “composite” attribute state for the duplexed structure.

- VOLATILITY

  If either structure is nonvolatile, the composite state is nonvolatile.
- FAILURE-ISOLATION

  If the connector is failure-isolated from either structure instance, the composite state is failure-isolated.
- PHYSICAL STRUCTURE VERSION NUMBERS

  There is no composite state of this attribute. The physical structure version numbers for both structure instances (primary and secondary) will be returned.

  See “Understanding the Structure Version Numbers” on page 252 for additional information about structure version numbers.
- LOGICAL STRUCTURE VERSION NUMBER

  There is no composite state of this attribute. The logical version number will be the same for the primary and secondary structure in a duplex pair.
- CFNAME

There is no composite state of this attribute. The CFNAME is for the coupling facility where the primary instance of the structure is allocated.

- CFLEVEL

  When establishing duplexing, the secondary structure must be allocated in at least as high a CFLEVEL as that which has been previously reported back to any connector as the primary structure's CFLEVEL. However, the secondary structure's actual CFLEVEL may validly be lower than the actual CFLEVEL of the primary structure.

  Therefore, when a new connector connects to the duplexed structure, the composite CFLEVEL reported is the lower of the connector's requested CFLEVEL or the lower of the actual CFLEVELs for the primary or secondary structure.

- EXCLUSION LIST

  The exclusion list indicator is not applicable to a connect during the Duplex Established phase.

- DUPLEXING STATE INDICATORS

  There are two attributes that pertain to duplexed structures. The indicators are:

  – An indicator of whether the structure is simplex or duplexed.
  – If duplexed, an indication of whether the primary structure instance is duplex failure-isolated form the secondary.

  When connecting to a structure during the Duplex Established phase, the structure will be presented as duplexed, with the structures either being duplex failure-isolated from one another or not.

When connecting to a structure in the Duplex Established phase of system-managed duplexing rebuild, the fields in IXLYCONA that pertain to connecting during user-managed duplexing rebuild do not apply. In particular, the CONAREBUILD and CONAREBUILDDUPLEX indications are off.

Disconnecting or Failed Connections

The system allows connectors to disconnect from a structure during any phase of the system-managed process.

- Rebuild

  During a system-managed rebuild, failure of a connector is similar to the disconnect of a connector. Although connectors do not actively participate in the rebuilding of the structure, peer connectors must do whatever recovery processing is appropriate for the failing connector. The surviving connectors can respond to the DISCFAILCONN event without impacting the progress of the rebuild.

  The system frees the user-related resources associated with both the old and the new structures. Remaining existing connections to the structure receive a Disconnected or Failed (DISCFAILCONN) event in their event exits. Processing for this event is as follows:

  – The remaining connections must confirm the DISCFAILCONN event before the disconnect can complete. When designing an application, consider the effect of requiring surviving connectors to perform any operation that requires structure access before confirming the DISCFAILCONN event. Access to the structure is quiesced during a system-managed rebuild, and therefore attempts to access the structure could cause completion of the disconnect to be delayed until the system-managed rebuild completes.
  – Even though structure access might be required to confirm the DISCFAILCONN event, surviving connectors must not attempt to defer confirmation of the event until after the system-managed rebuild completes. Instead, connectors should initiate recovery processing when they receive the DISCFAILCONN event, even if the Structure Temporarily Unavailable event has also been received.

    The following scenario describes why connectors should not defer peer recovery processing when they receive a Structure Temporarily Unavailable event. A connector could fail early enough during the system-managed rebuild that the system will have presented a Structure Temporarily Unavailable event to the failing connector, but before that connector was able to respond. The rebuild cannot continue until the system receives a response to the Structure Temporarily Unavailable event from

all connectors, including the failing one. The peer connectors must proceed with failure recovery because they cannot know whether the failing connector had responded to the Structure Temporarily Unavailable event before failing, and must assume that the failing connector did not respond.

- If the failing connector had not responded to the Structure Temporarily Unavailable event, the system-managed rebuild could not have proceeded to the point at which coupling facility requests for the structure would be quiesced. Any structure access required before confirmation of the DISCFAILCONN event would complete normally, assuming no other failures. Only after the peer connectors confirm the DISCFAILCONN event does the system implicitly confirm the Structure Temporarily Unavailable event on behalf of the failing connector. If the peer connectors had deferred their response to the DISCFAILCONN event, the system-managed rebuild would have hung.
- If the failing connector had responded to the Structure Temporarily Unavailable event, and all other connectors had responded as well, the system would defer until completion of the rebuild any structure requests submitted by peer connectors to perform recovery. Eventually the rebuild would complete, peer recovery would finish, and the disconnection of the failing connector would complete.

– If the disconnecting user is the last connection to a non-persistent structure (STRDISP=DELETE was specified on IXLCONN), the disconnect will cause deallocation of the structure and will stop the rebuild. However, if the disconnecting user is the last connection to a persistent structure, the system-managed rebuild will continue to completion with no active connectors.

- Duplexing Rebuild

  During a system-managed duplexing rebuild:

  – If all active connectors disconnect or fail while in the Duplex Established phase and the structure is persistent, the structure will remain allocated and remain in the Duplex Established phase.
  – If all active connectors disconnect or fail while the system is in the process of establishing duplexing and the structure is persistent, the duplexing rebuild process will continue, if possible, until the Duplex Established phase is reached. The structure will then remain allocated and remain in the Duplex Established phase.
  – If all active connectors disconnect or fail while in the process of switching or stopping out of the Duplex Established phase and the structure is persistent, the system will continue to switch or stop until the structure is no longer in system-managed duplexing rebuild. The simplex structure will remain allocated.
  – If all active connectors disconnect or fail during duplexing rebuild, there are no failed-persistent connectors, and the structure is non-persistent, the system deallocates both structure instances and the structure is no longer in duplexing rebuild.
  – If a connector disconnects or fails when a phase confirmation is expected from either the connector or XES on behalf of the connector, the confirmation is marked immediately as implicitly received for all phase confirmations with the exception of Startup. As in system-managed rebuild, it is necessary to provide the system-managed duplexing rebuild phase confirmation immediately rather than waiting until all peer connectors provide the confirmation to the disconnect event to avoid deadlock.

Handling Loss of Connectivity during System-Managed Processing

System-managed rebuild is designed primarily for use in a planned reconfiguration environment. It provides only limited capability for recovery for loss of connectivity.

System-managed duplexing rebuild is designed to handle loss of connectivity in a transparent way while two instances of the structure exist in the Duplex Established phase. Loss of connectivity prior to and subsequent to the Duplex Established phase are handled in a manner consistent with system-managed rebuild processing.

The following sections describe how a connector's loss of connectivity to a structure is handled during key points in the system-managed process:

- Before a system-managed rebuild or duplexing rebuild is initiated
- Before the system commits to the new structure
- After the system commits to the new structure
- During the stop of system-managed rebuild

Loss of Connectivity Before a System-Managed Process Is Initiated

System-managed rebuild and duplexing rebuild will not be initiated if, at the time of the start request, any active or terminating connector has lost connectivity to the target structure. (A terminating connector is one that has disconnected but whose peer connectors have not yet responded to the Disconnect event.)

Neither system-managed rebuild nor system-managed duplexing rebuild will be initiated in response to a loss of connectivity. An IXLREBLD request to start a system-managed process cannot specify STARTREASON=LOSSCONN, and the specification of REBUILDPERCENT in the CFRM policy is ignored.

Loss of Connectivity Before the System Commits to the New Structure

For system-managed rebuild, the presentation of the Structure State Change event (Cleanup phase) is the point at which the system-managed process commits to the new structure. When a loss of coupling facility connectivity occurs prior to this point, a system-managed rebuild will continue across the loss of connectivity to the old or the new structure only if the failure does not affect systems on which there are active connectors to the structure being rebuilt. A loss of connectivity might force the system to select other systems to carry out system-managed processing that was disrupted by the failure.

For system-managed duplexing rebuild, before the Duplex Established phase, the primary structure is the only viable copy of the structure. When active connections have lost connectivity to the primary structure, or when all systems have lost connectivity such that system-based processing cannot complete, duplexing rebuild will stop to fall back to the primary structure and then report a Loss of Connectivity event upon completion of the Rebuild Stop process.

To summarize:

- When a connector loses connectivity to the **old (primary) structure** before the system-managed rebuild commits to the new structure, or for duplexing rebuild, before the duplex established phase, the rebuild is stopped. After stop processing is complete, and after the Structure Available event is presented, LOSSCONN events are presented for the connectors that lost connectivity. The affected connectors must disconnect in response to this event.
- When a connector loses connectivity to the **new (secondary) structure** before the system-managed rebuild commits to the new structure, or for duplexing rebuild, before the duplex established phase, the rebuild is stopped. No LOSSCONN events are presented because the connectors knew nothing about the new structure and because the structure never actually came into use.

  When the stop processing completes, the system automatically may initiate duplexing rebuild for the structure if it is defined in the CFRM active policy as DUPLEX(ENABLED).

Loss of Connectivity After the System Commits to the New Structure

Once the system-managed rebuild has committed to the new structure (cleanup phase), the process cannot be stopped even if connectivity to the old or the new structure is lost.

Loss of connectivity during the Duplex Established phase is handled by the system internally so that the only externally visible effect is the Structure State Change Notification event that is presented when the structure falls out of duplexing into simplex mode.

Only losses of connectivity that affect systems with active connectors to the duplexed pair of structures will cause system-managed duplexing rebuild to transition into simplex mode. A loss of connectivity reported on a system that has no active connectors to a given structure will have no effect on the duplexed state of the structure.

For both system-managed rebuild and duplexing rebuild, if connectivity is lost to the old (primary) structure:

- The loss of connectivity is not reported to the affected connectors because the connectors cannot go back to the old structure. Deferred requests will be driven successfully to the secondary structure after the switch completes.
- For duplexing rebuild, the secondary structure is now a simplex structure.

If connectivity is lost to the new (secondary) structure, neither rebuild method can fall back to using the old (primary) structure.

- For system-managed rebuild, the loss of connectivity is reported for the affected connectors when the system-managed rebuild completes and after the Structure Available event is presented. The affected users must disconnect in response to the LOSSCONN event.
- For system-managed duplexing rebuild, a LOSSCONN event will be presented after the switch to secondary is complete. Deferred requests will be driven unsuccessfully to the secondary structure (now a simplex structure) after the switch completes; they will indicate loss of connectivity completion.

Loss of Connectivity During the Stop of System-Managed Processing

During the stop of a system-managed process:

- When a connector loses connectivity to the **old structure** while a system-managed rebuild is being stopped, or during a duplexing rebuild stop to use the secondary structure, the loss of connectivity is reported to each connector after the Structure Available event. The affected connectors must disconnect in response to the LOSSCONN event.
- When a connector loses connectivity to the **new structure** while a system-managed rebuild is being stopped, or during a duplexing rebuild stop to use the primary structure, the loss of connectivity is not reported to the connector because all connectors are reverting to the old structure. Deferred requests will be driven to the primary structure after the stop completes.

Handling structure failure during system-managed processes

How the system handles structure failure during a system-managed process depends on when in the process the failure occurs and which of the structures failed.

- Failure of the Old Structure

  If the system has not yet committed to the new structure (before the Cleanup phase), or, for duplexing rebuild, before the Duplexed Established phase, failure of the old structure is reported to active connectors with the Structure Available event followed by the STRFAILURE event. Users should respond to this event by disconnecting from the structure with REASON=FAILURE.

  For both rebuild and duplexing rebuild, once the system has committed to the new structure, failure of the old structure is irrelevant. No STRFAILURE events are reported to any active connectors.

- Failure of the New Structure

  For rebuild, once the system has committed to the new structure (Cleanup phase), failure of the new structure is reported to active connectors with the Structure Available event followed by the STRFAILURE event. Users should respond to this event by disconnecting from the structure with REASON=FAILURE. If the system has not yet committed to the new structure (before the Cleanup phase), active connectors receive the Structure Available event to indicate that the old structure can be used.

  Similarly, for duplexing rebuild, once the system has committed to the new structure (Switch phase), structure failure is reported to connectors in the same manner.

- Structure Failure During a Duplex Established Phase

  To handle structure failures in a manner that is transparent to the connector, XES temporarily quiesces access to the structure and captures requests that are already in progress. Once duplexing rebuild has fallen out of duplexing back to simplex mode on the unaffected structure, the system can redrive these captured or quiesced requests. Connectors are made aware of the transition to simplex mode through the Structure State Change Notification event.

- Structure Failure During Switch or Stop of Duplexing Rebuild

  When switching to the secondary (new) structure:

  – If the primary structure fails, no structure failure event is presented. The system drives deferred requests to the secondary structure (now a simplex structure) after the switch completes.
  – If the secondary structure fails, a structure failure event is presented after the switch to secondary is complete. The system will drive deferred requests to the secondary structure after the switch completes; the requests will be unsuccessful and will indicate structure failure.

  When stopping to the primary (old) structure:

  – If the primary structure fails, a structure failure event is presented after the stop to primary is complete. The system will drive deferred requests to the primary structure after the stop completes; the requests will be unsuccessful and will indicate structure failure.
  – If the secondary structure fails, no structure failure event is presented. The system drives deferred requests to the primary structure (now a simplex structure) after the switch completes.

Dumping Considerations during System-Managed Processes

The structure instances contained in an SVC dump taken during a system-managed process depend on whichever instance or instances of the structure are allocated at the time of the dump request.

- If the new (or secondary) structure has not yet been allocated, the dump contains only the old structure.
- If the new structure has been allocated and the old structure has not yet been deallocated, (or, for duplexing rebuild, if the structure is in the Duplex Established phase), the dump contains both instances of the structure.
- If the old structure has been deallocated, the dump contains only the new structure.

Structure dumps are associated with a physical structure instance. If a structure dump exists for a structure, and then the structure is rebuilt or duplexed, the structure dump remains associated with the old (primary) structure only and is not copied as part of the system-managed processing. When the rebuild completes or should a switch occur, the system will keep the old (primary) instance because of the associated structure dump. When the structure dump either is written out to a dump data set or is explicitly forced, the old (primary) instance will be deallocated.

For system-managed rebuild, if the dump serialization interferes with coupling facility operations generated by the system when the system is allocating the new structure, connecting users to it, or performing other system processing in support of the rebuild, the system will stop the rebuild. However, if dump serialization is held during the copying of the structure data from the old to the new structure, system-managed rebuild will continue.

For system-managed duplexing rebuild, structure dumping is allowed during any phase of system-managed duplexing rebuild. However, during the Allocate and Attach phases, the system cannot tolerate structure dump serialization and the system-managed duplexing rebuild process will be stopped.

Some comparisons between user-managed and system-managed duplexing rebuild

- FORCE

  For both user-managed and system-managed duplexing rebuild, FORCE is allowed only during a Duplex Established phase when switch processing has not been requested. Otherwise, the FORCE request will fail.

- Volatility

  User awareness of changes in the volatility state differs between user-managed and system-managed duplexing rebuild. In user-managed duplexing rebuild, the system presents the Volatility State Change event to inform the user of changes to either of the structure instances. In system-managed duplexing rebuild, the system presents the Volatility State Change event whenever the “composite” volatility state

changes. For example, a duplexed structure is considered to be nonvolatile if either instance of the structure in nonvolatile. If a volatility state change affects one of the coupling facilities in which one of the structure instances is allocated, the system will report the Volatility State Change event to users.

A structure that is transitioning into and out of a Duplex Established phase could also result in a volatility state transition. Volatility state transitions caused by starting or stopping duplexing rebuild are reported to users through the Structure State Change Notification event.

- Out of Synch Condition

  An “out of synch” condition applies to system-managed duplexing rebuild only. Specifically, this condition applies to synchronous duplexing. When a command that has been duplexed detects an “out of synch” condition, the integrity of the entire structure is at risk and therefore both instances of the structure are considered to have failed. The system-managed duplexing rebuild is stopped and a Structure Failure event is delivered.

- Cf-to-Cf Connectivity Lost

  Cf-To-Cf connectivity applies to system-managed duplexing rebuild only. During synchronous structure duplexing, this loss of connectivity is detected when a duplexed request fails. During asynchronous structure duplexing, the system-managed duplexing rebuild is stopped for this reason.

- Asynchronous Duplexing Processing Stalled

  A "stalled" condition applies to system-managed asynchronous structure duplexing only. When the system detects that the secondary structure is no longer being updated by the primary structure coupling facility, the system-managed duplexing rebuild will be stopped.

- Duplexing Processing Inactive

  An inactive condition applies to system-managed duplexing rebuild only. During synchronous structure duplexing, duplexing inactive is detected when a duplexed request fails. During asynchronous structure duplexing, the system-managed duplexing rebuild is stopped for this reason.

Summary of System-Managed Rebuild Processing

Figure 29 on page 323 illustrates the sequence of events during a system-managed rebuild.

The following list summarizes that process:

1. Rebuilding for a structure is initiated through SETXCF START,REBUILD or IXLREBLD REQUEST=START.
2. System reports Structure Temporarily Unavailable event to all active connector's event exits.
3. Connector responds to the STRTEMPUNAVAIL event with either IXLEERSP or IXLYEEPL.
4. When all responses are received, the system quiesces activity to the structure for access requests.

   Requests that are already in progress are completed. New requests are queued.
5. When all connectors are quiesced, the system allocates a new structure instance.
6. The system connects all active users of the old structure to the new structure.
7. One system attaches all failed-persistent users to the new structure.
8. The system copies structure objects from the old structure to the new structure.
9. The system reports the Structure State Change Notification event to all connectors and deallocates the old structure instance.
10. The system unquiesces access to the structure and drives all queued requests against the new structure.
11. The system delivers the Structure Available event to all connectors.
12. The system delivers the Alter Begin and Alter End events to all connectors.

System-Managed Rebuild Timeline

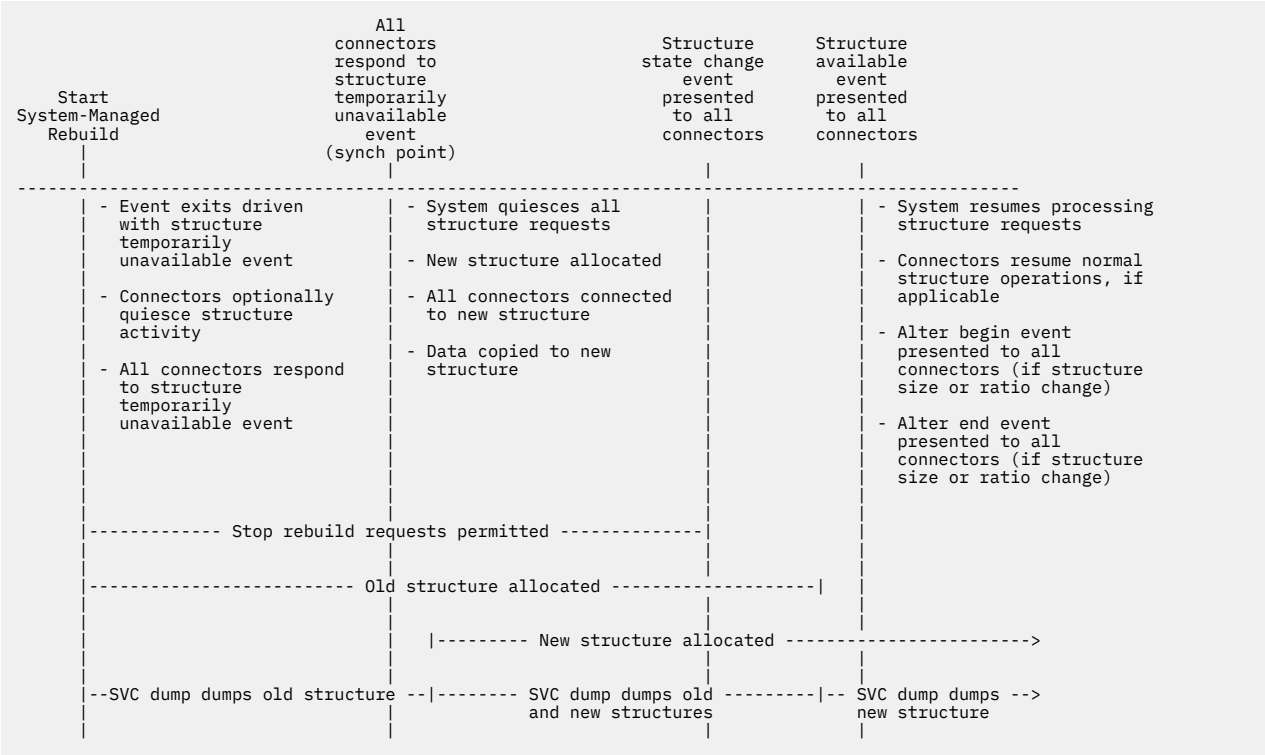


Figure 29. Sequence of Events During System-Managed Rebuild

Summary of System-Managed Duplexing Rebuild Processing

Figure 30 on page 324 illustrates the sequence of events during a system-managed duplexing rebuild.

The following list summarizes that process:

1. Duplexing rebuild for a structure is initiated through SETXCF START,REBUILD,DUPLEX or IXLREBLD REQUEST=STARTDUPLEX, of internally by MVS.
2. The system reports Structure Temporarily Unavailable event to all connectors' event exits.
3. Connector responds to the STRTEMPUNAVAIL event with either IXLEERSP or IXLYEEPL.
4. When all responses are received, the system quiesces activity to the structure for access requests. Requests that are already in progress are completed. New requests are queued.
5. When all connectors are quiesced, the system allocates a secondary instance of the structure, attaches the active connectors to the secondary structure, and copies structure objects from the old to the new structure.
6. The system unquiesces access to the structure and drives all queued requests in duplex mode.
7. The system reports the Structure State Change Notification event to all connectors.
8. The system delivers the Structure Available event to all connectors.
9. The system issues ENF 35 for the structure.
10. Connectors continue in duplexed mode until a request is received to stop the duplexing and either fall back to the old structure or forward complete (switch) to the new structure.
11. If a fallback to the primary structure or a switch to the secondary structure is requested, the system quiesces use of the structure and defers future requests.
12. The system reports the Structure State Change Notification event to all connectors.
13. The system deactivates duplexing for the appropriate structure and deallocates the other structure.
14. The system unquiesces use of the structure and drives all deferred requests in simplex mode.

15. The system issues ENF 35 for the structure.

System-Managed Duplexing Rebuild Timeline

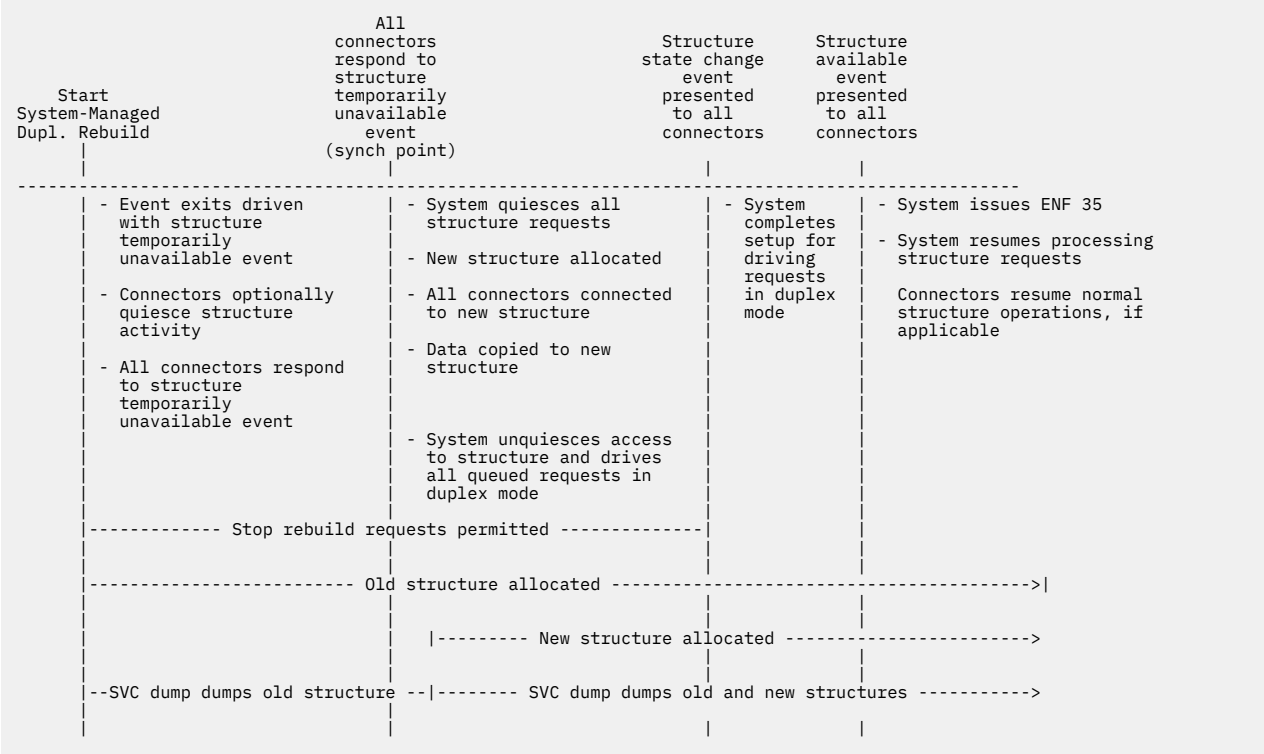


Figure 30. Sequence of Events During System-Managed Duplexing Rebuild

Altering a Coupling Facility Structure

The structure alter function allows a coupling facility structure to be reconfigured with minimal disruption to connectors. Structure alter provides for the expansion or contraction of the size of a structure, the reapportionment of the entry-to-element ratio of the structure's storage, and the alteration of the percentage of structure storage set aside for event monitor controls (EMCs). The structure being altered is not deallocated and then re-allocated; it is altered in place.

All functions of structure alter can be initiated by an authorized program; structure size only can be initiated by an operator command. Both methods are supported on an SP 5.2 and higher system. It is highly recommended that applications support the structure alter protocol, especially those applications whose only rebuild method supported is system-managed (IXLCONN ALLOWREBLD=NO,ALLOWAUTO=YES).

Note: You can use the SETXCF MODIFIY command to disable alter processing for the structure.

Overview of Structure Alter Processing

Structure alter is a non-disruptive process to connectors to the structure being altered. The structure alter function requires the following combination of hardware and software support:

- The structure alter function requires a coupling facility with CFLEVEL=1 or higher. You must add one or more coupling facilities with CFLEVEL=1 or higher to the structure's preference list in the CFRM policy. This enables XES to allocate the structure in a coupling facility that supports structure alter.

  Note that event monitor controls are supported only for keyed list structures allocated in a coupling facility of CFLEVEL=3 or higher. For this type of structure:

  – When allocated in a coupling facility of CFLEVEL=3, only the size and the entry-to-element ratio can be altered.

  - When allocated in a coupling facility of CFLEVEL=4 or higher, the size, entry-to-element ratio, and the percentage of EMC storage can be altered.
- The structure alter function requires that MVS SP 5.2 or higher be running on all systems on which applications plan to use the function. All connectors to the structure must have specified ALLOWALTER=YES on the IXLCONN macro.

Limiting the Scope of Structure Alter

Connectors that specify ALLOWALTER=YES have the ability to limit the changes that can be made to a structure during structure alter processing.

- The RATIO keyword indicates whether the connector allows the entry-to-element ratio to be changed. Specifying RATIO=NO prevents changes to the entry-to-element ratio and also prevents the structure from being contracted to less than its minimum size. The entry-to-element ratio could change if the structure were reduced in size to less than its minimum.

  The RATIO keyword also indicates whether the connector allows the percentage of EMC storage to be changed. Specifying RATIO=NO prevents changes to the percentage of EMC storage.

  See "Specifying the Structure Size" on page 212 for information about how the size of a structure is determined.
- The MINENTRY and MINELEMENT keywords allow the connector to specify minimum threshold levels for entries and elements allocated in the structure. The MINEMC keyword allows the connector to specify the minimum threshold level of EMCs allocated in the structure. The values specified for MINENTRY, MINELEMENT, and MINEMC are percentage values, used by the system when a structure alter is initiated either to contract a structure or to reapportion the structure's ratio of entries to elements or its percentage of storage available for allocation of EMCs.
  - The values specified for MINENTRY and MINELEMENT are percentage values of entries and elements that are to be available at the completion of structure alter processing — for a list structure, the percentage of "currently in-use" entries and elements and for a cache structure, the percentage of "currently in-use and changed" entries and elements. The connector thus is able to maintain a buffer of available entries and elements for subsequent use.
  - The value specified for MINEMC is a percentage value of EMCs that are to be available at the completion of structure alter processing — for a keyed list structure, the percentage of "currently-in-use" EMCs. The connector thus is able to maintain a buffer of available EMCs for subsequent use.

  If the reapportionment of the structure would decrease the amount of storage available for entries, elements, or EMCs, the MINENTRY, MINELEMENY, and MINEMC values are used to prevent the alter from making the structure unusable to the application. If the alter processing tries to contract or reapportion more free space than specified by these percentage values, the system stops the alter.

  The system does not use the MINENTRY, MINELEMENT, or MINEMC values during requests to expand the structure or during reapportion requests that will increase the amount of storage available for the particular set of entries, elements, or EMCs.

Determining a Structure's Composite Values

Before performing a structure alter, XES determines "composite values" for a structure, based on structure attributes specified at connect time. The composite values provide a limit on how the system can alter the structure. Each connection can specify an ALLOWALTER, RATIO, MINENTRY, MINELEMENT, and MINEMC value, and each connection could specify different values. XES merges the values from each connection. XES determines the most restrictive requirements and uses these as the composite values to limit the changes to the structure for the structure alter request.

For instance:

- If any connection specifies ALLOWALTER=NO, the alter request is rejected.
- If any connection specifies RATIO=NO, a request to change the entry-to-element ratio or percentage of EMC storage is rejected.

- For MINENTRY, MINELEMENT, and MINEMC values, the most restrictive value is the highest percentage of in-use elements that must be available upon completion of structure alter. Therefore, the highest percentage value set by any connection is used to limit the structure alter request.

Changing the Structure Size

The structure alter function can expand or contract a structure within the range of its maximum and minimum size. The maximum size (MSS) of a structure is set at the time of structure allocation and remains constant as long as this instance is allocated. The structure allocation algorithm determines the MSS based on whether the structure is allocated by counts or by size/ratios. The actual structure size may be less than or equal to the MSS value. The minimum size is set by the coupling facility, and is determined by calculating the minimum amount of space required for the coupling facility to allocate the structure with the specified entry-to-element ratio. See “Specifying the Structure Size” on page 212 for a discussion about the maximum and minimum structure size.

Structure alter cannot change the maximum size of a structure. However, the minimum structure size could change if you reapportion the structure with an entry-to-element ratio that is different from the previous ratio or with an EMC storage percentage that is different from the previous percentage.

For example, when allocating the structure, the coupling facility determines the marginal structure size — the true minimum size at which the structure can be allocated. The marginal structure size is less than the minimum structure size and does not take into consideration the entry-to-element ratio. The system will process a request to alter a structure to the marginal structure size value only if a change to the structure's entry-to-element ratio is permitted. When such a structure alter occurs, the minimum structure size value also changes to the altered structure size.

Note: When contracting a structure, the composite MINENTRY, MINELEMENT and/or MINEMC value, if applicable, might limit the extent to which the structure can be contracted.

Changing the Structure Entry-to-Element Ratio

The number of entries and elements in a structure is a function of the attributes specified at connect time. The structure alter function can change the ratio of entries to elements in a structure. The request to change the entry-to-element ratio is limited by the composite RATIO, MINENTRY, and MINELEMENT values.

A request to alter the size of a structure (expand or contract) might also result in a change to the entry-to-element ratio of the structure. When a structure is allocated initially, the system determines the target entry-to-element ratio and attempts to allocate the number of entries and elements accordingly. If the target ratio cannot be satisfied, the system maintains the current count of entries and elements along with the original target ratio. When an IXLALTER request is received to expand or contract the structure size, the system checks the current entry and element counts to determine if the current entry-to-element ratio equals the original target ratio. If the current ratio is not the same as the target ratio, the system uses the target ratio to calculate the target entry and element counts that will be used when altering the structure's size.

Changing the Percentage of Event Monitor Controls

The amount of available list storage set aside for EMCs in a keyed list structure is a function of the percentage specified at connect time. The structure alter function can change that percentage for keyed list structures allocated in a coupling facility with CFLEVEL=4 or higher. The request to change the percentage is limited by the composite RATIO and MINEMC values.

Altering a duplexed structure

A structure in a Duplex Established phase of a user-managed duplexing rebuild or system-managed duplexing rebuild can be altered.

If a structure is in user-managed duplexing rebuild processing when the IXLALTER request is received, the alter is applied to both instances serially. The primary (old) structure is altered first. When the alter for the old structure instance is complete, the secondary (new) structure is altered. The connectors are

notified that the alter has started and completed for each structure instance individually through their event exits.

If a structure is in system-managed duplexing rebuild processing when the IXLALTER request is received, the alter is applied to both instances serially. Typically, the primary (old) structure is altered first, followed by the alter of the secondary (new) structure. However, the connectors are notified that the alter has started and completed only once, treating the alter operation as if it were performed against one virtual structure.

Structure Type Considerations

The IXLALTER service imposes the following restrictions for each type of structure:

- Cache Structure

  A cache structure originally allocated with data can be altered to a structure without data and back again. If the cache structure was originally allocated without data, then structure alter cannot be used to create data.

  The system rejects a request to change the EMC percentage for a cache structure and sets the EEPLALTERENDREQEXCEPTION bit in the Structure Alter End event information.

  When a cache structure is altered, active reclaim vectors for the structure are deactivated. The system resumes using the default reclaim algorithm that was in effect at the start of the alter process for all storage classes.

  During the alter process, the system rejects any attempt to activate a reclaim vector for the structure with IXLCACHE SET_RECLVCTR.

  When the alter process completes for the structure, connectors can once again activate a reclaim vector. Connectors must explicitly activate any reclaim vectors that were deactivated at the time the alter process began; the reclaim vectors are not reactivated automatically. When reactivating the reclaim vectors, take into account the current entry and element counts, which might have been altered.

- List Structure

  A list structure originally allocated with data cannot be altered to a structure without data. If the list structure was originally allocated without data, then structure alter cannot be used to create data.

  When a list structure is altered, the list limit for each list is adjusted to equal the total number of entries or elements in the structure at the completion of the alter process when the user has never set a list limit for the list.

  List limits that have been modified by a list services WRITE_LCONTROLS request are left untouched. It is the user's responsibility to set a new list limit for such lists when the alter process completes. The new list limit should take into account the current entry and element counts for the altered structure.

  A keyed list structure allocated in a coupling facility with CFLEVEL=4 or higher that was initially allocated without EMCs can be altered to a structure containing EMCs. If the list structure was originally allocated with EMCs, then structure alter can be used to change to a list structure with no EMCs. Structure alter cannot be used to change the presence of EMCs in a keyed list structure allocated in a coupling facility with CFLEVEL=3.

- Lock Structure

  For a lock structure, you can only change the size of the structure. The system rejects a request to change the ratio or EMC storage percentage and sets the EEPLALTERENDREQEXCEPTION bit in the Structure Alter End event information.

  If the lock structure was allocated initially with record data, changing the size of the structure either increases or decreases the number of record data elements in the structure. To change the number of lock table entries, you must use the structure rebuild service.

Starting the Structure Alter Process

You can initiate structure alter processing either by using the IXLALTER macro or by issuing the SETXCF START,ALTER command. The IXLALTER macro allows an authorized user to request a change to the structure's size, entry-to-element ratio, and percentage of storage allocated for EMCs. The SETXCF START,ALTER command allows the operator to request a change only to the structure's size. Recall, however, that a request to contract the structure's size might also affect the entry-to-element ratio and the percentage of EMC storage.

XES determines if structure alter is supported by the current set of connectors to the structure. XES accepts the alter request if:

- The structure to be altered is allocated in a coupling facility with the appropriate level (CFLEVEL=1 or higher for all structures, CFLEVEL=3 or higher for keyed list structures allocated with EMCs for which a change in size is requested, or CFLEVEL=4 for keyed list structures allocated with EMCs for which a change in the percentage of EMC storage is requested.
- The structure is not already in an alter process.
- The structure is not in a rebuild process, or in a user-managed duplexing rebuild process, or in the duplex established phase.
- The SETXCF MODIFY command was NOT used to disable alter for the structure. Starting CF structure alter processing for such structures is not permitted.
- The structure is persistent with no active or failed-persistent connectors.
- The structure has active or failed-persistent connectors, all of whom specified ALLOWALTER=YES.
- The structure is in the Duplex Established phase of user-managed duplexing rebuild or system-managed duplexing rebuild.
- The structure has no objects in storage-class memory and no augmented space other than the fixed augmented space in use.

Notifying Connectors of Structure Alter Initiation

If the request to START structure alter is valid, all active connectors to the structure are notified of the Structure Alter Begin event (EEPLALTERBEGIN) through their event exit. The connectors can examine the event exit parameter list (IXLYEEPL) to determine the requested target values, the ratio change indication, and the composite values for the minimum percentage of entries and/or elements and EMC storage percentage to be available. The information available in IXLYEEPL when the Structure Alter Begin event is presented is mapped by EEPLALTERBEGININFO.

If the request is to change the size of the structure, EEPLALTERSIZE contains the requested size. The connector can compare this size with that returned in IXLYCONA at connect time to determine how the size is to be altered. The connector might want to free up any in-use or in-use and changed structure resources to accommodate the alter process.

If the request is to alter a structure in the duplexing rebuild process, EEPLSTRSTATEREBUILDDUPLEX is a flag that indicates duplexing rebuild is in progress. EEPLSTRUCTUREVERSION is the structure version of the instance of the structure for which the Alter Begin event is issued. EEPLALTERBEGINDUPREBOLD or EEPLALTERBEGINDUPREBNEW identify the old or new instance of the structure that is being altered.

Connectors to the structure are not required to respond to the structure alter event.

Initiating Alter for a Structure with No Connectors

If there are no active, failed-persistent, or disconnecting connectors, there are no RATIO, MINENTRY, MINELEMENT, and MINEMC values from which to calculate a composite value. Thus, a operator request to alter a structure with no connectors might result in the structure having no available entries or elements or might provide a ratio different from what the structure's connectors can tolerate. XES allows this type of structure alter to be performed so that an installation can adjust a persistent structure when it is not being used.

When the persistent structure is needed again, its connectors receive current structure information in the connect answer area. A connector at that point could initiate structure alter to change the structure's size and/or apportionment as appropriate.

Initiating Alter for a Structure with Failed-Persistent Connectors

A structure with failed-persistent connectors can be altered, assuming that the failed-persistent connector had specified ALLOWALTER=YES. The RATIO, MINENTRY, MINELEMENT, and MINEMC specifications from each failed-persistent connector contribute to the determination of the composite values of the structure.

Completing the Structure Alter Process

The alter processing continues until one of the following situations occurs:

- The target size, ratio, or EMC storage percentage are satisfied (either completely or to the extent possible based on current use of the coupling facility's resources).
- XES receives a request to stop the structure alter process.
- One of the events that causes automatic alter termination occurs. See “Requesting that Structure Alter Be Stopped” on page 329 for more information.

When structure alter completes, XES sets the structure size, the entry-to-element ratio, and/or the EMC storage percentage to that achieved by the alter process at the time the process stopped. Connectors access this information in the event exit parameter list (EEPL).

Notifying Connectors of Structure Alter Completion

At the completion of the structure alter operation, active connectors again are notified through their event exit. The event is the Structure Alter End event (EEPLALTEREND). The EEPL contains the status of the structure resulting from the alter processing. The connectors should examine the EEPL to determine:

- Whether the alter request was able to complete
- Whether the targets were met
- The achieved structure size, entry and element counts, and EMC counts for the structure
- The minimum structure size, which might have changed from the structure's initial allocation.

This information is mapped by EEPLALTERENDINFO.

If the Structure Alter End event indicates the completion of a structure alter for a structure in the duplexing rebuild process, EEPLALTERENDINFO contains information identifying the instance of the structure that was altered (EEPLALTERENDDUPREBUILDOLD or EEPLALTERENDDUPREBUILDNEW). When the old structure has completed the alter process, the system automatically initiates the alter process for the new structure. Connectors therefore will receive two sets of Structure Alter events for the duplexed structure, an EEPLALTERBEGIN and EEPLALTEREND event for the old structure which is altered first, followed by an EEPLALTERBEGIN and EEPLALTEREND event for the new structure. Connectors can use this information to determine if they need to change their use of the structure now that it has been altered. For example, prior to the structure being altered, the connector might have set list controls, such as list limits for entries and elements, for a particular list based on the currently allocated structure. After the structure is altered, the total number of entries and elements might have changed, and the connector might need to reset the list limits accordingly.

Requesting that Structure Alter Be Stopped

Structure alter processing can be stopped by an authorized program invoking IXLALTER or by an operator issuing the SETXCF STOP,ALTER command.

Structure alter processing is automatically stopped when:

- A structure failure occurs
- A nonpersistent structure is deallocated

- XES receives a request to rebuild the structure
- An “old” structure during the rebuild process is deallocated
- All SP 5.2 systems in the sysplex lose connectivity to the coupling facility
- All SP 5.2 systems in the sysplex fail
- Structure objects are migrated into storage-class memory

In addition, alter processing for one instance of a duplexed structure is stopped if the duplexing rebuild is stopped such that the instance of the altered structure will deallocate at the end of the duplexing rebuild process. The alter of the other structure instance, if it is not already altered, will start when the alter of the first instance completes its stop processing. See “Alter/Duplexing Coordination” on page 331.

The following scenarios describe system actions that occur when structure alter processing cannot complete:

Structure Failure

If structure failure occurs while the structure is being altered, XES stops the alter process. Connectors to the structure are presented with the EEPLALTEREND event with the EEPLALTERENDSTRFAIL bit set to indicate that structure alter did not complete due to structure failure.

If the structure failure occurs during a duplexing rebuild, the alter of the remaining structure continues.

Connection Termination

If the last connector to a persistent structure disconnects during structure alter, XES continues processing the structure alter request. If the last connector to a non-persistent structure disconnects during structure alter, XES stops the structure alter when the structure is deallocated.

If the last connector to a structure in duplexing rebuild disconnects during structure alter, the alter process continues for both the old and new instances of the structure.

System Failure

If a system failure occurs while a structure is being altered, XES continues the alter process as long as:

- There is at least one SP 5.2 system that has connectivity to the coupling facility containing the structure
- The structure remains allocated.

If there are no SP 5.2 systems that have connectivity to the coupling facility, the state of the structure is set to “alter in progress”. Active connections to the structure are presented with the EEPLALTEREND event with the EEPLALTERENDLOSSCONN bit set to indicate that the alter did not complete due to loss of connectivity.

Structure alter stops when the first SP 5.2 system gains connectivity to the coupling facility where the structure is allocated.

Changes in Connectivity

If connectivity is lost to the coupling facility whose structure is being altered, the alter process continues as long as there is one SP 5.2 system that has retained connectivity to that coupling facility and the structure remains allocated. If all SP 5.2 systems have lost connectivity, the structure is placed in an “alter in progress” state. Each connector to the structure is presented with the EEPLALTEREND event with the EEPLALTERENDLOSSCONN bit set to indicate that the alter did not complete normally.

When the first SP 5.2 system in the sysplex regains connectivity to the coupling facility, the structure alter that was in progress is stopped.

Alter/Rebuild Coordination

When the system receives a request to rebuild a structure during a structure alter operation, the success of the structure alter and the timing of the EEPLALTEREND event presentation to connectors depend on the following:

- If the structure rebuild request is received while a structure alter is in progress, the structure alter stops when the system performing the alter recognizes that rebuild is in progress. Whether the EEPLALTEREND event is presented depends on when in the rebuild process the recognition occurs.
- If the structure alter is stopped before the rebuild cleanup phase is entered, the EEPLALTEREND event is presented to the structure connectors. The EEPL contains the EEPLALTERENDREBLD bit, which is set to indicate that the alter ended due to a rebuild request. Depending on the timing, it is possible for the structure alter to complete before recognizing the request to stop due to a rebuild and the connectors will see only the EEPLALTEREND event indicating that the alter is complete.
- If the structure alter is not stopped before the rebuild cleanup phase, connectors will not be presented with an EEPLALTEREND event in the event exit pertaining to the “old” structure.
- If a rebuild stop is processed, XES completes the stop for structure alter that was requested when rebuild was started. The connectors receive the EEPLALTEREND event with the EEPLALTERENDREBLD bit set on to indicate that the alter ended due to a rebuild.

Structure Objects in Storage-Class Memory

If structure objects are migrated into storage-class memory while a structure alter is in progress, the coupling facility terminates the alter processing. Connectors to the structure are presented with the EEPLALTEREND event.

Altering a user-managed duplexed structure

A request to alter a duplexed structure is handled serially, with the old structure completing the alter process before the alter of the new structure is started. The same set of parameters for the alter is applied to each instance of the structure. However, conditions in the coupling facility, such as the coupling facilities might be of different CFLEVELs or might not have the same amount of available free storage, might not allow both alter operations to complete with the same results. For example, if both instances of the structure are altered to different sizes, it is the responsibility of the connector to manage the difference — in this case, to issue another alter request to resize the larger structure to the size of the smaller structure. Completion of the second duplexing alter request would take less time than the first duplexing alter request because the smaller instance of the structure would already be at the requested size.

Alter/Duplexing Coordination

If a request to stop the duplexing process is received while structure alter is in progress, the following general rule applies: A structure alter is automatically stopped for the structure which will be deallocated. Structure alter continues, if not already started or completed, for that structure which will remain after the stop or switch. For example, a request to stop the duplexing process is received while structure alter is in progress:

- If the old structure is being altered and a Stop Duplex request is received to keep the old structure:
  - The system issues a request to stop the alter of the new structure (which is not yet in progress)
  - All connectors receive a RebuildStop event for the new structure and respond to it
  - The new structure is deallocated
  - Structure alter completes for the old structure
  - Structure AlterBegin and AlterEnd events are not delivered for the new structure, which has now been deallocated.
- If the old structure is being altered and a Stop Duplex request is received to keep the new structure:
  - The system issues a request to stop the alter of the old structure, and proceeds to initiate alter of the new structure
  - All connectors receive a RebuildStop event for the old structure and respond to it
  - The old structure is deallocated
  - Structure alter completes for the new structure

  - Structure AlterBegin and AlterEnd events are delivered for both the new and old structures.
- If the new structure is being altered and a Stop Duplex request is received to keep the old structure:
  - The system issues a request to stop the alter of the new structure
  - All connectors receive the RebuildStop event and respond to it
  - The new structure is deallocated
  - AlterBegin and AlterEnd events are delivered for the new structure, which has now been deallocated.
- If the new structure is being altered and a Stop Duplex request is received to keep the new structure:
  - All connectors receive the RebuildStop event and respond to it
  - The old structure is deallocated
  - AlterBegin and AlterEnd events are delivered for the new structure.

Information returned in IXLYEEPL

The following IXLYEEPL flags are set during a structure alter while a structure is in a Duplex Established phase:

- EEPLSTRUCTUREVERSION — Specifies the structure version of the structure being altered.
- EEPLSTRSTATESTRVERSIONFLAG — Identifies whether the old or new instance of the structure is being altered.
- EEPLALTERBEGINDUPREBLDOLD — Specifies that the values presented in EEPLALTERBEGININFO are for the old structure.
- EEPLALTERENDDUPREBLDOLD — Specifies that the values presented in EEPLALTERENDINFO are for the old structure.
- EEPLALTERBEGINDUPREBLDNEW — Specifies that the values presented in EEPLALTERBEGININFO are for the new structure.
- EEPLALTERENDDUPREBLDNEW — Specifies that the values presented in EEPLALTERENDINFO are for the new structure.

Note: If an AlterEnd event is delivered, EEPLSTRUCTUREVERSION will always accurately identify the structure instance for which alter has completed. However, the EEPLALTERENDDUPREBLDOLD and EEPLALTERENDDUPREBLDNEW flags cannot be set unless both structure instances are currently allocated. If the flags are not set, then one of the structure instances has been deallocated. EEPLSTRUCTUREVERSION indicates whether the AlterEnd event is for the structure that remains or the one that was recently deallocated.

Detecting ENF Code 35 for Structure Alter

At the completion of structure alter processing, the system issues at least one ENF signal to indicate that structure alter processing has ended.

- If the structure alter resulted in a structure with a smaller structure size (contract request), then the system issues a generic ENF event code 35. The ENF signal parameter list does not contain the structure name, but does imply that additional coupling facility resources might be available because the structure size is decreased. Connectors who have been unable to connect to a structure can listen for this event and then attempt the connect request again.
- The system also issues an ENF event code 35 when structure alter processing is complete. The signal does not imply that additional coupling facility resources are available, but simply indicates that a structure alter has completed. The ENF signal parameter list presented to an ENF listen exit will contain the name of the structure. Connectors who have been unable to connect to a structure can listen for this event.

Connectors wishing to connect to a structure can use the IXCQUERY macro to verify that the structure alter actually has completed because ENF 35 is issued for several different events.

Handling New Connections during Alter Processing

Structure alter processing is supported only by SP 5.2, and is available based on connection attributes specified on IXLCONN. When a connector tries to connect to a structure that is being altered, the following actions occur depending on whether the connector is on a 5.1 or 5.2 system:

- SP 5.1 connector

  XES rejects the IXLCONN request with return code IXLRETCODEPARMERROR, reason code IXLRSNCODECONNPREVENTED.

- SP 5.2 connector

  If the SP 5.2 connector does not allow structure alter (ALLOWALTER=NO), XES rejects the IXLCONN request with return code IXLRETCODEPARMERROR, reason code IXLRSNCODESTRALTERNOTALLOW.

  If the SP 5.2 connector allows structure alter, XES compares the threshold values specified on IXLCONN (RATIO, MINENTRY, and MINELEMENT) with the composite limits for the structure being altered:

  – Connections with more restrictive limits than the current composite are rejected with return code IXLRETCODEPARMERROR, reason code IXLRSNCODESTRALTERRESTRICT.

  – Connections with the same or less restrictive limits than the current composite are connected to the structure. The connector must examine the connect answer area to determine whether structure alter is in progress (CONAALTERINPROGRESS) and if so, check CONAALTERINFO for target and composite information.

Responding to Connection Events

Each connector to a coupling facility structure must specify the address of an event exit. MVS communicates information about certain structure and connection events to the event exit. These events include information about new and existing connections to a structure (including failed-persistent connections), operations to rebuild a structure, and changes to the structure that can affect processing (like loss of connectivity and other events). The event exit of the connected user gets control each time one of the events occurs. For a list of events presented to the event exit, see “Events Reported to the Event Exit” on page 335.

The system describes the events through the event exit parameter list (IXLYEEPL) for all connected users to the structure. IXLYEEPL contains the following types of information:

- Information about the connector whose event exit has been driven
- General information about the event
- Information about the connection that is the subject of the event
- Event specific information, such as loss of connectivity, data about connectors to a structure being rebuilt, user synchronization point data, and volatility change information.

Note: The EEPL contains information that refers to the target connector and the subject connector. The target connector is the connector whose event exit has been driven. Identify the target connector by its connect token (EEPLCONTOKEN). The subject connector is the connector to which the event applies. For example, in a loss of connectivity situation, the connector who had lost connectivity would be the subject connector. Identify the subject connector by its connect token (EEPLSUBJCONTOKEN). (In this LOSSCONN example, the target connector could also be the subject connector, because a LOSSCONN event generates event notification to all active connections to the structure, including those connections that have lost connectivity.)

Note that if a cache structure connector has specified SUPPRESSEVENTS=YES, peer connectors to that structure will not receive the New Connection, Rebuild New Connection, and Discontinued or Failed Connection events originated by that connector, regardless of whether the peer connectors had specified SUPPRESSEVENTS. See “Suppressing Certain Events for a Connector” on page 233. The cache structure connector that specified SUPPRESSEVENTS=YES will not receive the Existing Connection and Rebuild Existing Connection events. However, that connector is responsible for responding to those connection

events that are originated by other connectors to the structure that have specified of defaulted to SUPPRESSEVENTS=NO.

The order in which MVS reports events to a connected user is usually in the sequence in which the events occurred. For example, a connected user would be notified about a new connection event before the connection disconnect or failed event. Exceptions to this ordering are:

- A Rebuild Quiesce, Connect, Connects Complete, or Duplex Established event followed by Rebuild Stop event

  If the system has not notified all connected users about the Rebuild Quiesce, Connect, Connects Complete, or Duplex Established event and a Rebuild Stop event for the same structure occurs, only the Rebuild Stop event will be presented to the connected users. Note that because a Rebuild Stop event can supersede many of the rebuild events, some rebuild events might or might not be presented to the connected user prior to the Rebuild Stop event, depending on its timing.
- Structure Volatility State Change events

  If the volatility state of a coupling facility changes (volatile or non-volatile) and then changes back again, the timing of the event exit notification might present only the second change. (For this reason, you should check the CONAVOLATILE structure attribute flag in the connect answer area to determine the volatility state.)
- Structure Temporarily Unavailable events

  If the system has not notified all connected users about the Structure Temporarily Unavailable event and the system-managed process is stopped, only the Structure Available event will be presented to the connected users.

Some events require that the connected user provide a response. Users can respond to events in the event exit:

- By setting a return code X'00' or X'01' in IXLYEEPL, indicating that all necessary processing has been performed.
- By setting an IXLYEEPL return code X'08', indicating that processing will be performed asynchronously, and that the connector will subsequently respond to the event using IXLEERSP. (Some events require that IXLEERSP be used to provide a response in this manner.)

Using IXLYEEPL to Provide a Response

For some events, connections can handle the event synchronously (that is, at the time the event exit gets control) and need only set a return code in IXLYEEPL. For example, a disconnected or failed connection event requires that all active connectors to the structure provide an event exit response. Depending on the protocol, users respond by setting return codes in IXLYEEPL to handle the event. (MVS checks return codes for IXLYEEPL only if it expects a response.) If the connection is failed-persistent and the connection is able to recover, active connectors can set a return code X'00'. When all connectors have responded, the failed-persistent connection can attempt to reconnect to the structure. An active connection also can perform recovery for the failed-persistent connection and set a return code X'01' in IXLYEEPL to delete the connection.

Using IXLYEEPL and the IXLEERSP macro

If asynchronous processing is required to respond to the event (that is, users need to process the event at a later time) or if the event is a Rebuild Quiesce, Rebuild Cleanup, or Rebuild Stop event that requires an IXLEERSP response, users must set a return code X'08' in IXLYEEPL to indicate that they intend to provide a response through the IXLEERSP macro. Users then issue the IXLEERSP macro in task mode to indicate that they have handled the event.

For example, to handle a failed-persistent connection, an active connection can set a return code X'08' in IXLYEEPL to indicate that the active connection will issue the response to the event on IXLEERSP. At a later time, an active connection can perform recovery for the failed-persistent connector and issue IXLEERSP in task mode to release the failed-persistent connection. “Deleting Failed-Persistent Connections” on page 262 provides information on how to handle a failed-persistent connection event.

For events that require an IXLEERSP response, all active connectors to the structure must set an IXLYEEPL return code to indicate that the response will be handled by IXLEERSP.

The system expects connected users to respond to the following events. For events marked with an asterisk (*), the user must respond with the IXLEERSP macro. “Using IXLEERSP” on page 355 provides information on the IXLEERSP macro.

- Existing Connection (failed-persistent only)
- Disconnected or Failed Connection
- Rebuild Quiesce*
- Rebuild Connect Failure
- Rebuild Cleanup*
- Rebuild Stop*
- Stucture Temporarily Unavailable

Handling Outstanding Event Responses

If a connected user disconnects or fails before providing an expected response to an event, the system informs all connected users through the Disconnected/Failed Connection event. After all existing connections have responded to the Disconnected or Failed Connection event, the system implicitly provides any outstanding event responses that the failed connected user needed to provide.

Note however, that the system does not implicitly provide these outstanding responses until all surviving users have themselves responded to the Disconnected or Failed Connection event on behalf of the failing user. If these responses from the surviving users are not received in a timely manner, deadlocks can occur. Alternatively, the surviving users can explicitly provide “proxy” responses for those owed by the failing connector for Rebuild Stop and Rebuild Cleanup events.

Events Reported to the Event Exit

MVS reports the following specific events to the event exit of connected users to a structure. N/A in a column means that the information is not applicable.

For all events, connection information about the connection that is the subject of the event is passed in IXLYEEPL. The connector information includes connect name, connection identifier, and connection disposition.

Table 17. Summary of Events Reported to the Event Exit

Event/Description	Important Information Passed to Event Exit	Connections Notified	Response Required by Notified Connection	Valid IXLYEEPL Return Codes
New Connection EEPLNEWCONNECTION The connection (subject of the event) is new to the structure. Existing connected users might receive notification of a new connection before the IXLCONN request for the new connection completes.		All active connections (except the new connection) to the structure	None.	N/A

Table 17. Summary of Events Reported to the Event Exit (continued)

Event/Description	Important Information Passed to Event Exit	Connections Notified	Response Required by Notified Connection	Valid IXLYEEPL Return Codes
Existing Connection (active and failed-persistent) EEPLEXISTINGCONNECTION The connection (subject of the event) is currently defined to the structure. An existing connection can be either active or failed-persistent as indicated by field EEPLSTATEACTIVE in IXLYEEPL. Each existing connection event represents one connected user. New connections might receive notification of existing connections before the IXLCONN request for the new connection completes. The end of the list of existing connections is indicated by a dummy event exit parameter list. **Note:** The dummy IXLYEEPL is indicated by the EEPLDUMMYLASTEVENT flag. When the flag specifies that this is the last event, the only other information in the EEPL is the identification of the connector whose event exit has been driven and general information about the event.	Active or failed-persistent state of the existing connection, connection disposition, user-specified disconnect data.	A new connection to the structure	If existing connection is active, none. If failed-persistent, user protocol of the new connection determines the response.	rc=X'00', X'01', or X'08'. See "Return Specifications" on page 354.
Disconnected or Failed Connection EEPLDISCFAILCONNECTION One of the following has occurred: • The connected user (subject of the event) issued IXLDISC. Disconnection reason specified on IXLDISC can be NORMAL, FAILURE, or DELETESTR. • End of task, address space, or system has occurred before the connection (subject of the event) issued IXLDISC. Connection has failed.	Indication of lock resources if held by a lock structure. User-specified disconnect data. Indication of whether, during rebuild, the connection was connected to both the new structure and the old structure.	All active connections to the structure	Required	rc=X'00', X'01', or X'08'. See "Return Specifications" on page 354.
Loss of Connectivity to the structure EEPLLOSSCONN The connection has lost physical connection to the coupling facility.	If rebuild was in progress, indication of whether connectivity was lost to the original structure or to the structure allocated for rebuild. An indication of whether or not to delay action.	All active connections to the structure, including those subject connections that have lost connectivity.	None. Recommended that all connections that have lost connectivity issue IXLDISC with REASON= FAILURE when delaying action is not indicated.	N/A
Structure Failure EEPLSTRFAILURE Either a structure in the coupling facility or the coupling facility itself has failed. New connections are denied access. Note that if the failure is for a single structure rather than the entire coupling facility, XES deallocates the failed structure when either there are no active connections or when the last Rebuild Cleanup event exit has been received as part of the rebuild process.		All active connections to the structure	None. Recommended that all users issue IXLDISC with REASON= FAILURE or rebuild the structure.	N/A

Table 17. Summary of Events Reported to the Event Exit (continued)

Event/Description	Important Information Passed to Event Exit	Connections Notified	Response Required by Notified Connection	Valid IXLYEEPL Return Codes
Rebuild Quiesce EEPLREBUILDQUIESCE Rebuild processing has been initiated for the structure. Connections can participate in rebuilding the structure, stop the process, or disconnect.	• Reason for the rebuilding or duplexing request. • LESSCONN-ACTION and LOCATION attributes for the rebuilding or duplexing request.	All active connections to the structure	If connections decide to rebuild or duplex, connection must 1. Complete outstanding structure requests which, if based on a restart token, should be fully completed before quiescing use of the structure. 2. Stop activity to structure. 3. Prevent new IXLCACHE, IXLLIST, IXLLOCK, or IXLRT requests to the structure. 4. Provide an event exit response. **Event response required through IXLEERSP**.	rc=X'08'. See "Return Specifications" on page 354.
Rebuild Connect EEPLREBUILDCONNECT All connections have responded to the Rebuild Quiesce event for the structure.		All active connections to the structure	Required. To continue with the user-managed rebuild processing (rebuild or duplexing rebuild), each active connection must issue IXLCONN with the REBUILD option. Once connected to the new structure, connected users can perform coupling facility operations to the structure. To confirm that a connected user has completed its structure reconstruction or data propagation to the new structure, issue IXLREBLD REQUEST=COMPLETE.	N/A
Rebuild Connects Complete EEPLREBUILDCONNECTSCOMPLETE All connections eligible to rebuild-connect to the structure have issued IXLCONN REBUILD. The system indicates the number of successful and unsuccessful connections to the new structure and identifies the connections through the connection ids. Users can determine from their protocol if enough connections are available to continue rebuilding the structure or stop rebuilding.	Indication of the connections that successfully connected to the new structure and the connections that failed Total number of successful connections to the new structure Total number of unsuccessful connections to the new structure	All active connections to the structure. Connectors to a structure in the user-managed duplexing process do not receive this event.	None.	N/A

Table 17. Summary of Events Reported to the Event Exit (continued)

Event/Description	Important Information Passed to Event Exit	Connections Notified	Response Required by Notified Connection	Valid IXLYEEPL Return Codes
Rebuild New Connection EEPLREBUILDNEWCONNECTION The connection (subject of the event) to the structure is new. Existing connected users to the new structure might receive notification of a new connection before IXLCONN for the new user completes.	Connect name, connection identifier	Active connections that connected to the new structure (other than the one that is the subject of the event)	None	N/A
Rebuild Existing Connection EEPLREBUILDEXISTINGCONNECTION The connection (subject of the event) is currently connected to the new structure. Each Rebuild Existing Connection event represents one connected user. A new connection might receive notification of existing connections before IXLCONN for the new connection completes. The end of the list of existing connections is indicated by a dummy event exit parameter list. For this event, the system reports only successful connections; failed-persistent connections to the new structure are not reported.		The connection that just connected to the new structure	None	N/A
Rebuild Connect Failure EEPLREBUILDCONNECTFAILURE The IXLCONN REBUILD request fails because the task or address space of the requestor abnormally terminated during IXLCONN REBUILD processing and this task is different from the task which owns the original connection.		All active connections to the structure	Required. Connections must cleanup any control information about a successful rebuild connect request that is reported by either a Rebuild New Connection or Rebuild Existing Connection event.	rc=X'00'or X'08'. See "Return Specifications" on page 354.
Structure Duplexing Established EEPLREBUILDDUPLEXESTABLISHED Connectors to the duplexed structures can begin duplexed structure operations.		All active connections to the structure.	None Connectors to the duplexed structures should begin duplexed structure operations.	N/A
Stop Duplexing Rebuild to Switch EEPLREBUILDSWITCH Duplexing rebuild stop processing has been initiated for the structure. Connectors should prepare to switch to the new structure.		All active connectors to the structure.	Required. Connections must quiesce their use of the old and new structures and perform the necessary cleanup. A confirmation is required with IXLREBLD REQUEST= DUPLEXCOMPLETE.	N/A
Rebuild Cleanup EEPLREBUILDCLEANUP Connections to the structure being rebuilt have issued IXLREBLD with REQUEST=COMPLETE to indicate that the rebuild process is complete. Connections to the structure being duplexed have issued IXLREBLD with REQUEST=DUPLEXCOMPLETE to indicate that the duplexing process is complete.		All active connections to the structure	**Event response required through IXLEERSP.**	rc=X'08'. See "Return Specifications" on page 354.
Rebuild Complete EEPLREBUILDPROCESSCOMPLETE The structure has been successfully rebuilt.		All active connections to the structure	Connectors should resume normal structure operations.	N/A

Table 17. Summary of Events Reported to the Event Exit (continued)

Event/Description	Important Information Passed to Event Exit	Connections Notified	Response Required by Notified Connection	Valid IXLYEEPL Return Codes
Rebuild Stop EEPLREBUILDSTOP Rebuild stop processing has been initiated for the structure. For duplexing, the rebuild stop event implies KEEP=OLD.	Reason for stopping the rebuilding process	All active connections to the structure	Connections must 1. Complete outstanding structure requests to both original and new structure which, if based on a restart token, should be fully completed before quiescing use of each structure. 2. Stop activity to structure 3. Prevent new IXLCACHE, IXLLIST, IXLLOCK, or IXLRT requests to the structure. 4. Provide an event exit response. **Event response required through IXLEERSP.**	rc=X'08'. See "Return Specifications" on page 354.
Rebuild Stop Complete EEPLREBUILDSTOPPROCESS- COMPLETE Stop-rebuilding for the structure is complete.		All active connections to the structure	Connectors should resume normal structure operations.	N/A
User Synchronization Point EEPLUSERSYNCPOINT A connection has defined a new user synchronization point, or all confirmations have been received for a user synchronization point. Connections can use the IXLUSYNC macro to define synchronization points for different processing stages. For example, see "Using IXLUSYNC to Coordinate Processing of Events" on page 344.	Confirmation that the processing for the event is complete. Event associated with the synchronization point. Definition of next synchronization point if specified. Highest user-defined completion code value (or, for connections that disconnect or fail while owing a sync point confirmation, X'0000FFFF', implicitly set by XES).	All active connections to the structure	Required when the event indicates a new synchronization point:. Confirmation using IXLUSYNC REQUEST=CONFIRM or REQUEST= CONFIRMSET. Not required when the event indicates all confirmations have been received.	N/A
Coupling Facility Structure Volatility State Change EEPLVOLATILITYSTATECHANGE The current volatility state of a coupling facility structure has changed.	The current volatility state	All active connections to the structure	None Connection may want to initiate a rebuild of the structure.	N/A

Table 17. Summary of Events Reported to the Event Exit (continued)

Event/Description	Important Information Passed to Event Exit	Connections Notified	Response Required by Notified Connection	Valid IXLYEEPL Return Codes
XES Recommend Action EEPLXESRECOMMENDACTION MVS did not initiate rebuild based on the comparison of the rebuild percent specified for the structure and the value calculated by MVS when a loss of connectivity occurs. MVS uses system weights in the active SFM policy to either start rebuild for the loss or advise the connections to disconnect.	The percentage loss of connectivity, based on SFM policy weights.	Active connections that lost connectivity to the coupling facility containing the structure.	Disconnect from the structure.	N/A
Structure Alter Begin EEPLALTERBEGIN A request to alter the structure has been initiated.	The requested target values, the composite for the minimum available entries/elements and EMCs, and ratio change indication specified by the connections. If user-managed duplexing is in effect, status of the old and new structures.	All active connections to the structure.	None. If the requested change is to contract the structure size, cast out or otherwise free up in-use structure resources to facilitate the structure alter processing.	N/A
Structure Alter End EEPLALTEREND The altering of the structure has ended.	The status of the structure as the result of the structure alter processing. If user-managed duplexing is in effect, status of the old and new structures.	All active connections to the structure.	None. Adjust any limits set for your use of the structure based on the changes made to the size and/or apportionment of the altered structure.	N/A
Loss of Connectivity Percentage EEPLLOSSCONNPCTNOTIFY	The percentage loss of connectivity, based on SFM policy weights. If there is no active SFM policy in the sysplex, the percentage loss of connectivity reported is either 100% or 0. There is no guarantee that all connectors will receive the same value.	Active connections that lost connectivity to a coupling facility containing the structure in a user-managed duplexing rebuild.	None	N/A
Structure Temporarily Unavailable EEPLSTRTEMPUNAVAILABLE The structure is temporarily unavailable for processing coupling facility requests because a system-managed process such as rebuild has begun.	• The type of system- managed process that is precluding use of the structure. • The event sequence number (required for response).	All active connections to the structure.	Required, either through IXLEERSP or by setting return code in IXLYEEPL. IBM recommends that connections prevent new coupling facility requests (IXLCACHE, IXLLIST, IXLLOCK, IXLRT, or IXLSYNCH) to the structure before responding to this event.	rc=X'00', X'08'. See "Return Specifications" on page 354.

Table 17. Summary of Events Reported to the Event Exit (continued)

Event/Description	Important Information Passed to Event Exit	Connections Notified	Response Required by Notified Connection	Valid IXLYEEPL Return Codes
Structure State Change EEPLSTRSTATECHANGE The characteristics of the structure may have changed as the result of a system-managed process such as rebuild.	The type of process that caused the structure state change. Current structure characteristics, including: • CFLEVEL • CFNAME • Volatility state • Physical structure version numbers • Failure isolation state	All active connections to the structure	None. The connector may inspect the new characteristics of the structure and take appropriate action.	N/A
Structure Available EEPLSTRAVAILABLE A structure that had been temporarily unavailable for processing coupling facility requests because a system-managed process such as rebuild had begun is once again available for processing. Connections may receive notification of structure availability without ever having received a Structure Temporarily Unavailable event. The system presents the Structure Available event upon completion of a system-managed process to inform the connector that activity against the indicated structure may be resumed.	The type of system-managed process that had been precluding use of the structure.	All active connections to the structure.	None. On receipt of this event, connections that had quiesced their activity against the structure in response to the Structure Temporarily Unavailable event may resume submitting requests to the structure.	N/A

XES Monitoring of Event Responses

With OS/390 Release 8 and higher (as well as Releases 3 through 7 with APAR OW20623 installed), XES provides support for monitoring responses for certain structure rebuild, User Sync Point, and Disconnected or Failed Connection events. This support is intended to limit the extent of potential hang conditions when connectors do not provide an expected response to an event by notifying the operator or an automation package so that some action can be taken against the non-responding connector. When a specific response is not provided within a predetermined time limit, XES issues a message for each connector owing an expected response that is overdue indicating a connector's failure to confirm an event. The messages identify the connector with the outstanding response so that action can be taken.

Beginning at z/OS V1R12, XES may take automatic action to relieve hangs if permitted by the installation's SFM policy CFSTRHANGTIME specification. After a hang is declared (approximately 2 minutes after the event requiring the response occurs), XES initiates a diagnostic dump. It then waits the time specified by CFSTRHANGTIME and takes the first applicable action from the following list:

- Stop the rebuild, if one is in progress and was not initiated due to structure failure or loss of connectivity to the coupling facility.
- Stop signaling paths (XCF signaling structures only).
- Force disconnection from the structure (XCF signaling structures only).
- Terminate the connector's task (the task from which IXLCONN was issued).
- Terminate the connector's address space.
- Partition the connector's system.

If an action fails to resolve the hang condition, XES will escalate down the list to progressively more disruptive actions until either the hang is alleviated or no further automatic action is possible. If XES does not take automatic action, or the action is ineffective, the installation may find it necessary to cancel the connector's job. In either case (XES or installation action), the expectation is that terminating the hung process or connector will allow the application or subsystem as a whole to continue processing.

For certain event types, XES may be able to determine that although the application has not responded to the event, it is nevertheless making progress toward completion. In that case, XES does not declare a hang. Instead, if the time without a response exceeds the threshold, XES will issue a message to report a delay but take no further action. If the application at some point stops making progress, XES will transition into hang detection processing by collecting diagnostic data and initiating automatic action, if possible.

Installations should be aware that an expected response that is not received in a timely manner does not necessarily indicate that the connector is hung, but could mean that because of environmental conditions, the connector is simply taking a long time to respond. It is highly recommended that before cancelling the connector, the system programmer, operator, or automation program used for message handling, should examine some diagnostic data to confirm that the connector truly is hung.

"Events Monitored by XES" on page 342 lists the events for which XES monitoring is in effect.

Events Monitored by XES

To improve sysplex availability, XES monitors the following events to ensure that the indicated responses are received from all structure connectors in a timely manner.

Table 18. Events Monitored by XES

Event	Required Response
Rebuild Quiesce	IXLEERSP EVENT=REBUILDQUIESCE
Rebuild Connect	IXLCONN REBUILD and IXLREBLD REQUEST=COMPLETE
Rebuild Switch	IXLREBLD REQUEST=DUPLEXCOMPLETE
Rebuild Cleanup	IXLEERSP EVENT=REBLDCLEANUP
Rebuild Stop	IXLEERSP EVENT=REBLDSTOP
Structure Temporarily Unavailable	IXLEERSP EVENT=STRTEMPUNAVAILABLE or IXLYEEPL return code
Disconnected or Failed Connection	IXLEERSP EVENT=DISCFAILCONN or IXLYEEPL return code
Rebuild Connect Failure	IXLEERSP EVENT=REBLDCONNFAIL or IXLYEEPL return code
User Sync Point	IXLUSYNC REQUEST=CONFIRM or IXLUSYNC REQUEST=CONFIRMSET
Connecting during Rebuild Quiesce phase	IXLEERSP EVENT=REBLDQUIESCE
Connecting during Rebuild Connect phase	IXLCONN REBUILD and IXLREBLD REQUEST=COMPLETE

Table 18. Events Monitored by XES (continued)	
Event	**Required Response**
Connecting during Duplex Established phase	IXLCONN REBUILD
Connecting during Rebuild Switch process	IXLCONN REBUILD and IXLREBLD REQUEST=DUPLEXCOMPLETE
Connecting during a User Sync Point	IXLUSYNC REQUEST=CONFIRM or IXLUSYNC REQUEST=CONFIRMSET
Connecting during Rebuild Stop process	IXLEERSP EVENT=REBLDSTOP

Information Provided by XES Event Monitoring

XES issues either message IXL040E or IXL041E when a required response to an event has not been received from a particular structure connector within a predetermined time interval. Each message identifies the connector, jobname, and ASID of the non-responder, the event for which the response is required, and the name of the affected structure. The message also identifies the XES process that is unable to continue because the required response has not been received and the time that the system started waiting for the response.

The purpose of the message is to alert the operator, system programmer, or automation package of a potential hang condition caused by the connector who is not responding in a timely manner. Before taking any overt actions to remove the connector, the installation should use diagnostic procedures to verify whether the connector is truly in a hang condition or is simply slow to respond. Only after it is established that the connector is in a hang condition can a decision be made as to whether to cancel or shut down the connector.

XES also records a symptom record in the logrec data set at the time that an IXL040E or IXL041E message is issued. The symptom record contains the same information as is contained in the message.

The messages remain on the operator console screen until either the required response is received or becomes no longer expected. A required response is no longer expected once the connector fails, disconnects, or when system failure cleanup processing completes the removal of the failed system on which the connector is running. Once a response is no longer expected, the system DOMs message IXL040E or IXL041E, and issues message IXL042I or IXL043I.

Connection Considerations with XES Event Monitoring

XES event monitoring is also in effect when a connector attempts to connect to a structure during structure rebuild processing that is user-managed or User Sync Point processing. In each of these processes, the connector is required to provide an explicit response as part of participating in the ongoing rebuild or user sync point process that is active for the structure. If the response is not received within the predetermined time frame, XES will issue a message to the operator indicating the connector's failure to confirm this in a timely manner.

Discontinuing XES Event Monitoring

XES discontinues event monitoring for expected responses when either the expected response is received or the required response becomes no longer expected from connectors because they have failed, disconnected, or reside on a system that terminated. Additionally, the following events can trigger the discontinuation of XES monitoring for certain events:

- Rebuild Stop event

  Causes the monitoring of the Rebuild Quiesce, Rebuild Connect, and Rebuild Connect Failure events to be discontinued.

Causes the monitoring of connector(s) that connected during the Rebuild Quiesce, Rebuild Connect, and Duplex Established phases to be discontinued.

- Disconnected or Failed Connection Event

  Causes the monitoring of the Rebuild Connect Failure event to be discontinued.

- Structure Available event

  Causes the monitoring of the Structure Temporarily Unavailable event to be discontinued.

Any outstanding operator messages that were issued for the events being discontinued are deleted from the console and the followup message, IXL042I or IXL043I, is issued.

Using IXLUSYNC to Coordinate Processing of Events

User synchronization points are used to provide synchronization of processing among connectors to a structure. The IXLUSYNC service and the User Sync Point Event work together to synchronize processing.

IXLUSYNC allows connections

- To define a value for a synchronization point associated with a specific event (**REQUEST=SET**).
- To confirm that a connection has reached a synchronization point (**REQUEST=CONFIRM**).
- To confirm the completion of the current event and define a synchronization point for the next event (**REQUEST=CONFIRMSET**).

The User Sync Point event is presented to connectors when a new synchronization point is set successfully and when all connectors confirm that a synchronization point has been reached. The User Sync Point event does not require an event exit response.

Overview of IXLUSYNC Processing

Using IXLUSYNC, you can define a value for a synchronization point that is associated with an event. When a synchronization point value is defined by a connected user for an event, the system reports the synchronization point value to the event exit of all connected users. Connected users must establish protocols to handle the event associated with the synchronization point. When each connector completes processing associated with the event, the connector uses IXLUSYNC to confirm that its processing for the event is complete. The connector can also set a user-defined completion code when confirming with IXLUSYNC.

When all confirmations have been received, the system passes the synchronization point confirmation to the event exit of all connected users. The information includes the highest completion code value set by any connector when confirming the sync point.

For connectors that disconnect or fail while owing a confirmation for a sync point, the system implicitly confirms the sync point and sets a completion code of X'0000FFFF' for the disconnected or failed connector. Note that if a given user completion code is to take precedence over this completion code, it must be higher that X'0000FFFF' or, if the implicit completion code is to take precedence over a given user completion code, it must be less than X'0000FFFF'.

You also have the option to define a new synchronization point value to be associated with another event (**REQUEST=CONFIRMSET**) at the same time you confirm the current event. This allows users to construct “chains” of synchronized events to be processed in sequence. (The last connected user to confirm a synchronization point defines the new synchronization point. Users who attempt to CONFIRMSET, but who are not the last connected user, will have the CONFIRM accepted, but the system rejects the SET with reason code IXLRSNCODENOTLASTCONFIRMATION.) The system reports the new value to the event exit of all connected users for confirmation. Only one synchronization point value can be defined at a time by the entire set of connected users of a structure.

Information Returned in IXLYEEPL

The following chart illustrates the user sync point values in the event exit parameter list for the user sync point event. “Request=Set” information is received after a user sync point is successfully set.

"Request=Confirm" or "Request=ConfirmSet" information is received after all confirmations have been received for a user sync point.

Table 19. IXLYEEPL Data for IXLUSYNC

	REQUEST =SET	**REQUEST =CONFIRM**	**REQUEST =CONFIRMSET**
EeplCompletedUserEvent	0	Value of USEREVENT	Value of USEREVENT
EeplNextUserEvent	Value of USEREVENT	0	Value of NEXTUSEREVENT from the IXLUSYNC REQUEST=CONFIRMSET invocation
EeplCompletedUserState	0	Value of USERSTATE when first set	Value of USERSTATE when first set
EeplNextUserState	Value of USERSTATE or 0	0	Value of USERSTATE from the IXLUSYNC REQUEST=CONFIRMSET invocation
EeplCompletedUserCompCode	0	Highest completion code set by any confirming user. Note that a completion code of X'0000FFFF' is set by XES when a user who has not provided a confirmation either disconnects or fails.	Highest completion code set by any confirming user. Note that a completion code of X'0000FFFF' is set by XES when a user who has not provided a confirmation either disconnects or fails.

XES Monitoring of User Sync Point Event Responses

XES monitors the time required by the connector to respond to the user sync point event. If a response with either IXLUSYNC REQUEST=CONFIRM or IXLUSYNC REQUEST=CONFIRMSET is not received in a timely manner, XES issues a message for each connector owing a response to the event so that the system programmer or operator can take actions to allow processing to continue. See "XES Monitoring of Event Responses" on page 341.

Handling Connection Failures during Synchronization

Whenever a connection terminates, the system informs all connected users through the Disconnected or Failed Connection event. If the connection terminates while a user event is set, two options are available.

- An active connector confirms any outstanding user event confirmations on behalf of the disconnected or failed connection using PROXYRESPONSE=YES before all connectors have responded to the Disconnected or Failed Connection event.
- The system confirms all outstanding user event confirmations for the disconnected or failed connection as soon as all connectors have responded to the Disconnected or Failed Connection event.

Before responding to the Disconnected or Failed Connection event, the other connected users have an opportunity to complete the failed user's processing for the user event and respond to the event, if

required. An example of when this might be most useful is when the CONFIRMSET option is used. If the failed user was the last connector to issue IXLUSYNC and had specified CONFIRMSET, the next event might not have been set before the user terminated. In such a case, one or more of the peer connectors could issue a SET to define the next event.

To confirm a user event on behalf of a failed user, the active connector must provide the connect token of the failed user. The system makes this token available when reporting the Disconnected or Failed Connection event.

If all connections terminate, the system resets the user event.

"Connecting to a Structure when a Synchronization Point Is Set" on page 257 describes XES processing when new connectors connect to a structure while a user synchronization point is set.

Disconnecting from a Coupling Facility Structure

A connected user can disconnect from a coupling facility structure when you no longer require access to the structure or when you recognize a failure such as loss of connectivity. Once disconnected, you cannot access the structure through any XES services.

Overview of Disconnect Processing

Users disconnect from a coupling facility structure either for normal processing or because of a failure. The system invalidates the disconnecting user's connect token and notifies other connectors connected to the structure about the disconnect event. When all connections to the structure have acknowledged the disconnect request through the event exit or IXLEERSP, the disconnect is complete.

Coding the IXLDISC Macro

The IXLDISC macro allows you to disconnect from a structure. You can disconnect from only one structure at a time. If you wish to disconnect from multiple structures, issue IXLDISC once for each structure.

IXLDISC requires that you provide the connect token (CONTOKEN) that XES returned when the initial connection to the structure was made with IXLCONN. You also must invoke IXLDISC from the same task that issued IXLCONN for the connection. During the rebuild process, if a connection disconnects during rebuild and a rebuild connect is issued from a different task, the subsequent disconnect must be done from the task that did the original connect.

The IXLDISC macro allows you to specify the reason for your disconnection. If you do not specify a reason, the system assumes this is a normal disconnection. You can also communicate information about your disconnection to surviving peer connectors by using the DISCDATA keyword. This eight bytes of data is passed to the event exits of surviving peers when you disconnect.

For the complete syntax of the IXLDISC macro, see *z/OS MVS Programming: Sysplex Services Reference*.

Disconnect Events and the Event Exit

When a connected user disconnects from a coupling facility structure, other users connected to the structure receive notification of the event through their event exits. The other users must respond to the Disconnected or Failed Connection event, either by setting a return code in the event exit parameter list (IXLYEEPL) or by issuing the IXLEERSP macro. The system does initial cleanup of the connection before notifying the other connected users, but does not complete processing of the disconnect until all connected users have provided an event exit response. The disconnected or failed connection remains in the disconnecting or failing state while the system waits for the event exit responses.

Before responding to the event, the other connected users have the responsibility of cleaning up all references to the disconnected or failed connection and performing recovery processing, if necessary. This cleanup might include handling locks that the connection held or setting and/or confirming a user synchronization point event. Other connected users determine what type of recovery is necessary by examining the IXLYEEPL, which indicates how the user terminated (either normally or abnormally) and what the persistence attribute of the connection is.

Suppressing Certain Event Notifications

On systems with OW38840 installed or which are at OS/390 Release 9 and higher, a connector to a cache structure may request that the system suppress certain connection and disconnection events generated by the connector. The Disconnected or Failed Connection event is one of the events that can be suppressed. If a cache structure connector has specified SUPPRESSEVENTS=YES and subsequently disconnects or fails, peer connectors will not receive the Disconnected or Failed Connection event and therefore will not provide the required response. Instead, the system immediately performs the cleanup that would have been performed after all responses to the event had been received, if the event had not been suppressed. See "Suppressing Certain Events for a Connector" on page 233 for information about event suppression.

Retrieving Information from IXLYEEPL

The following fields in IXLYEEPL provide pertinent information for the Disconnected or Failed Connection event:

EEPLCONNINFOSUBJECT
General information about the connection that is the subject of the Disconnected or Failed Connection event.

EEPLTERMINATEDABNORMAL
Type of termination — Did the connection terminate normally or abnormally?

EEPLSUBJDISPOSITIONKEEP
Persistence — Is the connection persistent or non-persistent?

EEPLDISCWITHLOCKRESOURCES
For lock user (lock or serialized list structure) — Was disconnection made with locks still held?

EEPLSUBJDISCDATA
Did the disconnecting user provide any disconnect-time data when invoking the IXLDISC macro?

Responding to a Disconnected or Failed Connection Event

You can respond to the Disconnected or Failed Connection event either by specifying that XES is to complete its cleanup of the connection or that XES is to complete its cleanup and also release the failed-persistent connection. Specifying that XES is to release the failed-persistent connection implies that you have done whatever recovery processing is necessary for the failed connection. You respond to the event either by setting a return code in IXLYEEPL or by invoking the IXLEERSP macro.

In IXLYEEPL, the return codes are:

IXLRCEVENTEXITRESPONSE
XES is to complete its cleanup of the connection.

IXLRCEVENTEXITRELEASECONN
XES is to complete its cleanup of the contention and also release the failed-persistent connection.

IXLRCEVENTEXITLATERESPONSE
You will respond to the event at a later time using the IXLEERSP macro.

To respond with IXLEERSP, the parameters are:

EVENT=DISCFAILCONN,RELEASECONN=NO,...
XES is to complete its cleanup of the connection.

EVENT=DISCFAILCONN,RELEASECONN=YES,...
XES is to complete its cleanup of the contention and also release the failed-persistent connection.

XES Monitoring of Disconnected or Failed Connection Event Responses

XES monitors the time required by the connector to respond to the DISCFAILCONN event. If a response is not received in a timely manner, XES issues a message for each connector owing a response to the event so that the system programmer or operator can take actions to allow processing to continue. See "XES Monitoring of Event Responses" on page 341.

Persistence Considerations

Both connection and structure persistence are defined at connect time with the IXLCONN macro. CONDISP=KEEP and CONDISP=DELETE specify whether a connection is to remain defined after a failure; STRDISP=KEEP and STRDISP=DELETE specify whether a structure is to become not-defined after all users have disconnected.

Normal Disconnection

A connected user who disconnects from a structure because normal structure processing is complete specifies REASON=NORMAL on IXLDISC. The system releases the connection to the structure when normal disconnection occurs. (The connection disposition refers only to processing that is to occur if a failure occurs; therefore, whether the connected user specified CONDISP=KEEP or CONDISP=DELETE on IXLCONN, the connection is released.)

Note: If a connected user disconnects with REASON=NORMAL on IXLDISC while still holding locks, XES treats the disconnect as if the user had specified REASON=FAILURE on IXLDISC.

Disconnection Because of Failure

In an error recovery situation, you can disconnect with REASON=FAILURE. If you had defined CONDISP=KEEP on IXLCONN and disconnect with REASON=FAILURE, your connection will be placed in a failed-persistent state when all peer connections respond to the Disconnected or Failed Connection event. To determine how to handle failed-persistence, see “Deleting Failed-Persistent Connections” on page 262.

If CONDISP=DELETE, the failed connection will be placed in the not-defined state when all peer connections respond to the Disconnected or Failed Connection event.

Disconnection to Delete the Structure

A connected user who disconnects from a structure in order to delete the structure specifies REASON=DELETESTR on IXLDISC. The system releases the connection to the structure.

Note: The connection disposition specified on the IXLCONN does not apply.

This disconnect reason is processed the same way as a normal disconnection, except when the disposition of the structure is KEEP. If the DELETESTR disconnect request results in no active or failed-persistent connections to the structure, the structure is deleted even if the structure has a disposition of KEEP.

The DELETESTR keyword is useful in many circumstances. For example, it is useful as part of the recovery actions for ensuring that the correct instance of an otherwise-persistent structure is deleted when there are no more connectors.

Note: If a connected user with RECORD=YES disconnects with REASON=DELETESTR on IXLDISC while still holding locks, XES treats the disconnect as if the user had specified REASON=FAILURE on IXLDISC.

Handling Resources for a Disconnection

After all active users have disconnected from the structure and all failed-persistent connections are released, XES either deletes (STRDISP=DELETE) or retains (STRDISP=KEEP) the structure depending on the structure disposition specified on IXLCONN. See “Defining the Structure Attributes” on page 205.

Whether the disconnection is normal or the result of an error, MVS cleans up resources depending on the type of structure (cache, list, or lock) and whether the connection is being made failed-persistent or not.

- Cache structure
  - The local cache vector is released.
  - Cast-out locks held by the terminating connection are reset.

    - For castout locks held by the terminating connection in the read-for-castout state (that is, as the result of a CASTOUT_DATA request), the cast-out lock and cast-out lock state are reset to zero, the change bit for the entry is set to one to overindicate the “changed” state for the entry, and the parity is reset to the null value.
    - For cast-out locks held by the terminating connection in the write-with-castout state (that is, as the result of a WRITE_DATA CHANGED=NO GETCOLOCK=YES request), the entry is deleted from the cache structure along with all data and registered interest.
  - Registered interest in named directory entries is cleaned up.
  - All outstanding asynchronous cross-invalidations will be completed.
    - Reconnecting a failed-persistent connection will ensure structure detach processing has completed and therefore all outstanding asynchronous cross-invalidations will be complete.
- List and serialized list structure
  - List monitoring interest, if registered for, is released.
  - Event queue monitoring interest, if registered for, is released.
  - The user's event queue and all event monitor controls objects used to monitor sublists, if applicable, are released.
  - For a serialized list structure, if the connection is being made failed-persistent, the lock resources are kept. If the connection is being made not-defined, the lock resources are released.
- Lock structure
  - XES releases locks held by the failed connector when all responses are received for the Disconnected or Failed Connection event. If the connection is being made failed-persistent, the record data associated with the failed connector is kept. If the connection is being made not-defined but still has associated record data, the record data is released.
  - When z/OS is running on a server that supports recovery process boosts, coupling facility disconnect processing takes advantage of System Recovery Boost's recovery process boost support to expedite lock structure connector cleanup processing performed by the system and active connected users. When a connected user is disconnected from a lock structure while holding lock resources, managing contention with no lock resources currently held or is disconnected implicitly as the result of task termination, address space termination or a system being removed from the sysplex, a System Recovery Boost will be requested on the systems where active connected users receive the Disconnected or Failed Connection event.

    For more information on System Recovery Boost, see: IBM Z System Recovery Boost Content Solution (www.ibm.com/support/z-content-solutions/system-recovery-boost/) or IBM Documentation (www.ibm.com/docs/en/zos).

If the disconnected structure is a lock or a serialized list, the system leaves the XCF group that it joined when the user first issued the IXLCONN request.

In certain instances, XES must quiesce the activity of user exits in order to perform cleanup processing. For example, when a user disconnects or abnormally terminates, XES will force to completion any user exits executing on behalf of that user by issuing a PURGEDQ against the appropriate units of work. Note that if a connector terminates while a rebuild is in progress, any exits pertaining to both the original and the new structures will be forced to completion. In addition to forcing the currently executing user exits to completion, XES will also prevent any new invocations of these exits by cancelling any events that are pending presentation.

A user exit must be sensitive to conditions that can occur as a result of actions taken by XES and must be able to handle these as appropriate. For example, if a user exit has suspended itself, when the PURGEDQ is issued the system abends the user exit's unit of work with a retryable X'47B' abend and gives control to the user exit's recovery routine. (Note that although the recovery routine can retry, the user exit can not re-suspend itself because the system will fail any request to suspend a unit of work that has been the target of a PURGEDQ.) If the recovery routine percolates back to the system, its associated connection is terminated.

Dumping Considerations

If the last user disconnects from a structure that has an SVC dump associated with it, the system does not delete the structure (regardless of the STRDISP of the structure) until the dump is successfully written out to a dump data set.

- If STRDISP=DELETE, deallocation of the structure remains pending until after the structure dump is deleted. If an attempt is made to allocate and connect to the structure, a new instance of the structure is allocated.
- If STRDISP=KEEP, the structure is not deallocated. If a new connection connects to the structure, it is connected to the current instance of the structure.

Successful Completion of a Disconnection

The system invalidates the user's connect token before returning control to the user who issued the IXLDISC macro. This ensures that the user cannot issue additional XES mainline requests. If the user does use the invalidated token to issue a request for XES services, the request fails with reason code IXLRSNCODEBADCONTOKEN.

The system considers the disconnection complete when all connected users to the structure have acknowledged the disconnection. Active connections to the structure respond to the disconnect event through their event exits. Connections that have failed before responding to the event are handled by XES cleanup processing. (XES implicitly confirms on behalf of the failed connections, with an implied RELEASECONN=NO.)

Upon successful disconnect from a structure, ENF event code 35 is issued.

Forcing the Deletion of a Coupling Facility Object

The IXLFORCE service forces the deletion of objects in the coupling faclility. The objects may be coupling facility structures, connections to a structure, a structure dump associated with a structure, or the structure dump serialization for a structure dump associated with a structure.

The IXLFORCE service is intended to be used for clean-up purposes. For that reason, issuers of the IXLFORCE macro do not have to be connected XES users. Note that forcing a structure without understanding how the structure is being used might cause loss of data or data integrity.

To determine which coupling facility objects are candidates for deletion, use the IXCQUERY macro. IXCQUERY returns information about the status of a structure, the state of structure connection, and whether or not a structure dump is associated with a structure. The DISPLAY XCF operator command also can be used to display this structure information.

Deleting a Coupling Facility Structure

A persistent structure can be deleted only if there are no active or failed-persistent connections to the structure, no failed-persistent connections pending reconciliation into the CFRM active policy, and no structure dump associated with the structure. Deallocation of the structure occurs as follows:

- On systems with OW33615 installed or which are at OS/390 Release 9 or higher, when an attempt is made to connect to an inaccessible structure, and there are no active connectors to the structure, and there is no connectivity to the coupling facility containing the inaccessible structure by any system in the sysplex, the system will force the structure and its failed-persistent connectors. This allows the system to allocate a new instance of the structure so that the connector can successfully connect to it.
- On systems that do not have OW33615 installed or which are below OS/390 Release 9, any attempt to connect to an inaccessible structure will fail until connectivity has been reestablished, or the structure and its failed-persistent connectors have been forced.

  In either case (with or without OW33615 or whether at OS/390 Release 9 or not), deallocation of the structure remains pending until connectivity to the coupling facility is reestablished.

- If there is a structure dump associated with the structure, deallocation of the structure remains pending until the structure dump is deleted. This normally occurs upon completion of the SVC dump that created the structure dump, although you can also delete the structure dump using the IXLFORCE service.

You cannot delete a structure while the rebuild process is in effect for the structure. However, during user-managed or system-managed duplexing rebuild, you can force a structure while it is in the Duplex Established phase. Both instances of the structure are forced.

Deleting a Coupling Facility Connection to a Structure

You can delete one or more failed-persistent connections to a coupling facility structure with the IXLFORCE service. An active connection cannot be deleted. Once the last connection to a non-persistent structure has been deleted, the structure also is deleted. Deletion of a failed-persistent connection occurs as follows:

- All connectors active at the time that the failed-persistent connection terminated must have provided an event exit response acknowledging the termination of the failed-persistent connector.
- When multiple failed-persistent connections to a structure are to be deleted with one invocation of IXLFORCE, each failed-persistent connection is treated as a single IXLFORCE request.

On system with OW33615 installed or which are at OS/390 Release 9 or higher, when connectivity to a coupling facility is lost by all systems in the sysplex and a request to connect to a structure in the coupling facility is received, the system will automatically force all failed-persistent connectors to the inaccessible structure and the structure itself. This allows the system to allocate a new instance of the structure so the connector can successfully connect to it.

You cannot delete a connection to a structure while the rebuild process is in effect for the structure. However, you can delete a failed-persistent connection to a structure in the Duplex Established phase of user-managed or system-managed duplexing rebuild, as long as a request to switch or stop the duplexing rebuild is not in progress. The connector is forced from both instances of the structure.

Deleting a Structure Dump

You can delete a structure dump associated with either an active coupling facility structure or a structure pending deallocation. Identify the structure dump by specifying the structure dump ID. A structure dump ID of zero designates the structure dump(s) associated with an active instance of the structure. This includes, for a structure being rebuilt, any dumps associated with the rebuild old structure, the rebuild new structure, or both. A non-zero structure dump ID designates the structure dump whose structure dump ID matches the specified value.

Requests to delete a structure dump for structures that are pending deallocation will not be processed unless a non-zero structure dump ID is specified.

If SVC Dump was in the process of capturing information into the structure dump at the time of the IXLFORCE request, the dump will not include any information pertaining to that structure. If SVC Dump was in the process of writing the captured information to the dump data set from the structure dump, the dump will be truncated for that structure.

Deleting Structure Dump Serialization

You can delete structure dump serialization for a structure dump associated with an active structure. This includes, for a structure being rebuilt, any dumps associated with the rebuild old structure, the rebuild new structure, or both. Release of dump serialization for a structure pending deallocation is not supported because the structure would have no active connectors to be impacted by dump serialization. Identify the structure dump for which serialization is to be released by specifying the structure dump ID.

When serialization for the structure dump is released, the structure dump that was in progress for the structure will be truncated. If SVC Dump was in the process of capturing information into the structure dump at the time of the IXLFORCE request, SVC Dump does not capture any additional data, but all the captured information is written to a dump data set. If SVC Dump was in the process of retrieving entry

data serialized, the entry data will be included in the dump, but it may change as it is being written to the dump data set.

Authorizing the Use of IXLFORCE

The security administrator may want to protect the integrity of the data contained in coupling facility structures. The default processing for IXLFORCE is to allow all force requests; the security administrator can override this default with the use of RACF or another security product. See "Authorizing Coupling Facility Requests" on page 204.

Forcing a Structure with Failed-Persistent Connections

A non-persistent structure is deleted when the last connection to the structure is deleted. If, however, the connection to the structure has failed, the connection must be deleted before the system can delete the structure. Consider your environment when deciding how to delete structures with failed-persistent connections:

- In a production environment where data integrity is important, you might restart the application to cleanup or reconnect the failed-persistent connections and use the applications' normal shut down procedure to cause the application to disconnect and stop using the structure. Once there are no longer any connections in the active or failed-persistent state, you can issue the SETXCF FORCE command to force the deletion of the structure.
- In a test environment or when data integrity is not important, use the SETXCF FORCE command to delete each individual failed-persistent connection. When no active or failed-persistent connections remain, you can use the SETXCF FORCE command to force the deletion of the structure.

Coding Exit Routines for Connection Services

All three structure types require both an event exit and a complete exit. The event exit requirements are described here; requirements for the complete exit are described with each of the structure services.

Coding the Event Exit

The event exit receives control in SRB mode with an event exit parameter list (IXLYEEPL) that describes the event being reported. Some events reported by the event exit require that you respond to the event, by setting a return code in IXLYEEPL. One return code that you can set specifies that you intend to do additional asynchronous processing and respond to the event at a later time, using the IXLEERSP macro.

Upon return from the event exit, the connected user no longer can access IXLYEEPL. However, IXLYEEPL contains information that will be required by IXLEERSP if that is the method by which you are responding. You must ensure that you copy the relevant IXLYEEPL data into a control block of your own for subsequent use by IXLEERSP.

Exit Routine Environment

The event exit receives control in the following environment:

Authorization:	Supervisor state and PSW key 0
Dispatchable unit mode:	SRB
Cross memory mode:	PASN = HASN = SASN. The primary address space equals the primary address space of the caller of IXLCONN.
Amode:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held

Exit Recovery

Routines that require recovery must establish their own. If SDWA recording is necessary for the recording of failure information, the exit's recovery routine must provide it. If an SVC dump is required, the recovery routine must also provide for that. Be aware that if the event exit fails for any reason and the XES recovery routine receives control before the exit's recovery routine (or the exit's recovery routine percolated to the XES recovery routine), the XES recovery routine will terminate the connector with a X'026' abend.

Exit Routine Processing

The system reports events to the event exit as they occur so it is possible that the exit of a connected user can receive control before the IXLCONN macro for the connection that is the subject of the event has completed. Therefore, ensure that before you issue IXLCONN, you have the event exit established along with any control structures necessary to complete the exit's processing.

The system passes information to the event exit routine in a parameter list and in registers.

Processing Considerations

Consider the following when writing an event exit routine:

- The event exit routine must be a reentrant program.
- If a connection can perform required processing for an event synchronously in the event exit (that is, the user can respond to the event at the time it occurs), exit processing should not be long-running and should not suspend the event exit SRB. As long as the event exit is running, the connection cannot receive information about other events as they occur because XES serializes its invocations of the event exit on a connection basis.
- It is advisable to check the connection level of the subject of the event. If you are in an environment with connectors of mixed connection levels, it is possible that you might be notified of an event that your event exit has not provided for (because you are processing at a “lower level” of MVS). If that situation occurs, you should set the EEPLRETCODE to X'00' and return to MVS, rather than to treat the unknown or unexpected event as an error.

Macro Instructions and Restrictions

The following restriction applies to the event exit routine:

- Because the event exit runs in SRB mode, the event exit routine cannot issue any macros that issue an SVC or that require the caller to be in task mode.

Input Register Information

On entry to the event exit routine, the general purpose registers (GPRs) contain:

Register
Contents

0
Does not contain any information for use by the event exit

1
Address of a fullword that contains the address of the event exit parameter list (IXLYEEPL)

2-12
Does not contain any information for use by the event exit

13
Address of a 72-byte work area for use by the event exit routine. The exit routine does not have to save or restore registers in this work area. The exit routine can use this work area in any way it chooses.

14
Return address

15
Entry point address

When the event exit receives control, the access registers (ARs) contain no information for use by the event exit.

Output Register Information

When control returns to XES, there are no requirements for the GPRs or ARs to contain any particular value.

Parameter List Contents

The parameter list that the system passes to the event exit routine is mapped by the IXLYEEPL mapping macro. GPR 1 contains the address of a fullword that points to IXLYEEPL. The parameter list is addressable from the primary address space in which the event exit routine runs, and includes the following:

- Information about the connection whose event exit gets control
- Event code.
- Event sequence number.
- Event exit return codes set by the user during exit processing. See "Return Specifications" on page 354.
- Console ID and command-and-response token (CART) for operator-initiated events.
- Information about the connection that is the subject of the event.
- Information about the specific event.

When control returns to the program from the event exit, the connected user can no longer access IXLYEEPL. If the user intends to respond to the event using IXLEERSP, the user must save the following IXLYEEPL information to provide to the IXLEERSP macro:

- Event type
- Event sequence number
- CONTOKEN for the connection that is the subject of an existing connection event
- SUBJCONTOKEN for the Disconnected or Failed Connection event and the Rebuild Connect Failure event.

Return Specifications

Return to XES with the address that was in register 14 upon entry to the event exit.

Depending on the event, set the following return codes in the EEPLRETCODE field in IXLYEEPL:

Return code
Meaning

0
The connected user confirms the event reported to its event exit.

1
The connected user confirms the Existing Connection event or the Disconnected or Failed Connection event and requests that the system release the connection if it is failed-persistent.

8
The connected user does not confirm the event, but intends to issue the IXLEERSP macro to provide a response at a later time.

For information about IXLYEEPL, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Using IXLEERSP

The IXLEERSP macro allows a connected user to provide a response to an event. IXLEERSP must be issued in task mode and can be used to respond asynchronously to events. The connected user should perform the necessary processing for the reported event before issuing the IXLEERSP macro.

A connected user responds to the following events using IXLEERSP:

- A disconnected or failed connection.

  The user can confirm that recovery for the connection is complete and if the connection is failed-persistent, release the failed-persistent state of the connection. Or the user can request that the system continue processing for the failed connection, in which case the connection disposition is not affected.
- An existing connection that is failed-persistent.

  The user can inform the system to release a connection in a failed-persistent state.
- Rebuild Quiesce

  The user is participating in the user-managed structure rebuild process (rebuild or duplexing rebuild) for the structure and has completed the necessary processing to quiesce activity to the structure.
- Rebuild Connect Failure

  The user is participating in the user-managed structure rebuild process (rebuild or duplexing rebuild) for the structure and must clean up any control information about a successful rebuild connect request that was reported by a Rebuild New Connection event or a Rebuild Existing Connection event.
- Rebuild Cleanup

  The user is participating in the user-managed structure rebuild process (rebuild or duplexing rebuild) for the structure and has cleaned up all information about the original structure.
- Rebuild Stop.

  The user confirms the request to stop the user-managed structure rebuild process (rebuild or duplexing rebuild).
- Structure Temporarily Unavailable.

  The user is not required to take any action before responding to the Structure Temporarily Unavailable event, but may optionally quiesce activity against the structure before responding. IBM recommends that connectors quiesce structure activity when presented with the Structure Temporarily Unavailable event to minimize system resources consumed during the system-managed process.

The following data (saved from the IXLYEEPL) must be provided on the IXLEERSP invocation:

- Event type
- Event sequence number
- CONTOKEN for the connection that is the subject of an existing connection event.

The following table shows the synchronous IXLYEEPL return code response to a Disconnected or Failed Connection event or an Existing Connection event and how a user accomplishes the same response asynchronously using IXLEERSP:

Table 20. Comparison of IXLYEEPL and IXLEERSP

Event	IXLYEEPL return code	IXLEERSP keyword	Event response
Disconnected or Failed Connection	0	For a failed connection: EVENT=DISCFAILCONN RELEASECONN=NO	User confirms the event; connection disposition unaffected.

Table 20. Comparison of IXLYEEPL and IXLEERSP (continued)

Event	IXLYEEPL return code	IXLEERSP keyword	Event response
Disconnected or Failed Connection	1	For a failed connection that is failed-persistent: EVENT=DISCFAILCONN RELEASECONN=YES	User confirms the failed connection and releases the connection.
Existing Connection	1	For an existing connection that is failed-persistent EVENT=EXISTINGCONN RELEASECONN=YES	User confirms the failed connection and releases the connection.

Chapter 7. Using Cache Services (IXLCACHE)

This chapter discusses the cache structure, the IXLCACHE macro, and its services. It describes how to use the cache services to access and manage the cache structure and storage. In addition to IXLCACHE, other services are available to users for managing and using a cache structure. (See "Other Services Used with IXLCACHE" on page 402.)

Benefits of Using Cache Services

A cache structure and its related services provide sysplex users with data consistency and high-speed access to data. Data consistency means that users can use cache services and develop protocols to ensure the validity of the data that they share. High-speed access means that users can use cache services to develop data sharing programs and protocols with improved performance.

- **Data Consistency**

  You can store data to be shared among multiple users in the cache structure on the coupling facility. You can also use the cache structure to keep track of data that resides in permanent storage and in local storage but is not stored in the cache structure itself.

  However you store data that multiple users share, each user of the cache structure is expected to maintain a local cache buffer to contain a **copy** of the data. Through the use of a directory in the cache structure and a mechanism called "cross-invalidate" to inform users of changes to data, each MVS system in the sysplex can keep track of whether locally cached copies of the data are valid (that is, whether the copies contain the latest changes).

  The directory allows you to refer to named data items that you can store in the cache structure itself or in local storage. Cross-invalidate processing involves setting an indicator in a local cache vector for each of the users to indicate whether the locally cached copy of the data is valid. Users must test the indicator to determine the validity of their copy, and if the data is no longer valid, users must read the data (either from the coupling facility or permanent storage) to obtain the most current copy.

- **High-speed Access to Shared Data**

  You can use the cache structure to store and access data that users can share, or to keep track of shared data that users maintain in their local cache buffers. Accessing data stored in the local cache buffer is the quickest way for a user to access the shared data. However, if the system has invalidated the local copy because another user has updated the data, you must gain access to the data in another way. Accessing data from the cache structure in the coupling facility is the next fastest way for the user to access the shared data.

  Data in the cache structure is directly accessible to any system in the sysplex that has access to the structure. If you do not store the data in the cache structure, you must read the data from permanent storage (like DASD), which is not as fast as accessing the data from the local cache buffer or from the cache structure in the coupling facility.

Elements of A Cache System

A cache system consists of four major elements:

- Cache structure
- Permanent storage
- Local cache buffers
- Local cache vector

Figure 31 on page 358 shows the four elements and their relationship to each other.

© Copyright IBM Corp. 1994, 2023

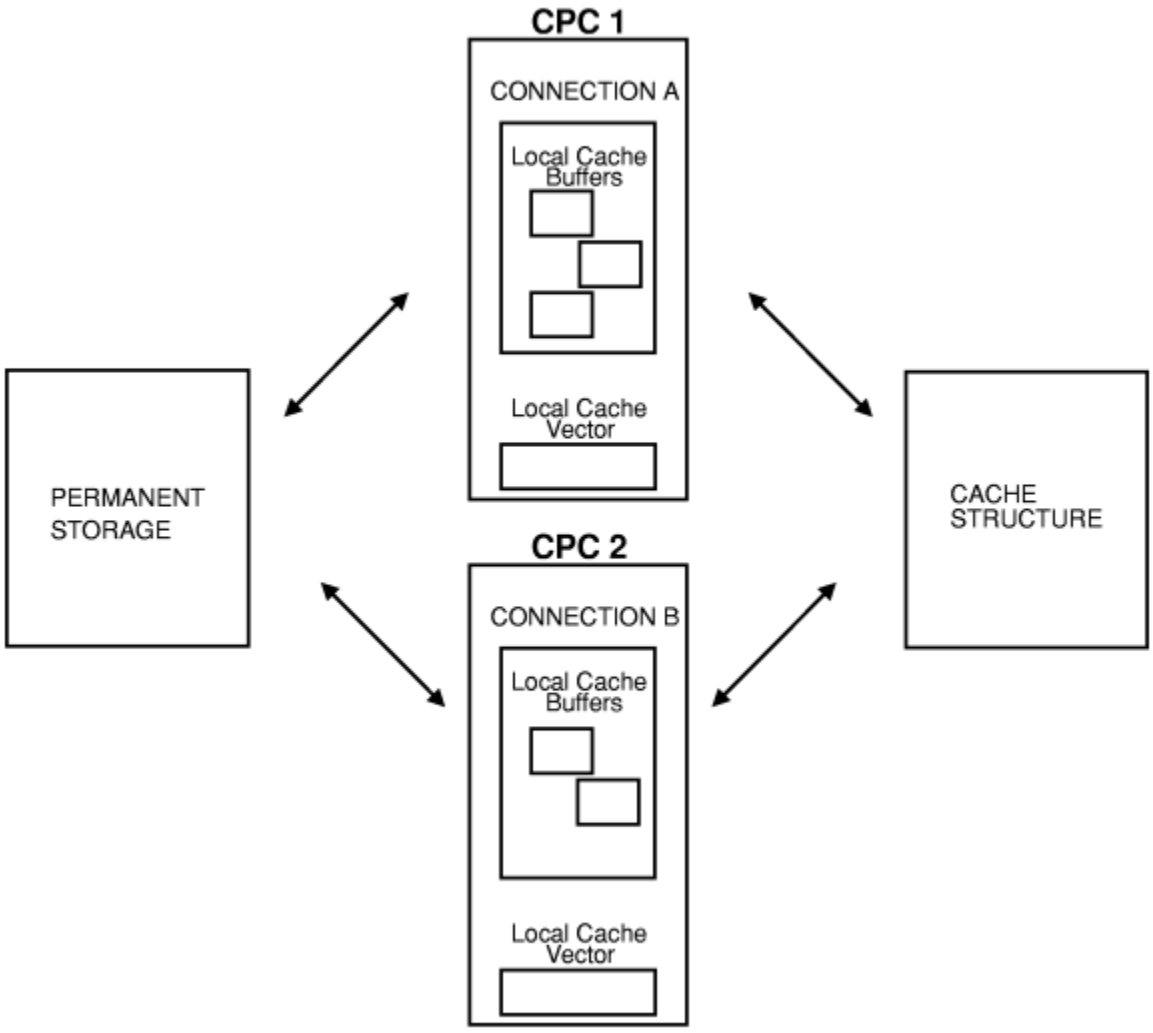


Figure 31. Elements of a Cache System

Each piece of shared data, referred to as a “data item” throughout this chapter , can be stored in different locations within the cache system. Copies of shared data items are stored in the local cache buffers (fastest access) belonging to each cache user. The shared data also resides either in the cache structure on the coupling facility (next fastest access), on permanent storage (slower access to the data than from either the local cache or cache structure) or both the cache structure and permanent storage. In general, how quickly you can access the data depends on where it is stored.

A description of each element of a cache system follows:

- **Local cache buffers** — Local cache buffers are storage buffers that users allocate in their own storage area. They contain copies of data that is shared among cache users. Users read data from permanent storage or from the cache structure to their local cache buffers, and write data from their local cache buffers to permanent storage, to the cache structure, or to both locations. Each user who accesses the cache structure must have a set of local cache buffers to accommodate the data items to be shared.
- **The cache structure** — The cache structure is a structure in the coupling facility that contains:
  - A directory to keep track of named data items that are shared among cache users

– Optionally, data entries that hold data items

Users who are connected to the cache structure can use cache services to access and manage shared data.

- **Permanent storage** — Permanent storage is storage that is the final repository for the data that users share, and might be on a direct access storage device (DASD). Users can read the data from permanent storage to local storage buffers for their use, and then either write the data to the cache structure and maintain the data there, or maintain the data in the local buffers and use the “directory-only” caching method to track the validity of the data. After users make updates to the locally-cached data, they are responsible for ensuring that the changes are made to the permanent storage copy of the data. They make these changes to permanent storage either immediately after the update or at a later time, depending on the cache protocol.
- **Local cache vector** — The local cache vector is a user-defined vector that provides a way for cache users to determine the validity of data in their local cache buffers. There is one local cache vector per user of the cache. Each vector is divided into separate entries with each entry corresponding to a local cache buffer. Each vector entry contains an indicator that the system sets to indicate whether the data in the corresponding local cache buffer is valid. Users must test the indicator to determine the validity of the data in their local cache buffers.

  Because the local cache vector is in system storage and not directly addressable by the user, the system provides the IXLVECTR service. IXLVECTR allows the user to test the entries in the vector to determine whether the corresponding local cache buffer is valid, and to dynamically change the number of entries in the vector.

Elements of a Cache Structure

A cache structure consists of the following major elements:

- A directory consisting of one or more directory entries
- Optional data entries consisting of one or more data elements
- Optional adjunct areas

Figure 32 on page 359 shows the elements of a cache structure. Two different users on separate processors (CPCs) in the sysplex access the cache structure in the coupling facility. A description of each of the cache structure elements follows the figure.

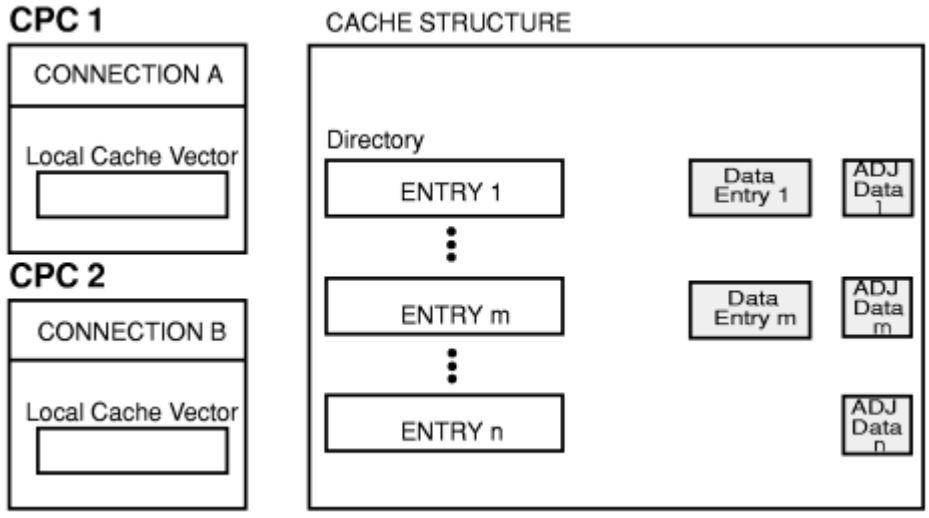


Figure 32. Major Elements of a Cache Structure

- **Directory** — The directory is a directory for the cache structure where the system keeps control information about data items shared among cache users. There is one directory entry for each data

item that users share. Data items can be stored in the cache structure, maintained in each user's local cache buffers, or maintained in both locations.

If a directory entry exists in the cache for a data item (that is, the system has assigned a directory entry to the data item), the data item is said to be "identified" to the cache structure, whether the data item is stored in the cache structure or not. When a data item is identified to the cache structure, each user receives notification through the local cache vector about the validity of the data item. The data item does not need to reside in the cache structure for the system to indicate through the local cache vector whether the data item has been changed. As long as the data item is identified by a directory entry, the system can indicate to users that the data item associated with the directory entry has been changed and is therefore no longer valid. A cache structure that contains directory entries but no data items is referred to as a "directory-only" cache.

For a complete list of the information contained in a directory entry, see "Format of Returned Directory Information" on page 463.

- **Data entry** — A data entry is storage in the cache structure where the system stores a data item that a user writes to the cache structure. (For a "directory-only" cache, the data is not actually stored in the cache structure, so the cache structure contains directory entries but no data entries.) For a data item that exists in the cache structure, the data entry for the data item can consist of from 1 to 16 data elements for a cache structure allocated in a coupling facility of CFLEVEL=0, from 1 to 255 for a cache structure allocated in a coupling facility of CFLEVEL=1 or higher, or from 0 to 255 for a cache structure allocated in a coupling facility of CFLEVEL=4 or higher. Each data element is of a fixed length (from 256 to 4096 bytes). The fixed size of each data element is defined when the structure is allocated and cannot be changed for the life of the structure.

  When a user writes a data item to the structure for the first time, the user specifies the number of data elements that are associated with the data entry. (1 to 16 for CFLEVEL=0, 1 to 255 for CFLEVEL=1 or higher, or 0 to 255 for CFLEVEL=4 or higher.) If the data entry is subsequently overwritten, you can increase or decrease the number of data elements associated with the data entry. You specify the maximum number of data elements that a data entry in the cache structure can support on the MAXELEMNUM keyword of the IXLCONN macro.
- **Adjunct area** — An adjunct area is storage that is separate from the data entry to which users can write data and from which users can read data. Adjunct areas are optional. If you specify adjunct areas, the system provides one 64-byte adjunct area for each allocated directory entry. Users can provide additional data for data entries in the adjunct area or extend the user-data fields of associated directory entries.

Table 21 on page 360 summarizes characteristics of data entries, data elements, and adjunct areas.

Table 21. Data Entry, Data Element, and Adjunct Area Characteristics

Component	Size	When Attributes Are Determined	When Attributes Can Be Changed
Data element	256, 512 1024, 2048, or 4096 bytes	The first user to connect to the structure determines the fixed element size.	Element size is fixed for the life of the cache structure.

Table 21. Data Entry, Data Element, and Adjunct Area Characteristics (continued)			
Component	**Size**	**When Attributes Are Determined**	**When Attributes Can Be Changed**
Data entry		Each user designates the number of data elements as part of each write operation.	Each user can change the number of data elements each time the user writes data to the cache structure.
	0 to 16 elements; **CFLEVEL=0**	The first connector to the structure specifies the actual maximum number of data elements per data entry (**16 or less**) using the MAXELEMNUM parameter of the IXLCONN macro	
	0 to 255 elements **CFLEVEL=1 or higher**	The first connector to the structure specifies the actual maximum number of data elements per data entry (**255 or less**) using the MAXELEMNUM parameter of the IXLCONN macro	
	0 to 255 elements **CFLEVEL=4 or higher**	With a coupling facility of CFLEVEL=4 or higher, the user can specify that 0 elements are to be allocated when writing data to the cache structure only when CHANGED=NO is specified on the WRITE_DATA request.	
Adjunct area	64 bytes	The first user to connect to the structure determines whether the cache structure has adjunct areas.	The presence or absence of adjunct areas is fixed for the life of the cache structure.

Important Terms

The following is a list of terms that you need to understand. These terms describe basic concepts important to the understanding of the cache structure and cache services.

Table 22. Terms for Caching	
Term	**Definition**
cast out/casting out	Process of writing **changed data** that is in the cache structure to permanent storage. Casting out is implemented through the association of data items with **cast-out classes**.
cast-out class	Class assigned to a data item used with cast-out processing. Users of the **store-in** method of caching must assign data items in the cache structure to **cast-out classes**. Cast-out class assignments simplify the **cast-out** process by grouping data items together with similar characteristics. Users must also develop their own cast-out algorithms that make use of these cast-out classes when they cast out data.
cast-out lock	Lock used with a data item for cast-out processing. The user must obtain the data item's **cast-out lock** to serialize the update to permanent storage. When the cast-out lock is held for a data item, the data item is said to be **locked for cast-out**. When a data item is locked for cast-out, the cast-out lock (composed of the connection identifier of the holder of the cast-out lock and, optionally, the process identifier of the task or process that holds the lock) is part of the directory entry for the data item. Any user can still make updates to the data item even when the data item is locked for cast out.

Table 22. Terms for Caching (continued)

Term	Definition
changed data (changed data item)	A **data item** in the cache structure that is an updated version of the same data item on permanent storage. When a user updates the copy of a data item in the local cache buffer and then writes the updated data to the cache structure, the data item is considered **changed** data. If a user has written to the cache structure but has not yet cast out the data to permanent storage, the data in the cache structure is said to be **changed**. A data item that is locked for cast-out processing is also considered changed until the update is made to permanent storage and the cast-out lock is released. An **unchanged** data item is a data item in the cache structure that is the same as the version on permanent storage.
data item	A single unit of information that is referred to by a single name in local cache buffers, the cache structure, and on permanent storage. If a data item is in the cache structure, it is contained in a **data entry**. A user will keep a copy of a data item in a local cache buffer. Wherever copies of the same data item exist, that data item is referred to by a single name.
deregistration/deregistering interest	A way to indicate to users information about the validity of a data item. Users with **registered interest** in a data item can have their interest **deregistered** if the data item has changed and the local copy of the data is no longer valid. When a shared data item is updated, the system indicates to interested users, through the users' associated local cache vector entry, that the data item has been changed. The copy of the data item in users' local cache buffer is then considered **not valid**. This process is also referred to as **invalidation** of local cache copies of data items.
directory-only cache	A cache structure that contains directory entries but not data items. Contrast with **store-in** and **store-through cache**. See "Directory-only Cache" on page 365.
invalidation	See **deregistration/deregistering interest**.
reclaim	The management of resources in the cache structure. When a user writes a data item to the cache structure and a resource like a directory entry or data entry is unavailable, the system attempts to **reclaim** an existing directory entry or data entry to satisfy the request. Not all resources are available for reclaim. For example, a data entry containing changed data cannot be reclaimed. Reclaim is implemented through the association of data items with **storage classes**. Users can define a reclaim vector and use IXLCACHE to control reclaim processing. Otherwise, a system default for reclaim is in effect.
registration/registering interest	A way to indicate to users information about the validity of a data item. Users that use the cache structure can **register** interest in a data item. When a user registers interest in a data item, an association is formed between the local cache vector entry associated with the user's local copy of the data item and the directory entry for the data item in the cache structure. When interest has been registered, the system uses the local cache vector entry to indicate the validity or invalidity of the data in the user's local cache buffer. If a user has registered interest in a data item, the copy of that data item in the user's local cache buffer is considered **valid**.
storage class	Class assigned to a data item in the cache structure used in the reclaim process. Each data item that is defined to the cache structure (either through a directory-only cache structure or a cache structure that contains both directory entries and data entries) must be assigned to a **storage class**. Storage class assignments simplify the **reclamation** of resources by grouping together data items with similar characteristics.

Table 22. Terms for Caching (continued)	
Term	**Definition**
store-in cache	A cache structure in which data items are stored in data entries. Users of a store-in cache write changed data to the cache structure but not to permanent storage at the same time. Users perform an independent cast-out process after the updates have been made and then make the changes to permanent storage. Contrast with **store-through cache** and **directory-only cache**. See "Store-in Cache" on page 363.
store-through cache	A cache structure in which data items are stored in data entries. Users of a store-through cache write changed data to the cache structure and to permanent storage at the same time, that is, under the same serialization. Contrast with **store-in cache** and **directory-only cache**. See "Store-through Cache" on page 364.
valid data	The state of data in a user's local cache buffer. If a user's copy of a data item is **valid**, the copy contains the latest changes. If a data item copy is **not valid**, it does not reflect the latest changes. See also **registration/registering interest**.
validation	See **registration/registering interest**.

Using the Cache Structure

There are three ways to use the cache structure in a cache system:

- As a store-in cache
- As a store-through cache
- As a directory-only cache

Before you use the cache services, evaluate the different characteristics of each method as they apply to your data sharing application.

Store-in Cache

Store-in cache users store data in the cache structure on the coupling facility. The data can be changed data (different from the data on permanent storage) or unchanged (the same as data on permanent storage). What distinguishes the store-in method from other cache methods is that store-in cache users write changed data to the cache structure but do not at the same time write the data to permanent storage (called hardening the data). This means that at any time, the data in the cache structure might contain changes not yet stored (or hardened) in permanent storage. Store-in users must periodically read the changed data from the cache structure and write, or "cast-out" the changed data to permanent storage.

Accessing the Data

The store-in cache user needs to access permanent storage less frequently than do the users of the other methods.

- **Reading a data item** - Users read from permanent storage as a "last resort." First, they check the local cache buffer to determine if the local buffer contains a valid copy of the data. If the data item is not valid in the local cache buffer (that is, the system indicates that the data is not valid as a result of an action taken by another user of the data), they next read the cache structure for the data item. If the data item is not in the cache structure, users finally read the data from permanent storage.
- **Writing a data item** - Users write the changed data to the cache structure. Periodically, they must cast out the data to permanent storage.

Casting out Data from the Cache Structure

The store-in user must develop a protocol for casting out changed data to permanent storage. This protocol includes assigning data items to cast-out classes and developing a cast-out algorithm. (For information, see "CASTOUT_DATA: Casting Out Data from a Cache Structure" on page 427 and "CASTOUT_DATALIST: Casting Out a List of Data Items" on page 432.)

Assigning Storage Classes

The store-in user must assign data items to storage classes to direct the system in reclaiming resources, such as data entries and directory entries, from the cache structure. (For information, see "Assigning and Using Storage Classes" on page 381.)

Recovery

Your program must provide recovery of data in the cache structure. Because the latest changes to data might exist only in the cache structure on the coupling facility, recovery of the data is crucial if the coupling facility or structure fails. When you use the IXLCONN macro to connect to the structure, you might also consider specifying that the cache structure be allocated in a non-volatile coupling facility.

Store-through Cache

Store-through cache users also store changed or unchanged data in the cache structure. Unlike the store-in method, the store-through user writes changed data to the cache structure and to permanent storage ***at the same time*** and under the same serialization so that at any time, the data in the cache structure matches the data in permanent storage.

Accessing the Data

The store-through user generally needs to access permanent storage more frequently than the store-in user:

- **Reading a data item** - Reading a data item for the store-through cache is the same as reading a data item for the store-in cache. See "Store-in Cache" on page 363 considerations above.
- **Writing a data item** - To write a data item, most store-in users write to the cache structure and permanent storage at the same time every time a data item is written.

Casting out Data from the Cache Structure

Because the data is hardened to permanent storage at the same time it is updated in the cache, most store-through users do not need to develop a protocol for casting out changed data to permanent storage or assign data items to cast-out classes.

Assigning storage Classes

All store-through users must assign data items to storage classes to direct the system in reclaiming resources, such as data entries and directory entries, from the cache structure.

Recovery

The store-through method provides improved data availability in comparison to the store-in method because users store data to permanent storage and the cache structure simultaneously. Failure of the coupling facility or the cache structure does not result in lost data; therefore, there is less need to keep data on a non-volatile coupling facility than there is with the store-in method.

Directory-only Cache

Directory-only cache users ***do not*** store data in the cache structure. The directory-only users use the cache structure and cache structure services only to maintain the consistency of data in their local caches.

Accessing the Data

The directory-only user needs to access permanent storage more frequently than the users of the other cache methods:

Reading a data item - Users check the local cache vector entry that corresponds to the data item to determine if the copy of the data is valid. If the local cache buffer does not contain a valid copy, users must read from permanent storage.

Writing a data item - Users must write to permanent storage and use cross-invalidation to invalidate other users' local copies of the data item.

Casting out Data from the Cache Structure

Directory-only users do not need to develop a protocol for casting out changed data to permanent storage or assign data items to cast-out classes.

Assigning Storage Classes

Directory-only users must assign data items to storage classes to direct the system in reclaiming resources, specifically, directory entries from the cache structure.

Recovery

Because users store data to permanent storage only, the directory-only method provides improved data availability in comparison to the store-in method. Failure of the coupling facility or the cache structure does not result in lost data; therefore, there is less need to keep data on a non-volatile structure than there is with the store-in method.

Sizing the Structure

The directory-only cache structure might be very small compared to the other methods because there are no data entries in it.

Focus of this chapter

The remainder of this chapter focuses on how a store-in cache user uses cache structure services. Unlike the store-through or directory-only cache users, the store-in user tends to use all available cache services. When the use of a service or function depends on the cache method being used, the text provides an appropriate explanation for each of method.

Summary of IXLCACHE Requests

To request cache services, you issue the IXLCACHE macro. You identify the service you want by specifying the name of the service on the REQUEST keyword. Table 23 on page 366 identifies, for various IXLCACHE services, how to code the REQUEST keyword, and indicates the cache methods that typically use the service. An “X” indicates the request is typically used with the corresponding cache method. Where necessary, notes provide additional clarification. The table also provides references to the topics where the individual requests are discussed in detail.

Table 23. Description of IXLCACHE Services

To request a service to:	Code REQUEST=	Store-in	Store-through	Directory-only	Where described
Define and write a new data item to the cache structure, and register interest in the data item.	WRITE_DATA	X	X		"WRITE_DATA: Writing a Data Item to a Cache Structure" on page 402
Write a changed data item to the cache structure and invalidate any copies of the data item that are in other users' local cache buffers.	WRITE_DATA	X	See note "1" on page 367		"WRITE_DATA: Writing a Data Item to a Cache Structure" on page 402
Write data from a list of entries to the cache structure, and register interest in the data items.	WRITE_DATALIST	X	X		"WRITE_DATALIST:Writing Multiple Data Items to a Cache Structure" on page 412
Read a data item from a cache structure to your local cache buffer and register interest in the data item.	READ_DATA	X	X		"READ_DATA: Reading a Data Item from a Cache Structure" on page 417
Define a directory entry for a new data item to the cache structure and register interest in that data item.	READ_DATA			X	"READ_DATA: Reading a Data Item from a Cache Structure" on page 417
Register interest in a list of data items.	REG_NAMELIST See note "3" on page 367	X	X	X	"REG_NAMELIST: Registering Interest in a List of Data Items" on page 421
Lock a data item for cast-out and read the data item from a cache structure to your local cache buffer for the purpose of writing the data to permanent storage. Also mark the data item as unchanged.	CASTOUT_DATA	X			"CASTOUT_DATA: Casting Out Data from a Cache Structure" on page 427
Lock a set of data items for cast-out and read the data items from a cache structure to your local storage for the purpose of writing the data to permanent storage. Also mark the data item as unchanged.	CASTOUT_DATALIST	X			"CASTOUT_DATALIST: Casting Out a List of Data Items" on page 432
Unlock cast-out locks that you previously obtained.	UNLOCK_CASTOUT	X	X		"UNLOCK_CASTOUT: Releasing Cast-Out Locks" on page 434
Unlock a single cast-out lock that you previously obtained.	UNLOCK_CO_NAME	X	X		"UNLOCK_CO_NAME: Releasing a Single Cast-Out Lock" on page 439
Invalidate other user's local copies of a data item.	CROSS_INVAL			X	"CROSS_INVAL: Invalidating Other Users' Copies of Data Items" on page 449
Invalidate other user's local copies of a set of data items.	CROSS_INVALLIST			X	"CROSS_INVALLIST: Invalidating a List of Data Items" on page 450
Delete one or more data items from a cache structure and deregister all users' interest.	DELETE_NAME	X	X	X	"DELETE_NAME: Deleting Data Items From a Cache Structure" on page 442
Delete one or more data items from a cache structure and deregister all users' interest.	DELETE_NAMELIST	X	X	X	"DELETE_NAMELIST: Deleting a List of Data Items" on page 446

Table 23. Description of IXLCACHE Services (continued)

To request a service to:	Code REQUEST=	Store-in	Store-through	Directory-only	Where described
Activate, deactivate, or change a reclaim vector.	SET_RECLVCTR	X	X	See note "2" on page 367	"SET_RECLVCTR: Overriding or Restoring the Default Reclaim Algorithm" on page 452
Mark as recently referenced one or more data items, and move the data item(s) to the end of the storage class queue as most recently used.	PROCESS_REFLIST	X	X	See note "2" on page 367	"PROCESS_REFLIST: Marking Data Items as Referenced" on page 457
In the associated directory entry for the specified data item(s) in the cache structure, indicate as not recently referenced and return a count of the number of data entries that currently have the reference bit set.	RESET_REFBIT	X	X	X	"RESET_REFBIT: Marking Data Items as Unreferenced" on page 459
Read directory information for one or more data items.	READ_DIRINFO	X	X	X	"READ_DIRINFO: Reading Cache Directory Entries" on page 461
Read cast-out class information for one or more data items.	READ_COCLASS	X			"READ_COCLASS: Reading A Cast-Out Class" on page 465
Read cast-out class statistics for one or more cast-out classes.	READ_COSTATS	X			"READ_COSTATS: Reading Cast-Out Class Statistics" on page 468
Read storage class statistics for a specified storage class.	READ_STGSTATS	X	X	See note "2" on page 367	"READ_STGSTATS: Reading Storage Class Statistics" on page 472

Note:

1. Store-in users mark the data as changed. Store-through users mark the data as unchanged because they intend to immediately update the data on permanent storage.
2. Directory-only users might use the request to manage reclamation of directory storage.
3. REG_NAMELIST users can also use the request to define a directory entry for a new data item and register interest in the data item, as with a READ_DATA request.

Cache Structure Allocation and Connection

Before each user can use the cache structure, the user needs to issue the IXLCONN macro to connect the user's instance or image of the application to the structure. When the first user connects to the cache structure, the system allocates resources for the structure and assigns structure characteristics. The coupling facility resource management (CFRM) policy defines the names of cache structures to the systems in the sysplex. The CFRM policy also defines, among other structure characteristics, the maximum amount of coupling facility storage that you can allocate to a structure.

Note: If the amount of storage requested for the structure is not available, the system allocates as much storage as is available and issues messages to indicate how much storage has been allocated.

When the user connects to a cache structure, the user identifies the structure by name. The name must be defined in the active CFRM policy. If a structure by that name is already allocated, the system connects the user to the structure. If the structure has not been allocated and the user wishes to allocate the structure, and if coupling facility resources are available, the system allocates coupling facility resources for the structure, connects the first user to the structure, and assigns attributes for the structure and the connection specified on the IXLCONN macro. Once a structure is defined, other users can connect to the structure.

- **Defining Structure and Connection Characteristics for Cache**

  Characteristics that the user can specify on the IXLCONN macro for the structure include:

- Structure disposition
- Structure size
- Amount of storage available for the directory and for data expressed as a ratio of directory entries-to-data elements
- Maximum number of data elements per data entry and the data element size
- Whether the structure supports adjunct areas
- Maximum number of storage classes and cast-out classes available to the structure
- Whether the structure supports user data field (UDF) queues for each cast-out class for the structure (requires CFLEVEL=5 or higher).
- Whether the structure supports the logical grouping of name classes. Name classes can be used in conjunction with the NAMECLASSMASK specified on IXLCONN for more efficient use of some cache requests (requires CFLEVEL=7 or higher). See "Using Name Classes in a Coupling Facility" on page 444 for additional information.

Structure characteristics remain fixed for the life of the cache structure (that is, as long as the structure remains allocated.) Whenever a user connects to a previously existing cache structure, the user cannot change the structure characteristics. However, a user is able to change some of the structure characteristics by rebuilding or altering the structure. For information, see the IXLCONN and IXLREBLD macros.

Characteristics that the user can specify on the IXLCONN macro for the connection to a cache structure include:

- Connection name
- Connection disposition
- Size of the local cache vector
- Cross-invalidation operation mode processing (synchronous or asynchronous) for local cache cross-invalidations generated by commands issued by the connection

With the exception of the size of the local vector, connection characteristics remain fixed for the life of the connection (that is, as long as the user remains connected to the structure). Other users can connect to an existing structure and define their own connection characteristics. To change the vector size, users can issue the IXLVECTR at any time during the connection.

For more information about defining a CFRM policy and about allocating and connecting to a cache structure, see:

- *z/OS MVS Setting Up a Sysplex*
- Chapter 6, "Connection Services," on page 203

- **Specifying the Appropriate CFLEVEL**

  When you connect to a cache structure, you should be aware of your application's CFLEVEL requirements. Different levels of coupling facility control code (CFCC) support different coupling facility functions. For example, if your application is going to use the IXLALTER service to change the structure size, you should specify CFLEVEL=1 or higher on your IXLCONN invocation.

- **Defining the Local Cache Vector**

  When you connect to a cache structure, one of the characteristics you specify is the length of the local cache vector. The local cache vector is a mechanism for determining if your locally cached data is valid. Each cache structure user must have a local cache vector allocated. The user of IXLCACHE services needs one vector entry for each local cache buffer. Or, put another way, the vector length needs to be the same as the maximum number of data items that you intend to have concurrently available in your private storage.

  The amount of storage available for local cache vectors is finite. Therefore, you need to define a vector length that is only as large as the length you actually need. If you need to change the storage for the vector (for example, at some point you might need to keep track of more or fewer data items), you can use the IXLVECTR macro to increase or decrease the size of the vector.

For More Information

For more information about local cache vectors and the IXLVECTR macro, see “Maintaining Data Consistency” on page 373. For information about the IXLVECTR macro keywords, see “Using the IXLVECTR Macro” on page 680.

Accessing and Managing Data Within a Cache System

When you initially allocate a cache structure using the IXLCONN macro, the structure contains no user-defined data. If you plan to use a store-in or store-through cache method, you store the data in the cache structure. First, you read the data from permanent storage to your local cache buffers. Next, you use the IXLCACHE macro to write the data from the local cache buffers to the cache structure. (See Table 23 on page 366 for a summary of IXLCACHE request types and services to use with the cache structure.)

- **Providing the Connect Token (CONTOKEN)**

  Each user must issue any IXLCACHE request in the connector's address space, that is, from the address space where the IXLCONN macro for the connection is issued. To identify the connection, your IXLCACHE request MUST include the CONTOKEN keyword. (CONTOKEN must contain the connect token that the system returns to the answer area of IXLCONN when the user issues the IXLCONN macro to establish the connection to the cache structure. The system returns the connect token in the CONACONTOKEN field of the answer area for IXLCONN.)

- **Providing a Request Identifier (REQID)**

  To identify your request, you can optionally use the REQID keyword. Coding REQID is useful for recovery routines, or for developing protocols to use with a resource manager that needs to purge coupling facility requests from the system through the use of the IXLPURGE macro. One way to use IXLPURGE is to purge only those requests for a specified connect token (that is, requests associated with a specified connector to the cache.) Specifying the REQID keyword on an IXLCACHE request provides a means for the resource manager to further limit or filter the set of requests that it purges to include only requests for both the specified connect token and the REQID. Users of each connection are responsible for establishing protocols for the use of the REQID keyword and the IXLPURGE macro.

Managing Local Cache Buffers

You are responsible for maintaining local cache buffers for data items. To refer to the data items and allow the system to track the data in the local cache buffers, you need to define a local vector entry index. You assign an index value to correspond to each data item in a local cache buffer. By using the local vector index value for the data item on IXLCACHE requests to the cache structure, the system can communicate to all users whether a user registers interest in the data item and whether the data in the local cache buffer for the data item is valid.

The number of local cache buffers that you define depends on how many data items you want to have concurrently available in your private storage. You can use one local cache buffer to share one data item concurrently among users, two buffers to share two data items, and so forth.

You can change the local cache buffers for a data item. As a result, you need to indicate that change to the cache structure. For example, if you assign a buffer for data item A to a new local cache buffer called data item B and plan to use the same local vector entry index to refer to the data, you need to deregister interest in data item A and register interest in data item B in the cache structure. IXLCACHE provides an OLDNAME keyword to allow you to deregister interest on read, write, or cast-out requests. If you plan to reassign the local vector entry index for a data item to another data item, you also need to reflect that change so the system can invalidate the local vector entry index value for the original data item.

Note that with a coupling facility of CFLEVEL=2 or higher, you can control the processing of a WRITE_DATA request by specifying VECTORINDEX on the WRITE_DATA,WHENREG=YES request. If you have not already registered interest in the data item, or if the VECTORINDEX does not match the local vector index with which you previously registered interest, the WRITE_DATA request will fail with reason code IXLRSNCODENOENTRY.

For More Information

For more information about managing the local cache buffers, see:

- "Selecting a Data Buffer For a Request" on page 388
- "Design Considerations for Choosing the Buffer Format" on page 391
- "Specifying the Vector Entry Index on IXLCACHE Requests" on page 397

For information about IXLVECTR, see "Using the IXLVECTR Macro" on page 680.

Identifying a Data Item to the Cache Structure

If a data item is in the cache structure, it is said to be "identified" to the cache. A data item is identified to the cache structure when a user allocates a directory entry for the data item. (For the directory-only cache method, the data item itself does not reside in the cache structure.)

When you identify a data item to the cache structure, you assign the data item a name. This name identifies the data item to the cache structure. All references to the data item must be by the assigned data item name.

Reading, Writing, or Registering Interest in a Data Item

You can identify a data item to the cache structure by writing the data item to the structure, reading the data item from the structure, or registering interest in a list of data items with a REG_NAMELIST request. You can use the WRITE_DATA or WRITE_DATALIST request on IXLCACHE to write the data item to the cache structure. (The data item can be new or changed.) If a directory entry for a named data item does not exist in the cache structure, you can use a READ_DATA request or a REG_NAMELIST request on IXLCACHE to allocate a directory entry for the data item(s) in the cache structure. (The READ_DATA request allows you to define directory entries in the cache structure for use in a directory-only cache.) If the data item exists in the cache structure, the READ_DATA request on IXLCACHE reads the data into the local cache buffer for the named data item. The REG_NAMELIST request returns an indication as to whether there is data associated with the entry along with other entry state information.

Determining the Validity of a Data Item

When you identify a data item through READ_DATA, REG_NAMELIST, WRITE_DATA, or WRITE_DATALIST requests on IXLCACHE, the system also registers your connection as having interest in the data item. Having registered interest ensures that the system can indicate, through each user's local cache vector, whether the user's locally cached copy of the data for the data item is valid.

Defining a Storage Class for a Data Item

Whenever you use IXLCACHE requests to create a directory entry for a data item or to write a data item, you must also specify a storage class for the data item. The system uses the data item's storage class assignment to reclaim storage in the cache structure for new requests.

For More Information

For information about identifying a data item to the cache structure, see:

- "WRITE_DATA: Writing a Data Item to a Cache Structure" on page 402
- "WRITE_DATALIST:Writing Multiple Data Items to a Cache Structure" on page 412
- "READ_DATA: Reading a Data Item from a Cache Structure" on page 417
- "REG_NAMELIST: Registering Interest in a List of Data Items" on page 421

For information about registering interest in a data item, see "Maintaining Data Consistency" on page 373.

For information about assigning a storage class, see "Managing Cache Structure Resources" on page 380.

Changing a Data Item in the Cache Structure

Consider a store-in cache that includes a cache structure with data items allocated to data entries. When you read a data item from the cache structure, you read it into your local cache buffer. Once the data item is in your local cache buffer, you can use it as is or change the data. (In a cache system, a data item is considered changed if the copy in the cache structure contains data that is not the same as the data in the buffer or the data on permanent storage.)

You use the IXLCACHE REQUEST=WRITE_DATA or REQUEST=WRITE_DATALIST to write the changed data item back to the cache structure. On the request, you indicate that the data item has changed. The system modifies the directory entry for the data item to indicate that the data is changed and invalidates the locally cached copies of the same named data item for other users. When other users reference the data item and test for validity, the system indicates that their local cache copies of the data item are not valid, and they must refresh their local cache buffers to reflect the changes to the data item.

The system has a safeguard that prohibits you from overwriting changed data in the cache structure. For instance, if the data you read into your local cache buffer is changed, and you indicate on the write request to the cache structure that the data item is unchanged, the system fails the request.

Casting out Changed Data

When you write changed data to the cache, you must assign the data to a cast-out class. Cast-out class assignments help to implement the cast-out process whereby changed data from the cache structure is written to permanent storage. Until the data item is successfully cast out from the cache structure, the system cannot reclaim resources for the changed data item. The system considers a data item to be changed, and thus ineligible for reclaim, if either of the following conditions is true:

- The data item's directory entry is marked changed.
- The data item's cast-out lock is held. (When the cast-out lock is released, the data item is considered unchanged.)

For more information about casting out data, see “Casting out Data or Updating Permanent Storage” on page 372.

Considerations Using the Store-through Cache Method

If you use the store-through cache method, you write copies of the same data to the cache structure and permanent storage. Therefore, when you write a changed data item to the cache structure, you indicate on the write request that the data item is unchanged because, at the same time, you intend to write the same data item to permanent storage. (Remember that data is considered changed in a cache system when any copy of the data in the cache structure has been updated and is no longer the same as the data in permanent storage.) Using IXLCACHE, you can also request the cast-out lock for the data item to serialize the update to permanent storage, and request that the system invalidate copies of the data item in the local buffers of the other users.

Considerations Using the Directory-only Cache Method

Directory-only users do not write data to the cache structure. The directory-only user identifies a data item to the cache structure on the IXLCACHE READ_DATA or REG_NAMELIST request and creates a directory entry for the data item. (The user assigns a directory entry to the data item by specifying ASSIGN=YES on the READ_DATA request or by setting an “assignment control” bit on the REG_NAMELIST request.) Because there is only a directory entry for the data item and no data in the cache structure, the directory entry cannot be marked as changed.

For more information about writing a changed data item to the cache structure, see:

- “Casting out Data or Updating Permanent Storage” on page 372
- “WRITE_DATA: Writing a Data Item to a Cache Structure” on page 402
- “WRITE_DATALIST:Writing Multiple Data Items to a Cache Structure” on page 412

Casting out Data or Updating Permanent Storage

The process of writing changed data from the cache structure to permanent storage is called ***casting out*** data. Casting out data does not delete the data from the structure.

Considerations for Cast-out Using the Store-in Cache Method

In the store-in cache method, you must assign changed data items to cast-out classes. You must determine the criteria to use with these cast-out class assignments. You can group data items with similar "cast-out frequency" in the same cast-out class. For instance, you could assign data items that are to be cast out frequently to one cast-out class, and data items that could be cast-out infrequently to a different cast-out class. Whenever you are ready to cast out, you cast out all the data items belonging to a certain cast-out class at the same time.

Before you cast out data by cast-out class, you can use IXLCACHE REQUEST=READ_COSTATS to obtain information about data items in the castout class. When you have a sufficient number of data items to cast out, you can use IXLCACHE REQUEST=READ_COCLASS to determine the names of the data items. Then, when you are ready to write the data items to permanent storage, you issue IXLCACHE REQUEST=CASTOUT_DATA once for each data item or IXLCACHE REQUEST=CASTOUT_DATALIST for a list of data items.

If the cache structure is allocated with user data field (UDF) order queues (supported by CFLEVEL=5 or higher), the system maintains a queue for each cast-out class for which user-defined data was written to the directory entry. The queue is ordered in ascending order by the UDF field value. You can use the REQUEST=READ_COCLASS,COSTATSFMT=COSTATSLIST invocation to request that the system return for each cast-out class, the count of data elements and the user data on the queue with the smallest user data value. How you use this data is determined by your own protocol.

The cast-out service of IXLCACHE allows you to read the data item that you intend to write to permanent storage from the cache to your local cache buffer. The service also gives you the cast-out lock for the data item so you can serialize the update of the data item on permanent storage. While you hold the lock, the data item is said to be ***locked for cast-out***, and other users cannot cast out the data item. However, any user can update the data item in the cache even if it is locked for cast-out. Resources for a data item that is locked for cast out cannot be reclaimed.

The CASTOUT_DATA and CASTOUT_DATALIST requests update the directory entry of each data item to indicate unchanged data. To write the data item to permanent storage, use the access method that you normally use to access permanent storage. After completing the write operation, use IXLCACHE REQUEST=UNLOCK_CASTOUT or REQUEST=UNLOCK_CO_NAME to release the cast-out lock. You can issue the UNLOCK_CASTOUT request once to free multiple locks that belong to data items within a certain cast-out class or you can issue the UNLOCK_CASTOUT request to free a single lock. The UNLOCK_CO_NAME request allows you to free only a single lock, and is a more efficient method than UNLOCK_CASTOUT for releasing a single lock.

Considerations for Cast-out Using the Store-through Cache Method

With the store-through cache method, you write to permanent storage and to the cache structure at the same time and with the same serialization. Thus, you do not need to obtain the cast-out lock to serialize the update to permanent storage.

For recovery in a multisystem environment, you can optionally obtain the lock through the IXLCACHE REQUEST=WRITE_DATA with the GETCOLOCK=YES option. If the system that performs the cast out obtains the lock and fails, users on other systems can recognize that data items locked by the user might not be valid as a result of the failure.

When you write the changes to the cache structure, you request that the system invalidate the copies of the data item for other users. To write the data item to permanent storage, use the access method that you usually use to access permanent storage. After completing the write operation, use IXLCACHE REQUEST=UNLCOCK_CASTOUT or REQUEST=UNLOCK_CO_NAME to free the cast-out lock. You can use the UNLOCK_CASTOUT request once to free multiple locks at the same time you can issue the

UNLOCK_CASTOUT request once to free a single lock. The UNLOCK_CO_NAME request allows you to free only a single lock, and is a more efficient method than UNLOCK_CASTOUT for releasing a single lock.

Considerations for Cast-out Using the Directory-only Cache Method

If you use the directory-only cache method, you only write updates to permanent storage. You do not write data to the cache structure and, as a result, do not need to issue requests for cast out. To write the data item to permanent storage, use the access method that you normally use to access permanent storage. Immediately before or after you write the data item to permanent storage, ensure that you invalidate copies of the data item that other users maintain in their local cache buffers by using IXLCACHE REQUEST=CROSS_INVAL. The CROSS_INVAL request must be invoked under the same serialization used to update the data item.

For more information about updating permanent storage, see:

- “WRITE_DATA: Writing a Data Item to a Cache Structure” on page 402
- “CASTOUT_DATA: Casting Out Data from a Cache Structure” on page 427
- “CASTOUT_DATALIST: Casting Out a List of Data Items” on page 432
- “UNLOCK_CASTOUT: Releasing Cast-Out Locks” on page 434
- “UNLOCK_CO_NAME: Releasing a Single Cast-Out Lock” on page 439
- “CROSS_INVAL: Invalidating Other Users' Copies of Data Items” on page 449
- “CROSS_INVALLIST: Invalidating a List of Data Items” on page 450
- “READ_COCLASS: Reading A Cast-Out Class” on page 465
- “READ_COSTATS: Reading Cast-Out Class Statistics” on page 468

Maintaining Data Consistency

Each time a connecting user issues the IXLCACHE macro to read data, write data, or optionally, cast out data, the system registers the interest of the user in the data item. It also indicates, in a local cache vector entry that the user specifies, that the copy of the data item is valid. (A valid copy of a data item is one that contains the latest updates to the data item that other users might have made.)

Registering interest allows the system to “remember” that the local cache buffer of the user contains a valid copy of the data item. If a user changes the data item and writes the data item to the cache structure, the system deregisters interest in the data item for the other users and indicates in their local cache vector entry that the copy is no longer valid. Each connecting user must test the validity of the locally cached copy by testing the vector entry associated with the data item. Each user also needs to ensure that there is external serialization for the data item between the time the user invokes IXLVECTR to test the validity of the data item and the time when the user makes use of the data.

Registering Interest in a Data Item and Validating Local Copies

When you register interest, you must specify an entry in the local cache vector (VECTORINDEX keyword) that you have assigned to the data item. The system uses the vector entry to indicate the validity of the associated data item in your local cache buffer. Figure 33 on page 374, shows data item X in the local storage buffer of the connecting user A. The data item is valid because vector entry 2 — the vector entry that connection A assigned to data item X, indicates that the data is valid.

The system keeps track of users, the validity of copies of their data items, and the vector entries for each user in the directory entry for each data item in the cache structure. In Figure 33 on page 374, the directory entry for data item Z shows that connecting users A and B have registered interest in data item Z (that is, the connections have valid copies of data item Z). If a third connection updates Z in the cache structure, the system uses the assigned vector entries (entry 5 for connection A and entry 4 for connection B) to invalidate the local copies belonging to connections A and B.

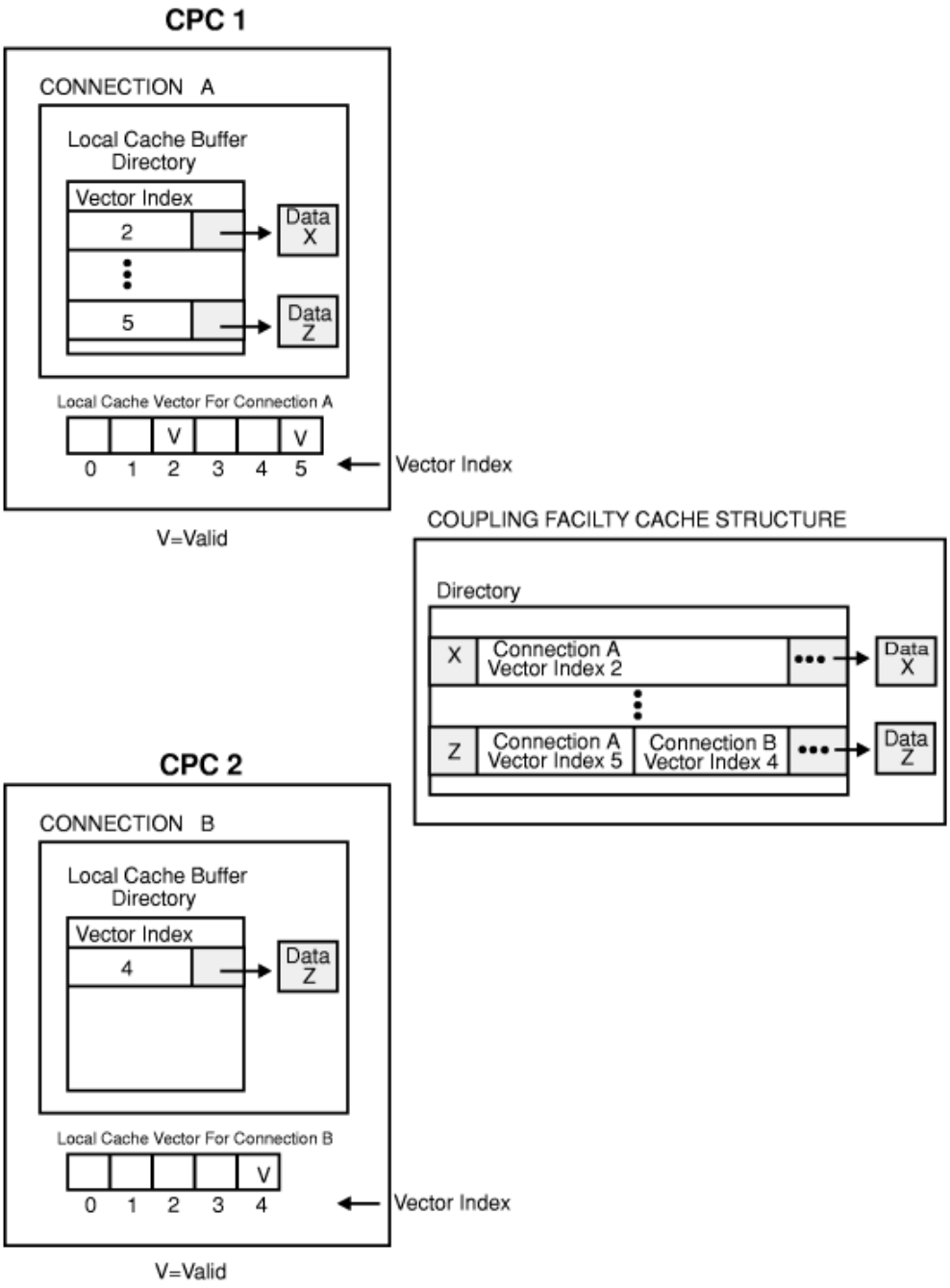


Figure 33. Registered Interest in Data Items

Maintaining Connections between the Local Cache Vector and Data Items

Although the system maintains the connection between cached data and your local cache vector, you must establish and maintain the connection between the local cache vector and your locally cached data. Figure 33 on page 374 shows how users use a local cache buffer directory to maintain the connection between vector entries and locally cached data items.

Registering Interest in a Data Item

When you perform any of the following tasks, you cause the system to register or re-register your interest in a data item and update your local cache vector to indicate that your locally cached copy of the data item is valid:

- Read a data item from the cache structure (REQUEST=READ_DATA).
- Write a data item to the cache structure (REQUEST=WRITE_DATA,WHENREG=NO).
- Read a data item from the cache structure for cast-out and request to register interest (REQUEST=CASTOUT_DATA,REGUSER=YES).
- Specify a list of data items to register interest in (REQUEST=REG_NAMELIST)

When re-registering interest in a data item and the specified local cache vector entry replaces the previously specified local cache vector entry for the connection and data item, the old local cache vector entry associated with the local cache copy of the data item is invalidated.

Deregistering Interest in a Data Item and Invalidating Local Copies

Figure 34 on page 376 shows what happens when connection A updates data item Z in the cache structure. The system invalidates the copy of data item Z belonging to connection B using local cache vector entry 4—the vector entry that connection B assigned to data item Z. Notice also that the cache structure directory shows that only connection A has registered interest in data item Z; connection B has been deregistered.

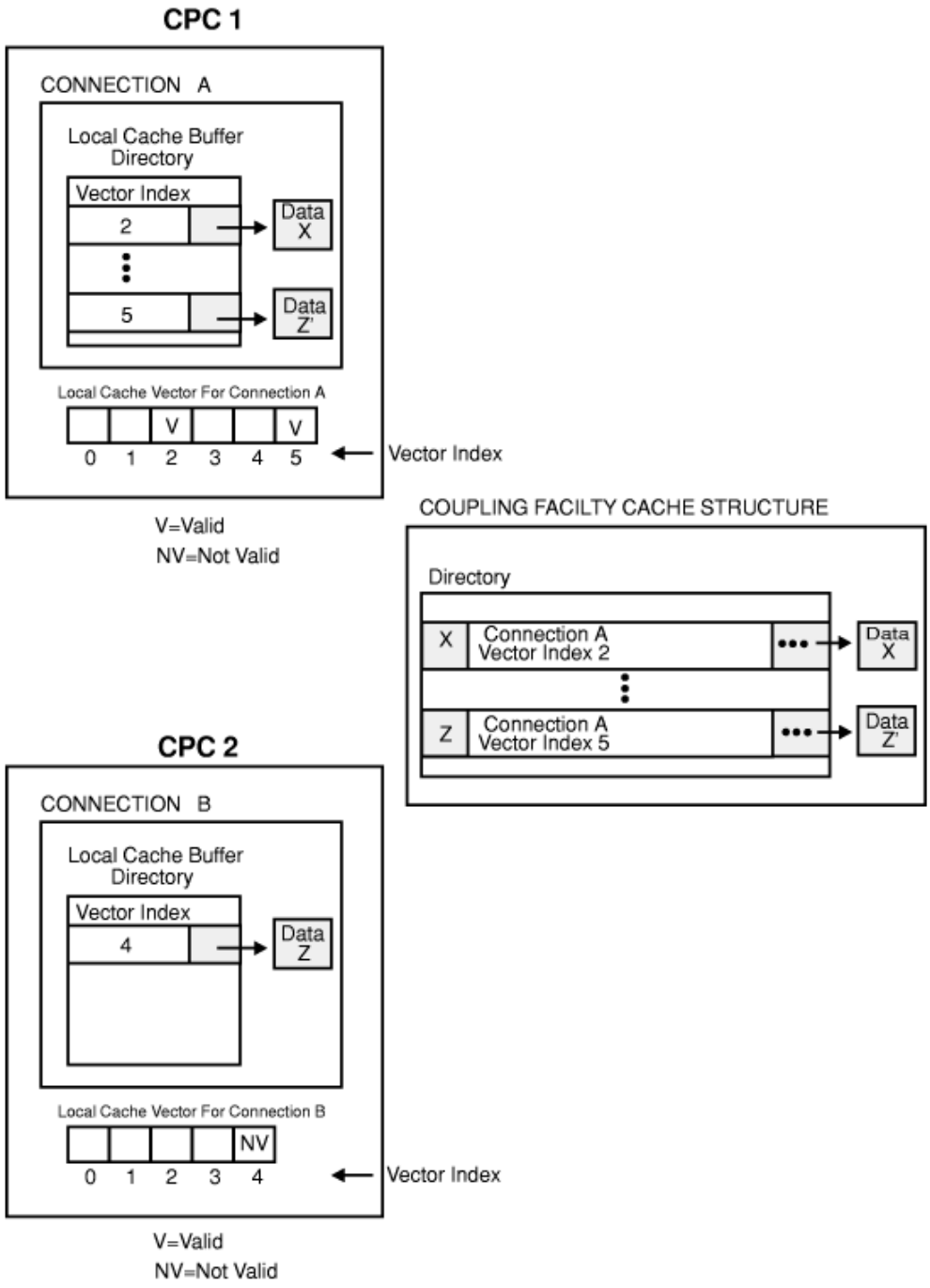


Figure 34. Invalidating Local Cache Copy of a Data Item

Invalidating Local Cache Copies of a Data Item

When any of the following events occur, the system invalidates your local copy of a data item and deregisters your interest in the data item:

- A user writes an updated copy of the data item to the cache structure (REQUEST=WRITE_DATA,CHANGED=YES, REQUEST=WRITE_DATALIST).
- A user requests that the system invalidate copies of the data item (REQUEST=CROSS_INVAL, REQUEST=CROSS_INVALLIST).

- A user deletes a data item from the cache structure (REQUEST=DELETE_NAME, REQUEST=DELETE_NAMELIST).
- A user requests that the system write unchanged data and invalidate copies of the data item (REQUEST=WRITE_DATA,CHANGED=NO,CROSSINVAL=YES).
- You reassign the vector entry for the data item to another data item (REQUEST=WRITE_DATA, REQUEST=READ_DATA, or REQUEST=CASTOUT_DATA).
- The system reclaims directory entry resources for the data item.

Determining the Validity of a Data Item through IXLVECTR

Before you use your copy of a data item, you must test the local cache vector entry assigned to that data item to determine if the copy is current. IXLVECTR REQUEST=TESTLOCALCACHE enables you to test the vector entry for the validity of a data item in your local buffer.

In Figure 34 on page 376, connecting user B checks vector entry 4 to test the validity of data item Z. In the example, the vector entry indicates that the data is not valid. If a data item in your local cache buffer is not valid, IXLVECTR returns an appropriate response. To refresh data item Z in the local buffer, connection B must read the data item from the cache structure and again register interest in data item Z.

The IXLVECTR macro also allows you to test for connectivity failure between your system and the coupling facility through the VALIDATE=YES option. If connectivity to the coupling facility is interrupted, specifying VALIDATE=YES on IXLVECTR allows the system to invalidate the local cached copy of the data item. This helps ensure data integrity because cross-invalidate might not have occurred during the temporary loss of connectivity.

Changing the Size of the Local Cache Vector

The number of entries in the local cache vector determines the number of data items for which you can have concurrently registered interest. By maintaining a local cache vector that contains only the number of entries you need, you can optimize the use of vector storage that other applications might need. You can increase or decrease the vector size to meet the needs of your data sharing. To change the vector size, use IXLVECTR REQUEST=MODIFYVECTORSIZE.

For more information about the use of the IXLVECTR macro, see “Using the IXLVECTR Macro” on page 680.

Serializing and Managing Access to Shared Data

When you share data with other users, you must establish protocols for serializing the use of, and updates to, the shared data. The objectives of these protocols are to ensure that:

- Changes made to cached data by one user are not subsequently overwritten with down-level data by another user.
- Data that you are using (that is, data in your local cache buffer) contains the most recent changes made by other users.

To meet these objectives, IBM recommends that you serialize accesses to cached data.

Whether you serialize access to shared data and the serialization methods that you use are your decisions. While it is generally true that cache services do not automatically provide serialization, cache services might provide some serialization to suit your needs. When you write data to the cache, you can optionally specify that your data be written only if your local copy of the data item is still valid (REQUEST=WRITE_DATA,WHENREG=YES). If another user updates the data in the cache after you read it (which causes your local copy to be invalidated), you cannot overwrite the update. However, to use this method, you do not hold serialization on reading the data, so the data in your local cache buffer might be downlevel data from the data in the cache structure.

You can provide other forms of serialization outside the scope of the IXLCACHE macro. For example, you can use locking services available through the IXLLOCK macro to serialize cache resources. See Chapter 10, “Using Lock Services (IXLLOCK),” on page 625.

The following scenarios show ways to serialize and manage shared data access for store-in, store-through, and directory-only cache methods.

Using but not Updating Data in a Store-in Cache

You are a store-in user who plans to use a data item but not update it. Serializing this process ensures that no one updates the data item while you hold the lock.

1. Obtain a shared lock by using the IXLLOCK macro. Holding a shared lock enables others to use, but not update, the data item.
2. If there is a copy of the data item in your local cache buffer, use the IXLVECTR macro to determine if the copy is valid. If the copy is valid, go to step "5" on page 378.
3. If a copy does not exist in the local cache buffer or the copy is no longer valid, use IXLCACHE REQUEST=READ_DATA to read the data item from the cache structure. If the data item is in the cache structure, go to step "5" on page 378. If a local cache buffer is not assigned to the named data item, assign the local cache buffer and use a protocol to assign a vector entry to the buffer.
4. If the data item is not in the cache structure, read it from permanent storage into the local cache buffer.
5. Use the data item as needed.
6. If the data item has been read from permanent storage, use IXLCACHE REQUEST=WRITE_DATA to write the data item to the cache structure. The system registers your interest in the data item. Otherwise, skip this step and go directly to step "7" on page 378.
7. Use the IXLLOCK macro to free the shared lock.

Updating Data in a Store-in cache

You are a store-in user who plans to update the data item and then write it back to the cache structure. Serializing this process ensures that no other user can use or update the data item while you hold the lock.

1. Obtain an exclusive lock by using the IXLLOCK macro. Holding an exclusive lock ensures that no other user can use or update the data item.
2. If there is a copy of the data item in your local cache buffer, use the IXLVECTR macro to determine if the copy is valid. If the copy is valid, go to step "5" on page 378.
3. If a copy does not exist in your local cache buffer or the copy is no longer valid, use IXLCACHE REQUEST=READ_DATA to read the data item from the cache structure. If the data item is in the cache structure, go to step "5" on page 378. If a local cache buffer is not assigned to the named data item, assign the local cache buffer and use a protocol to assign a vector entry to the buffer.
4. If the data item is not in the cache structure, read it from permanent storage into the local cache buffer.
5. Update the data item in the local cache buffer.
6. Use IXLCACHE REQUEST=WRITE_DATA to write the updated data item to the cache structure. To invalidate copies of the data item for other users, specify CHANGED=YES on the IXLCACHE request.
7. Use the IXLLOCK macro to free the exclusive lock.

Using but not Updating Data in a Store-through Cache

You are a store-through user who plans to use a data item but not update it. Serializing this process ensures that no one updates the data item while you hold the lock.

1. Obtain a shared lock by using the IXLLOCK macro. Holding a shared lock enables others to also use, but not update, the data item.
2. If there is a copy of the data item in your local cache buffer, use the IXLVECTR macro to determine if the copy is valid. If the copy is valid, go to step "5" on page 379.

3. If a copy does not exist in the local cache buffer or the copy is no longer valid, use IXLCACHE REQUEST=READ_DATA to read the data item from the cache structure. If the data item is in the cache structure, go to step "5" on page 379. If a local cache buffer is not assigned to the named data item, assign the local cache buffer and use a protocol to assign a vector entry to the buffer.
4. If the data item is not in the cache structure, read it from permanent storage into the local cache buffer.
5. Use the data item as needed.
6. If the data item has been read from permanent storage, use IXLCACHE REQUEST=WRITE_DATA to write it to the cache so the system can register your interest. Otherwise, skip this step and go directly to step "7" on page 379.
7. Use the IXLLOCK macro to free the shared lock.

Updating Data in a Store-through Cache

You are a store-through user who plans to update the data item and then write it back to the cache structure. Serializing this process ensures that no other user can use or update the data item while you hold the lock.

1. Obtain an exclusive lock by using the IXLLOCK macro. Holding an exclusive lock ensures that no other user can use or update the data item.
2. If there is a copy of the data item in your local cache buffer, use the IXLVECTR macro to determine if the copy is valid. If the copy is valid, go to step "5" on page 379.
3. If a copy does not exist in the local cache buffer or the copy is no longer valid, use IXLCACHE REQUEST=READ_DATA to read the data item from the cache structure. If the data item is in the cache structure, go to step "5" on page 379. If a local cache buffer is not assigned to the named data item, assign the local cache buffer and use a protocol to assign a vector entry to the buffer.
4. If the data item is not in the cache structure, read it from permanent storage into the local cache buffer.
5. Update the data item in the local cache buffer.
6. Use IXLCACHE REQUEST=WRITE_DATA to write the updated data item to the cache structure. On the IXLCACHE macro, specify CROSSINVAL=YES to invalidate copies of the data item for other users, and CHANGED=NO to mark the data item as unchanged. If necessary, you can optionally specify GETCOLOCK=YES to obtain the cast-out lock.
7. Write the updated data item to permanent storage.
8. If the cast-out lock is held, use IXLCACHE REQUEST=UNLOCK_CASTOUT or REQUEST=UNLOCK_CO_NAME to release it.
9. Use the IXLLOCK macro to free the exclusive lock.

Using but not Updating Data in a Directory-only Cache

You are a directory-only user who plans to use a data item but not update it. Serializing this process ensures that no one updates the data item while you hold the lock.

1. Obtain a shared lock by using the IXLLOCK macro. Holding a shared lock enables others to also use, but not update, the data item.
2. If there is a copy of the data item in your local cache buffer, use the IXLVECTR macro to determine if the copy is valid. If the copy is valid, go to step "4" on page 379.
3. If a copy does not exist in the local cache buffer or the copy is no longer valid, read the data item from permanent storage into the local cache buffer. If a local cache buffer is not assigned to the named data item, assign the local cache buffer and use a protocol to assign a vector entry to the buffer.
4. Use the data item as needed.

5. If the data item has been read from permanent storage, use IXLCACHE REQUEST=READ_DATA to identify the data item to the cache and get your interest in the data item registered. Otherwise, skip this step and go directly to step "6" on page 380.
6. Use the IXLLOCK macro to free the shared lock.

Updating Data in a Directory-only Cache

You are a directory-only user who plans to update a data item and then write it back to permanent storage. Serializing this process ensures that no other user can use or update the data item while you hold the lock.

1. Obtain an exclusive lock by using the IXLLOCK macro. Holding an exclusive lock ensures that no other user can use or update the data item.
2. If there is a copy of the data item in the local cache buffer, use the IXLVECTR macro to determine if the copy is valid. If the copy is valid, go to step "4" on page 380.
3. If a copy does not exist in the local cache buffer or the copy is no longer valid, read the data item from permanent storage into the local cache buffer. If a local cache buffer is not assigned to the named data item, assign the local cache buffer and use a protocol to assign a vector entry to the buffer.
4. Update the data item in the local cache buffer.
5. Use the IXLCACHE macro REQUEST=CROSS_INVAL to invalidate copies of the data item for other users.
6. Write the updated data item to permanent storage.
7. Use the IXLLOCK macro to free the exclusive lock.

Serialization considerations when requesting asynchronous cross-invalidations

When a cache structure connector exploiting asynchronous cross-invalidates fails, it is important not to release serialization held by the connector to serialize the cache requests until it is known that all possible asynchronous cross-invalidations have completed. A cache structure exploiter can rely on structure detach processing to know for certain that any asynchronous cross-invalidations have completed for a failed connector. Structure detach processing is an internal XES process that detaches the connector from the cache structure in the coupling facility. Structure detach processing will outwait all asynchronous cross-invalidations that are in progress for that connection. Structure detach is an asynchronous process that is initiated when all peer connectors have acknowledged the disconnected or failed connection event for the failed connector. The only way to know for certain that the detach processing has completed is the ability for the same connector to reconnect to the cache structure. Considering that fact, a possible serialization scheme would call for connections to connect to the cache structure with CONDISP=KEEP allowing the connection to become failed-persistent after a failure. Holding serialization until the failed-persistent connection has reconnected ensures that all asynchronous cross-invalidations for the failed connection have completed.

Managing Cache Structure Resources

Because the amount of storage available to a cache structure is finite, you need to use the storage efficiently. You manage the use of cache structure storage through:

- The assignment and use of storage classes and storage reclaim algorithms to help control storage reclaim
- The cast-out process for changed data items that returns the data items to an unchanged state, allows you to commit the changes to permanent storage, and thereby makes the storage for cast-out data items suitable for reclaim.

Storage Reclaim

Whenever you write a data item to the cache structure, the system first attempts to allocate cache structure resources (a data entry, a directory entry, or both) that are unallocated or not in use. When there are insufficient resources available, the system attempts to reclaim currently allocated resources in the cache structure to satisfy the request.

The system reclaims only those resources that are either:

- Allocated to unchanged data items that are not “locked for cast-out”
- Allocated to named data items that do not contain data
- Allocated to named data items that contain data but have no registered interest.

Data items that are marked changed or that are locked for cast-out cannot be reclaimed. If the system cannot reclaim sufficient resources to satisfy the request, the request fails.

Assigning and Using Storage Classes

Each time you read a data item from the cache structure or write a data item to the cache structure, you must assign the data item to a storage class. Storage classes allow you to control reclaim processing. By grouping data items with similar attributes into a storage class, you can control from which group (that is, storage class) the system will reclaim resources.

For each storage class, the system maintains a queue of entries that identifies the data items for that storage class. Entries on the queue are kept in a least recently used (LRU) order. When the system needs to reclaim from a particular storage class, the system reclaims resources that are used least recently.

You need to develop algorithms to determine the number of storage classes and the data items to assign to each storage class. You might define only one storage class that meets your needs, or you might define multiple storage classes and base the assignment of data items to different storage classes based on the importance of the data items to your application. For instance, storage class one might identify data entries for data to which the application does not need fast access. Storage class two might identify data entries for data that your application must be able to access quickly. Whenever you read or write the data item, you can also change the data item's storage class.

IXLCACHE REQUEST=READ_STGSTATS returns statistics for a specified storage class. These statistics provide information about the use of cache structure storage.

Storage Reclaim Considerations and the Directory-only Cache Method

For the directory-only cache method, each data item that you identify to the cache structure requires cache structure storage for the associated directory entry. When you define a data item, you must assign the data item to a storage class. You use the storage class assignments as a way to manage reclaim processing for the directory entries.

Storage Reclaim Algorithm

You can optionally define a reclaim algorithm for any storage class that you use. If you do not specify your own algorithm to reclaim storage, the system uses a default. By default, the system attempts to reclaim the least recently used resources that belong to data items in the storage class specified on the read or write request.

Defining a Reclaim Algorithm for a Storage Class

You define a reclaim algorithm for a specified storage class by using IXLCACHE REQUEST=SET_RECLVCTR. This request defines and activates a reclaim vector. Each reclaim vector corresponds to one storage class and controls the reclaim of resources for data items in that storage class. If you want to control the reclaim process for two different storage classes, you define two reclaim vectors, one for each storage class.

When you define the reclaim vector for a given storage class, you specify a number of reclaim attempts that the system makes from target storage classes to satisfy a request for any data item in the storage

class controlled by the vector. The location of an entry in a reclaim vector determines the storage class that is the target of the reclaim attempt. The first vector entry corresponds to storage class 1, the second vector entry corresponds to storage class 2, and so forth.

When defining a reclaim vector, you also indicate on the REPEAT keyword how many times the system can use the vector before it deactivates it and begins to use the default reclaim algorithm. You can also use IXLCACHE REQUEST=SET_RECLVCTR to deactivate the reclaim vector at any time, in which case, the system immediately resumes use of the default reclaim algorithm.

Example of a Reclaim Vector

For example, suppose you have two storage classes and you want to specify a reclaim vector for each of them. Storage class one identifies data items that your application does not need to quickly access. Storage class two identifies data items that your application must access quickly.

Table 24 on page 382 shows how you might define two reclaim vectors (one for each of the storage classes) described as follows:

- The reclaim vector for **storage class 1** specifies the following: 5 reclaims from storage class 1 and 0 reclaims from storage class 2.
- The reclaim vector for **storage class 2** specifies the following: 6 reclaims from storage class 1 and 1 reclaim from storage class 2.
- The repeat factor specified on the REPEAT keyword is 2. For each of the storage reclaim vectors described above, the system goes through the process twice for reclaims before it deactivates the vector and uses the default.

Table 24. Two Reclaim Vectors

Storage class reclaim vector	Number of reclaims from storage class 1 resources	Number of reclaims from storage class 2 resources	Repeat factor
For storage class 1	5	0	Attempts reclaims as indicated by the vector 2 times.
For storage class 2	6	1	Attempts reclaims as indicated by the vector 2 times.

Rationale

In this example, the overall effect is to prevent resources in the cache structure associated with storage class 2 (the more important storage class) from being reclaimed more often than resources from storage class 1.

How the Reclaim Vector for Storage Class 1 Works

For data items in storage class 1, the system can make 5 reclaims for resources from storage class 1 to satisfy requests for storage. For each reclaim from the storage class, the system subtracts from the counter (which equals the value specified for the storage class, 5 in the example) until the value equals zero. Then the system reads the vector value for the next storage class (storage class 2, in the example). For data items in storage class 2 (considered to be the more important storage class), the system can make no (0) reclaims to satisfy requests. At this point, the system has made one pass through the vector. The repeat factor indicates the number of times the system reads the reclaim values for storage classes specified on the vector. Before the system reads the vector on the second pass, it resets the original reclaim values for storage class 1 (5 reclaims) and storage class 2 (0 reclaims). With each pass through the vector, the system resets the original vector values and subtracts from the repeat counter (2, in this example). When the repeat counter equals 0, the vector is deactivated and the system default for reclaim is in effect.

How the Reclaim Vector for Storage Class 2 Works

For data items in the more important storage class 2, the system can make 6 reclaims for resources from storage class 1. With each reclaim from the storage class, the system subtracts from the counter (6 in this example) until the value equals zero. Then the system reads the vector value for the next storage class (storage class 2). For data items in storage class 2, the system can make 1 reclaim to satisfy requests. At this point, the system has made one pass through the vector. The system subtracts 1 from the repeat counter of 2, the original values in the vector are reset (6 for storage class 1 and 1 for storage class 2), and the system starts the second pass through the vector. When the repeat counter equals 0, the vector is deactivated and the system default for reclaim is in effect.

Considerations when Defining the Reclaim Vector

You do not need to use a reclaim vector for each storage class you have defined in the structure. You can define a reclaim vector for some storage classes while allowing other storage classes to use the system default reclaim algorithm. However, the number of entries in any reclaim vector must equal the number of storage classes defined, even if you have not defined reclaim vectors for some storage classes.

For more information about defining your own reclaim algorithm or restoring the default reclaim algorithm, see "SET_RECLVCTR: Overriding or Restoring the Default Reclaim Algorithm" on page 452.

Managing Storage Reclaim for Specific Data Items

When the system reclaims storage, it tries to reclaim resources for data items that are the least recently used and have no registered interest in the target storage class. The system maintains information about data items in the cache structure in least recently used queues for each storage class. The last item on the queue indicates that the data item is the least recently used or referenced data item and is suitable for reclaim.

Based on the least recently used queue for the storage class, the longer an unchanged cached data item remains unused (or unreferenced) in the structure, the greater the chance that the system can reclaim its resources. The system handles reclaim processing as follows:

- When a reclaim of only data entry resources is required, the system reclaims those resources from the least recently used queue for the storage class and does not automatically reclaim or invalidate the associated directory entry.
- When a reclaim of directory entry resources is required, the system reclaims those resources from the least recently used queue for the storage class, invalidates the locally cached copies of the data item for all users, and frees the associated data entry resources, if any.

When reclaiming resources from the least recently used queues, the system gives preference to reclaiming those entries that have data but no registered interest over those that have data and do have registered interest.

If storage is reclaimed for a data item and a user then references that data item (for example, with a READ_DATA request), the following occurs:

- If the system had reclaimed only data entry resources for the data item, then the user's read reference of the data item succeeds and the user is registered in the data item. The system returns an indication that the data cannot be read (reason code IXLRSNCODENOREADDATA), since no data entry exists for the data item.
- If the system had reclaimed directory entry resources for the data item (and thus freed the associated data entry resources as well), then the user's read reference of the data item will not succeed. If the user did not request assignment of a new directory entry for the data item, the user will not be registered in the data item and the system returns an indication that the data item does not exist (reason code IXLRSNCODENOENTRY). If the user requested assignment of a new directory entry for the data item, and if assignment of a new directory entry is successful, the user will be registered in the data item and the system will return an indication that the data cannot be read (reason code IXLRSNCODENOREADDATA) because no data entry exists for the data item.

In all of the above cases, no data is read into the user's local cache buffer. In general the user should then read the data from permanent storage to the local cache buffer. Depending on the user's caching protocols, the user may need to write the data back to the cache structure as well, causing data entry resources to be assigned to the data item.

To avoid having to frequently refresh the data item in the cache structure from permanent storage, you can periodically read the data item from the cache structure, write the data item to the cache structure, or issue a PROCESS_REFLIST request for the data item. Any of these options causes the system to do the following:

- Update the reference bit in the directory entry of the data item to indicate that the data item has been recently referenced.
- Move the data item to the recently referenced end of its storage class queue.

The effect of issuing these requests is to make the data item less suitable for reclaim processing so that you can continue to reference the data item in your local cache buffer.

Using PROCESS_REFLIST Requests

PROCESS_REFLIST allows you to mark the copy of the data item in the cache structure as recently referenced. If you are using a local cache copy of the data item, but are not referencing the data item in the cache structure, the system might be more likely to consider resources for the data item in the cache structure as eligible for reclaim. When the system reclaims data item resources, the system invalidates the local cache copy of the data item. If you are using the local cache copy of the data item, but are not referencing the data item in the cache structure, you need to continue to issue the IXLVECTR macro to test the validity of the data item in your local cache while you are using it.

To avoid the reclaiming of resources in the cache structure for a data item that you are referencing in your local cache but not in the cache structure itself, you can use PROCESS_REFLIST to mark the data item as recently referenced. As a result, the system is less likely to reclaim the resources for the data item. PROCESS_REFLIST allows you to process multiple data items at the same time. You can keep a record of the data items that you are using in your local cache buffers, and, at certain intervals, or once you have collected a certain number of data item names in a list, pass the list to the PROCESS_REFLIST request.

For more information about managing cache structure resources for specific data items, see:

- “PROCESS_REFLIST: Marking Data Items as Referenced” on page 457
- “RESET_REFBIT: Marking Data Items as Unreferenced” on page 459.

Deleting Data Items and Reclaim Processing

For data items in the cache structure that users no longer need to access, you can use IXLCACHE REQUEST=DELETE_NAME or REQUEST=DELETE_NAMELIST. These requests delete the data item from the cache structure and free the allocated resources. Deleting a data item from the cache causes the system to automatically invalidate the locally cached copy of the data item for all users. For more information about deleting data items, see “DELETE_NAME: Deleting Data Items From a Cache Structure” on page 442 and “DELETE_NAMELIST: Deleting a List of Data Items” on page 446.

If a user wants to deregister interest in a data item, the user does not invoke the DELETE_NAME or DELETE_NAMELIST request. To deregister interest, the user can register interest in another data item, and specify the vector index that is currently assigned to the original data item. The user also reassigns the local cache buffer to the new data item. For a complete description, see the following sections on registering interest in a data item:

- “Registering Interest in the Data Item for WRITE_DATA Requests” on page 403
- “Registering Interest in the Data Item for READ_ DATA Requests” on page 418
- “Registering Interest in the Data Item for CASTOUT_DATA Requests” on page 429.

Casting out Data Items and Reclaim Processing

If you have changed data in the cache structure, you need to cast out the data to permanent storage. Because the system does not reclaim changed data items, developing an efficient protocol for casting out data is essential for managing the reclaim of resources for the cache structure.

Each time you write a changed data item to the cache structure, the system marks the data item as changed. A data item that is marked as changed remains that way until you successfully cast-out the data item. When you free the cast-out lock after having cast out the data but are unable to write the data item to permanent storage, you can issue an IXLCACHE request to free the cast-out lock and indicate that the data item remain marked as changed. As long as the data item is marked as changed or locked for cast out processing, the system does not reclaim resources for the data item.

When a high percentage of data items are marked as changed, the amount of storage available for reclaim is limited. If the amount of free storage available for new data items is limited, you might be unable to define new data items to the cache structure.

For more information about casting-out data items, see

- "Casting out Data or Updating Permanent Storage" on page 372
- "CASTOUT_DATA: Casting Out Data from a Cache Structure" on page 427
- "CASTOUT_DATALIST: Casting Out a List of Data Items" on page 432

Assigning Cast-Out Classes

When you use the store-in cache method, each time you write changed data to the cache, you must assign it to a cast-out class. Consider grouping data items with similar cast-out frequency requirements in the same cast-out class.

Establishing a Cast-Out Process

To ensure that the system can reclaim storage in the cache structure for subsequent requests, you must periodically cast out changed data items. You can develop a protocol by assigning multiple cast-out classes based on frequency. You might set a "fast" timer to trigger cast-out processing for data items belonging to the "frequently updated" storage class, and a "slow" timer to trigger cast-out processing for infrequently updated data items. Or, you could base your cast-out algorithm on how many changed data items there are in a certain cast-out class. When the number of changed data items reaches that limit, you can cast out the data items.

For each specified data item on the IXLCACHE READ_DIRINFO request, the system returns the cast-out class for the data item and an indication whether the data item is changed. Using this and other information can help you make decisions about how to perform cast-out processing. Using the IXLCACHE macro, you can obtain:

- Directory information for specified data items (REQUEST=READ_DIRINFO).
- Cast-out statistics for specified cast-out classes (REQUEST=READ_COSTATS)
- Cast-out information for a specified cast-out class (REQUEST=READ_COCLASS)
- Storage statistics for specified storage classes (REQUEST=READ_STGSTATS)

You can use the following information that these requests return to make your cast-out decisions:

- The cast-out class for a data item
- The changed/unchanged state of the data item
- The total number of changed or locked for cast-out data items in a specified storage class
- The total number of data elements allocated to the data items in a specified storage class
- The total number of data elements allocated to the data items in a specified cast-out class
- The names of data items belonging to a cast-out class
- The user-data associated with data items belonging to a cast-out class

Based on monitoring cast-out information that the system returns for pre-defined thresholds, you can invoke a process to cast-out selected data items.

For more information about obtaining information that can help you manage a cast-out process, see:

- "READ_DIRINFO: Reading Cache Directory Entries" on page 461
- "READ_COSTATS: Reading Cast-Out Class Statistics" on page 468
- "READ_COCLASS: Reading A Cast-Out Class" on page 465
- "READ_STGSTATS: Reading Storage Class Statistics" on page 472

Releasing Cast-Out Locks

When you cast out data for a data item, you must obtain the cast-out lock for the data item. After you have cast out the data and written it to permanent storage, you must free the cast-out lock; otherwise, the system is unable to reclaim resources associated with the data item.

To release a cast-out lock for a data item, use IXLCACHE REQUEST=UNLOCK_CASTOUT or REQUEST=UNLOCK_CO_NAME. You can free the cast-out lock for one data item at a time or unlock multiple cast-out locks for multiple data items with REQUEST=UNLOCK_CASTOUT. To reduce processing overhead, you might want to cast out a number of data items, then free all of the cast-out locks with one invocation of IXLCACHE REQUEST=UNLOCK_CASTOUT.

For more information about unlocking cast-out locks, see "UNLOCK_CASTOUT: Releasing Cast-Out Locks" on page 434 and "UNLOCK_CO_NAME: Releasing a Single Cast-Out Lock" on page 439.

Measuring Cache Structure Resource Usage

Every time you access a data item in the cache structure (through a READ_DATA or WRITE_DATA request), the system sets the directory reference bit to indicate that the data item is recently referenced. You can use IXLCACHE RESET_REFBIT to test and reset the directory reference bit settings. Using RESET_REFBIT can help determine how efficiently you are using cache structure storage. If the number of recently referenced data items is low compared to the total number of cached data items, your cache structure might be too big, and you might not be making good use of cache structure resources. If the percentage of recently referenced data items relative to the total number of data items in the cache structure is high, your cache structure might be too small.

For each recently referenced data item that the system scans, the RESET_REFBIT request resets the reference bit to make them appear to be "unreferenced." (When you issue RESET_REFBIT to reset the reference bit for a data item, the system does not change the order of the data entry on the storage class queue.) After a set interval, you can test again to measure resource usage.

Understanding Synchronous and Asynchronous Cache Operations

You can specify whether to allow the system to process an IXLCACHE request synchronously or asynchronously. For asynchronous processing of a request, you can specify how you want the system to notify you about request completion. To control synchronous or asynchronous processing, use the MODE parameter on IXLCACHE requests. Table 25 on page 387 lists the options for the MODE parameter.

- **Synchronous Processing**

  Synchronous processing of an IXLCACHE request means that your program regains control only when the IXLCACHE request has completed processing. To specify synchronous processing, you can specify one of the following options for MODE:

  - SYNCECB
  - SYNCTOKEN
  - SYNCEXIT
  - SYNCSUSPEND

The system might need to suspend your program to be able to process the IXLCACHE request synchronously. If you specify MODE=SYNCSUSPEND, the system suspends the program, if necessary, to process the request synchronously. If you specify another synchronous option for MODE and the request cannot be processed synchronously, the system processes the request asynchronously.

The following conditions can cause the system to process a synchronous IXLCACHE request asynchronously:

- The necessary resources for the request (for example, a subchannel) are not currently available
- The BUFFER on the request specifies more than 4096 bytes of buffer storage.
- The BUFLIST parameter specifies more than one buffer, regardless of the total amount of data for the request.
- A dump of the structure is in progress.
- The system might also choose to convert synchronous requests to asynchronous processing, based on performance considerations or other criteria.

The system indicates its intention to process your synchronous request asynchronously by returning a return code of IXLRETCODEWARNING with a reason code of IXLRSNCODEASYNCH when you issue the IXLCACHE request.

- **Asynchronous Processing**

  When the system processes a request asynchronously, your program regains control after it issues the request, and the request runs independently. To specify asynchronous processing, you can specify one of the following options for MODE:

  - ASYNCECB
  - ASYNCTOKEN
  - ASYNCEXIT
  - ASYNCNORESPONSE

  When the request runs asynchronously, you need to determine when it has completed processing. For synchronous requests other than MODE=SYNCSUSPEND, you need to specify how you want to be informed of an asynchronous request completion if the system processes the request asynchronously. You can specify how the system is to inform you when it processes an IXLCACHE request asynchronously in one of the following ways:

  - MODE=SYNCECB or MODE=ASYNCECB to post an event control block (ECB).
  - MODE=SYNCTOKEN or MODE=ASYNCTOKEN to return the request token specified on the IXLFCOMP macro. You issue IXLFCOMP after you issue the IXLCACHE request to obtain information about the results of the request.
  - MODE=SYNCEXIT or MODE=ASYNCEXIT to give control to the complete exit for your program.

  If you do not want to be informed about the completion of an asynchronous request, you can code the following option for some types of requests:

  - MODE=ASYNCNORESPONSE

The MODE Parameter — Summary

Table 25 on page 387 summarizes the synchronous and asynchronous options that you can specify on the MODE parameter.

Table 25. Options for IXLCACHE Request Processing and Completion Notification

MODE Parameter Value	Actions Specified
SYNCECB	Attempt to process the request synchronously but if the request must be processed asynchronously, post an ECB to indicate request completion.
ASYNCECB	Process the request asynchronously and post an ECB to indicate request completion.

Table 25. Options for IXLCACHE Request Processing and Completion Notification (continued)	
MODE Parameter Value	**Actions Specified**
SYNCTOKEN	Attempt to process the request synchronously but if the request must be processed asynchronously, return an asynchronous request token representing the request. To obtain request results, invoke the IXLFCOMP macro with the asynchronous request token you received. For more information, see "Using the IXLFCOMP Macro with MODE=ASYNCTOKEN or MODE=SYNCTOKEN" on page 516.
ASYNCTOKEN	Process the request asynchronously and return an asynchronous request token representing the request. To obtain request results, invoke the IXLFCOMP macro with the asynchronous request token you received. For more information, see "Using the IXLFCOMP Macro" on page 388.
SYNCEXIT	Attempt to process the request synchronously but if the request must be processed asynchronously, give control to the complete exit when the request completes. For more information about the complete exit, see "Coding a Complete Exit" on page 583.
ASYNCEXIT	Process the request asynchronously and give control to the complete exit when the request completes.
SYNCSUSPEND	Process the request synchronously. If necessary, suspend the program until the request completes processing. Note that this is the only MODE option that could cause your program to be suspended. To use this option, your program must be enabled for I/O and external interrupts.
ASYNCNORESPONSE	Process the request asynchronously. Do not provide notification of request completion.

You can issue multiple IXLCACHE requests with MODE=ASYNCTOKEN or MODE=SYNCTOKEN to allow you to continue with other work while the requests are being processed asynchronously and obtain request results through the IXLFCOMP macro.

Using the IXLFCOMP Macro

If you specify MODE=ASYNCTOKEN or MODE=SYNCTOKEN, and your request is processed asynchronously, you must invoke the IXLFCOMP macro to obtain the results of your IXLCACHE request. You can use IXLFCOMP to determine whether your request has completed or to have your task suspended until the request completes.

If the return code from IXLFCOMP indicates that your request has completed, the results are available in the output areas you have specified on the IXLCACHE macro.

For more information about the IXLFCOMP macro, see "Using the IXLFCOMP Macro" on page 679.

Selecting a Data Buffer For a Request

You can pass or receive structure entry or control data for IXLCACHE requests in buffers. On the request, you can specify a single buffer (by using the BUFFER keyword) or multiple buffers (by using the BUFLIST keyword). To specify buffers for passing or receiving adjunct data, you must use the ADJAREA keyword.

The type of information in the buffers depends on the IXLCACHE request. For instance, on a WRITE_DATA request, the buffer holds data for a data item to be written to a data entry in the cache structure. On an UNLOCK_CASTOUT request, the buffer contains a list of the names of data entries with cast-out locks that you want to free. Both the BUFFER and BUFLIST parameter options enable you to pass or receive up to 65,536 (64K) bytes of structure entry or control data. The ADJAREA parameter allows you to pass or receive up to 64 bytes of data for an adjunct area.

BUFFER Keyword

The BUFFER keyword specifies a single contiguous buffer. It consists of a virtual storage area containing the data that the user passes to the cache structure or data that the system returns from the cache structure. Only 31-bit addressable virtual storage areas (below 2GB) are supported by the BUFFER keyword. High virtual storage areas (above 2GB) can only be specified with the BUFLIST keyword.

For a single buffer less than or equal to 4096 bytes in size, the storage must have the following characteristics:

- The buffer size can be 256, 512, 1024, 2048, or 4096 bytes.
- The buffer must start on a 256-byte boundary.
- The buffer must not cross a 4096-byte (page) boundary.
- The buffer must not start below storage address 512.

For a single buffer greater than 4096 bytes, the storage must have the following characteristics:

- The buffer size can be up to 65,536 bytes and must be a multiple of 4096.
- The buffer must start on a 4096-byte boundary.

The following IXLCACHE requests MUST specify a buffer size of 4096 or greater:

- UNLOCK_CASTOUT
- PROCESS_REFLIST
- READ_COCLASS
- READ_DIRINFO
- READ_COSTATS
- CASTOUT_DATALIST

To specify the size of the buffer, use the following parameter:

- **BUFSIZE**

BUFLIST Keyword

The BUFLIST parameter specifies the address of a storage area that contains the addresses of up to 16 buffers. These buffers do not have to be contiguous. The system transfers data to and from the set of buffers in the list in order of ascending buffer number. Figure 35 on page 390 illustrates a buffer list used with a cache structure.

Either 31-bit addressable (below 2GB) or 64-bit addressable (above 2GB) real or virtual storage areas are supported for the BUFLIST keyword, depending on the specifications for the BUFADDRTYPE and BUFADDRSIZE keywords. However, pageable high virtual storage areas (above 2GB) may not be used.

Storage for the buffer list must have the following characteristics:

- The buffer list consists of a maximum 128-byte storage area that can contain a list of 0 to 16 buffer addresses.
- Each entry in the buffer list consists of an 8-byte field. For 31-bit addresses, the high-order (left-most) 4 bytes are reserved and the low-order (right-most) 4 bytes contain the real or virtual address of a buffer. For 64-bit addresses, the entire 8 bytes contain the real address of a buffer.

To specify the ALET of each buffer in the buffer list, use the following parameter:

- **BUFALET**

The BUFALET parameter specifies an access list entry token (ALET) to be used in referencing all of the BUFLIST entries. All the buffers must be in the same address or data space.

To specify the number of buffer entries in the list, use the following parameter:

- **BUFNUM**

Note: The system ignores any other buffer entries in the list greater than the number of buffers specified on BUFNUM.

Each buffer specified by BUFLIST must have the following characteristics:

- The buffer size must be 256, 512, 1024, 2048, or 4096 bytes.
- All of the buffers must be the same size.
- The buffer must start on a 256-byte boundary.
- The buffer must not cross a 4096-byte boundary.
- The buffer must not reside below storage address 512.

For the following IXLCACHE requests, each buffer in the list must be 4096 bytes long and must start on a 4096-byte boundary:

- WRITE_DATALIST
- UNLOCK_CASTOUT
- PROCESS_REFLIST
- READ_COCLASS
- READ_DIRINFO
- READ_COSTATS

To specify the number of 256-byte increments in each BUFLIST buffer for all requests except those that must start on a 4096-byte boundary, use the following parameter:

- **BUFINCRNUM**

Valid values are 1, 2, 4, 8, and 16. For example, if you specify BUFINCRNUM=4, each buffer in the buffer list is 4 x 256 bytes, or 1024 bytes.

To specify whether the buffer addresses are real or virtual addresses, use one of the following parameters:

- **BUFADDRTYPE=REAL**
- **BUFADDRTYPE=VIRTUAL**

To specify whether the BUFLIST entry address in real storage is a 31-bit or a 64-bit address, use one of the following parameters:

- **BUFADDRSIZE=31**
- **BUFADDRSIZE=64**

Figure 35 on page 390 and Figure 36 on page 391 show examples of a buffer list:

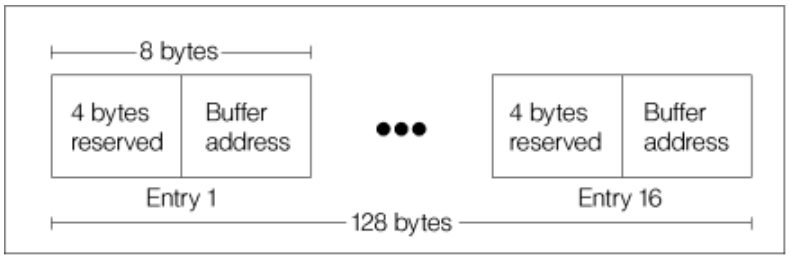


Figure 35. Format of Buffer List Specified by the BUFLIST Parameter

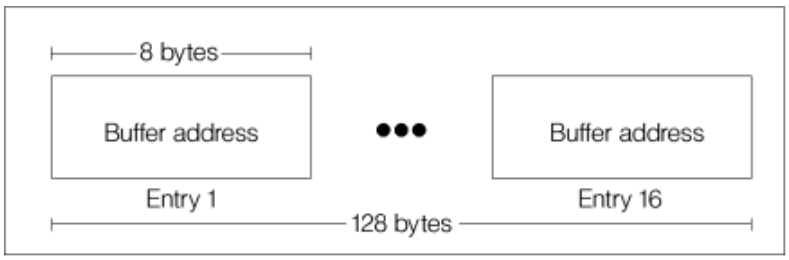


Figure 36. Format of Buffer List (BUFLIST) - 64-bit Addresses

ADJAREA

ADJAREA specifies a 64-byte area containing the information to be written to or read from the data entry's adjunct area.

Design Considerations for Choosing the Buffer Format

To help you evaluate options when you specify buffers, consider the following questions:

- How much buffer storage should I use?
- Should I use BUFFER or BUFLIST?
- If I use BUFLIST, how many buffers should I use?
- What is the relationship between the organization of data in my buffers and data elements in the structure?

Buffer Sizes

You should specify just the buffer storage you need to hold the data you are passing or receiving. Because the system transfers the entire buffer storage that you specify, specifying more buffer space than is needed to hold the data can affect performance.

If you are writing data to a data entry and you wish to create a data entry with extra space for use later, specify a greater number of data elements (ELEMNUM keyword) than you need to hold your data. Specifying more data elements than your data requires does not affect performance.

Specifying BUFFER or BUFLIST

Whether you use a single buffer or multiple buffers depends on whether you are issuing IXLCACHE multiple times, whether all the data resides in contiguous storage, and whether performance is a major factor. If you want to provide real buffer addresses, you can only use BUFLIST.

When you pass IXLCACHE a single buffer, IXLCACHE creates a buffer list for that buffer in the same manner as if you were specifying BUFLIST. If you invoke IXLCACHE multiple times, you can obtain better performance if you use BUFLIST instead of BUFFER and allow IXLCACHE to build the buffer list on each invocation. Using BUFLIST also lets you avoid having to move data from multiple storage areas into a single buffer before passing it to IXLCACHE.

Performance Considerations Using Buffers

If you choose to use multiple buffers, you must determine how many buffers to use and the size of the buffers. To achieve the best performance, use the fewest buffers possible. For example, a few large buffers provide better performance than many small ones.

Buffers and Structure Data Elements

The size of your buffers does not have to correspond to the size of a structure data element. To create a buffer size equal to a structure element, specify the same value for BUFINCRNUM as was specified on the ELEMINCRNUM keyword of the IXLCONN macro for the structure. Establishing this one-to-one relationship is not required because IXLCACHE automatically “remaps” data that is arranged differently as it is transferred between buffer areas and structure elements.

Design Considerations for Defining Buffer Storage Areas

The IXLCACHE request types that allow you to specify buffer storage areas generally result in data being transferred directly between the data buffer storage and the coupling facility storage. The coupling facility transfers data using real storage addresses; therefore, the data buffer storage must be fixed in a specific, known real storage location and remain so until the coupling facility has transferred all data for the request.

When defining the buffer storage areas for an IXLCACHE request, consider the following:

- The cross-memory mode of your application
- The use of real versus virtual storage

The data buffers for an IXLCACHE request can be addressable in the caller's primary, secondary, or home address space, from the PASN access list, or from the DU access list. The system assigns ownership of a data buffer to the address space either in which the buffer storage resides or that has an associated data space in which the buffer storage resides.

Defining Buffer Storage Areas for a WRITE_DATALIST Request

Special buffer requirements exist for the IXLCACHE REQUEST=WRITE_DATALIST request.

- Each buffer must be 4096 bytes long and start on a 4096-byte boundary.
- The area specified by BUFFER must be addressable in the caller's primary address space or from the caller's PASN access list.
- The area specified by the first entry in BUFLIST is the only BUFLIST area that can contain write-operation-blocks. That area must reside in 31-bit virtual storage. All other buffers pointed to by BUFLIST may reside in 31-bit or 64-bit virtual storage. Real storage addressed cannot be used.

Determining Buffer Storage Ownership

XES always assumes that the storage for the data buffers is owned by the home address space (the “requestor's” or “client's” address space) at the time of the IXLCACHE request. However, XES also allows the buffers to be owned by the primary address space (the “connector's” or “server's address space”) at the time of the request when the following conditions both exist:

- The connector's space is not equal to the requestor's home space
- The connector's space is non-swappable.

Thus, the possible address space environments for your application are:

- Requestor (Home) equals Connector (Primary)
- Requestor (Home) does not equal Connector (Primary) with buffer storage owned by Connector's address space
- Requestor (Home) does not equal Connector (Primary) with buffer storage owned by Requestor's address space.

In general, the IXLCACHE service allows you to designate your data buffer storage using real or virtual storage addresses. However, it is of the utmost importance that XES is aware of the specific location of the data buffer storage and that the location remains so until all data transfer is complete.

Using Real Versus Virtual Storage

The IXLCACHE service allows you to designate the data buffer storage in three different ways:

- By **real** storage address
- By **pageable** virtual storage address (including pageable subpools, disabled-reference (DREF) subpools, and page-fixed storage that might not remain page-fixed in a particular real storage location until the completion of the request). High shared virtual storage areas (above 2GB) may not be used.
- By **nonpageable** virtual storage address (including fixed subpools and storage that might not remain page-fixed in a particular real storage location until the completion of the request).

(For information about whether a subpool is pageable, fixed, or DREF storage, see *Authorized Assembler Programming Guide*.)

Specifying the PAGEABLE parameter with BUFFER and BUFLIST is a way to identify to the system whether the storage area you pass is in pageable or potentially pageable storage.

Real storage address

When data buffer storage is designated by real address, XES takes no responsibility for its ownership or its attributes. The IXLCACHE invoker is entirely responsible for management of the storage binds.

For example, suppose a swappable connector

- Obtains a pageable virtual storage buffer in storage associated with the connector's space
- Pagefixes the storage
- Loads the real address of the buffer storage
- Passes those real storage addressses to XES on a request.

If the connector's address space were to be swapped out at some point after loading the real addresses, the system could free and then reassign the real storage frames backing the data buffer. (Page-fixed storage does not remain fixed in real storage when the owning address space is swapped out.) Then, if those real addresses were subsequently used to transfer data to or from the coupling facility, the results would be unpredictable because XES is unaware that the bind between the real addresses and the data buffer virtual storage has been broken.

To summarize: When data buffer storage is passed by real address, it is the caller's responsibility to manage the binds between the data buffer virtual storage and the real storage addresses provided to the coupling facility. The caller must ensure that the data buffer virtual storage remains bound to the real storage addresses provided until the request completes.

Pageable virtual storage address

When data buffer storage is designated by pageable virtual storage address (PAGEABLE=YES on the IXLCACHE request), XES takes full responsibility for the ownership and its attributes regardless of what address space owns the storage. XES performs the required page fixing to fix the buffer in real storage while the IXLCACHE request transfers data to or from the coupling facility. XES establishes the storage binds between the data buffer virtual storage and the real storage backing it and then releases those binds when the data transfer is complete.

If the storage-owning address space were to be swapped out while the XES-established storage binds exist, XES does not allow the swap-out to complete until those storage binds have been broken. The following three scenarios describe actions taken by XES at the time of the swap-out:

1. Coupling facility data transfer has not yet been initiated.

   XES breaks the real storage binds associated with the request. When the address space is swapped-in again, XES re-establishes the storage binds for the request by once again fixing the data buffer virtual storage in real storage (which most likely is a different real storage location than the data buffer previously occupied). XES subsequently uses these real storage addresses for the coupling facility data transfer.

2. Coupling facility data transfer is actively in progress.

   XES delays the swap-out until the coupling facility data transfer completes. When the address space is swapped-in again, the data transfer for the request is complete and there is no need to re-establish the storage binds for the request.

3. Coupling facility data transfer has completed.

   XES breaks the real storage binds associated with the request (or, the storage binds might already have been broken, depending on when the swap-out occurred). When the address space is swapped-in again, the data transfer for the request is complete and there is no need to re-establish storage binds for the request.

To summarize: When data buffer storage is passed by pageable virtual storage address, XES is responsible for managing the binds between the data buffer virtual storage and the real storage used to transfer data to or from the coupling facility.

Nonpageable virtual storage address

When data buffer storage is designated by non-pageable virtual storage address (PAGEABLE=NO on the IXLCACHE request), XES takes full responsibility for the ownership and its attributes if and only if the storage is owned by the requestor's or connector's address space. XES establishes the storage binds between the data buffer virtual storage and the real storage backing it and then releases those binds when the data transfer associated with the request is complete.

If the storage-owning address space (the requestor's or connector's address space) were to be swapped out while the XES-established storage binds exist, XES does not allow the swap-out to complete until those storage binds have been broken. The following three scenarios describe actions taken by XES at the time of the swap-out:

1. Coupling facility data transfer has not yet been initiated.

   XES breaks the real storage binds associated with the request. When the address space is swapped-in again, XES re-establishes the storage binds for the request (which most likely is a different real storage location than the data buffer previously occupied). XES subsequently uses these real storage addresses for the coupling facility data transfer.
2. Coupling facility data transfer is actively in progress.

   XES delays the swap-out until the coupling facility data transfer completes. When the address space is swapped-in again, the data transfer for the request is complete and there is no need to re-establish the storage binds for the request.
3. Coupling facility data transfer has completed.

   XES breaks the real storage binds associated with the request (or, the storage binds might already have been broken, depending on when the swap-out occurred). When the address space is swapped-in again, the data transfer for the request is complete and there is no need to re-establish storage binds for the request.

To summarize: When data buffer storage is passed by nonpageable virtual storage address, XES is responsible for managing the binds between the data buffer virtual storage and the real storage used to transfer data to or from the coupling facility if and only if the storage is owned by the requestor's or connector's address space.

Note:

1. If you specify PAGEABLE=NO and your request is processed synchronously, you can free storage as soon as control returns from IXLCACHE. You must check the return code to verify if the system handled the request synchronously.
2. Table 26 on page 394 shows how long you must keep storage areas fixed for asynchronous processing of a request. It shows the MODE (including synchronous requests that might be processed asynchronously) and when the storage can be made pageable during request processing:

Table 26. When Storage Areas Passed to IXLCACHE Can Be Made Pageable

MODE Value	For Asynchronous Processing, when Storage Can Be Made Pageable
ASYNCECB or SYNCECB	After ECB is posted
ASYNCTOKEN or SYNCTOKEN	When your program regains control from the IXLFCOMP service **and the request has completed**.
ASYNCEXIT or SYNCEXIT	When your completion exit receives control.

Design Considerations for Page-Fixed Storage

Allowing the system to page-fix storage (PAGEABLE=YES) is faster than using the PGSER services. However, specifying PAGEABLE=YES results in slower IXLCACHE performance than specifying

PAGEABLE=NO because it takes more time for IXLCACHE to ensure that the virtual storage is backed by central storage. Additionally, if you issue IXLCACHE multiple times and reuse the same storage areas to pass information, you might obtain better performance if you issue a single PGSER invocation and specify PAGEABLE=NO than if you specify PAGEABLE=YES and allow the system to fix storage on each IXLCACHE invocation.

When selecting an option, consider how many requests you will issue and whether you plan to use the same storage buffers on multiple requests. For example, if you plan to write changes from the buffer to permanent storage as part of the store-through cache method, and must page-fix the storage yourself for such a write, specifying PAGEABLE=YES on the read request for the data in the cache structure has no effect on the page fixing for the write to permanent storage. In such a scenario, because you must provide the page fixing for the write to permanent storage anyway, for improved performance, you might specify PAGEABLE=NO on the IXLCACHE request and page-fix the storage yourself when you write the data from local buffers to the cache structure.

See "Using Real Versus Virtual Storage" on page 392 for more information about specifying pageable and nonpageable virtual storage.

Specifying the Buffer Storage Key

You can specify the BUFSTGKEY parameter with BUFFER or BUFLIST and PAGEABLE=YES to identify and associate a storage key with the buffers. Specifying a storage key helps provide data integrity by allowing IXLCACHE services to check that the buffer is accessible in the key intended by the caller.

Storage key checking is important when the buffer is owned by a client address space that relies on a server address space to invoke IXLCACHE services for data requests. IXLCACHE performs the storage key check so that before passing the data to IXLCACHE, the server address space does not need to transfer the data of the client address space into its own storage.

If you omit BUFSTGKEY with PAGEABLE=YES, the system uses the PSW key of the IXLCACHE requestor as the default storage key and performs key checking using the caller's PSW key.

You cannot specify the BUFSTGDEY parameter with PAGEABLE=NO. The system does not do any storage key checking when non-pageable buffers are used. It is the IXLCACHE invoker's responsibility to do any storage key checking that might be required for non-pageable buffer storage.

Receiving Information from a Request

You receive information from an IXLCACHE request through return and reason codes and the answer area. Depending on the type of request, you might receive information in other storage locations provided by the request. For a description of where to find information returned for each IXLCACHE requests, see the topic in this chapter that discusses the request.

Requesting Return and Reason Codes

All IXLCACHE requests provide a return code in register 15. The system returns reason codes, if they exist, in register 0. (Not all return codes have reason codes.) Optionally, you can define the return code keyword (RETCODE) and the reason code keyword (RSNCODE) in your program.

If the IXLCACHE request defines an answer area, the answer area also contains the return code (in the CAARETCODE field) and the reason code (in the CAARSNCODE field).

Defining an Answer Area (ANSAREA)

All IXLCACHE requests allow you the option to provide an answer area. When you provide an answer area, the system uses it to return information about the request. For a mapping of the answer area fields for the IXLCACHE macro, see macro IXLYCAA in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

If you provide an answer area, you must identify it on each IXLCACHE request through the ANSAREA keyword. You must also indicate the length of the answer area through the ANSLEN keyword.

The following are restrictions that apply when you specify an answer area:

- You must provide an answer area if you specify MODE=SYNCTOKEN or MODE=ASYNCTOKEN.
- You must provide an answer area if you specify AXIOVERRIDE=0 (or default to 0) on an IXLCACHE request that can generate asynchronous cross-invalidations and specified ASYNCXI=1 on the IXLCONN invocation when connecting to the cache structure. The IXLCACHE requests that can generate asynchronous cross-invalidations are:
  - CROSS_INVAL
  - CROSS_INVALLIST
  - DELETE_NAME
  - DELETE_NAMELIST
  - READ_DATA
  - REG_NAMELIST
  - WRITE_DATA
  - WRITE_DATALIST
- Do not specify the same answer area for more than one request at the same time.
- Re-use or free answer area storage for a request only after you determine that the request is complete, either synchronously, or through the asynchronous specification for MODE on the request.

Specifying the IXLYCAA Level

The IXLCACHE Answer Area (IXLYCAA) supports several levels of information that IXLCACHE returns. Certain IXLCACHE requests might provide data that was not returned when the IXLCACHE service was first made available. For these request types, you must check the level of the IXLYCAA and ensure that the length of the answer area that you provide is capable of receiving all the data that the IXLCACHE request returns. For example:

- Extended restart tokens might be returned for restarting a request. An extended restart token requires that the level-1 version of IXLYCAA be used and that its length be specified as CAALEVEL1LEN or larger, or
- An asynchronous cross-invalidation sequence number may be returned when cross-invalidates are performed asynchronously to the completion of a request. An asynchronous cross-invalidation sequence number requires that a level-2 version of IXLYCAA be used and that its length be specified as CAALEVEL2LEN or larger.

The IXLYCAA macro provides the following mappings:

CAA
: Data area that contains the macro level of the version of the IXLYCAA macro, the offset from the beginning of the mapping to the answer area data and the length of the answer area data. An answer area length of CAALEVEL1LEN (144 bytes) or greater is required when the version (PLISTVER) of the IXLCACHE macro is greater than three (3).

CAA2
: Data area that contains the same content as the CAA mapping with additional information such as an asynchronous cross-invalidation sequence number (CaaAsyncXiSeqNum). An answer area length of CAALEVEL2LEN (256 bytes) or greater is required when using a CAA2 mapping.

You must use at least the required minimum answer area size for the request information that may be returned by the IXLCACHE request.

IBM recommends that you use at least the level-2 version of IXLYCAA to accommodate future enhancements to the IXLCACHE service. Note that as the level version of the IXLYCAA mapping increases, so does the corresponding CAALEVELxLEN value increase.

Determining Valid Information in the Answer Area

There are instances when the answer area might not be updated with valid information for a request. For example, if you issue an IXLCACHE request and the system handles the request asynchronously, the system issues a return and reason code to indicate asynchronous processing for the request. As a result, the user must assume that the data in the answer area or other storage location associated with the request is not valid. Only when the user is sure that the request has completed can the data in the answer area be considered valid. For the return and reason code descriptions of each request, see *z/OS MVS Programming: Sysplex Services Reference*.

Specifying the Vector Entry Index on IXLCACHE Requests

You specify a vector entry index to refer to the data item in your local cache buffer on the following IXLCACHE requests:

- IXLCACHE REQUEST=WRITE_DATA
- IXLCACHE REQUEST=READ_DATA
- IXLCACHE REQUEST=CASTOUT_DATA

The system tracks data items in the local cache buffers through each user's vector entry index that corresponds to the local buffer for the named data item. Users are responsible for defining and maintaining the vector entry index values and for specifying on the IXLCACHE request the index for the data item. On the request, you can specify the vector entry index currently assigned to the data item in the cache structure, a vector entry index that is not currently assigned to the data item in the cache structure, or a vector entry index that is currently assigned to another data item in the cache structure.

- **Specifying an Assigned Vector Entry Index**

  To specify the vector entry index currently assigned to the data item, code the vector entry index on VECTORINDEX and the name of the data item on the NAME keyword. The system registers your interest in the data item. It is your responsibility to keep track of the vector entry index you have assigned to the data item and to ensure that you specify that vector entry index.

- **Specifying a Currently Unassigned Vector Entry Index**

  To specify a vector entry index that is currently unassigned to a data item, code the unassigned vector entry index on VECTORINDEX and the name of the data item on NAME. The system registers your interest in the data item. The data item can be a new data item that currently does not have an assigned vector entry index, or a data item that is currently assigned a different vector entry index, in which case, the system invalidates the existing vector entry for the data item.

- **Specifying a Vector Entry Index that is Assigned to Another Data Item**

  To specify a vector entry index that is currently assigned to a data item to another data item, code the vector entry index for the data item on VECTORINDEX and the name of the data item to which you are assigning the vector entry index on NAME. On the OLDNAME keyword, specify the name of the data item to which the vector entry index is currently assigned. The system registers your interest in the data item specified on NAME and deregisters your interest in the data item specified on OLDNAME. The data item for NAME can be a new data item that currently does not have a vector entry index assigned or a data item that is currently assigned a different vector entry index, in which case, the system invalidates the existing vector entry index for the data item.

For a description of the vector entry index and IXLCACHE REQUEST=WRITE_DATA, see “Registering Interest in the Data Item for WRITE_DATA Requests” on page 403.

For a description of the vector entry index and IXLCACHE REQUEST=READ_DATA, see “Registering Interest in the Data Item for READ_ DATA Requests” on page 418.

For a description of the vector entry index and IXLCACHE REQUEST=CASTOUT_DATA, see “Registering Interest in the Data Item for CASTOUT_DATA Requests” on page 429.

Using Filters for Names on Requests

You can specify a NAME and a NAMEMASK filter for the following requests:

- IXLCACHE REQUEST=CROSS_INVAL
- IXLCACHE REQUEST=DELETE_NAME
- IXLCACHE REQUEST=RESET_REFBIT
- IXLCACHE REQUEST=READ_DIRINFO
- IXLCACHE REQUEST=READ_COCLASS

With these requests, you can specify either a single data item name (NAME) or a NAME and NAMEMASK that defines a filter or character selection pattern for multiple data item names.

- **Using a Character Selection Pattern**

  Optionally, you can code both the NAME and NAMEMASK keywords to provide a character selection pattern. The NAMEMASK keyword defines a selection bit-mask. The selection bit-mask together with the name specified on the NAME keyword defines a character pattern that the system uses to select data item names. The technique enables you to select multiple data item names from the cache structure.

  The selection process works as follows: The data item name specified on the NAME keyword is 16-characters long. The bit-mask specified on the NAMEMASK keyword is a bit string that is 16-bits long. Each bit in the bit-mask corresponds to the same relative character position in the data item name. For example, the high-order bit in the mask corresponds to the high-order character in the data item name.

  The value of each bit in the mask determines whether the corresponding character in the NAME keyword is used in the selection process. If the mask bit is B'1', the corresponding character in both the cached data item name and the name specified on the NAME keyword must match exactly. If the mask bit is B'0', the corresponding character in the cached data item name can be any value.

  Consider the following bit-mask values:

  – If the mask contains all B'1's (which is also the system default), the system selects only the cached data item whose name matches exactly the name specified on the NAME keyword.
  – If the mask contains all B'0's, the system selects all cached data items.
  – If the mask contains a combination of B'0's and B'1's, the system selects only those names that satisfy the selection criteria.

  For examples of using NAME and NAMEMASK, see “Identifying Data Items to Delete” on page 443.

Restarting a Request that Ends Prematurely

Some IXLCACHE requests can complete prematurely (that is, without fully completing the requested service) if the request exceeds the time-out criteria for the coupling facility or the user's buffer is filled before all data is returned. (Time-out criteria for a coupling facility is model-dependent.) The following IXLCACHE requests can complete prematurely if they exceed time-out criteria:

- IXLCACHE REQUEST=DELETE_NAME
- IXLCACHE REQUEST=DELETE_NAMELIST
- IXLCACHE REQUEST=CROSS_INVAL
- REQUEST=CROSS_INVALLIST
- IXLCACHE REQUEST=RESET_REFBIT
- IXLCACHE REQUEST=READ_DIRINFO
- IXLCACHE REQUEST=READ_COCLASS
- IXLCACHE REQUEST=READ_COSTATS
- IXLCACHE REQUEST=UNLOCK_CASTOUT

- IXLCACHE REQUEST=REG_NAMELIST
- IXLCACHE REQUEST=CASTOUT_DATALIST
- IXLCACHE REQUEST=WRITE_DATALIST

There are two methods by which the system enables restart of a prematurely completed request. One method uses a restart token, and the other uses an index value. Both methods require that you provide an answer area, mapped by the IXLYCAA macro.

Using the Restart Token

The following IXLCACHE requests use the restart token method when restarting a prematurely completed request:

- IXLCACHE REQUEST=DELETE_NAME
- IXLCACHE REQUEST=CROSS_INVAL
- IXLCACHE REQUEST=RESET_REFBIT
- IXLCACHE REQUEST=READ_DIRINFO
- IXLCACHE REQUEST=READ_COCLASS

To enable restart of a prematurely completed request, the system provides a restart token in the answer area. The restart token can be either 8 or 16 bytes long. The standard restart token (RESTOKEN) is 8 bytes long and is returned in the CAARESTOKEN field of the answer area. The extended restart token (EXTRESTOKEN) is 16 bytes long and is returned in the CAAEXTRESTOKEN field of the answer area. Requestors that specify IXLCONN ALLOWAUTO=YES must use the extended restart token. Requestors that specify or default to ALLOWAUTO=NO must use the standard restart token.

On the first invocation of any IXLCACHE request, you can optionally specify a RESTOKEN or EXTRESTOKEN of all zeros to indicate that the request is invoked for the first time. If the request completes prematurely, it returns a restart value to the CAARESTOKEN or CAAEXTRESTOKEN field in the answer area of the request. To restart processing, you must specify RESTOKEN or EXTRESTOKEN on the next invocation of the request and reset either RESTOKEN or EXTRESTOKEN with the value returned in the answer area from the previous request. To ensure that you do not alter the intent of the request that completed prematurely, the restarted request needs to specify the same keywords and values as those of the original request (with the exception of RESTOKEN or EXTRESTOKEN which is now the value returned in CAARESTOKEN or CAAEXTRESTOKEN from the original request). The system restarts the request from the point at which it completed prematurely on the original request.

Be sure to process the information from this request before reissuing the request.

When using an extended restart token, users should be particularly aware of the structure instance being processed. For example, if a structure has undergone a system-managed rebuild and the user then specifies the extended restart token returned from processing the original structure, the system returns the reason code IXLRSNCODEBADEXTRESTOKEN. An appropriate action to take when this situation occurs would be to start the process again with an EXTRESTOKEN of zero.

Restarting Requests Multiple Times with Restart Tokens

It is possible for a request to complete prematurely multiple times. Each time, you must restart the request until it completes normally. The following series of events shows how to handle these requests:

1. The request completes prematurely and the system returns a restart token in the answer area.
2. Process any information that may have been returned from the request.
3. Issue IXLCACHE to restart the request. You must code RESTOKEN or EXTRESTOKEN to specify the restart token. All other keywords coded on the original request need to be coded on the restart request.
4. The request again completes prematurely and the system returns a restart token in the answer area.
5. Process any information that may have been returned from the request.
6. Issue IXLCACHE with the RESTOKEN or EXTRESTOKEN to restart the request.

Continue this process until the system completes processing all the data specified by the request.

To avoid coding separate IXLCACHE invocations with RESTOKEN or EXTRESTOKEN each time you need to restart the request, code a single IXLCACHE invocation with the restart token initialized to all zeros on the original request. Every time you need to restart the request, you can set the restart token equal to the value returned in the CAARESTOKEN or CAAEXTRESTOKEN field of the answer area on the previous request.

Using an Index Value

The following IXLCACHE requests use the index value method when restarting a prematurely completed request:

- IXLCACHE REQUEST=UNLOCK_CASTOUT
- IXLCACHE REQUEST=REG_NAMELIST
- IXLCACHE REQUEST=DELETE_NAMELIST
- IXLCACHE REQUEST=WRITE_DATALIST
- IXLCACHE REQUEST=CASTOUT_DATALIST
- IXLCACHE REQUEST=CROSS_INVALLIST

To enable restart of a prematurely completed request, the system provides a index value in the index field of the answer area. Use this index value to restart the request so it can process the remaining data. Be sure to process the information returned from this request before reissuing the request.

- For an UNLOCK_CASTOUT request, the system returns an index value into the list of name elements in the CAAULINDEX field. See “Processing an UNLOCK_CASTOUT Request that Ends Prematurely” on page 436.
- For a REG_NAMELIST request, the system returns an index value into the list of registration blocks in the CAARNLINDEX field. See “Restarting a REG_NAMELIST Request that Ends Prematurely” on page 426.
- For a DELETE_NAMELIST request, the system returns an index value into the list of name elements in the CAADNLINDEX field. See “Restarting a DELETE_NAMELIST Request that Ends Prematurely” on page 447.
- For a WRITE_DATALIST request, the system returns an index value into the list of entries to be written in the CAAWDLINDEX field. See “Restarting a WRITE_DATALIST request” on page 416.
- For a CASTOUT_DATALIST request, the system returns an index value into the list of data entries in the CAACDLINDEX field. See “Restarting a REQUEST=CASTOUT_DATALIST Request that ends prematurely” on page 433.
- For a CROSS_INVALLIST request, the system returns an index value into the list of name elements in the CAACILINDEX field. See “Restarting a CROSS_INVALLIST Request that ends prematurely” on page 451.

Restarting Requests Multiple Times with Index Values

It is possible for a request to complete prematurely multiple times. Each time, you must restart the request until it completes normally. The following series of events shows how to handle these requests:

1. The request completes prematurely and the system returns an index value in the answer area.
2. Process any information that may have been returned from the request.
3. Issue IXLCACHE to restart the request. You must reinitialize the starting index based on the index value returned. All other keywords coded on the original request need to be coded on the restart request.
4. The request again completes prematurely and the system returns an index value in the answer area.
5. Process any information that may have been returned from the request.
6. Issue IXLCACHE with the reinitialized starting index value to restart the request.

Continue this process until the system completes processing all the data specified by the request.

Understanding the Cache Data Entry Version Number

When a cache structure is allocated in a coupling facility with CFLEVEL=5 or higher, several IXLCACHE requests allow you to associate a version number with a data entry. You can use the version number field to indicate when the contents of a data entry have changed, to select data entries for certain types of IXLCACHE requests, or to implement a serialization mechanism (similar to compare and swap) on a single data entry basis.

Setting the Cache Entry Version Number

The WRITE_DATA request allows you to set up or change the version number of the target data entry by specifying the VERSUPDATE parameter. The version number can be:

- Assigned a particular value (VERSUPDATE=SET,NEWVERS=*newvers*)
- Incremented by one (VERSUPDATE=INC)
- Decremented by one (VERSUPDATE=DEC).

Note: When a data entry is created, its version number is set to zero. If you specify VERSUPDATE=INC or VERSUPDATE=DEC when you create a new cache entry, the system uses zero as the value to be incremented or decremented.

Using the Version Number to Select Data Entries for Processing

With structures allocated in a coupling facility with CFLEVEL=5 or higher, on a WRITE_DATA request, you can require the target data entry to compare successfully with a version number and type of comparison that you specify in order to be selected for processing. You can specify that a version number be equal or less-than-equal to a designated version number with the VERSCOMPTYPE keyword. If the version number for the target data entry does not meet the version comparison criteria you specify, the IXLCACHE request fails with no resultant change to the structure. The system returns the version number that did not meet the required comparison criteria in the cache answer area.

On DELETE_NAME and DELETE_NAMELIST requests, you can require that all selected data entries have a version number which compares successfully with a version number and type of comparison you specify. If the comparison fails on a DELETE_NAME request, no processing is performed for the current entry and processing continues with the next entry to be considered. When a version number comparison fails on a DELETE_NAMELIST request, the ERRORACTION keyword allows you to specify that either processing is to continue with the next entry or the request is to be stopped. If stopped, the index of the entry that caused the error is returned in the cache answer area.

Using the Version Number to Serialize Data Entry Operations

By adhering to a protocol of updating the version number when you update a cache entry's contents, you can avoid corrupting or deleting changes made to the entry by other users. For instance, you could establish the following procedure for updating data entries:

- Read a data entry
- Update its contents
- Increment, decrement, or set the version number of the updated copy of the data entry
- Write the changes back to the data entry using the VERSCOMP parameter to ensure that the data entry is updated only if its version number is still the same as when you read it or is less than or equal to a specified value.

Note that the use of VERSCOMP is needed to ensure that updates to the version number requested through the VERSUPDATE keyword are not processed multiple times as a result of XES internal request redrive logic. When VERSCOMP is requested along with VERSUPDATE to update the version number, then if the initial execution of the request succeeds, any subsequent internal redrive of the request will fail due to a version number miscompare, preventing multiple updates from occurring on the request. Conversely, if the initial execution of the request was unsuccessful, any subsequent internal redrive of the request will be able to execute successfully and update the version number only once.

In either of these cases, if the request is internally redriven and experiences a version number miscompare on the redrive, a return and reason code of IXLRSNCODESTATUSUNKNOWN will be returned. This reflects the fact that it is not known whether the observed version number miscompare:

- Resulted from the version number update succeeding on the original issuance of the request (causing the miscompare on the redriven request), or
- Was present all along, or
- Resulted from a version number update made by another request.

When this return and reason code is returned, it is up to the user to determine whether or not the requested update has actually occurred, and take the appropriate recovery action.

If the version number comparison fails, the write request is not performed and you must start the update process again after re-reading the current data entry.

Other Services Used with IXLCACHE

Besides the IXLCACHE services, several other services are available to users for managing and using a cache structure. The following is a list of services and exits:

- IXLCONN macro — Used to define characteristics of the cache structure and to connect to the structure
- IXLVECTR — Used to determine the validity of locally cached data and to manage the local cache vector
- IXLLOCK macro — Used to serialize access to data that is shared among users of the cache structure
- IXLFCOMP macro and the complete exit — Used to handle the completion of IXLCACHE requests that run asynchronously.

WRITE_DATA: Writing a Data Item to a Cache Structure

To define a data item and write it to a cache structure, or to update a previously written data item, use the WRITE_DATA request. When you write data to a cache structure, you can:

- Write only a data item from your local cache buffers to a data entry in the cache structure
- Write only adjunct data to the adjunct area, if the data entry has an adjunct area
- Write both adjunct data and a data item

Additionally, when updating a data item, you can:

- Write user-defined data to the associated directory entry
- Write zero data to a data entry, thus causing the user to disassociate a data item and adjunct from the entry.

With a cache structure allocated in a coupling facility with CFLEVEL=5 or higher, these additional functions are available. You can:

- Write a version number, update a version number, and compare version numbers.
- Write data without registering interest and optionally, deregister interest in a different directory entry.

With a cache structure allocated in a coupling facility with CFLEVEL=9 or higher, an additional feature allows you to specify whether the system is to suppress the creation of a new data entry when an existing data entry is not found.

With a cache structure allocated in a coupling facility that supports write suppression based on local cache registration, an additional feature allows you to specify whether the system is to suppress the write request when the user's connection (local cache) is the only registered interest in the data item in the cache structure, and no subsystem data for the data item is cached.

Extended Function

With a coupling facility of CFLEVEL=12 or higher, it is possible to specify a list of up to 256 data items to be written. See “WRITE_DATALIST:Writing Multiple Data Items to a Cache Structure” on page 412.

Guide to the Topic

"WRITE_DATA: Writing a Data Item to a Cache Structure" on page 402 is divided into three sections .

The firstsection , "IXLCACHE Functions for REQUEST=WRITE_DATA" on page 403, applies to all WRITE_DATA requests and includes the following major topics:

- "Registering Interest in the Data Item for WRITE_DATA Requests" on page 403
- "Specifying the Data Item Name" on page 405
- "Specifying the Changed or Unchanged State of the Data Item" on page 405
- "Assigning a Changed Data Item to a Cast-Out Class" on page 407
- "Specifying Parity of a Changed Data Item" on page 407
- "Writing User-Defined Data" on page 408
- "Obtaining the Cast-Out Lock on Write Requests" on page 406
- "Assigning a Storage Class" on page 408
- "Specifying the Size of the Data Entry to Hold the Data" on page 408
- "Selecting the Buffering Method" on page 409
- "Design Considerations for Choosing the Buffer Format" on page 391
- "Specifying Data on a Write Request" on page 409
- "Receiving Answer Area Information" on page 410

The second section , "Defining and Writing a New Data Item: Summary" on page 410 summarizes a procedure for defining a new data item and writing it to the cache structure.

The third section , "Updating an Existing Data Item: Summary" on page 411 summarizes a procedure for updating a data item that is already defined to the cache structure.

IXLCACHE Functions for REQUEST=WRITE_DATA

The following functions apply when you specify REQUEST=WRITE_DATA.

Registering Interest in the Data Item for WRITE_DATA Requests

Users indicate on the WRITE_DATA request whether the user requires current registration of interest in the data item for the request to succeed. You can specify WHENREG=YES (which is also the system default), or WHENREG=NO. For an illustration of registered interest in data items, see Figure 33 on page 374.

Using the WHENREG=YES Option

WHENREG=YES provides a way to serialize updates without obtaining a lock. For example, you might select this option if you are updating the data item and you want to be sure that the copy in your local cache buffer is still valid at the time the write operation takes place. If the copy in your local cache buffer is not valid, the request fails, and you must read the data item from the cache structure and request that your interest be re-registered. You can make updates to the copy in your buffer and write the updated data item to the cache structure. (Note that WHENREG=YES does not actually serialize the use of the data the way an external lock does, but only prevents you from writing data that is not valid in your local cache buffer to the cache structure.)

The VECTORINDEX keyword with WHENREG=YES is not supported with coupling facilities of CFLEVEL 0 or 1 and will be ignored. However, with a coupling facility of CFLEVEL=2 and higher, you can optionally specify the vector entry assigned to the data item with the VECTORINDEX keyword. If you code WHENREG=YES and your interest in the data item is registered with the same vector index as is specified on VECTORINDEX, the WRITE_DATA request will be processed. If you code WHENREG=YES and your interest in the data item is either not registered or registered with a different vector index, the WRITE_DATA request will fail with an IXLRSNCODENOENTRY reason code. In the latter case (where the

vector index is different from that specified by VECTORINDEX), the system returns the vector index with which you are currently registered at the time of the failed request in the cache answer area.

- CAALCVI is set ON to indicate that the value of the vector index specified on the request is different from the vector index with which you are currently registered.
- CAALCVINUM contains the value of the vector index with which you are currently registered.

The system defaults are WHENREG=YES and VECTORINDEX=NO_VECTORINDEX.

Using the WHENREG=NO Option

When writing a new data item to the cache structure, code WHENREG=NO to indicate that you do not have registered interest in the data item. Also, code WHENREG=NO if you want to update the cached copy of the data item regardless of whether you are currently registered. If the data item is new and you code WHENREG=NO, the system allocates cache structure resources when they are available, writes the data item to the cache structure, and registers your interest in the data item. If unused cache structure resources are unavailable, the system attempts to reclaim resources currently in use.

With a cache structure allocated in a coupling facility with CFLEVEL=5 or higher, you can optionally specify REGUSER to indicate whether the WRITE_DATA request should register interest in the entry. If you specify REGUSER=NO, keep in mind that the system gives preference to reclaiming those data items for which there is data but no registered interest.

With a cache structure allocated in a coupling facility with CFLEVEL= 9 or higher, code WHENREG=NO,ASSIGN=NO to specify that you do not want the system to write a data entry to the cache structure if the data item is new and an existing data entry is not found. The system does not create the data entry and returns IXLRSNCODENOENTRY to the user if the conditions are met. Coding or defaulting to WHENREG=NO,ASSIGN=YES for a new data item results in the system allocating cache structure resources when they are available, writing the data item to the cache structure, and registering your interest in the data item.

With a cache structure allocated in a coupling facility that supports write suppression based on local cache registration, code WHENREG=NO, ASSIGN=NO, LOCALREGCNTL=YES, CHANGED=YES to specify that you do not want the system to write changed data to the cache structure if the user requesting the write operation (local cache) is the only registered interest in the data item in the cache structure, and no subsystem data for the data item is cached. If conditions are met, the write request is suppressed, no data entry is written to the cache structure and the request completes with a reason code of IXLRSNCODELOCALREGWRTSUPPRESS.

Using the REGUSER Options

Use the REGUSER option with WHENREG=NO to indicate whether you want to have interest registered in the data item. Data entries in the structure that contain data with no registered interest are higher-priority candidates for being reclaimed than entries with some registered interest.

- Specify REGUSER=NO when you want to write data without registering interest and optionally, with the same request, deregister interest in a different directory entry.
  - Use the NAME keyword to identify the data entry to be written without having interest registered.
  - Use the OLDNAME keyword to identify the entry for which deregistration is to be performed.
  - Use the VECTORINDEX keyword to identify the vector index of the entry which is to be deregistered. VECTORINDEX is required if OLDNAME is specified.
- Specify REGUSER=YES when interest is to be registered in the entry. With REGUSER=YES, you must also specify a value for VECTORINDEX.

When you code WHENREG=NO, you must specify the vector entry assigned to the data item. For a given local cache vector, the vector entries start at 0. For example, if a vector contains 3 entries, they are numbered 0, 1, and 2.

You specify the vector entry on the VECTORINDEX keyword. If you code WHENREG=NO when your interest in the data item is currently registered, you can specify the vector entry that is currently assigned

to the data item. You can also specify a vector entry that is currently unassigned, or specify a vector entry that is currently assigned to another data item.

For example, consider the data items A and B. The vector entry index for A is 1 and the vector entry index for B is 2. To reassign vector entry index 1 to B, code the following keywords:

```
VECTORINDEX=1
NAME=B
OLDNAME=A
```

The system deregisters your interest in data item A, associates vector entry 1 to data item B, registers your interest in B, and writes the data item to the cache structure.

Scenario

Consider specifying a vector entry that is currently assigned to a data item to another data item if you need to contract the size of your local cache buffer and you want to remap your vector entry indexes to data items so you can keep frequently referenced data items in the contracted local cache buffers.

In the following is a scenario, a protocol maps each vector entry to a named buffer: for example, vector entry 1 maps to BUFONE, vector entry 2 maps to BUFTWO, and so forth. BUFONE contains data item X, BUFTWO contains data item Y, and BUFTHREE contains data item Z. You want to free the space allocated to BUFTHREE and you want to keep data items X and Z in the local cache buffers:

1. Move data item Z from BUFTHREE to BUFTWO.
2. Issue the following request to write data item Z to the cache structure, to assign vector entry 2 to data item Z, and to deregister interest in data item Y:

```
IXLCACHE REQUEST=WRITE_DATA,WHENREG=NO,VECTORINDEX=VECTOR2,         X
         NAME=NNAME,OLDNAME=ONAME,…
⋮
VECTOR2  DC  F'2'        VECTOR ENTRY
NNAME    DC  CL16'Z'     NEW NAME
ONAME    DC  CL16'Y'     OLD NAME
⋮
```

3. Free the storage allocated to BUFTHREE.
4. Compress the vector, using IXLVECTR MODIFYVECTORSIZE, so that the unneeded entry 3 is released.

For general information on specifying the vector index entry, see “Specifying the Vector Entry Index on IXLCACHE Requests” on page 397.

Specifying the Data Item Name

All WRITE_DATA requests must specify the name of the data item. Specify the data item name on the NAME keyword.

Specifying the Changed or Unchanged State of the Data Item

When you write a data item to the cache structure, you must indicate the changed or unchanged state of the data item on the CHANGED keyword.

An unchanged data item is one that is identical to the data item on permanent storage. For example, if you read a data item from permanent storage to your local cache buffer, and then, without changing the data item, write it to cache, the data item is considered unchanged. To indicate that you are writing an unchanged data item to the cache structure, specify CHANGED=NO, or omit the CHANGED keyword. If you read the data item from permanent storage or the cache structure to your local cache buffer, change the data that is in the buffer, and then write the buffer to the cache structure without also writing the data item back to permanent storage, the data item in the cache is considered changed because it is unlike the data item on permanent storage.

To indicate that you are writing a changed data item to the cache structure, specify CHANGED=YES. When you specify CHANGED=YES, the system invalidates any copies of the data item that are in local cache buffers of other users and deregisters their interest in the data item.

WRITE_DATA Requests and Unchanged Data

If a data item in the cache structure is marked changed, and you attempt to issue a WRITE_DATA request with CHANGED=NO, the system fails the request. (The system does not let you overwrite changed data with unchanged data in the cache structure.)

Changed Data and Storage Reclaim

If the system has marked a data item as changed, or a user holds the cast-out lock for the data item, the data item is not eligible for reclaim. The system might reclaim resources to satisfy a request from any user to either define a new data item or increase the number of data elements associated with the data item. If the system reclaims resources for a data item, the local copies of the data item for all users are invalidated and their interest deregistered. A subsequent user must read that data item from permanent storage and store it back to the cache structure.

Casting out Changed Data

To make efficient use of cache structure storage, you need to cast-out the changed data in a timely way by using REQUEST=CASTOUT_DATA and ensure that you release the lock for the data items you have cast out by using REQUEST=UNLOCK_CASTOUT. Once the changed data item is cast out and the lock for the data item is released, the resources for the data item are eligible for reclaim. For information, see “Reasons for Casting out Data” on page 428.

Recovery and Changed Data Items

If the coupling facility or structure fails, you cannot cast out the data you have changed from the cache structure. Unless you provide recovery in such situations, you might lose the changed data. To guarantee that you do not lose changed data in the event of a coupling facility or structure failure, provide the necessary recovery routines.

Obtaining the Cast-Out Lock on Write Requests

When you write a data item to the cache structure, you can request the cast-out lock for the data item.

To obtain the cast-out lock, code GETCOLOCK=YES on the IXLCACHE request. When you obtain the cast-out lock, you identify your connection and, optionally, a process (such as a task) as the holder of the lock. You specify your process on the PROCESSID keyword. (The system can return the id, along with the cast-out lock, on certain IXLCACHE requests to the answer area.) While you hold the cast-out lock, if another user invokes a cache service that returns the value of the cast-out lock in the answer area, that user can identify, not only the connection, but also the task or process that holds the lock.

Note: Depending on the IXLCACHE request, two cast-out lock states exist. One is associated with the WRITE_DATA described in this section , and one is associated with CASTOUT_DATA. See “Identifying the Cast-Out Locks to Release” on page 435.

Writing Changed and Unchanged Data items to the Cache

The following topics describe writing changed and unchanged data to the cache structure depending on whether you use the store-through or store-in cache system. (With the directory-only cache, you do not write data items to the cache structure.)

Store-in Cache System

In a store-in cache system, changed and unchanged data items might be handled as follows:

- When writing a new data item that is identical to the copy on permanent storage, code CHANGED=NO. You can also code CROSSINVAL=NO and GETCOLOCK=NO, or omit those keywords and use the system defaults.
- When writing a data item that you have read from permanent storage or the cache structure and updated, or when writing an updated data item back to the cache structure, code CHANGED=YES. The system marks the cached data item as changed and invalidates other users' copies of the data item that

are in their local cache buffers. The system considers resources for changed data items as ineligible for reclaim.

Store-through Cache System

In a store-through cache system, changed and unchanged data items might be handled as follows:

- When writing a new data item that is identical to the copy on permanent storage, code CHANGED=NO. You can also code CROSSINVAL=NO and GETCOLOCK=NO, or omit these keywords and use the system defaults.
- When writing a data item that you have read from permanent storage or the cache structure and updated, or when writing an updated data item back to the cache structure:
  - Code CHANGED=NO to mark the data item as unchanged. Remember, in a store-through cache system you intend to immediately write the data item to permanent storage. You can mark the data item as unchanged to indicate to the system that the data item resources are eligible for reclaim.
  - Code CROSSINVAL=YES to cause the system to invalidate copies of the data item that might be in local cache buffers of other users.

When you write unchanged data items to the cache structure in the store-through cache system, you can also code GETCOLOCK=YES to obtain the cast-out lock for the data item. Obtaining the cast-out lock serializes the update to permanent storage. If another user attempts to obtain the same cast-out lock, that user's request fails. Whether or not you serialize your permanent updates to storage by obtaining the cast-out lock depends on your protocol.

Assigning a Changed Data Item to a Cast-Out Class

Each time you write a changed data item (CHANGED=YES) to the cache structure, you must assign the data item to a cast-out class. Specify the cast-out class on the COCLASS keyword.

You can define the total number of cast-out classes on the IXLCONN macro. The first user who connects to the structure determines the number of cast-out classes for the structure. Cast-out classes are numbered consecutively from 1 to *n* where *n* is the number of cast-out classes specified on IXLCONN.

The data item remains assigned to this cast-out class until one of the following events occur:

- A subsequent WRITE_DATA request for the data item assigns a different cast-out class.
- A CASTOUT_DATA request casts out the data item from the cache structure, and you issue the UNLOCK_CASTOUT request to release the cast-out lock. (You can issue UNLOCK_CASTOUT and specify that the system remark the data entry as changed, in which case, the data item remains associated with the storage class to which it was assigned. See “Changing the Directory Entry for the Data Item” on page 437.)
- A subsequent DELETE_NAME request deletes the data item from the cache structure.

For information on the selection and use of cast-out classes, see “Casting out Data Items and Reclaim Processing” on page 385.

Specifying Parity of a Changed Data Item

When you write a data item to the cache structure and specify CHANGED=YES, you can specify bits (called parity bits) in the directory entry of the data item. The system writes the parity bits only when the value of the bits in the directory entry are null as follows:

```
B'11'
```

Otherwise, the parity bits are left unchanged:

```
B'01' or B'10'
```

The system returns the parity bits as part of the directory entry when you issue IXLCACHE REQUEST=READ_DIRINFO with the DIRINFOFMT=DIRENTRYLIST keyword. The system does not use the parity bits. You establish your own protocol to use the parity bits.

Writing User-Defined Data

When you write a changed data item (CHANGED=YES) to the cache structure, you can also write eight bytes of user-defined data to the directory entry for the data item. (User-defined data might identify the user, typically a process or task identifier, that updates the data item.)

The system writes the user-defined data only if one of the following occurs:

- The data item name is currently undefined in the cache structure.
- The data item name is defined in the cache structure but there is no data stored in the data entry.
- The data item is currently stored in the cache structure and is marked as unchanged.

The system does not use the user-defined data. You establish your own protocol to make use of the data. For cache structures allocated in a coupling facility with CFLEVEL=5 or higher, you can, however, request that the system maintain a queue of the data items for which user-defined data was written to the directory entry. Then, when reading the cast-out class statistical information with REQUEST=READ_COSTATS, the system returns for each cast-out class, the count of data elements and the user data for the UDF order queue entry having the smallest value.

To enable UDF (user data field) order queues, the following conditions must be met:

- The structure must be allocated in a coupling facility with CFLEVEL=5 or higher.
- The initial IXLCONN invocation to connect to the structure must specify UDFORDER=YES. After the structure's allocation, an indicator in IXLYCONA indicates whether UDF order queues are supported.
- The IXLCACHE REQUEST=READ_COSTATS invocation must specify COSTATSFMT=COSTATSLIST.

Note that if a structure is allocated in a coupling facility with CFLEVEL=5 or higher and the IXLCACHE invocation specifies COSTATSFMT=COSTATSLIST, but UDF order queues are not supported by the structure, the system returns the user data of the first entry in the cast-out class queue.

Assigning a Storage Class

Each time you write a data item to the cache structure, you must assign the data item to a storage class. To specify the storage class, specify the STGCLASS keyword. If the data item is currently assigned to a storage class, you can assign it to the same class or reassign it to a different class.

The system determines the number of storage classes for the structure based on the value specified on the first invocation of IXLCONN that allocates the structure. The system ignores any subsequent specifications made by subsequent connectors to the structure as long as the structure remains allocated. Storage classes are numbered consecutively from 1 to *n* where *n* is the number of storage classes specified on IXLCONN.

For information on how to use storage classes to manage resource reclamation, see “Managing Cache Structure Resources” on page 380.

Specifying the Size of the Data Entry to Hold the Data

Whether you update an existing data entry or create a new one, you must always specify the number of data elements to allocate for the data entry to hold the data you are providing. To specify the number, specify the ELEMNUM keyword. Each write request causes the contents and the size of the data entry to be redefined.

The element size and the maximum number of elements that you can allocate to a data item are defined on the IXLCONN macro of the first user who connects to the structure. The size of a data element affects the number of data elements you specify. The first user to connect to the cache structure, selects the data element size. The size is fixed for the life of the structure. Possible sizes are 256, 512, 1024, 2048, or 4096 bytes. Table 27 on page 409 shows the result of specifying a number of data elements that is more

than, less than, or exactly the number necessary to contain the data you are passing by means of BUFFER or BUFLIST.

Table 27. Results of Specifying the Number of Data Elements

Number of Data Elements Specified	Result
Enough to hold data	Specified number of data elements is allocated.
More than number needed to hold data	Specified number of data elements is allocated. Extra space is padded with binary zeros.
Fewer than number needed to hold data	The data is truncated to fit the allotted space.

Selecting the Buffering Method

You can write data to a data entry, write data to the adjunct area when the structure is defined with adjunct areas, or write data to both a data entry and adjunct area for a data item. You pass data to be written to the data entry in a buffer specified on the BUFFER and BUFSIZE keywords, or multiple buffers specified on the BUFLIST, BUFNUM, BUFALET, and BUFINCRNUM keywords. (BUFALET allows you to specify an access list entry token or ALET for use in referencing BUFLIST buffers.) Both methods enable you to pass up to 65536 (64K) bytes of data. You pass data to be written to the adjunct area in a single 64-byte storage area (the ADJAREA keyword).

For information about whether to use a single buffer or multiple buffers and for information on selecting buffer attributes, see “Selecting a Data Buffer For a Request” on page 388.

Specifying Data on a Write Request

When you write to the cache structure, you must specify whether you are writing only the data item to a data entry, only adjunct data, or both.

To write a data item for a data entry only, omit the ADJAREA keyword from the request. If the cache structure definition supports an adjunct area, the system writes binary zeros to the adjunct area. To specify the local cache buffer for the data item, use either BUFFER or BUFLIST.

To write adjunct data only, code the ADJAREA keyword. You must also specify BUFLIST, and BUFNUM must equal 0. The system writes the data specified by the ADJAREA keyword to the adjunct area and leaves the related data entry unchanged.

To write both a data item and adjunct data, specify ADJAREA and use either BUFLIST or BUFFER.

Specifying that No Data Is to Be Written on a Write Request

For cache structures allocated in a coupling facility with CFLEVEL=4 or higher, you can issue a WRITE_DATA request that effectively specifies that no data is to be written. This allows you to remove unchanged data and adjunct from the coupling facility without invalidating all other's local buffers.

To accomplish this, you must specify CHANGED=NO on the WRITE_DATA request. Neither BUFFER nor BUFLIST is required, and ELEMNUM must be zero. If ELEMNUM is specified with a value greater than zero, the system will write data to the entry. If BUFFER or BUFLIST are not specified, the data written will contain all binary zeros.

Specifying the Cache Entry Version Number on a WRITE_DATA Request

For information about:

- Using the entry version number to maintain data integrity on a WRITE_DATA request
- Updating the version number on a WRITE_DATA request

see “Understanding the Cache Data Entry Version Number” on page 401.

Receiving Answer Area Information

On most IXLCACHE requests, the system returns information related to the request in the answer area. You specify the answer area on the ANSLEN and ANSAREA keywords. With certain events, the information in the answer area might not be valid. See "Determining Valid Information in the Answer Area" on page 397.

When the request completes, the system returns information to the answer area. When the request is not valid, the system returns non-zero return and reason codes.

For the mapping of the answer area, see the IXLYCAA mapping macro described in the *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary). For a description of answer area fields and return and reason codes for the request, see *z/OS MVS Programming: Sysplex Services Reference*.

Defining and Writing a New Data Item: Summary

A previously undefined data item is one that is not defined to the cache structure. This topic summarizes a way to use the IXLCACHE REQUEST=WRITE_DATA to define and write a new data item.

- To write an undefined data item to the cache structure, specify NAME for the data item and WHENREG=NO to indicate that you currently do not have registered interest in the data item.
- Assign a vector entry on the VECTORINDEX keyword. If the vector entry you assign is currently associated with another data item, specify OLDNAME to identify the other data item.

The system attempts to allocate the cache structure resources to satisfy the write request. If the allocation is successful, the system creates a directory entry and registers your interest in the data item. If the resources are unavailable to satisfy the allocation, and the system is unable to reclaim resources currently in use, the request fails.

Your request must indicate whether the data item you are writing is the same as the copy on permanent storage. If it is the same, specify CHANGED=NO or omit the parameter to use the system default. The system does not allow you to overwrite changed data with unchanged data.

Considerations for a Store-through Cache System

If you change the data item before writing it to the cache structure and at the same time you intend to write the data item to permanent storage, specify CHANGED=NO to mark the data item as unchanged. The technique is typically used in a store-through cache environment when you are writing the changed data item to both the cache structure and to permanent storage. In this case, you must also code CROSSINVAL=YES to invalidate copies of the data item in the local cache buffers of other users and deregister their interest in the data item.

You can optionally specify GETCOLOCK=YES to obtain the cast-out lock for the data item. By holding the cast-out lock, you serialize the update to permanent storage. If another user makes a request to obtain the cast-out lock that you hold, the request fails. You can also use PROCESSID to identify your task or process as the holder of the cast-out lock. After successfully writing the data item to permanent storage, issue a REQUEST=UNLOCK_CASTOUT or REQUEST=UNLOCK_CO&NAME to free the cast-out lock.

Considerations for a Store-in Cache System

If you write a changed data item to the cache structure without also writing the data item to permanent storage, specify CHANGE=YES. This technique is typically used in a store-in cache system. You must also specify COCLASS to assign the data item to a cast-out class. Optionally, you can specify PARITY to assign parity bits to the data item and USERDATA to provide user-defined data for the directory entry.

Specifying Storage Class and Data Element Numbers

You must code STGCLASS to assign the data item to a storage class, and ELEMNUM to specify the number of data elements that are to be allocated to the data item.

Data that you write to the data item must be in the local cache buffer. You identify the buffer by coding either BUFLIST or BUFFER and related keywords. Data that you write to the adjunct area must be in a storage area identified on the ADJAREA keyword, and the cache structure must be allocated with adjunct areas when you connect to the structure.

When the WRITE_DATA request completes, the system provides a return code, a reason code, and appropriate answer area information. Examine the information that the system returns, and take the action that is appropriate for your program.

For a discussion of keywords applicable to all IXLCACHE requests, see:

- "Understanding Synchronous and Asynchronous Cache Operations" on page 386
- "Accessing and Managing Data Within a Cache System" on page 369
- "Requesting Return and Reason Codes" on page 395
- "Defining an Answer Area (ANSAREA)" on page 395

Updating an Existing Data Item: Summary

An existing data item is one whose name is currently defined to the cache structure. This topic summarizes a way to use the IXLCACHE REQUEST=WRITE_DATA to update an existing data item.

To ensure that you do not overwrite changes that another user might have made to the data item and to ensure that the copy of the data item in your local cache buffer is current, you can code WRITE_DATA with WHENREG=YES without having to use locks to serialize the updates. When coding the WRITE_DATA request for an existing data item, specify the NAME keyword to identify the data item and WHENREG=YES to request that the system perform the update only when you have a registered interest in the data item.

Otherwise, you can use IXLLOCK to serialize your updates and specify WHENREG=NO with the WRITE_DATA request, and the system performs the update regardless of whether you have registered interest in the data item or not.

Your request must indicate whether the data item you are writing is the same as the copy on permanent storage. If it is the same, specify CHANGED=NO or omit the parameter to use the system default. The system does not allow you to overwrite changed data with unchanged data.

Considerations for a Store-through Cache System

If you change the data item before writing it to the cache structure and at the same time you intend to write the data item to permanent storage, specify CHANGED=NO to mark the data item as unchanged. The technique is typically used in a store-through cache environment when you are writing the changed data item to both the cache structure and to permanent storage. In this case, you must also code CROSSINVAL=YES to invalidate copies of the data item in the local cache buffers of other users and deregister their interest in the data item.

You can optionally specify GETCOLOCK=YES to obtain the cast-out lock for the data item. By holding the cast-out lock, you serialize the update to permanent storage. If another user makes a request to obtain the cast-out lock that you hold, the request fails. You can also use PROCESSID to identify your task or process as the holder of the cast-out lock. After successfully writing the data item to permanent storage, issue a REQUEST=UNLOCK_CASTOUT or REQUEST=UNLOCK_CO_NAME to free the cast-out lock.

Considerations for a Store-in Cache System

If you write a changed data item to the cache structure without also writing the data item to permanent storage, specify CHANGE=YES. This technique is typically used in a store-in cache system. You must also specify COCLASS to assign the data item to a cast-out class. Optionally, you can specify PARITY to assign parity bits to the data item and USERDATA to provide user-defined data for the directory entry.

Specifying Storage Class and Data Element Numbers

You must code STGCLASS to assign the data item to a storage class, and ELEMNUM to specify the number of data elements that are to be allocated to the data item.

Data that you write to the data item must be in the local cache buffer. You identify the buffer by coding either BUFLIST or BUFFER and related keywords. Data that you write to the adjunct area must be in a storage area identified on the ADJAREA keyword, and the cache structure must be allocated with adjunct areas when you connect to the structure.

When the WRITE_DATA request completes, the system provides a return code, a reason code, and appropriate answer area information. Examine the information that the system returns, and take the action that is appropriate for your program.

For a discussion of keywords applicable to all IXLCACHE requests, see:

- "Understanding Synchronous and Asynchronous Cache Operations" on page 386
- "Accessing and Managing Data Within a Cache System" on page 369
- "Requesting Return and Reason Codes" on page 395
- "Defining an Answer Area (ANSAREA)" on page 395

WRITE_DATALIST:Writing Multiple Data Items to a Cache Structure

You can use the WRITE_DATALIST request to specify that data should be written to the cache structure for a given set of entries for the connection identified by CONTOKEN. Using this request type provides performance benefits by reducing the number of accesses to the coupling facility.

The data to be written is described by a write-operation-block, which contains the relevant information to describe the attributes of the write operation. The result of the write operation is contained in the write-operation-response-block, which contains the relevant information about the completion of the write request.

The number of write-operation-blocks that can be specified is dependent upon whether BUFFER or BUFLIST is specified.

The WRITE_DATALIST request type is valid only for a structure allocated in a coupling facility of CFLEVEL=12 or higher.

Guide to the Topic

"WRITE_DATALIST:Writing Multiple Data Items to a Cache Structure" on page 412 is divided into two sections .

The first section , "IXLCACHE Functions for WRITE_DATALIST" on page 412, applies to all WRITE_DATALIST requests and includes the following major topics:

- "Specifying the entries to be written" on page 413
- "Providing information in the write-operation-block" on page 413
- "Selecting a buffering method" on page 413
- "Specifying the index values" on page 413
- "Providing a storage area for Returned WOB processing" on page 414
- "Situations that Cause WOB Processing to be Discontinued" on page 414
- "Receiving answer area information" on page 416
- "Restarting a WRITE_DATALIST request" on page 416

The second section , "Writing a list of data items: Summary" on page 416 summarizes a procedure for defining a list of data items and writing them to the cache structure.

IXLCACHE Functions for WRITE_DATALIST

The following functions apply when you specify REQUEST=WRITE_DATALIST.

Specifying the entries to be written

The set of entries to be written to the cache structure is identified by write-operation-blocks that are contained in the storage area defined by BUFFER or BUFLIST. Each write-operation-block, mapped by the IXLYWOB macro and refered to here as a WOB, contains the information needed to identify an individual cache entry to be written.

The write-operation-blocks are numbered starting with 1. Each WOB has a length of 256 bytes. When BUFFER is specified, 1 to 256 WOBs can be provided by the user. When BUFLIST is specified, 1 to 16 WOBs can be provided by the user.

The data area containing the entries to be written to the cache structure is referenced through the DATAOFFSET keyword. DATAOFFSET specifies an offset in 256-byte increments into the BUFFER or BUFLIST storage areas that identifies the first data area.

Providing information in the write-operation-block

The purpose of the WOB is to provide all pertinent information required when writing an entry to the cache structure. For example, the name of the entry, the data area size in terms of its number of elements, and the storage class is contained in the WOB. Indicators specify such information as whether changed data is to be written, whether a directory entry should be assigned, and whether cross-invalidate processing should be performed.

Other information that you specify in the WOB refers to registration of interest in the entry. If you do not want to register interest, set the suppress registration indicator. If you do want to register interest, then you specify a vector index. When registration is performed, if connection interest is already registered, the specified vector index replaces any previously-specified vector index for the entry. To deregister interest for a different entry, specify OLDNAME in the WOB. If both NAME and OLDNAME are specified in the WOB for an entry, are not equal, and the name replacement control indicator is set, any registered interest for the specified local cache vector index for the entry specified by OLDNAME will be deregistered prior to registering interest for the named entry.

See IXLYWOB in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Selecting a buffering method

You can choose to create the write-operation-blocks for the WRITE_DATALIST request in the storage area referenced either by BUFFER or BUFLIST. Each buffer area must be 4096 bytes long. BUFFER defines an area containing an array of from 1 to 256 WOBs, followed by the corresponding data areas to be written to each entry specified in the WOBs. The buffer area must be addressable in the caller's primary address space or from the caller's PASN access list.

BUFLIST defines a set of buffers that will contain an array of from 1 to 16 WOBs, followed by the corresponding data areas to be written to each entry specified in the WOBs. WOBs can only be specified in the **first** buffer pointed to by the **first** BUFLIST entry.

Specifying the index values

The system references the WOBs by an index into BUFFER or BUFLIST and an offset specifying the location of the data area to be processed. The entries are processed sequentially beginning with the WOB specified by the first index entry (STARTINDEX) and ending with the WOB specified by the last index entry (ENDINDEX). The WOB entries are numbered starting with 1.

DATAOFFSET contains the offset in 256-byte increments into the data block in the storage area specified by BUFFER or BUFLIST of the first data area to be processed.

A DATAOFFSET value of 0 implies that there is no actual data being passed in the storage area specified by BUFFER or BUFLIST. When this occurs, all WOBs also have to specify an ELEMNUM of 0 to indicate that no data is being passed. If any WOB specifies an ELEMNUM value other than 0, the WRITE_DATALIST request will fail with return code IXLRETCODEPARMERROR, reason code IXLRSNCODEBADELEMNUM.

Providing a storage area for Returned WOB processing

For each WOB specified in the input array, there is a corresponding array of write-operation-response-blocks that the system returns in the storage area defined by WORBAREA. A write-operation-response-block is mapped by IXLYWORB and is referred to here as a WORB. The WORB contains information based on the processing of the WOB. For example, for each WOB processed, the WORB will contain the cast-out count for the cast-out class to which data was just written, the total changed count for the storage class to which data was just written, and the invalidated local cache vector index, if any.

See IXLYWORB in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Situations that Cause WOB Processing to be Discontinued

The following scenarios describe the results of WOB processing when specific indicators in the WOB are set.

- Assignment suppression control indicator

  If set, no directory entry will be assigned and the write operation will be suppressed if the entry does not exist. If not set, a directory entry will be assigned for the named entry and the write will be performed if the entry does not exist. When LOCALREGCNTL=NO is specified or defaulted to and the write operation is suppressed due to assignment suppression, the data is not written, the index of the write-operation block that failed, and the offset in the data block of the data area for the write-operation block being processed is returned in the ANSAREA. The WRITE_DATALIST request completes prematurely with a return code IXLRETCODEPARMERROR, reason code IXLRETCODENOENTRY. All prior write-operation blocks were processed.

  When LOCREGCNTL=YES is specified and the assignment suppression control indicator is set, the indication as to whether a write operation was suppressed for a WOB can be found in the write- suppressed vector returned in the CAAWRITESUPPRESSEDVECTORfield of the answer area. The CAAWRITESUPPRESSEDVECTOR should be processed whenever the WRITE_DATALIST request completes with a return code of IXLRETCODEOK or the return and reason code indicates that the WRITE_DATALIST request completed prematurely and all the write operations prior to the premature completion were processed.

  For more information on processing results when a request completes prematurely and restarting IXLCACHE requests, see “Restarting Requests Multiple Times with Index Values” on page 400 and “Restarting a WRITE_DATALIST request” on page 416.
- Change control indicator is set

  The entry will be written as changed and assigned to the specified cast-out class. Also, with the exception of the connection specified by CONTOKEN, all connections with registered interest in the entry have interest deregistered and a cross-invalidate performed against their local caches. The ELEMNUM specification in the WOB must be greater than or equal to 1.

  When the get cast-out lock indicator is also set, the data is not written, the cast-out lock is not obtained, and the index of the failing WOB is returned in the answer area. None of the specified WOBs were processed. (Processing of the entire WRITE_DATALIST request was suppressed.)

  When a cast-out class that is not valid is specified in the WOB, the data is not written, the index of the WOB that failed, the invalidated local cache validity vector, and the offset in the data block of the data area for the WOB being processed are returned in the answer area and the WORB returned is set to zero. All prior WOBs were processed.

  When parity bits that are not valid are specified in the WOB, the data is not written, the index of the WOB that failed, the invalidated local cache validity vector, and the offset in the data block of the data area for the WOB being processed are returned in the answer area and the WORB returned is set to zero. All prior WOBs were processed.
- Change control indicator is not set

When the cross-invalidate control indicator is set, with the exception of the connection specified by CONTOKEN, all connections with registered interest in the entry will have interest deregistered and a cross-invalidate performed against their local caches.

If data is already cached, it must be cached as unchanged. If the entry is already marked as changed or locked for cast-out, the data is not written, the index of the WOB that failed, the changed indicator, the castout lock state, the castout lock value, the local cache vector index, the invalidated local cache validity vector, and the offset in the data block of the data area for the WOB being processed are returned in the answer area and the WORB returned is set to zero. All prior WOBs were processed.

- Get cast-out lock control indicator is set

  If the cast-out lock is already held through a REQUEST=CASTOUT_DATA or REQUEST=CASTOUT_DATALIST invocation, or if the cast-out lock is already held by another connector, the data is not written, the index of the WOB that failed, the castout lock state, the castout lock value, and the offset in the data block of the data area for the WOB being processed are returned in the answer area. All prior WOBs were processed.

- Comparative version number

  If the comparative version number and the version comparison request type are specified in the WOB, version numbers will be compared after the deregistration operation is performed for the WOB. If version-number comparison is requested and the name is assigned and the version number comparison is successful, the data will be written. If the name is assigned and the version number comparison fails, the data is not written, the version number from the directory entry, the index of the WOB that failed, the invalidated local cache validity vecotr, and the offset in the data block of the data area for the WOB being processed are returned in the answer area and the WORB returned is set to zero. All prior WOBs were processed.

- Incorrect data area size

  When a data area size that is not valid is specified in the WOB with WOB_ELEMNUM, the data is not written, the index of the WOB that failed, the invalidated local cache validity vector, and the offset in the data block of the data area for the WOB being processed are returned in the answer area and the WORB returned is set to zero. All prior WOBs were processed.

- Storage class

  If a storage class that is not valid is specified in the WOB, the data is not written, the index of the WOB that failed, the invalidated local cache validity vector, and the offset in the data block of the data area for the WOB being processed are returned in the answer area and the WORB returned is set to zero. All prior WOBs were processed.

  When a WRITE_DATALIST request fails due to the inability to obtain structure resources from the storage class specified in the WOB, the data is not written, the index of the WOB that failed, the invalidated local cache validity vector, the offset in the data block of the data area for the WOB being processed, and the target storage class number from which resources could not be reclaimed are returned in the answer area and the WORB returned is set to zero. All prior WOBs were processed.

- Incorrect local cache vector index

  When a local cache vector index that is not valid is specified in the WOB, the data is not written and the index of the WOB containing the incorrect local cache vector index is returned in the answer area. None of the specified WOBs was processed. (Processing of the entire WRITE_DATALIST request was suppressed.)

- Incorrect ELEMNUM

  When the ELEMNUM specified in the WOB does not match the size of the data block corresponding to the WOB being processed, the data is not written, the index of the WOB that failed, the invalidated local cache validity vector, and the offset in the data block of the data area for the WOB being processed are returned in the answer area and the WORB returned is set to zero. All prior WOBs were processed.

- Entry does not exist

  If the entry does not exist and the assignment suppression control indicator is set, the data is not written, the index of the WOB that failed, the invalidated local cache validity vector, and the offset in the

data block of the data area for the WOB being processed are returned in the answer area and the WORB returned is set to zero. All prior WOBs were processed.

Receiving answer area information

On most IXLCACHE requests, the system returns information related to the request in the answer area. You specify the answer area on the ANSLEN and ANSAREA keywords. With certain events, the information in the answer area might not be valid. See "Determining Valid Information in the Answer Area" on page 397.

When the request completes, the system returns information to the answer area. When the request is not valid, the system returns non-zero return and reason codes.

For the mapping of the answer area, see the IXLYCAA mapping macro described in the *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary). For a description of answer area fields and return and reason codes for the request, see *z/OS MVS Programming: Sysplex Services Reference*.

Restarting a WRITE_DATALIST request

The IXLCACHE REQUEST=WRITE_DATALIST request can complete prematurely. When a request completes prematurely, the system will not have processed all the WOBs identifying the cache structure entries and associated data to be written. To continue processing the WOBs, you must restart the request.

When a request completes prematurely, the system returns an index value into the list of WOB entries. This value is returned in the CAAWDLINDEX field of the answer area. Use this index value to restart the request.

Reasons for which a WRITE_DATALIST request can complete prematurely are:

- A write operation was suppressed due to the assignment suppression control indicator being set in a WOB and a cache structure entry not existing. The index of the next WOB to be processed is returned in the answer area. All WOBs preceding this one are processed.
- The request has exceeded the model-dependent time-out criteria. The index of the next WOB to be processed is returned in the answer area. All WOBs preceding this one are processed.
- The WORBAREA is full prior to completing the processing of the WOBs.
- A write operation was suppressed due to the assignment suppression control indicator being set in a WOB and a cache structure entry not existing. The index of the next WOB to be processed is returned in the answer area. All WOBs preceding this one are processed.

Writing a list of data items: Summary

For cache structures allocated in a coupling facility of CFLEVEL=12 or higher, use the WRITE_DATALIST request to specify a set of data entries to be written to the cache structure. Each data entry to be written is described by a write-operation-block, mapped by IXLYWOB. The write-operation-blocks are placed in the storage area specified by BUFFER or BUFLIST. The result of the write operation is placed in a write-operation-response-block, mapped by IXLYWORB.

The system references the write-operation-blocks by an index into BUFFER or BUFLIST and processes them sequentially.

For a discussion of keywords applicable to all IXLCACHE requests, see:

- "Understanding Synchronous and Asynchronous Cache Operations" on page 386
- "Accessing and Managing Data Within a Cache System" on page 369
- "Requesting Return and Reason Codes" on page 395
- "Defining an Answer Area (ANSAREA)" on page 395

READ_DATA: Reading a Data Item from a Cache Structure

You read a data item (REQUEST=READ_DATA) to either:

- Define (allocate a directory entry for) a new data item to the cache structure and register interest in that data item
- Read a currently defined data item from the cache structure to your local cache buffer and register or re-register interest in the data item.
- Register interest in a currently defined data item without reading the data element from the cache structure to your local cache buffer (only for cache structures allocated in a coupling facility with CFLEVEL=4 or higher).
- Read a data item without registering interest in the data item (only for cache structures allocated in a coupling facility with CFLEVEL=5 or higher).

When you define a new data item, the system tries to allocate and initialize a directory entry for the data item. If resources for allocating a directory entry are unavailable and cannot be reclaimed, the system fails the request. If the system can allocate the directory entry, the system also registers your interest in the data item and validates your local copy of the data item. There is no data actually transferred from the cache to your local cache buffer or adjunct area.

Note: The system validates your local copy even when there is currently no data in the local buffer.

When you read a currently defined data item from the cache structure, you can:

- Read only the data item into your local cache buffer
- Read only the adjunct data to your adjunct area
- Read both the data item and adjunct data
- Register interest in the data item without reading the data into your local cache buffer (only for cache structures allocated in a coupling facility with CFLEVEL=4 or higher).

The system registers your interest in the data item, transfers the requested data, if it is stored in the cache structure, to your storage, and validates your local copy.

If your protocol relies on external serialization, you need to hold a lock to serialize your read operation. For serialization recommendations and sample scenarios that show how to establish serialization, see “Serializing and Managing Access to Shared Data” on page 377.

Guide to the Topic

“READ_DATA: Reading a Data Item from a Cache Structure” on page 417 is divided into three sections .

The first section , “IXLCACHE Functions for REQUEST=READ_DATA” on page 418, applies to all READ_DATA requests and includes the following major topics:

- “Specifying the Data Item Name” on page 418
- “Registering Interest in the Data Item for READ_ DATA Requests” on page 418
- “Specifying a New or Existing Data Item” on page 418
- “Assigning a Storage Class” on page 419
- “Selecting the Buffering Method” on page 419
- “Specifying the Data to be Read” on page 419
- “Receiving Answer Area Information” on page 420

The second section , “Defining a New Data Item: Summary” on page 420 summarizes a procedure for defining a new data item to the cache structure.

The third section , “Reading a Data Item: Summary” on page 421 summarizes a procedure for reading a data item that is already defined to the cache structure.

IXLCACHE Functions for REQUEST=READ_DATA

The following functions apply when you specify REQUEST=READ_DATA.

Specifying the Data Item Name

All READ_DATA requests must specify the name of the data item. Specify the data item name on the NAME keyword.

Registering Interest in the Data Item for READ_ DATA Requests

Prior to coupling facilities at CFLEVEL=5, all READ_DATA requests either register or re-register your interest in the data item. To enable the system to register or re-register your interest, each request must specify a vector entry index on the VECTORINDEX keyword. If you currently have a vector entry index assigned to the data item, you can specify that vector entry. Optionally, you can specify a vector entry that is currently assigned to another data item or one that is currently unassigned.

With a cache structure allocated in a coupling facility with CFLEVEL=5 or higher, you can read a data item with a READ_DATA request without having to register interest in the data item, using the REGUSER=NO specification. It should be noted, however, when choosing not to register interest in a data item, that entries with no registered interest are higher-priority candidates for reclaim processing than entries with registered interest.

The VECTORINDEX keyword is not required when using REGUSER=NO to indicate that interest is not to be registered. The VECTORINDEX keyword is required when REGUSER=YES is specified or defaulted to, and whenever OLDNAME is specified to deregister interest in a data item other than the one being read.

For a given local cache vector, the vector entries start at 0. For example, if a vector contains 3 entries, they are numbered 0, 1, and 2. For an illustration of registered interest in data items, see Figure 33 on page 374.

Consider specifying a vector entry that is currently assigned to a data item to another data item if you need to contract the size of your local cache buffer and you want to remap your vector entry indexes to data items so you can keep frequently referenced data items in the contracted local cache buffers.

The following is a scenario:

- You have a protocol that maps each vector entry to a named buffer: for example, vector entry 1 maps to BUFONE, vector entry 2 maps to BUFTWO, and so forth.
- BUFONE contains data item X, BUFTWO contains data item Y, and BUFTHREE contains data item Z. You no longer need data item Y and you want to free as much local cache buffer storage as possible.
- Issue the following request to read data item Z into BUFTWO, to associate vector entry 2 with data item Z, and deregister interest in data item Y:

```
IXLCACHE REQUEST=READ_DATA,ASSIGN=NO,VECTORINDEX=VECTOR2,          X
         OLDNAME=ONAME,NAME=NNAME,…
⋮
VECTOR2  DC  F'2'       VECTOR ENTRY
NNAME    DC  CL16'Z'    NEW NAME
ONAME    DC  CL16'Y'    OLD NAME
⋮
```

- Free the storage allocated to BUFTHREE.
- Compress the vector, using IXLVECTR MODIFYVECTORSIZE, so that the unneeded entry 3 is released.

Specifying a New or Existing Data Item

On each READ_DATA request, you specify whether you want the system to define the data item to the structure. If you do not know whether the data item is currently assigned a directory entry in the cache structure, specify ASSIGN=YES (which is also the system default). If a directory entry for the data item does not exist in the cache structure, this request defines the directory entry to the cache structure. Directory-only cache systems use this option to allocate only the directory entry for a data item in the structure.

If you specify ASSIGN=YES and cache structure resources are available, the system allocates a directory entry for the named data item. If resources are unavailable for the directory entry, and currently allocated resources cannot be reclaimed, the system fails the request. If the data item already has a directory entry allocated, the request does not define a second directory entry, and the system registers the user's interest in the data item. If the data item has a directory and a data entry associated with it, the request reads the data to your local cache buffer and the system re-registers user interest in the data item.

If you do not want the system to define the data item to the structure, code ASSIGN=NO. Store-in and store-through cache users use this option to read a currently cached data item.

If the data is available, ASSIGN=NO causes the system to transfer the requested data to your storage. If the named data item is currently undefined and you code ASSIGN=NO, the system fails the request.

The system registers interest in the data item if the READ_DATA request allocates a directory entry or re-registers interest in the data item if the directory entry is already allocated.

For general information on specifying the vector index entry, see "Specifying the Vector Entry Index on IXLCACHE Requests" on page 397.

Assigning a Storage Class

Each time you read a data item, you must assign the data item to a storage class. To specify the storage class, specify the STGCLASS keyword. If the data item is currently assigned to a storage class, you can assign it to the same class or reassign it to a different class.

The system determines the number of storage classes for the structure based on the value specified on the first invocation of IXLCONN that allocates the structure. The system ignores any subsequent specifications made by subsequent connectors to the structure as long as the structure remains allocated. Storage classes are numbered consecutively from 1 to *n* where *n* is the number of storage classes specified on IXLCONN.

For information on how to use storage classes to manage resource reclamation, see "Managing Cache Structure Resources" on page 380.

Selecting the Buffering Method

On read requests, the system returns data from the cache structure to the local cache buffers. (Optionally, at CFLEVEL=4 or higher, you can specify that the system is not to return data from the cache structure. See "Specifying that No Data Is To Be Read" on page 420.) You can receive data in either a single buffer (the BUFFER keyword) or in multiple buffers (the BUFLIST keyword). Both methods enable you to receive up to 65536 (64K) bytes of data. Adjunct area information associated with the data item is returned in the 64-byte buffer specified by the ADJAREA keyword. If you use the READ_DATA request to register interest in the named data item, you need not specify any buffers to receive data.

You must ensure that your local cache buffer can hold the largest data item that you plan to read. If you read data items of different sizes into the same buffer, ensure that the buffer is as large as the largest data item you read. If you attempt to read a data item that is larger than the buffer, data is not returned to the buffer, and the system returns appropriate return and reason codes.

For information about whether to use a single buffer or multiple buffers and for information on selecting buffer attributes, see "Selecting a Data Buffer For a Request" on page 388.

Specifying the Data to be Read

You issue a READ_DATA request to either define a new data item or read a currently defined data item. When you define a new data item, there is no data transferred to either your local cache buffer or your adjunct area.

When you read a currently defined data item, you can read the data item, read adjunct data, or read both. To read the data item only, identify the local cache buffers by coding either the BUFFER or BUFLIST keywords and their related keywords. Omit the ADJAREA keyword.

- If there is data in the cache structure for the requested data item, the system transfers the data item to your local cache buffer.
- If there is no data in the data entry, your local cache buffer is left unchanged.

To read adjunct data only, code the ADJAREA keyword and omit the BUFFER keyword. Optionally, you can code the BUFLIST keyword and its related keywords. If you code BUFLIST, BUFNUM must specify a value of zero.

- If the cache structure supports adjunct data, the system returns the adjunct data to the area specified on the ADJAREA keyword.
- If the cache structure does not support adjunct data, the area specified on the ADJAREA keyword remains unchanged, and appropriate return and reason codes are returned.

To read both the data item and adjunct data, code BUFFER or BUFLIST and their related keywords, and ADJAREA.

Specifying that No Data Is To Be Read

For cache structures allocated in a coupling facility of CFLEVEL=4 or higher, you can issue a READ_DATA request with the RETURNDATA=NO keyword to suppress the read function so that no data is returned. Instead, the READ_DATA request will register interest in the entry without returning the associated data. Note however, that if the cache structure supports adjunct data and the data exists, the READ_DATA request will return the adjunct data in the area specified on the ADJAREA keyword. If you do not specify the ADJAREA keyword, the system does not return the adjunct data even if it exists. The CAAADJAREAVALID bit in the cache answer area indicates the presence of adjunct data in the area specified by ADJAREA.

Receiving Answer Area Information

On most IXLCACHE requests, the system returns information related to the request in the answer area. You specify the answer area on the ANSLEN and ANSAREA keywords. With certain events, the information in the answer area might not be valid. See “Determining Valid Information in the Answer Area” on page 397.

When the request completes, the system returns information to the answer area. When the request is not valid, the system returns non-zero return and reason codes.

For the mapping of the answer area, see the IXLYCAA mapping macro described in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary). For a description of answer area fields and return and reason codes for the request, see *z/OS MVS Programming: Sysplex Services Reference*.

Defining a New Data Item: Summary

This topic summarizes one way to define a new data item using IXLCACHE REQUEST=READ_DATA:

- To name the data item, code the NAME keyword.
- To request that the system create a directory entry if one does not exist, code ASSIGN=YES, or omit the ASSIGN keyword. The system attempts to allocate the cache structure resources needed to satisfy the request. If the allocation is successful, the system creates a directory entry and registers your interest in the data item. If unallocated resources are unavailable and currently allocated resources cannot be reclaimed to satisfy the allocation, the request fails.
- When you define a new data item, there is no data in the cache structure for the data item, and it is unnecessary to code the BUFFER, BUFLIST, or ADJAREA keywords.
- Code the VECTORINDEX keyword.

You must assign a vector index entry by coding the VECTORINDEX keyword. If you assign a vector index entry that is currently associated with another data item, specify the name of that data item on the OLDNAME keyword. The system creates an association between the new data item and the vector index

entry and registers your interest in the data item. If you also specified the OLDNAME keyword, the system deregisters your interest in the data item specified on OLDNAME.

For More Information

There are other keywords that are required and some that are optional. Some of these keywords apply to all IXLCACHE requests and others apply to just READ_DATA requests. For a description of keywords applicable to all IXLCACHE requests, see:

- "Understanding Synchronous and Asynchronous Cache Operations" on page 386
- "Accessing and Managing Data Within a Cache System" on page 369
- "Requesting Return and Reason Codes" on page 395
- "Defining an Answer Area (ANSAREA)" on page 395

Reading a Data Item: Summary

This topic summarizes ways to read a data item or create a directory entry for a data item if one does not exist using IXLCACHE REQUEST=READ_DATA:

- To identify the data item, code the NAME keyword.
- To indicate that a directory entry for the data item is to be created if it does not already exist, code ASSIGN=YES. Directory-only users of the cache can specify this keyword.
- To read an existing data item in the cache to the local cache buffer, code ASSIGN=NO. If the data item does not exist, the system fails the request. Store-in or store-through cache users can specify this keyword.
- To read an existing data item, identify your local cache buffers by coding either BUFFER, or BUFLIST and their related keywords. The system transfers the data item from the data entry to your local cache buffers. If the data entry is empty (that is, data does not exist for the data item in the cache structure), the system does not transfer data to your local cache buffers, but registers interest for the data item in the directory entry.
- If the cache structure supports adjunct data, use the ADJAREA keyword to read adjunct data. The system transfers the adjunct data from the cache structure to the buffer specified on the ADJAREA keyword.

You must assign a vector index entry by coding the VECTORINDEX keyword. If you assign a vector index entry that is currently associated with another data item, specify the name of that data item on the OLDNAME keyword. The system creates an association between the new data item and the vector index entry and registers your interest in the data item. If you also specified the OLDNAME keyword, the system deregisters your interest in the data item specified on OLDNAME.

For More Information

There are other keywords that are required and some that are optional. Some of these keywords apply to all IXLCACHE requests and others apply to just READ_DATA requests. For a description of keywords applicable to all IXLCACHE requests, see:

- "Understanding Synchronous and Asynchronous Cache Operations" on page 386
- "Accessing and Managing Data Within a Cache System" on page 369
- "Requesting Return and Reason Codes" on page 395
- "Defining an Answer Area (ANSAREA)" on page 395

REG_NAMELIST: Registering Interest in a List of Data Items

You might want to identify a list of data items to the cache structure with one operation. The REG_NAMELIST request allows you to:

- Define (allocate directory entries for) up to 32 new data items to the cache structure and register interest in those data items, and
- Register or re-register interest in up to 32 currently defined data items in the cache structure.

The REG_NAMELIST request type is valid only for a structure allocated in a coupling facility with CFLEVEL=2 or higher.

As with the READ_DATA request, when you define each new data item, you can specify that the system is to try to allocate and initialize a directory entry for the data item. If the system can allocate the directory entry, the system also registers your interest in the data item and validates your local copy of the data item. If the system is unable to allocate the directory entry because resources are unavailable and cannot be reclaimed, the system will terminate the request, perhaps without having processed all the data items you have specified.

For each data item in which you want to register interest, you build a registration block identifying the data item, its associated local cache vector index, and other information specific to the data item. When processing of the REG_NAMELIST request completes, the system returns status information about each of the data items identified by a registration block. This status information indicates whether the user was successfully registered for the data item.

- If successful, the system returns directory information about the entry, and the user's local vector is marked valid.
- If not successful, the system does not return directory information about the entry, and the user's local vector is marked invalid.

Guide to the Topic

"REG_NAMELIST: Registering Interest in a List of Data Items" on page 421 is divided into two sections .

The first section , "IXLCACHE Functions for REQUEST=REG_NAMELIST" on page 422, applies to all REG_NAMELIST requests and includes the following major topics:

- "Specifying a Data Item for Registration Block Processing" on page 422
- "Specifying the Registration Block Buffer" on page 423
- "Specifying the Index Values for Registration Block Processing" on page 424
- "Providing a Storage Area for Returned Registration Information" on page 424
- "Receiving Answer Area Information" on page 424
- "Description of Returned Registration Information" on page 424
- "Restarting a REG_NAMELIST Request that Ends Prematurely" on page 426

The second section , "Registering Interest in a List of Data Items: Summary" on page 427 summarizes a procedure for specifying a list of entries to be registered.

IXLCACHE Functions for REQUEST=REG_NAMELIST

The following functions apply when you specify REQUEST=REG_NAMELIST.

Specifying a Data Item for Registration Block Processing

You identify each data item in a registration block, which you build in the area identified by the BUFFER keyword. You can build up to 32 registration blocks in the BUFFER area. Each registration block is mapped by the mapping macro IXLYCRRB. For a description of IXLYCRRB, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Table 28 on page 423 shows the information that each registration block contains. The third column contains a reference to the READ_DATA function that is analogous to the REG_NAMELIST function.

Table 28. IXLCACHE Registration Block Information		
Field Name	**Description**	**READ_DATA Keyword**
CRRBSTGCLASS	Storage class to which this entry should be assigned.	STGCLASS keyword
CRRBASSIGNCNTL	Directory entry assignment: **0** Do not assign a directory entry for this entry if one does not currently exist. **1** Assign a directory entry for this entry if one does not currently exist.	ASSIGN=YES|NO keyword
CRRBNAMEREPLACECNTL	Name replacement control. **0** Do not deregister interest for the entry specified by CRRBOLDNAME. **1** Deregister interest for the specified local cache vector index and the entry specified by CRRBOLDNAME. The deregistration of the CRRBOLDNAME entry occurs only if the user is currently registered in the CRRBOLDNAME entry with the vector index specified in CRRBVECTORINDEX.	Specification of OLDNAME keyword
CRRBNAME	Name of this entry	NAME keyword
CRRBOLDNAME	Name of the old entry to which the vector index specified by CRRBVECTORINDEX was previously assigned. The registration of this vectorindex for the old entry will be deregistered.	OLDNAME keyword
CRRBVECTORINDEX	Local cache vector index. Used in both the registration of the CRRBNAME entry and the deregistration of the CRRBOLDNAME entry.	VECTORINDEX keyword

Specifying the Registration Block Buffer

When you issue a REQUEST=REG_NAMELIST request, you must identify the buffer that contains the set of registration blocks that specify the data items. Note that the BUFFER specification for REG_NAMELIST requests differs in its addressability requirements from other IXLCACHE requests that use BUFFER. You must use a single buffer (the BUFFER keyword), which is addressable from your primary address space or from your PASN access list. The size of the buffer can be larger than that actually required to hold the maximum (32) number of registration blocks. However, creating a buffer larger than required could result in a performance degradation.

For information on selecting buffer attributes, see “Selecting a Data Buffer For a Request” on page 388.

Specifying the Index Values for Registration Block Processing

On a REG_NAMELIST request, you specify a starting and ending index value for registration block processing with the STARTINDEX and ENDINDEX keywords. Both keywords specify an index value into the set of registration blocks. The registration blocks in the storage area are numbered starting with 1.

- Use STARTINDEX to identify the first registration block in the storage area that the system is to process.
- Use ENDINDEX to identify the last registration block that the system is to process.

The system starts with the registration block indicated by the index for STARTINDEX and attempts to process all registration blocks through the one indicated by the index for ENDINDEX.

Providing a Storage Area for Returned Registration Information

The REG_NAMELIST request must identify a 256-byte storage area where the system can return status information about the results of the registration block processing. To identify the storage area, code the NSBAREA keyword. The NSBAREA area must be addressable in your primary address space or from your PASN access list.

Additional information about the request might be returned in the cache answer area.

Receiving Answer Area Information

On most IXLCACHE requests, the system returns information related to the request in the answer area. You specify the answer area on the ANSLEN and ANSAREA keywords. With certain events, the information in the answer area might not be valid. See “Determining Valid Information in the Answer Area” on page 397.

Below is a description of the answer area information returned when the answer area is valid. The answer area is mapped by the IXLYCAA macro, which is shown in the *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

CAARETCODE
The return code from the IXLCACHE macro. Return code values are defined in the IXLYCON macro.

CAARSNCODE
The reason code associated with the return code from the IXLCACHE macro. Reason code values are defined in the IXLYCON macro.

CAASTGCLFULL
The storage class from which a reclaiming operation failed, thus causing the failure of the REG_NAMELIST request because the system could not obtain directory resources to satisfy the request.

CAARNLINDEX
Index of the current registration block. A value of zero indicates that no registration blocks were successfully processed. See “Restarting a REG_NAMELIST Request that Ends Prematurely” on page 426 for a description of the CAARNLINDEX value when specific reason codes are returned.

Description of Returned Registration Information

The system returns state information for each processed data item included in your registration block area. Mapping macro IXLYNSB maps the information. See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for a description of IXLYNSB.

IXLYNSB contains two arrays. The first array (the “NSB array”) contains state information for the corresponding named cache entry, including whether the registration was successful. The second array (the “NSBINVLCVINUM array”) contains the invalidated vector index when the corresponding named cache entry's prior registration was invalidated as a result of the REG_NAMELIST request. There is a one-to-one correlation between a registration block entry in the BUFFER area and an element in each of the arrays in the NSBAREA area. Therefore, the same index number that designates an IXLYCRRB entry in the BUFFER will also designate the corresponding NSB array entry and NSBINVLCVINUM array entry.

Table 29 on page 425 describes the information returned for each data item.

Table 29. IXLCACHE Registration Block Returned Information

Field Name	Description
NSBCHANGED	Change status of cached subsystem data **0** Unchanged **1** Changed
NSBDATACACHED	Indicator of whether the associated data entry is cached or is a directory-only entry. **0** Data not cached **1** Data cached
NSBPARITY	Parity as recorded in the item's directory entry
NSBCOLOCKSTATE	State of the castout lock **00 (CAACOLS_RESET)** Reset state, which is entered when the name is assigned to the directory entry or when the castout lock is reset to zeros. **01 (CAACOLS_READFORCASTOUT)** Read-for-castout state, which is entered when the castout lock is obtained by a CASTOUT_DATA request. **10 (CAACOLS_WRITEWITHCASTOUT)** Write with castout, which is entered when the castout lock is obtained by a WRITE_DATA request specifying GETCOLOCK=YES.
NSBINVLCVI	Indicator of whether a local cache vector index was invalidated because interest for the associated item was re-registered using a different vector index. **0** The associated NSBINVLCVINUM array entry is not valid. **1** The associated NSBINVLCVINUM array entry contains the invalidated local cache vector index number.
NSBREGPERFORMED	Indicator of whether the registration was successfully performed. **0** The registration was not successfully performed. For this data item: • No directory entry for the name exists • The user is not registered in the entry • The user's local cache buffer for the entry was marked invalid • The other NSB information for the named entry was not returned. **1** The registration was successful for the entry name and local cache vector index in the corresponding registration block. For this data item: • A directory entry for the name exists • The user was registered in the named entry as requested • The user's local cache buffer for the entry was marked valid • The other NSB information for the named entry was returned.

Table 29. IXLCACHE Registration Block Returned Information (continued)	
Field Name	**Description**
NSBELEMNUM	The entry size expressed as the number of elements in the entry. (This value is returned only when the cache structure is allocated in a coupling facility of CFLEVEL=4 or higher.)
NSBINVLCVINUM	The value of the local cache vector index that was invalidated when interest for the data item was re-registered using a different vector index. NSBINVLCVI indicates the validity of this value.

Restarting a REG_NAMELIST Request that Ends Prematurely

The IXLCACHE REQUEST=REG_NAMELIST request can complete prematurely. When a request completes prematurely, the system will not have processed all the registration blocks identifying the data items. To continue processing the registration blocks, you must restart the request.

Be sure to process the information returned from this request before reissuing the request. The data returned from this request will be overwritten if you specify the same buffer address. Continue to reissue the request until the return code indicates that all processing has completed.

When a request completes prematurely, the system returns an index value into the list of registration block entries. This value is returned in the CAARNLINDEX field of the answer area. Use this index value to restart the request.

Reasons for which a REG_NAMELIST can complete prematurely are:

- The request has exceeded the model-dependent time-out criteria. The index of the next registration block to be processed is returned in the answer area (ANSAREA). All registration blocks preceding this one are processed.
- The user has specified an incorrect storage class or the target storage class is full. The index of the failing registration block is returned in the answer area. All registration blocks preceding the failing registration block are processed.
- The user has specified an incorrect local cache vector index. The index of the failing registration block is returned in the answer area. None of the registration blocks are processed.

The information in the NSBAREA for the registration blocks processed before the premature completion of the REG_NAMELIST request might or might not contain meaningful information. The following return and reason codes from the REG_NAMELIST request indicate which registration blocks were processed prior to the request's premature completion.

IXLRETCODEOK
: All NSB array and NSBINVLCVINUM array entries that have an index value greater than or equal to STARTINDEX and less than or equal to ENDINDEX contain meaningful information.

IXLRSNCODETIMEOUT
: All NSB array and NSBINVLCVINUM array entries that have an index value greater than or equal to STARTINDEX and less than CAARNLINDEX contain meaningful information. To process the remaining registration blocks, update STARTINDEX with the value in CAARNLINDEX and reissue the REG_NAMELIST request.

IXLRSNCODESTRFULL IXLRSNCODEBADSTGCLASS
: All NSB array and NSBINVLCVINUM array entries that have an index value greater than or equal to STARTINDEX and less than CAARNLINDEX contain meaningful information. The registration block indexed by CAARNLINDEX was not processed either because the target storage class was full or because an incorrect storage class was specified. If possible, correct the error in the registration block. To process the remaining registration blocks (including the corrected registration block), update STARTINDEX with the value in CAARNLINDEX and reissue the REG_NAMELIST request. If it is not possible to correct the error in the registration block, update STARTINDEX with the value in CAARNLINDEX plus one and reissue the REG_NAMELIST request.

IXLRSNCODEBADVECTOROP
No NSB array or NSBINVLCVINUM array entries contain meaningful information. The registration block indexed by CAARNLINDEX contains the invalid vector index. Correct that vector index value and reissue the REG_NAMELIST request with the same STARTINDEX and ENDINDEX values.

A restarted request can also complete prematurely due to either a timeout or a failure on a later registration block in the list. Restart the request using the procedure described.

Registering Interest in a List of Data Items: Summary

You use a register name list request to register interest in up to 32 data items.

The request must identify the data items by building a list of registration blocks in a buffer. Each registration block contains information about the data item, such as name, storage class, and whether a directory entry should be assigned.

The request must indicate the first and last registration block in the list of registration blocks that the system is to process.

- To identify the first registration block, use the STARTINDEX keyword.
- To identify the last registration block, use the ENDINDEX keyword.

The request can complete prematurely for the following reasons:

- The coupling facility timed-out.
- A registration block specified an incorrect storage class or the target storage class was full.
- A registration block specified an incorrect local cache vector index.

Each time a request completes prematurely, the system returns an index value. You can use the index value to identify the registration block that might have caused the premature completion. You can also use the index value to restart the request.

There are other keywords that are required and some that are optional. Some of these keywords apply to all IXLCACHE requests and others apply to just REG_NAMELIST requests. For a description of keywords applicable to all IXLCACHE requests, see:

- “Understanding Synchronous and Asynchronous Cache Operations” on page 386
- “Accessing and Managing Data Within a Cache System” on page 369
- “Requesting Return and Reason Codes” on page 395
- “Defining an Answer Area (ANSAREA)” on page 395

CASTOUT_DATA: Casting Out Data from a Cache Structure

Casting out a changed data item means reading it from the cache structure and writing it to permanent storage. When you cast out a data item from the cache structure, the data item is not deleted from the structure, but remains in the cache structure.

When you cast out a data item, you obtain the cast-out lock to prevent other users from casting out the same data item. The system considers the data for a data item that is cast out as unchanged. Even though the data item is marked unchanged, it is unavailable for reclaim until its cast-out lock is released. Once the data item is cast out and a user releases the cast-out lock for the data item, the storage resources for the data item are available for reclaim.

Note: Depending on the IXLCACHE request, two cast-out lock states exist. One is associated with UNLOCK_CASTOUT, and one is associated with WRITE_DATA. See “Identifying the Cast-Out Locks to Release” on page 435 and “Obtaining the Cast-Out Lock on Write Requests” on page 406.

Extended Function

With a coupling facility of CFLEVEL=12 or higher, it is possible to specify a list of up to 8 data entries for cast-out processing. See “CASTOUT_DATALIST: Casting Out a List of Data Items” on page 432.

Reasons for Casting out Data

Periodic casting out of changed data items from the structure can improve the likelihood that the system can reclaim storage resources for new requests. The number of data items that can be defined to the cache structure at any given time is finite. If you try to define a new data item to the cache structure or increase the size of an existing data entry, and there is insufficient unused cache structure storage available for the new data item, the system attempts to reclaim storage.

The system attempts to reclaim storage from unchanged data items stored in the cache structure. When you cast out data items and release the lock, the data items are considered unchanged and available for reclaim. If there are no unchanged data items, or insufficient storage is available from unchanged data items, the request fails.

How you cast out data items depends on the cast-out class you assign to each data item, and how your protocol uses these cast-out class assignments. For information on how to assign cast-out classes and on developing cast-out protocol, see “Casting out Data Items and Reclaim Processing” on page 385.

It is also important to cast out changed data from the cache structure because the changed data might be lost if the coupling facility or structure fails. The more often you cast out changed data, the fewer changed data items you lose if a failure occurs, and you thus minimize the amount of recovery processing you need to perform.

Cast-out Requests

To read the data item for cast-out, you issue a REQUEST=CASTOUT_DATA request. The system locks the data item for cast-out on your behalf, marks the data item unchanged and transfers the requested data to your storage. Locking the data item for cast-out prevents another user from concurrently casting out the same data item. Locking a data item for cast-out, however, does not prevent another user from reading the data item from the cache structure or from updating the data item in the cache structure.

After completion of the REQUEST=CASTOUT_DATA request, you must write the data item to permanent storage. After the write operation completes, you must release the cast-out lock by issuing either a REQUEST=UNLOCK_CASTOUT request or a REQUEST=UNLOCK_CO_NAME request. If other users have not updated the data while the cast-out lock is held, the system releases the cast-out lock and removes the data item from the cast-out class to which it had been assigned.

If you are unable to write the data item to permanent storage, you can request that the system mark the data item as changed when you release the lock on an UNLOCK_CASTOUT request or UNLOCK_CO_NAME request. By marking the data item as changed, you ensure that the data item's cache structure resources are not reclaimed before you, or another user, casts out the data item. Also, the data item remains associated with its cast-out class. (See “Changing the Directory Entry for the Data Item” on page 437.)

The data item is also marked as changed if another user updates the data item while you hold the cast-out lock. While you hold the lock, the data item being cast out is still available in the cache structure to be read or updated. If another user updates the data item in the cache structure while you hold the lock, that user's request causes the data item to be marked changed and to be assigned to the specified cast-out class. When you issue the UNLOCK_CASTOUT or UNLOCK_CO_NAME request for the data item, the system still considers the data item to be associated with the cast-out class that the user specified when the data item was updated.

The cast-out process involves three major tasks:

- **Reading the data item for cast-out from the cache structure**: To read a data item for cast-out, you issue a REQUEST=CASTOUT_DATA request. See “IXLCACHE Functions for CASTOUT_DATA” on page 429.
- **Writing the data item to its permanent storage**: IXLCACHE does not provide services for writing the data item to permanent storage. You must use other system services to perform this task. See the documentation for the method you use to access permanent storage.
- **Unlocking the cast-out lock**: You must unlock the cast-out lock by issuing a REQUEST=UNLOCK_CASTOUT or REQUEST=UNLOCK_CO_NAME request. See “UNLOCK_CASTOUT:

Releasing Cast-Out Locks” on page 434 and “UNLOCK_CO_NAME: Releasing a Single Cast-Out Lock” on page 439.

Plan to process an entire cast-out class at a time. Perform the first two steps for each data item in the class, and then the third step, passing all the names that were cast out in a list.

Guide to the Topic

“CASTOUT_DATA: Casting Out Data from a Cache Structure” on page 427 is divided into two sections .

The first section , “IXLCACHE Functions for CASTOUT_DATA” on page 429, applies to all CASTOUT_DATA requests and includes the following major topics:

- “Specifying the Data Item Name” on page 429
- “Registering Interest in the Data Item for CASTOUT_DATA Requests” on page 429
- “Specifying a Process Identifier” on page 430
- “Selecting the Buffering Method” on page 430
- “Specifying the Data to be Cast Out” on page 430
- “Receiving Answer Area Information” on page 431

The second section , “Casting Out A Data Item: Summary” on page 431 summarizes a procedure for casting-out data from a cache structure.

IXLCACHE Functions for CASTOUT_DATA

The following topics apply to casting out data from the cache structure when you specify REQUEST=CASTOUT_DATA.

Specifying the Data Item Name

All CASTOUT_DATA requests must identify the data item for cast-out. To identify the data item, specify the data item name on the NAME keyword.

Registering Interest in the Data Item for CASTOUT_DATA Requests

Users can perform cast-out processing for a data item without having to register interest in the data item. To cast-out a data item for which you do not want to register interest, code REGUSER=NO (which is the system default). The system does not register interest, and if you currently have registered interest in the data item, the system does not deregister your interest unless another user updates the data item while you hold the cast-out lock. Using WHENREG=NO on the CASTOUT_DATA request allows you to develop a cast-out protocol that is independent of the regular registration/deregistration of interest in shared data items that occur with other IXLCACHE requests. (For a description of the registration/deregistration process, see Figure 33 on page 374.)

To register interest on a CASTOUT_DATA request, specify REGUSER=YES and the VECTORINDEX keyword to specify a vector entry for the data item. You can specify the vector entry index currently assigned to the data item in the cache structure, a vector entry index that is not currently assigned to the data item in the cache structure, or a vector entry index that is currently assigned to a data item in the cache structure to another data item.

Specify REGUSER=NO to indicate that interest in a data item is not to be registered. With a structure allocated in a coupling facility with CFLEVEL=5 or higher, you can also optionally deregister interest in the data item specified by OLDNAME. The VECTORINDEX keyword is required whenever OLDNAME is specified to deregister interest in a data item other than the one being read.

For general information on specifying the vector index entry, see “Specifying the Vector Entry Index on IXLCACHE Requests” on page 397.

Specifying a Process Identifier

Optionally, you can identify your task or process as the holder of the cast-out lock for the named data item. You identify your task or process on the PROCESSID keyword. When you obtain the cast-out lock, the process identifier becomes part of the cast-out lock along with your connection identifier. If another user invokes a service that returns the value of the cast-out lock in the answer area while your connection holds the cast-out lock, that user can identify, not only your connection, but also the task or process that holds the lock.

Selecting the Buffering Method

The system returns the data read for cast-out to your local cache buffers. You can receive data in either a single buffer (the BUFFER keyword) or in multiple buffers (the BUFLIST keyword). Both methods enable you to receive up to 65536 (64K) bytes of data. The system returns adjunct information read for cast-out to the 64-byte buffer specified by the ADJAREA keyword.

You must ensure that your local cache buffer can hold the largest data item that you plan to cast out. If you read data items of different sizes into the same buffer, ensure that the buffer is as large as the largest data item you cast out. If you attempt to read a data item that is larger than the buffer, data is not returned to the buffer, and the system returns appropriate return and reason codes. Even if your buffer is too small to contain the data, the request still registers your interest in the data item if you have specified REGUSER=YES, and the system still obtains the cast out lock for the data item.

For information about whether to use a single buffer or multiple buffers and for information on selecting buffer attributes, see “Selecting a Data Buffer For a Request” on page 388.

For information about whether to use a single buffer or multiple buffers and for information on selecting buffer attributes, see “Design Considerations for Choosing the Buffer Format” on page 391.

Specifying the Data to be Cast Out

When you cast out data, you can cast out only the data for the data item, cast out only adjunct data, or cast out both.

For example, if the data for a data item needs to be backed up on permanent storage, but the adjunct data for the data item contains control information that does not need to be backed up, you need only cast out the data for the data item. On the other hand, for adjunct data that must be backed up on permanent storage, you need to cast out both the data and the adjunct data for the data item.

To cast-out only the data for the data item, identify the local cache buffers by coding either BUFFER or BUFLIST (you must specify one of these keywords) and their related keywords. Omit the ADJAREA keyword.

To cast-out adjunct data only, code the ADJAREA keyword and omit the BUFFER keyword. Optionally, you can code the BUFLIST keyword and its related keywords. If you code BUFLIST, BUFNUM must specify a value of zero.

To cast out both the data item and adjunct data, code BUFFER or BUFLIST and their related keywords, and ADJAREA.

Consider the following when you issue these requests:

- If there is data in the cache structure for the requested data item, the system transfers the data to your local cache buffer.
- If you specify ADJAREA and the cache structure supports adjunct areas, the system returns the adjunct data to the area specified on ADJAREA.
- If the cache structure does not support ADJAREA, the area specified on ADJAREA remains unchanged.
- If there is no data in the data entry or no adjunct data, your local cache buffer and adjunct area are left unchanged.

Cast out and Unchanged Data

If the data entry does not contain changed data, the system does not obtain the cast-out lock for the data item and does not cast out the data to your local cache buffers. Return and reason codes indicate the error, and the system does not register interest for the user in the data item.

Receiving Answer Area Information

On most IXLCACHE requests, the system returns information related to the request in the answer area. You specify the answer area on the ANSLEN and ANSAREA keywords. With certain events, the information in the answer area might not be valid. See "Determining Valid Information in the Answer Area" on page 397.

When the request completes, the system returns information to the answer area. When the request is not valid, the system returns non-zero return and reason codes.

For the mapping of the answer area, see the IXLYCAA mapping macro described in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary). For a description of answer area fields and return and reason codes for the request, see *z/OS MVS Programming: Sysplex Services Reference*.

Casting Out A Data Item: Summary

The three major tasks of a cast-out operation are:

- Reading the data item for cast-out from the cache structure and obtaining the cast-out lock.
- Writing the data item to permanent storage.
- Releasing the data item's cast-out lock.

To identify the data item, code the data item name on the NAME keyword.

You can cast-out a data item regardless of whether you have registered interest in the data item. You also have the option to register interest as part of the cast-out operation. If you do not want to register interest, code REGUSER=NO. To register interest, code REGUSER=YES and VECTORINDEX to assign a vector entry. If you assign a vector entry that is currently assigned to another data item, specify that data item name on the OLDNAME keyword. The system deregisters your interest in the data item specified on the OLDNAME keyword.

Optionally, you can identify your task or process as the holder of the data item's cast-out lock. To do this, specify the task or process identifier on the PROCESSID keyword. By providing this identifier, you enable other users to determine which task or process holds the data item's cast-out lock.

To read a data item for cast-out, identify your local cache buffers by coding either BUFFER, or BUFLIST and their related keywords. The system transfers the data item from the data entry to your local cache buffers. If the data entry does not contain changed data, the system does not obtain the cast-out lock for the data item and does not cast out the data to your local cache buffers. Return and reason codes indicate the error, and the system does not register interest for the user in the data item.

If the cache structure supports adjunct data, you can read the adjunct data for cast-out by coding the ADJAREA keyword. The system transfers the adjunct data from the cache structure to the buffer specified on the ADJAREA keyword.

There are other keywords that are required and some that are optional. For a description of keywords that are applicable to all IXLCACHE requests, see:

- "Understanding Synchronous and Asynchronous Cache Operations" on page 386
- "Accessing and Managing Data Within a Cache System" on page 369
- "Requesting Return and Reason Codes" on page 395
- "Defining an Answer Area (ANSAREA)" on page 395

CASTOUT_DATALIST: Casting Out a List of Data Items

You might want to cast-out a set of data entries with one operation. The CASTOUT_DATALIST request allows you to identify a set of up to 8 entries for cast-out processing.

The CASTOUT_DATALIST request specifies that a cast-out lock be obtained for the set of entries for the connection and optionally, the process, identified by CONTOKEN and PROCESSID. The directory entry change bit will be updated for each entry indicating that each entry contains unchanged data.

The CASTOUT_DATALIST request type is valid only for a structure allocated in a coupling facility with CFLEVEL=12 or higher.

Guide to the Topic

"CASTOUT_DATALIST: Casting Out a List of Data Items" on page 432 is divided into two sections .

The first section , "IXLCACHE Functions for REQUEST=CASTOUT_DATALIST" on page 432, applies to all CASTOUT_DATALIST requests and includes the following major topics:

- "Specifying the data items to be cast-out" on page 432
- "Specifying a Process Identifier" on page 432
- "Selecting a Buffering Method" on page 432
- "Situations that cause CASTOUTLIST entry processing to be discontinued" on page 433
- "Specifying the Index Values for Data Item Processing" on page 433
- "Restarting a REQUEST=CASTOUT_DATALIST Request that ends prematurely" on page 433
- "Receiving answer area information" on page 433

The second section , "Casting out a list of data items: Summary" on page 434 summarizes a procedure for casting-out a list of data items from a cache structure.

IXLCACHE Functions for REQUEST=CASTOUT_DATALIST

The following functions apply when you specify REQUEST=CASTOUT_DATALIST.

Specifying the data items to be cast-out

The list of entries to be cast-out is contained in the 128-byte CASTOUTLIST field. Each entry is 16-bytes; up to eight entries can be included in CASTOUTLIST.

Specifying a Process Identifier

You can optionally specify a process identifier that will be placed in the cast-out lock along with the connection identifier. The one-byte PROCESSID is a user-defined value.

Selecting a Buffering Method

When you issue a CASTOUT_DATALIST request, you must specify a storage area to contain the data to be cast-out. You can specify either a single buffer (the BUFFER keyword) or multiple buffers (the BUFLIST keyword). Both methods enable you to receive up to 64K bytes of data.

Understanding the Use of the Buffer Areas

The output from a CASTOUT_DATALIST request is placed in several different areas. The first entry in CASTOUTLIST that is processed is handled differently from the remaining entries. For the first entry, the DEIB is placed in DEIBAREA and adjunct data, if it is present, is placed in ADJAREA. The data buffer for the first entry is placed in the storage area pointed to by either BUFFER or BUFLIST. For subsequent entries in CASTOUTLIST, the storage area will contain the DEIB, adjunct data if it exists, and the data area for each entry.

When the size of the BUFFER or BUFLIST is not large enough to contain the data area for the first entry in CASTOUTLIST referenced by STARTINDEX, no entries are processed. The number of elements in the entry specified by STARTINDEX is returned in the answer area.

When the remaining space in the BUFFER or BUFLIST is not large enough to contain the data area for the current entry in the CASTOUTLIST, the index of the name in CASTOUTLIST being processed is returned in the answer area, along with the number of elements in the entry. All prior entries in the CASTOUTLIST were processed.

Specifying the Index Values for Data Item Processing

CASTOUTLIST can contain the 16-byte names of up to eight entries. Use the STARTINDEX and ENDINDEX keywords as index values to identify the entry names to be processed. The entry names are numbered starting with 1. The entry names are processed sequentially beginning with STARTINDEX and continuing through ENDINDEX.

Situations that cause CASTOUTLIST entry processing to be discontinued

The following scenarios describe the results of processing an entry in the CASTOUTLIST that is unable to be cast-out.

- If another connection or process currently holds the cast-out lock for the entry being processed, the castout lock is not obtained. The index of the name in the CASTOUTLIST, the value of the cast-out lock, and the value of the cast-out lock state are returned in the answer area. All prior entries in the CASTOUTLIST were processed.
- If entry data is not cached, or the cached entry data is not changed, the index of the name in the CASTOUTLIST, the changed subsystem data indicator, and the data-cached indicator are returned in the answer area. All prior entries in the CASTOUTLIST were processed.
- If the entry name in the CASTOUTLIST being processed is not in the directory, IXLCACHE processing stops and the index of the name in the CASTOUTLIST is returned in the answer area. All prior entries in the CASTOUTLIST were processed.

Restarting a REQUEST=CASTOUT_DATALIST Request that ends prematurely

An IXLCACHE REQUEST=CASTOUT_DATALIST might complete prematurely if the request exceeds the time-out criteria for the coupling facility. (Time-out criteria is model-dependent.) Each time a request completes prematurely, the system returns an index value into the list of entries in the CAACDLINDEX field of the answer area. Use this index value to restart the request so it can process the remaining entries in the list specified by CASTOUTLIST. Reinitialize the STARTINDEX index value to the value returned in CAACDLINDEX. To restart a request, after reinitializing STARTINDEX, reissue IXLCACHE REQUEST=CASTOUT_DATALIST. To ensure that you do not alter the meaning of the request that completed prematurely, the restarted request should specify the same keywords and values (with the exception of the index value specified on STARTINDEX) as the request that completed prematurely.

Be sure to process the information returned from this request before reissuing the request. The data returned from this request will be overwritten if you specify the same buffer address. Continue to reissue the request until the return code indicates that all processing has completed.

For general information about restarting requests, see Restarting a Request that Ends Prematurely.

Receiving answer area information

On most IXLCACHE requests, the system returns information related to the request in the answer area. You specify the answer area on the ANSLEN and ANSAREA keywords. With certain events, the information in the answer area might not be valid. See “Determining Valid Information in the Answer Area” on page 397.

When the request completes, the system returns information to the answer area. When the request is not valid, the system returns non-zero return and reason codes.

For the mapping of the answer area, see the IXLYCAA mapping macro described in the *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary). For a description of answer area fields and return and reason codes for the request, see *z/OS MVS Programming: Sysplex Services Reference*.

Casting out a list of data items: Summary

For cache structures allocated in a coupling facility of CFLEVEL=12 or higher, use the CASTOUT_DATALIST function to specify up to 8 data items for cast-out processing. You identify the data items by coding the 16-byte entry names in the CASTOUTLIST storage area. The system references the entries in the CASTOUTLIST by an index into the list and processes the entries sequentially.

The output from the request is placed in the storage areas specified by BUFFER, BUFLIST, DEIBAREA, and ADJAREA, if appropriate. From these areas, the data can be written to permanent storage.

For a discussion of keywords applicable to all IXLCACHE requests, see:

- "Understanding Synchronous and Asynchronous Cache Operations" on page 386
- "Accessing and Managing Data Within a Cache System" on page 369
- "Requesting Return and Reason Codes" on page 395
- "Defining an Answer Area (ANSAREA)" on page 395

UNLOCK_CASTOUT: Releasing Cast-Out Locks

To release one or more cast-out locks held by your connection, issue an IXLCACHE REQUEST=UNLOCK_CASTOUT request. After you have cast out and written the data for the data item to permanent storage, you release a cast-out lock. If you fail to update permanent storage for the data item that you have cast out, you need to release the lock, but you indicate that the data item is changed on the UNLOCK_CASTOUT request. As a result, the system does not consider the data item eligible for reclaim. The data item then needs to be cast out again so that you can successfully write the changes to permanent storage.

You have the option to release one lock at a time or multiple locks. To reduce processing overhead, you might write a number of data items associated with a specific cast-out class, for example, to permanent storage, then release their cast-out locks with one invocation of REQUEST=UNLOCK_CASTOUT.

When you release a cast-out lock for a data item, the system updates the directory entry to indicate that the lock is released. If the data item has not been updated by another user while the lock is held, the system also disassociates the data item from the cast-out class.

As a user, you can also update the directory entry for the data item by providing data for the user-defined data field and by changing the parity bits. You can also indicate on the UNLOCK_CASTOUT request that the system mark the data as changed, which makes the resources unavailable for reclaim.

While you hold a cast-out lock for a data item, another user can write changed data to the data item. If you issue REQUEST=UNLOCK_CASTOUT after another user has changed the data, the system releases the cast-out lock, but does not update the directory entry with any of the data that you provide. Instead, the system considers the data item as changed and updates the directory entry as specified on the write request of the user that makes the change. The data item also remains associated with the cast-out class specified by the user on the WRITE_DATA request.

Guide to the Topic

"UNLOCK_CASTOUT: Releasing Cast-Out Locks" on page 434 is divided into two sections .

The first section , "IXLCACHE Functions for REQUEST=UNLOCK_CASTOUT" on page 435, applies to all UNLOCK_CASTOUT requests and includes the following major topics:

- "Identifying the Cast-Out Locks to Release" on page 435
- "Initializing Elements in the List of Name Elements" on page 435

- “Selecting a Buffering Method” on page 436
- “Processing an UNLOCK_CASTOUT Request that Ends Prematurely” on page 436
- “Receiving Answer Area Information” on page 437
- “Changing the Directory Entry for the Data Item” on page 437

The second section , “Releasing Cast-Out Locks: Summary” on page 438 summarizes a procedure for unlocking cast-out locks.

IXLCACHE Functions for REQUEST=UNLOCK_CASTOUT

The following functions apply when you specify REQUEST=UNLOCK_CASTOUT.

Identifying the Cast-Out Locks to Release

To identify the data items whose cast-out locks are to be released, you build a list of names in a buffer. The mapping macro IXLYCUNB maps each name element in the list. For a description of IXLYCUNB, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Each name element in the list identifies one data item and contains the following information:

- **Data item name:** The name of the data item whose cast-out lock is to be released.
- **User-defined data:** Data that is to replace the current user-defined data in the directory entry for the data item.
- **Parity bits:** Parity bits that are to replace the current parity bits in the directory entry for the data item.
- **Change indicator:** An indicator to allow the system to mark the data item as changed in the cache structure after the lock is released.

When you issue IXLCACHE REQUEST=UNLOCK_CASTOUT, you can release cast-out locks for all data items identified in the list of name elements or for a subset of the data items. To identify the set of data items whose cast-out locks are to be released, use the FIRSTNAME and LASTNAME keywords. Both keywords specify an index value into the list of name elements. Use FIRSTNAME to identify the first element for the first data item in the list that the system is to process and LASTNAME to identify the last element for the last data item that the system is to process. The system starts with the data item indicated by the index for FIRSTNAME and attempts to release the cast-out locks for all data items through the data item indicated by the index for LASTNAME.

For example, you have built a list of seven name elements that identify data items named A, B, C, D, E, F, and G. The name element that identifies data item A starts at buffer offset 1. All other name elements follow in contiguous storage:

- To release the cast-out locks for A, B, and C, FIRSTNAME must specify an index of 1 and LASTNAME an index of 3.
- To release the cast-out locks for C, D, and E, FIRSTNAME must specify an index of 3 and LASTNAME an index of 5.
- To release the cast-out lock for data item G only, FIRSTNAME and LASTNAME both must specify an index of 7.

Initializing Elements in the List of Name Elements

For each name element in the list, you must initialize the data item name. If you plan to use the user-defined data field and the parity bits, you must also initialize these fields. Otherwise, you can specify zeros for the user-defined data field and parity bits. You must also indicate whether the system marks the data item as changed. Indicating the changed status of a data item depends on whether you successfully write the data item to permanent storage.

During normal cast-out processing, you write changed data for each data item to permanent storage. For a successful write operation, issue the request to release the lock and ensure that the value of the change indicator informs the system to leave the change state as is. As long as no other user has updated the

data item while you held the lock, the system considers the data item as unchanged and the storage resources are eligible for reclaim. For an unsuccessful write operation to permanent storage, issue the request to release the lock and set the change indicator to mark the data item as changed. By marking the data item as changed, the system cannot reclaim data item resources for other requests so that you can preserve the changes until you are able to write the data item to permanent storage.

Specifying a Process Identifier

Optionally, you can identify your task or process as the lock holder for one or more cast-out locks for the named data items on the PROCESSID keyword. While you hold a cast-out lock for a data item, another user can invoke a service that returns the value of the cast-out lock in the answer area, and the user can identify, not only the connection, but also the task or process that holds the lock.

Selecting a Buffering Method

When you issue a REQUEST=UNLOCK_CASTOUT request, you must specify a buffer that contains the list of name elements. You can use a single buffer (the BUFFER keyword) or multiple buffers (the BUFLIST keyword). Either method enables you to build a maximum of 2048 name elements.

For information about whether to use a single buffer or multiple buffers and for information on selecting buffer attributes, see "Design Considerations for Choosing the Buffer Format" on page 391.

Processing an UNLOCK_CASTOUT Request that Ends Prematurely

The completion of IXLCACHE REQUEST=UNLOCK_CASTOUT request can be affected for the following reasons:

- The request exceeds the time-out criteria for the coupling facility. (Time-out criteria is model-dependent.)
- A name element specified a data item that is not defined to the cache structure.
- The connection specified on the CONTOKEN keyword, or the process or task specified on the PROCESSID keyword, does not hold the cast-out lock for one of the data items specified by the name element.
- A name element specified a data item with parity bits that are not valid.
- A name element specified a data item that holds a cast-out lock in a write-with-cast-out state. The write-with-cast-out lock state is not compatible with the change indicator for the data item.

Each time a request completes prematurely, the system returns an index value into the list of name elements in the CAAULINDEX field of the answer area. Use this index value in CAAULINDEX to:

- Locate the name element that specified the undefined data item, the data item with a connector, task, or process that does not hold the cast-out lock, the data item that specified parity bits that are not valid, or the data item that requested an incompatible update for the change indicator.
- For time-out problems, restart the request so it can process the remaining elements in the list of name elements.

Locating a Name Element

Use the index value to identify the data item that is not defined, the data item for which the connection, task, or process does not hold the cast-out lock, the data item that specified parity bits that are not valid, or the data item that requested an incompatible update for the change indicator.

For example, you specify a list of five name elements for data items named A, B, C, D, and E. The list starts at offset 1 of the buffer. When you issue the REQUEST=UNLOCK_CASTOUT request, FIRSTNAME specifies an index of 1 (for data item A) and LASTNAME an index of 5 (for data item E). If data item C is not defined to the cache structure, the system prematurely completes the request and returns an index value of 3 in CAAULINDEX that corresponds to data item C.

Restarting a Request

Use the index value to restart a prematurely completed request. Before restarting a request, you must reinitialize the index value that FIRSTNAME specifies. If the request exceeded a time-out value for the coupling facility, reinitialize the FIRSTNAME index value to the value returned in CAAULINDEX.

If the request specifies a data item that is not in the cache structure, a data item for which the connection, task, or process does not hold the cast-out lock, a data item that specified parity bits that are not valid, or a data item that requested an incompatible update for the change indicator, and the condition is unexpected, check to ensure that all users of the cache structure are following the established protocols. If your protocol expects these conditions to occur and you want to restart the request, increase the value in the CAAULINDEX by 1 (as long as the original value is not the last element in the list), so that it points to the next name element in the list. When you reissue the request, specify the new index value on FIRSTNAME.

In the previous example that described missing data item C indicated by index value 3 in CAAULINDEX, specify 4 in CAAULINDEX. Then for FIRSTNAME, also specify 4 and reissue the request. When you reissue the request, the system can start to release the cast-out lock starting with data item D.

Note: If the problem is the last name element in the list, ensure that the new index value for FIRSTNAME does not exceed the value for LASTNAME. For example, if the fourth element in a list of four caused the problem (CAAULINDEX returns a value of 4) and you want to restart the request with the first element, specify 1 for FIRSTNAME. Do not increase CAAULINDEX by 1 and specify that value (5) for FIRSTNAME, or you will receive an error.

To restart a request, after reinitializing FIRSTNAME, reissue IXLCACHE REQUEST=UNLOCK_CASTOUT. To ensure that you do not alter the intent of the request that completed prematurely, the restarted request should specify the same keywords and values (with the exception of the index value specified on FIRSTNAME) as the request that completed prematurely. For general information about restarting requests, see Restarting a Request that Ends Prematurely.

Receiving Answer Area Information

On most IXLCACHE requests, the system returns information related to the request in the answer area. You specify the answer area on the ANSLEN and ANSAREA keywords. With certain events, the information in the answer area might not be valid. See “Determining Valid Information in the Answer Area” on page 397.

When the request completes, the system returns information to the answer area. When the request is not valid, the system returns non-zero return and reason codes.

For the mapping of the answer area, see the IXLYCAA mapping macro described in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary). For a description of answer area fields and return and reason codes for the request, see *z/OS MVS Programming: Sysplex Services Reference*.

Changing the Directory Entry for the Data Item

If a user does not write changed data to a data item while you hold the cast-out lock, you can indicate to the system whether you want to mark the data item as changed or indicate that the system is not to change the status of the data item when you release the lock.

If you indicate that the system is not to change the status of the data item when you release the lock and no other user has updated the data item in the cache while the lock was held, the system is able to reclaim resources from the data item for other requests. If you indicate that the data item is changed when you release the lock or a user has updated the data item while the lock was held, the system considers the data item as changed, and the system cannot reclaim data item resources.

Indicating to the System not to Change the Status of the Data Item

If you do not want to change the status of the data item and the data item is unchanged, the system does the following:

- Updates the user-data and parity bits in the directory entry with the data you provide in the name element.
- Disassociates the data item from the cast-out class to which it was assigned.

Marking the Data Item as Changed

To request that the data item be marked as changed, do the following:

- Specify B'1' in the CUNBCHANGEOI field of the mapping macro IXLYCUNB for the name element of the data item in the list.

If you request that the system mark the data item as changed the system does the following:

- Marks the data item as changed.
- Updates the user-data and parity bits in the directory entry with the data you provide in the name element.
- Leaves the data item associated with the cast-out class to which it was assigned.

When Another User Updates the Data Item

If another user writes changed data to the data item while you hold the cast-out lock, the system ignores the data in the name element. Instead, the information provided on the other user's write request determines the directory update. The system:

- Marks the data item as changed.
- Updates the user-data field and the parity bits with the information supplied on the WRITE_DATA request.
- Assigns the data item to the cast-out class specified on the WRITE_DATA request.

The UNLOCK_CASTOUT request in this instance does not affect the directory entry for the data item and has no effect on the storage class specified for the data item on the WRITE_DATA request. The data item resources are marked as changed and are not available for storage reclaim.

Releasing Cast-Out Locks: Summary

You use an unlock cast-out request to unlock one or more locks that your connection, or optionally your process or task, holds.

The request must identify the data items whose cast-out locks are to be unlocked.

- To identify the data items, build a list of name elements in a buffer. Each name element contains a data item name, user-defined data, parity bits, and a change indicator (change-bit-overindication bit):
- To mark the data item as changed use the CUNBCHANGEOI field in the IXLYCUNB mapping macro for each name element in the list. The data item remains associated with its specified cast-out class, and the resources of the data item are not available for reclaim.
- If the data item is not changed, allow the system to leave the state of the data item as is, and the data item is disassociated with its cast-out class and its resources available for reclaim.

The request must indicate the first and last name elements in the list of name elements that the system is to process.

- To identify the first name element, use the FIRSTNAME keyword.
- To identify the last name element, use the LASTNAME keyword.

The system processes all of the name elements from FIRSTNAME through LASTNAME.

Optionally, specify the task or process identifier on the PROCESSID keyword to identify the task or process that holds the cast-out lock for the data item.

The request can complete prematurely for the following reasons:

- The coupling facility times-out.

- A name element specifies a data item that is not in the cache structure.
- A name element specifies a data item whose cast-out lock is not held by the specified connection or process or task identified by the PROCESSID.
- A name element specified a data item with parity bits that are not valid.
- A name element specified a data item that holds a cast-out lock in a write-with-cast-out state. The write-with-cast-out lock state is not compatible with the change indicator for the data item.

Each time a request completes prematurely, the system returns an index value. You can use the index value to identify the name element for the data item that might have caused the premature completion. You can also use the index value to restart the request.

The request can alter the directory entry for each data item named in a name element. If, while you hold the cast-out lock, no other user writes changed data to the data item, the system updates the directory entry with the information you supply in the name element. If another user writes changed data to the data item while you hold the cast-out lock, the system unlocks the cast-out lock but ignores your directory update information. Instead, the system updates the directory with information provided by the user who performed the update.

There are other keywords that are required and some that are optional. For a description of keywords that are applicable to all IXLCACHE requests, see:

- “Understanding Synchronous and Asynchronous Cache Operations” on page 386
- “Accessing and Managing Data Within a Cache System” on page 369
- “Requesting Return and Reason Codes” on page 395
- “Defining an Answer Area (ANSAREA)” on page 395

UNLOCK_CO_NAME: Releasing a Single Cast-Out Lock

To release a single cast-out lock held by your connection, issue an IXLCACHE REQUEST=UNLOCK_CO_NAME request. After you have (Note that although it is possible to release a single cast-out lock with the UNLOCK_CASTOUT request, it is more efficient to use the UNLOCK_CO_NAME request. After you have cast out and written the data for the data item to permanent storage, you release a cast-out lock. If you fail to update permanent storage for the data item that you have cast out, you need to release the lock, but you indicate that the data item is changed on the UNLOCK_CO_NAME request. As a result, the system does not consider the data item eligible for reclaim. The data item then needs to be cast out again so that you can successfully write the changes to permanent storage.

When you release a cast-out lock for a data item, the system updates the directory entry to indicate that the lock is released. If the data item has not been updated by another user while the lock is held, the system also disassociates the data item from the cast-out class.

As a user, you can also update the directory entry for the data item by providing data for the user-defined data field and by changing the parity bits. You can also indicate on the UNLOCK_CO_NAME request that the system mark the data as changed, which makes the resources unavailable for reclaim.

While you hold a cast-out lock for a data item, another user can write changed data to the data item. If you issue REQUEST=UNLOCK_CO_NAME after another user has changed the data, the system releases the cast-out lock, but does not update the directory entry with any of the data that you provide. Instead, the system considers the data item as changed and updates the directory entry as specified on the write request of the user that makes the change. The data item also remains associated with the cast-out class specified by the user on the WRITE_DATA request.

Guide to the Topic

“UNLOCK_CO_NAME: Releasing a Single Cast-Out Lock” on page 439 is divided into two sections .

The first section , “IXLCACHE Functions for REQUEST=UNLOCK_CO_NAME” on page 440, applies to all UNLOCK_CO_NAME requests and includes the following major topics:

- "Identifying the Cast-Out Lock to Release" on page 440
- "Initializing a Name Element" on page 440
- "Specifying a Process Identifier" on page 440
- "Receiving Answer Area Information" on page 441
- "Changing the Directory Entry for the Data Item" on page 441

The second section , "Releasing a Single Cast-Out Lock: Summary" on page 442 summarizes a procedure for unlocking a single cast-out lock.

IXLCACHE Functions for REQUEST=UNLOCK_CO_NAME

The following functions apply when you specify REQUEST=UNLOCK_CO_NAME.

Identifying the Cast-Out Lock to Release

To identify the data item whose cast-out lock is to be released, you create a name element record in the CUNBAREA. The mapping macro IXLYCUNB maps the name element. For a description of IXLYCUNB, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

The name element in the CUNBAREA contains the following information:

- **Data item name:** The name of the data item whose cast-out lock is to be released.
- **User-defined data:** Data that is to replace the current user-defined data in the directory entry for the data item.
- **Parity bits:** Parity bits that are to replace the current parity bits in the directory entry for the data item.
- **Change indicator:** An indicator to allow the system to mark the data item as changed in the cache structure after the lock is released.

Initializing a Name Element

You must initialize the data item name. If you plan to use the user-defined data field and the parity bits, you must also initialize these fields. Otherwise, you can specify zeros for the user-defined data field and parity bits. You must also indicate whether the system marks the data item as changed. Indicating the changed status of a data item depends on whether you successfully write the data item to permanent storage.

During normal cast-out processing, you write changed data for each data item to permanent storage. For a successful write operation, issue the request to release the lock and ensure that the value of the change indicator informs the system to leave the change state as is. As long as no other user has updated the data item while you held the lock, the system considers the data item as unchanged and the storage resources are eligible for reclaim. For an unsuccessful write operation to permanent storage, issue the request to release the lock and set the change indicator to mark the data item as changed. By marking the data item as changed, the system cannot reclaim data item resources for other requests so that you can preserve the changes until you are able to write the data item to permanent storage.

Specifying a Process Identifier

Optionally, you can identify your task or process as the lock holder for a cast-out lock for the named data item on the PROCESSID keyword. While you hold a cast-out lock for a data item, another user can invoke a service that returns the value of the cast-out lock in the answer area, and the user can identify, not only the connection, but also the task or process that holds the lock.

Selecting a Buffering Method

When you issue a REQUEST=UNLOCK_CASTOUT request, you must specify a buffer that contains the list of name elements. You can use a single buffer (the BUFFER keyword) or multiple buffers (the BUFLIST keyword). Either method enables you to build a maximum of 2048 name elements.

For information about whether to use a single buffer or multiple buffers and for information on selecting buffer attributes, see "Design Considerations for Choosing the Buffer Format" on page 391.

Receiving Answer Area Information

On most IXLCACHE requests, the system returns information related to the request in the answer area. You specify the answer area on the ANSLEN and ANSAREA keywords. With certain events, the information in the answer area might not be valid. See "Determining Valid Information in the Answer Area" on page 397.

When the request completes, the system returns information to the answer area. When the request is not valid, the system returns non-zero return and reason codes.

For the mapping of the answer area, see the IXLYCAA mapping macro described in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary). For a description of answer area fields and return and reason codes for the request, see *z/OS MVS Programming: Sysplex Services Reference*.

Changing the Directory Entry for the Data Item

If a user does not write changed data to a data item while you hold the cast-out lock, you can indicate to the system whether you want to mark the data item as changed or indicate that the system is not to change the status of the data item when you release the lock.

If you indicate that the system is not to change the status of the data item when you release the lock and no other user has updated the data item in the cache while the lock was held, the system is able to reclaim resources from the data item for other requests. If you indicate that the data item is changed when you release the lock or a user has updated the data item while the lock was held, the system considers the data item as changed, and the system cannot reclaim data item resources.

Indicating to the System not to Change the Status of the Data Item

If you do not want to change the status of the data item and the data item is unchanged, the system does the following:

- Updates the user-data and parity bits in the directory entry with the data you provide in the name element.
- Disassociates the data item from the cast-out class to which it was assigned.

Marking the Data Item as Changed

To request that the data item be marked as changed, do the following:

- Specify B'1' in the CUNBCHANGEOI field of the mapping macro IXLYCUNB for the name element.

If you request that the system mark the data item as changed the system does the following:

- Marks the data item as changed.
- Updates the user-data and parity bits in the directory entry with the data you provide in the name element.
- Leaves the data item associated with the cast-out class to which it was assigned.

When Another User Updates the Data Item

If another user writes changed data to the data item while you hold the cast-out lock, the system ignores the data in the name element. Instead, the information provided on the other user's write request determines the directory update. The system:

- Marks the data item as changed.
- Updates the user-data field and the parity bits with the information supplied on the WRITE_DATA request.
- Assigns the data item to the cast-out class specified on the WRITE_DATA request.

The UNLOCK_CO_NAME request in this instance does not affect the directory entry for the data item and has no effect on the storage class specified for the data item on the WRITE_DATA request. The data item resources are marked as changed and are not available for storage reclaim.

Releasing a Single Cast-Out Lock: Summary

You use an UNLOCK_CO_NAME request to unlock one lock that your connection, or optionally your process or task, holds.

The request must identify the data item whose cast-out lock is to be unlocked.

- To identify the data item, build a name element in the area specified by CUNBAREA, mapped by IXLYCUNB. Each name element contains the data item name, user-defined data, parity bits, and a change indicator (change-bit-overindication bit):
- To mark the data item as changed use the CUNBCHANGEOI field in the IXLYCUNB mapping macro. The data item remains associated with its specified cast-out class, and the resources of the data item are not available for reclaim.
- If the data item is not changed, allow the system to leave the state of the data item as is, and the data item is disassociated with its cast-out class and its resources available for reclaim.

Optionally, specify the task or process identifier on the PROCESSID keyword to identify the task or process that holds the cast-out lock for the data item.

The request can alter the directory entry for the data item. If, while you hold the cast-out lock, no other user writes changed data to the data item, the system updates the directory entry with the information you supply in the name element. If another user writes changed data to the data item while you hold the cast-out lock, the system unlocks the cast-out lock but ignores your directory update information. Instead, the system updates the directory with information provided by the user who performed the update.

There are other keywords that are required and some that are optional. For a description of keywords that are applicable to all IXLCACHE requests, see:

- “Accessing and Managing Data Within a Cache System” on page 369 for the connect token and the request identifier
- “Requesting Return and Reason Codes” on page 395
- “Defining an Answer Area (ANSAREA)” on page 395

DELETE_NAME: Deleting Data Items From a Cache Structure

To delete a data item from the cache structure and free the cache structure resources allocated to that data item, issue the IXLCACHE REQUEST=DELETE_NAME request. With one request you can delete a single data item or multiple data items whose names satisfy a specified character selection pattern. For cache structures allocated in a coupling facility of CFLEVEL=4 or lower, the system deletes the data entry and the directory entry for the data items identified on the request, and invalidates the copies of the data items that are in local cache buffers for all users (including the user who issues the request). The data items are no longer associated with their cast-out classes or storage classes, and the resources allocated to the data item are made available for reuse within the cache structure. Subsequent references to a deleted data item fail until the data item is redefined to the cache structure.

For cache structures allocated in a coupling facility of CFLEVEL=5 or higher, you have the option of specifying the type of resource deletion that is to be performed as well as whether version number comparison is required.

If your protocol relies on external serialization, you need to hold a lock to serialize access to data items. For serialization recommendations and sample scenarios that show how to establish serialization, see “Serializing and Managing Access to Shared Data” on page 377.

Timing and DELETE_NAME Requests

When you issue the DELETE_NAME request, consider the impact of timing issues on serialization. If another user creates a new data item in the cache after you have issued a DELETE_NAME request with criteria that matches the data item but before the request completes, the request might not delete the data item. If you want to ensure that when your request completes, any data item matching your criteria has been deleted from the cache structure, you must hold serialization throughout the request processing. Serialization needs to remain in effect for both the initial request, and any subsequent request restarts that might be required as a result of a timeout, and the scope of the serialization must prevent any other user from creating a new entry that matches the criteria on the DELETE_NAME request.

Guide to the Topic

"DELETE_NAME: Deleting Data Items From a Cache Structure" on page 442 is divided into two sections .

The first section , "IXLCACHE Functions for REQUEST=DELETE_NAME" on page 443, applies to all DELETE_NAME requests and includes the following major topics:

- "Identifying Data Items to Delete" on page 443
- "Specifying the Type of Deletion" on page 444
- "Restarting Requests" on page 445
- "Receiving Answer Area Information" on page 445

The second section , "Deleting Data Items: Summary" on page 445 summarizes a procedure for deleting data items from a cache structure.

IXLCACHE Functions for REQUEST=DELETE_NAME

The following functions apply when you specify IXLCACHE REQUEST=DELETE_NAME.

Identifying Data Items to Delete

To identify a single data item, specify the data item name on the NAME keyword and omit the NAMEMASK keyword. This causes the system to select only the data item whose name matches the name specified on the NAME keyword. For a general description of NAMEMASK and the character selection pattern, see "Using Filters for Names on Requests" on page 398.

Example 1

You want to select the data item named IXLG567. You can omit the NAMEMASK keyword or code it as shown:

```
          IXLCACHE …,NAME=DNAME,NAMEMASK=MASK,…
⋮
DNAME     DC   CL16'IXLG567'
MASK      DC   BL2'1111111111111111'
⋮
```

Example 2

You want to select only those data items whose name contains the characters 'RL' in the third and fourth character positions. The other characters in the name can be any character. You must provide the following values for the NAME and NAMEMASK keywords:

```
          IXLCACHE …,NAME=DNAME,NAMEMASK=MASK,…
⋮
DNAME     DC   CL16'XXRL'
MASK      DC   BL2'0011000000000000'
⋮
```

Note: The characters 'XX' in the data constant DINAME can be anything you choose because they are not used in the selection process.

Example 3

You want to select only those data items whose name begins with the character string "IXL1". The other characters in the name can be any character. You must provide the following values for the NAME and NAMEMASK keywords:

```
          IXLCACHE …,NAME=DNAME,NAMEMASK=MASK,…
⋮
DNAME     DC   CL16'IXL1'
MASK      DC   BL2'1111000000000000'
⋮
```

Specifying the Type of Deletion

Use the DELETETYPE keyword to indicate the type of delete processing to be performed. The default (DIRANDDATA) requests that all cache structure resources for the entry be released for reuse by the structure and also all applicable connections have their interest deregistered and a cross-invalidate performed against their local vector.

The other DELETETYPE keyword options all provide the ability to keep the data item's directory entry and not have the system perform the cross-invalidate against a connector's local vector. For each structure entry,

- UNCHDATA requests that all unchanged data only be released for reuse.
- CHDATA requests that all changed data be released for reuse and certain status fields be reset.
- ANYDATA requests that, whether changed or unchanged, the data is to be released for reuse and status fields are to be reset.

Using Name Classes in a Coupling Facility

At connect time, you can specify that the cache structure is to be allocated to support the logical grouping of cache entries into name classes. A coupling facility of CFLEVEL=7 or higher can assign entries to name classes based on the value of the NAMECLASSMASK that was specified when the structure was allocated. Using NAMECLASSMASK in conjunction with NAMEMASK may improve the efficiency of an IXLCACHE REQUEST=DELETE_NAME request.

For example, if your processing requires that at some point you will want to identify for deletion purposes all cache entries that adhere to a particular naming convention, the following method would accomplish that requirement:

1. Determine a naming convention that logically relates the entry names. Let's suppose that the naming convention specifies that the first four characters of the name determine the logical naming convention for these “related” entries. That is, at some point in your processing, you will want to delete all entries in the cache structure whose entry names start with a given four-character string, while leaving all other entries whose names start with a different four-character string unaffected.
2. Specify on IXLCONN a NAMECLASSMASK value of X'F000' to indicate that the first four characters are the ones in which you are interested. This allows the coupling facility (of CFLEVEL=7 or higher) to maintain separate name classes based on the first four characters of the name as the entries in the structure are referenced. Each separate name class maintained by the coupling facility contains only those entries whose names start with the same first four characters.
3. If, at some point in your processing, you want to delete a particular set of entries with the same first four characters, issue IXLCACHE REQUEST=DELETE_NAME with a NAME identifying the entries to be deleted and a NAMEMASK=X'F000' (equal to the NAMECLASSMASK value). The coupling facility can efficiently process this request because the cache entries have been logically grouped into name classes. For example, if the entries to be deleted all start with the characters ‘ABCD’, those entries identified by NAME=ABCDxxxxxxxxxxxx would have been logically grouped together and can be easily retrieved by the coupling facility for deletion.

   In contrast, again assuming a NAMECLASSMASK of X'F000', consider the following examples.

- If the entries to be deleted are identified by the NAME ‘ABxxxxxxxxxxxxxx’ and a NAMEMASK of X'C000' is specified, the coupling facility would have to scan the entire directory to locate those entries that matched the name ‘ABxxxxxxxxxxxxxx’. The coupling facility retrieval process would be significantly less efficient, depending on the size of the structure.
- If the entries to be deleted are identified by the NAME ‘ABCDEFGHxx’ and a NAMEMASK of X'FF00' is specified, because the NAMEMASK does not exactly match the NAMECLASSMASK specified on IXLCONN, the coupling facility would scan the entire directory to locate the entries with names identified by ‘ABCDEFGHxx’.
- If the structure does not support name classes (either because it is not allocated in a coupling facility of CFLEVEL=7 or higher or because NAMECLASSMASK was not specified on IXLCONN when the structure was allocated), the request will result in the coupling facility having to scan the entire directory to locate the entries to be deleted because they have not been logically grouped together.

Restarting Requests

IXLCACHE REQUEST=DELETE_NAME might complete prematurely because the request exceeds time-out criteria. When a request completes prematurely, the system might not have deleted all the data items specified on the request. Even if you expect to delete a single data item or are using name classes to optimize the performance of the REQUEST=DELETE_NAME request, you need to consider time-outs. (Time-outs will be much less likely to occur when using name classes, but still must be considered.) To delete one or more remaining data items for the request, you can restart the request. For general information about restarting a request, see Restarting a Request that Ends Prematurely.

Receiving Answer Area Information

On most IXLCACHE requests, the system returns information related to the request in the answer area. You specify the answer area on the ANSLEN and ANSAREA keywords. With certain events, the information in the answer area might not be valid. See “Determining Valid Information in the Answer Area” on page 397.

When the request completes, the system returns information to the answer area. When the request is not valid, the system returns non-zero return and reason codes.

For the mapping of the answer area, see the IXLYCAA mapping macro described in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary). For a description of answer area fields and return and reason codes for the request, see *z/OS MVS Programming: Sysplex Services Reference*.

Deleting Data Items: Summary

You delete a data item to remove it from the cache structure. Deleting a data item removes the name from the cache structure, marks the copies of the data item as not valid for all users, and frees the data item's cache structure resources for reuse.

- To identify the data item, specify the data item name on the NAME keyword. If you want to delete only the named data item, omit the NAMEMASK keyword.
- To delete all data items with names that match a specified character pattern, code both the NAME and NAMEMASK keywords. NAME must specify a data item name that contains the specified character pattern. NAMEMASK must specify a bit-string where the bits that correspond to the specified character pattern are set to B'1'. For examples that show how to use NAME and NAMEMASK together, see “Using Filters for Names on Requests” on page 398.

If the coupling facility model dependent time-out criteria is exceeded or a halt condition is found as a result of specifying HALTONCHANGED=YES, the delete request completes prematurely. For the premature completion of a request, the system returns a token that you can use to restart the request from the point at which it prematurely ended. The system returns the token in the CAARESTOKEN or CAAEXTRESTOKEN field of the answer area. To restart a request that completes prematurely, code the IXLCACHE REQUEST=DELETE_NAME request as you previously coded it with the exception of the

RESTOKEN or EXTRESTOKEN keyword. The RESTOKEN or EXTRESTOKEN keyword must specify the token that the system returned when the delete request ended prematurely.

There are other keywords that are required and some that are optional. For a description of keywords that are applicable to all IXLCACHE requests, see:

- “Accessing and Managing Data Within a Cache System” on page 369
- “Requesting Return and Reason Codes” on page 395
- “Defining an Answer Area (ANSAREA)” on page 395
- “Understanding Synchronous and Asynchronous Cache Operations” on page 386

DELETE_NAMELIST: Deleting a List of Data Items

To delete one or more data items from a cache structure, issue an IXLCACHE REQUEST=DELETE_NAMELIST request. The DELETE_NAMELIST request allows you to delete selective resources for a cache structure allocated in a coupling facility of CFLEVEL=5 or higher. The DELETE_NAMELIST also allows you to do version number comparisons and provides an option to control whether processing should continue after an error or miscomparison has occurred.

Guide to the Topic

“DELETE_NAMELIST: Deleting a List of Data Items” on page 446 is divided into two sections .

The first section , “IXLCACHE Functions for REQUEST=DELETE_NAMELIST” on page 446, applies to all DELETE_NAMELIST requests and includes the following major topics:

- “Identifying Data Items to Delete” on page 446
- “Identifying Data Items to Delete” on page 446
- “Specifying the Type of Deletion” on page 447
- “Requesting Version Comparison” on page 447
- “Handling Error Processing” on page 447
- “Restarting a DELETE_NAMELIST Request that Ends Prematurely” on page 447
- “Receiving Answer Area Information” on page 448

The second section , “Deleting a List of Data Items: Summary” on page 448 summarizes a procedure for deleting a list of data items from a cache structure.

IXLCACHE Functions for REQUEST=DELETE_NAMELIST

The following functions apply when you specify IXLCACHE REQUEST=DELETE_NAMELIST.

Identifying Data Items to Delete

To identify a data item for deletion processing, you build a list of name blocks in a buffer. The mapping macro IXLYDNNB maps each name block in the list. For a description of IXLYDNNB, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Each name block in the list identifies one data item and contains the following information:

- **Structure entry name:** The name of the structure entry for which delete processing is to be performed.
- **Comparative version number:** An optional version number to be used when version number comparison is requested.

Selecting a Buffering Method

When you issue a REQUEST=DELETE_NAMELIST request, you must specify a buffer that contains the list of name elements. You can use a single buffer (the BUFFER keyword) or multiple buffers (the BUFLIST keyword). Either method enables you to build a maximum of 2048 name elements.

For information about whether to use a single buffer or multiple buffers and for information on selecting buffer attributes, see “Design Considerations for Choosing the Buffer Format” on page 391.

Use the STARTINDEX and ENDINDEX keywords as index values to identify the name blocks in the buffer area to be processed. The name blocks are numbered starting with 1. The name blocks are processed sequentially beginning with STARTINDEX and continuing through ENDINDEX.

Specifying the Type of Deletion

Use the DELETETYPE keyword to indicate the type of delete processing to be performed. The default (DIRANDDATA) requests that all cache structure resources for the entry be released for reuse by the structure and also all applicable connections have their interest deregistered and a cross-invalidate performed against their local vector.

The other DELETETYPE keyword options all provide the ability to keep the data item's directory entry and not have the system perform the cross-invalidate against connectors' local vectors. For each structure entry,

- UNCHDATA requests that all unchanged data only be released for reuse.
- CHDATA requests that all changed data be released for reuse and certain status associated with the data's change state be reset.
- ANYDATA requests that, whether changed or unchanged, the data is to be released for reuse and status fields associated with the data's change state are to be reset.

Requesting Version Comparison

Use the VERSCOMPTYPE keyword if you require structure entry version comparison to be performed. The system compares the version number in the structure entry with the version number in the IXLYDNNB name block being processed. Valid conditions that you can specify are no comparison, equal comparison, or less than or equal comparison. If the comparison does not meet the condition specified, you can request that processing either continue with the next name block or be halted.

Handling Error Processing

If, while processing the IXLYDNNB name blocks, either an entry is not found or a version number miscompare occurs, you can specify whether processing is to continue with the next name block (ERRORACTION=CONTINUE) or stop (ERRORACTION=TERMINATE). If processing is halted, the index value of the entry that caused the error is returned in the CAADNLINDEX field of IXLYCAA. To restart processing after it is halted, increment the index value returned in CAADNLINDEX by 1, reinitialize STARTINDEX with the new index value, and resubmit the DELETE_NAMELIST request.

Restarting a DELETE_NAMELIST Request that Ends Prematurely

An IXLCACHE REQUEST=DELETE_NAMELIST request might complete prematurely if the request exceeds the time-out criteria for the coupling facility (time-out criteria is model-dependent) or a halt condition is found as a result of specifying HALTONCHANGED=YES. Each time a request completes prematurely, the system returns an index value into the list of name elements in the CAADNLINDEX field of the answer area. Use this index value in CAADNLINDEX to restart the request so it can process the remaining elements in the list of name elements. Reinitialize the STARTINDEX index value to the value returned in CAADNLINDEX. To restart a request, after reinitializing STARTINDEX, reissue IXLCACHE REQUEST=DELETE_NAMELIST. To ensure that you do not alter the intent of the request that completed prematurely, the restarted request should specify the same keywords and values (with the exception of the index value specified on STARTINDEX) as the request that completed prematurely. For general information about restarting requests, see Restarting a Request that Ends Prematurely.

For DELETETYPE=DIRANDDATA, a check of the changed status of data and cast-out lock state for a structure entry may be requested prior to processing the structure entry by specifying the HALTONCHANGED=YES keyword. If a structure entry is found to either contain changed data or for which the cast-out lock is currently held, processing of the DELETE_NAMELIST request is halted and the system returns the index value (CAADNLINDEX) into the list of name elements of the entry name meeting the halt criteria in the answer area. When the DELETE_NAMELIST request is halted as requested, the application is expected to take some action to change the state of the indicated structure entry data to unchanged (that is, the cast-out lock is not held and the status of the data is unchanged) before resuming the request. One such action could be to read the entry data for castout, write the data to permanent storage, then reset the cast-out lock to the not-held state. After taking such action, the DELETE_NAMELIST request may be started at the element in the list of name elements that originally caused the request to halt by setting input parameter STARTINDEX to CAADNLINDEX.

Receiving Answer Area Information

On most IXLCACHE requests, the system returns information related to the request in the answer area. You specify the answer area on the ANSLEN and ANSAREA keywords. With certain events, the information in the answer area might not be valid. See "Determining Valid Information in the Answer Area" on page 397.

When the request completes, the system returns information to the answer area. When the request is not valid, the system returns non-zero return and reason codes.

For the mapping of the answer area, see the IXLYCAA mapping macro described in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary). For a description of answer area fields and return and reason codes for the request, see *z/OS MVS Programming: Sysplex Services Reference*.

Deleting a List of Data Items: Summary

For cache structures allocated in a coupling facility with CFLEVEL=5 or higher, you can specify a list of data items for which structure resources are to be deleted. The resources are returned to the structure for reuse. You can specify the type of deletion to be performed and whether version number comparison is required. The DELETE_NAMELIST provides you with the option of deleting data resources without deleting the corresponding directory entry or having the system cross-invalidate connectors' local vectors.

If an entry is not found or a version number miscompare occurs, the DELETE_NAMELIST request provides an option for either continuing processing with the next entry or halting processing. If halted, the system returns an index value in the CAADNLINDEX field of the answer area. To continue processing the request, reinitialize the STARTINDEX keyword with the incremented index value and resubmit the DELETE_NAMELIST request.

If the coupling facility time-out criteria are exceeded, the DELETE_NAMELIST request completes prematurely. For the time-out of a request, the system returns an index value that you can use to restart the request from the point at which it timed-out. The system returns the index value in the CAADNLINDEX field of the answer area. To restart a request that completes prematurely, code the IXLCACHE REQUEST=DELETE_NAMELIST request as you previously coded it with the exception of the STARTINDEX keyword. The STARTINDEX keyword must specify the index value that the system returned when the delete request ended prematurely.

There are other keywords that are required and some that are optional. For a description of keywords that are applicable to all IXLCACHE requests, see:

- "Accessing and Managing Data Within a Cache System" on page 369
- "Requesting Return and Reason Codes" on page 395
- "Defining an Answer Area (ANSAREA)" on page 395
- "Understanding Synchronous and Asynchronous Cache Operations" on page 386

CROSS_INVAL: Invalidating Other Users' Copies of Data Items

To invalidate copies of one or more data items that other users have in their local cache buffers, use IXLCACHE REQUEST=CROSS_INVAL. The system invalidates any copies of the specified data items that are in the local cache buffers of *other* users and deregisters interest in the data item for those users. (Your own copy of the data item is not invalidated.) Typically, you use the cross-invalidate function in a directory-only cache environment when you update data items on permanent storage. For a description of cross-invalidation, see Figure 34 on page 376. The principles of invalidation are the same for a directory-only cache.

The request does not cause the specified data items to be deleted from the cache structure. Also, the system does not consider resources for a data item that is specified on the CROSS_INVAL request as eligible for reclaim.

If your protocol relies on external serialization, you need to hold a lock to serialize access to any data items. For serialization recommendations and sample scenarios that show how to establish serialization, see “Serializing and Managing Access to Shared Data” on page 377.

Extended Function

With a coupling facility of CFLEVEL=12 or higher, it is possible to specify a list of up to 4096 data items to be cross-invalidated. See “CROSS_INVALLIST: Invalidating a List of Data Items” on page 450.

Timing and CROSS_INVAL Requests

When you issue the CROSS_INVAL request, consider the impact of timing issues on serialization. If another user creates a new data item in the cache after you have issued a CROSS_INVAL request with criteria that matches the data item but before the request completes, the request might not invalidate copies of the new data item. If you want to ensure that when your request completes, all other users' copies of the data items matching your criteria have been invalidated, you must hold serialization throughout the request processing. Serialization needs to remain in effect for both the initial request, and any subsequent request restarts that might be required as a result of a timeout, and the scope of the serialization must prevent any other user from creating a new entry that matches the criteria on the CROSS_INVAL request.

Guide to the Topic

“CROSS_INVAL: Invalidating Other Users' Copies of Data Items” on page 449 is divided into two sections .

The first section , “IXLCACHE Functions for REQUEST=CROSS_INVAL” on page 449, applies to all CROSS_INVAL requests, and includes the following major topics:

- “Identifying Data Items to Cross-Invalidate” on page 449
- “Restarting a Request that Ends Prematurely” on page 450
- “Receiving Answer Area Information” on page 450

The second section , “Cross-Invalidating a Data Item: Summary” on page 450 summarizes a procedure for invalidating data items.

IXLCACHE Functions for REQUEST=CROSS_INVAL

The following functions apply when you specify REQUEST=CROSS_INVAL.

Identifying Data Items to Cross-Invalidate

To invalidate the local copies of a cached data item, specify the data item name on the NAME keyword and omit the NAMEMASK keyword. The system selects only the data item specified on NAME.

Optionally, you can code both NAME and NAMEMASK to provide a character selection pattern. The NAMEMASK keyword defines a selection bit-mask. The selection bit-mask together with the name

specified on the NAME keyword defines a character selection pattern that the system uses to select data item names. The technique enables you to select multiple data item names.

Restarting a Request that Ends Prematurely

IXLCACHE REQUEST=CROSS_INVAL might complete prematurely because the request exceed time-out criteria. When a request completes prematurely, the system might not have invalidated all the data items specified on the request. Even if you expect to invalidate copies of a single data item, you need to consider time outs. To invalidate one or more remaining data items for the request, you can restart the request.

For general information about restarting request, see Restarting a Request that Ends Prematurely.

Receiving Answer Area Information

On most IXLCACHE requests, the system returns information related to the request in the answer area. You specify the answer area on the ANSLEN and ANSAREA keywords. With certain events, the information in the answer area might not be valid. See “Determining Valid Information in the Answer Area” on page 397.

When the request completes, the system returns information to the answer area. When the request is not valid, the system returns non-zero return and reason codes.

For the mapping of the answer area, see the IXLYCAA mapping macro described in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary). For a description of answer area fields and return and reason codes for the request, see *z/OS MVS Programming: Sysplex Services Reference*.

Cross-Invalidating a Data Item: Summary

Use the cross-invalidate function to invalidate the copies of one or more data items for other users.

- To identify the data item, specify the data item name on the NAME keyword. If you want to invalidate only the named data item, omit the NAMEMASK keyword.
- To invalidate all data items whose name matches a specified character pattern, code both the NAME and NAMEMASK keywords. NAME must specify a data item name that contains the specified character pattern. NAMEMASK must specify a bit-string with the bits that correspond to the specified character pattern set to B'1'. For examples of how to use NAME and NAMEMASK, see “Using Filters for Names on Requests” on page 398.

If the coupling facility time-out criteria are exceeded, the cross-invalidate request completes prematurely. For the time-out of a request, the system returns a token that you can use to restart the request from the point at which it timed-out. The system returns the token in the CAARESTOKEN field of the answer area. To restart a request that completes prematurely, code the IXLCACHE REQUEST=CROSS_INVAL request as you previously coded it with the exception of the RESTOKEN keyword. The RESTOKEN keyword must specify the token that the system returned when the cross-invalidate request ended prematurely.

There are other keywords that are required and some that are optional. For a description of keywords that are applicable to all IXLCACHE requests, see:

- “Understanding Synchronous and Asynchronous Cache Operations” on page 386
- “Accessing and Managing Data Within a Cache System” on page 369
- “Requesting Return and Reason Codes” on page 395
- “Defining an Answer Area (ANSAREA)” on page 395

CROSS_INVALLIST: Invalidating a List of Data Items

You may want to perform a cross-invalidate operation on a list of entries with one operation. The CROSS_INVALLIST request allows you to cross-invalidate up to 4096 entries at one time. The result

of the cross-invalidate operation is that with the exception of the connection specified by CONTOKEN, all connections with registered interest in the specified entries will have interest deregistered and a cross-invalidate performed against their local caches.

The CROSS_INVALLIST request type is valid only for a structure allocated in a coupling facility with CFLEVEL=12 or higher.

Guide to the Topic

"CROSS_INVALLIST: Invalidating a List of Data Items" on page 450 is divided into two sections .

The first section , "IXLCACHE Functions for REQUEST=CROSS_INVALLIST" on page 451, applies to all CROSS_INVALLIST requests, and includes the following major topics:

- "Identifying the list of data items to be cross-invalidated" on page 451
- "Selecting a Buffering Method" on page 451
- "Handling Error Processing" on page 451
- "Restarting a CROSS_INVALLIST Request that ends prematurely" on page 451
- "Receiving answer area information" on page 452

The second section , "Cross-invalidating a list of data items: Summary" on page 452 summarizes a procedure for invalidating data items.

IXLCACHE Functions for REQUEST=CROSS_INVALLIST

The following functions apply when you specify REQUEST=CROSS_INVALLIST.

Identifying the list of data items to be cross-invalidated

The names of the data items for which a cross-invalidate operation is requested are contained in the storage area specified by either BUFFER or BUFLIST. Each entry in the list is a 16-byte field containing the structure entry name. Up to 4096 entries can be specified.

Selecting a Buffering Method

When you issue a REQUEST=CROSS_INVALLIST request, you must specify a buffer that contains the list of entry names to be cross-invalidated. You can use a single buffer (the BUFFER keyword) or multiple buffers (the BUFLIST keyword). Either method allows you to build a list of up to 4096 16-byte structure entry names.

For information about whether to use a single buffer or multiple buffers and for information on selecting buffer attributes, see "Design Considerations for Choosing the Buffer Format" on page 391.

Use the STARTINDEX and ENDINDEX keywords as index values to identify the entry names in the buffer area to be processed. The entry names are numbered starting with 1. The entry names are processed sequentially beginning with STARTINDEX and continuing through ENDINDEX.

Handling Error Processing

If, while processing the names on the list to be cross-invalidated, a name does not identify an existing structure entry, you can specify whether processing is to continue with the next name (ERRORACTION=CONTINUE) or stop (ERRORACTION=STOP). If processing is halted, the index value of the entry that caused the error is returned in the CAACILINDEX field of IXLYCAA. To restart processing after it is halted, increment the index value returned in CAACILINDEX by 1, reinitialize STARTINDEX with the new index value, and resubmit the CROSS_INVALLIST request.

Restarting a CROSS_INVALLIST Request that ends prematurely

An IXLCACHE REQUEST=CROSS_INVALLIST request might complete prematurely if the request exceeds the time-out criteria for the coupling facility. (Time-out criteria is model-dependent.) Each time a request

completes prematurely, the system returns an index value into the list of names in the CAACILINDEX field of the answer area. Use this index value to restart the request so it can process the remaining entries in the list of names. Reinitialize the STARTINDEX index value to the value returned in CAACILINDEX. To restart a request, after reinitializing STARTINDEX, reissue IXLCACHE REQUEST=CROSS_INVALLIST. To ensure that you do not alter the intent of the request that completed prematurely, the restarted request should specify the same keywords and values (with the exception of the index value specified on STARTINDEX) as the request that completed prematurely. For general information about restarting requests, Restarting a Request that Ends Prematurely.

Receiving answer area information

On most IXLCACHE requests, the system returns information related to the request in the answer area. You specify the answer area on the ANSLEN and ANSAREA keywords. With certain events, the information in the answer area might not be valid. See "Determining Valid Information in the Answer Area" on page 397.

When the request completes, the system returns information to the answer area. When the request is not valid, the system returns non-zero return and reason codes.

For the mapping of the answer area, see the IXLYCAA mapping macro described in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary). For a description of answer area fields and return and reason codes for the request, see *z/OS MVS Programming: Sysplex Services Reference*.

Cross-invalidating a list of data items: Summary

For cache structures allocated in a coupling facility of CFLEVEL=12 or higher, use the CROSS_INVALLIST function to invalidate copies of a list of data items.

- To identify the data items, list the names in the storage area specified by either BUFFER or BUFLIST. The names in the list are processed sequentially using an index value that you provide.
- A name that fails to identify an existing structure entry causes processing to either continue (and skip the failing name) or stop (allowing you to restart processing with a new index value) based on the specification of ERRORACTION.

If the coupling facility time-out criteria are exceeded, the CROSS_INVALLIST completes prematurely. For the time-out of a request, the system returns the index of the first unprocessed name in the list. Using this index, you can restart the CROSS_INVALLIST request.

For a discussion of keywords applicable to all IXLCACHE requests, see:

- "Understanding Synchronous and Asynchronous Cache Operations" on page 386
- "Accessing and Managing Data Within a Cache System" on page 369
- "Requesting Return and Reason Codes" on page 395
- "Defining an Answer Area (ANSAREA)" on page 395

SET_RECLVCTR: Overriding or Restoring the Default Reclaim Algorithm

As part of the process of managing cache structure resources, you can provide a reclaim vector that overrides the default resource reclaim algorithm. The reclaim vector applies to the storage class that you specify. You can override the default algorithm by providing a reclaim vector for some or all of the storage classes, or use the default algorithm for all storage classes.

When you write a new data item to the cache structure, or read a data item that is undefined in the cache, the system must allocate cache structure resources for that data item. By default, when you have not defined a reclaim vector and unused storage resources are not available, the system attempts to reclaim the least recently used resources that belong to data items in the storage class specified on the request. If those resources are unavailable from the storage class specified on the request, and you have not

provided a reclaim vector, the system fails the request. However, if you have provided a reclaim vector for the storage class, the system uses the vector specifications to try to obtain resources from other storage classes to satisfy the request.

Defining the Reclaim Vector

The reclaim vector defines the storage classes from which the system can reclaim resources to satisfy WRITE_DATA or READ_DATA requests with the specified storage class. The reclaim vector also defines how many times the system can reclaim from each storage class (repeat factor).

Figure 37 on page 453 shows three reclaim vectors, one for storage class 1, one for storage class 2, and one for storage class 3. In the example of the reclaim vector for storage class 1 requests, the first 3 reclaims to satisfy a request for a data item in storage class 1 come from data items in storage class 1.

The system maintains a counter so that each reclaim from a storage class causes the system to subtract 1 from the reclaim value until the value equals 0. Then the system attempts reclaims from the next storage class based on the reclaim value of that vector entry. The next 2 reclaims for storage class 1 requests come from storage class 2, followed by 5 reclaims from storage class 3.

When the system processes the last reclaim from storage class n as specified by the reclaim vector entry, it subtracts 1 from a counter based on the repeat factor. Based on the repeat factor specified, the system refreshes the reclaim values specified for each storage class entry in the vector and starts the reclaim process again from the beginning of the vector until the repeat counter equals 0. When the counter equals 0, the system deactivates the vector and uses the system default to satisfy requests for that storage class.

The same reclaim processing applies to the vectors specified for storage classes 2 and 3.

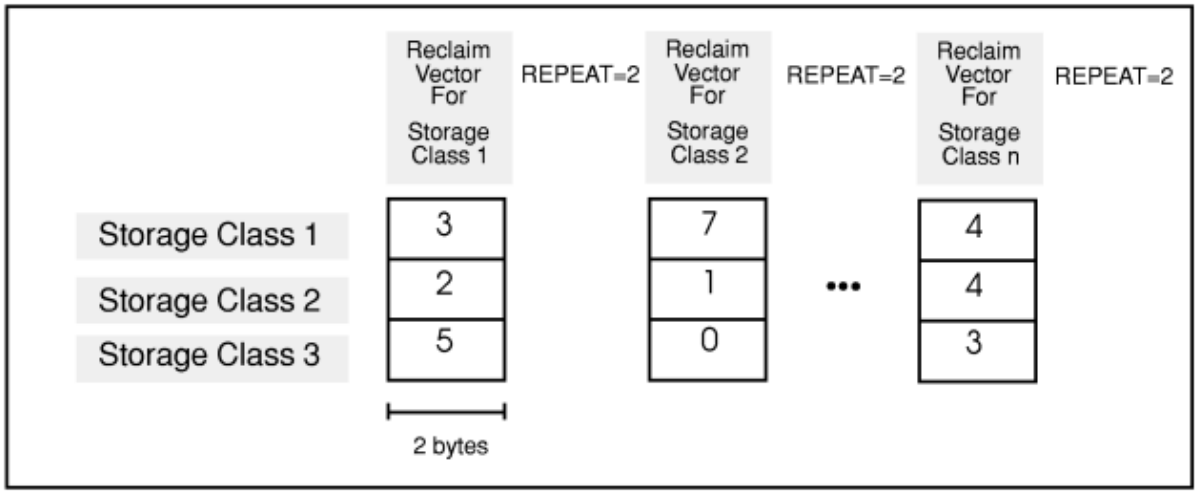


Figure 37. Three Reclaim Vectors

To provide a reclaim vector or to restore the default reclaim algorithm, issue an IXLCACHE REQUEST=SET_RECLVCTR request. When you provide a vector, you must also indicate how many times the system is to use the vector before resuming use of the default algorithm. The number of times that the system repeats the process indicated by the reclaim vector is called the repeat factor (specified on the REPEAT keyword). In Figure 37 on page 453, a repeat factor of 2 (REPEAT=2) for the storage class 1 reclaim vector indicates that the system process the vector twice— 3 reclaims from storage class 1, 2 reclaims from storage class 2, 5 reclaims from storage class 5, and repeat the sequence a second time before it uses the default reclaim algorithm.

Guide to the Topic

"SET_RECLVCTR: Overriding or Restoring the Default Reclaim Algorithm" on page 452 is divided into two sections .

The first section , “IXLCACHE Functions for REQUEST=SET_RECLVCTR” on page 454, applies to all SET_RECLVCTR requests, and includes the following major topics:

- “Specifying the Reclaim Vector” on page 454
- “Specifying the Storage Class” on page 454
- “Activating a Reclaim Vector” on page 456
- “Deactivating a Reclaim Vector” on page 456
- “Receiving Answer Area Information” on page 457

The second section , “Overriding or Restoring the Default Reclaim Algorithm: Summary” on page 457, summarizes how to use IXLCACHE REQUEST=SET_RECLVCTR.

IXLCACHE Functions for REQUEST=SET_RECLVCTR

The following functions apply when you specify REQUEST=SET_RECLVCTR.

Specifying the Storage Class

Each SET_RECLVCTR request must specify the storage class for which a reclaim vector is to be activated or deactivated. To specify the storage class, code the STGCLASS keyword.

Note: The first user to connect to the structure uses the IXLCONN macro to define the total number of storage classes available to the cache structure.

Specifying the Reclaim Vector

On each request to activate a reclaim vector, the request must include the RECLVCTR keyword to specify the reclaim vector. The reclaim vector consists of a contiguous series of two-byte elements. You must define as many elements as there are assigned storage classes. Each element corresponds to one storage class: the first element, at offset 0, corresponds to storage class 1, the second element to storage class 2, the third to storage class 3, and so forth.

Before you issue the SET_RECLVCTR request, you must initialize each element of the reclaim vector to a value that indicates the number of times the system can reclaim resources from the corresponding storage class.

Example Scenarios

The following scenarios illustrate how a reclaim vector algorithm works. Each scenario describes a user action, the cache structure environment at the time the user takes the action, the system response to the user action, and effects on the vector reclaims after the user action.

The user uses three storage classes, 1, 2, and 3. Before the user performs the first action, the user defines a reclaim algorithm for storage class 1 as follows:

```
⋮
          IXLCACHE REQUEST=SET_RECLVCTR,RECLVCTR=VECTCLS1,STGCLASS=SCLS,REPEAT=REPT,
⋮
 SCLS     DC  X'01' STORAGE CLASS
          DS  0H
 REPT     DC  H'2'  REPEAT FACTOR
 VECTCLS1 DC  H'2'  RECLAIMS TO BE MADE FROM STORAGE CLASS 1
          DC  H'0'  RECLAIMS TO BE MADE FROM STORAGE CLASS 2
          DC  H'1'  RECLAIMS TO BE MADE FROM STORAGE CLASS 3
⋮
```

The request specifies that the system activate a reclaim vector to override the system default for data items in storage class 1. The reclaim vector indicates that the system can reclaim resources to satisfy the request from storage class 1 two times, cannot reclaim resources from storage class 2, and can reclaim resources from storage class 3 one time. The system can repeat this process two times before the reclaim vector for storage class 1 is deactivated, at which time the system begins to use the default reclaim algorithm.

Scenario 1 - First Request

The user issues IXLCACHE REQUEST=WRITE_DATA to write a new and changed data item and assigns it to storage class 1.

Environment
Environment at the time of the user action:

- No free storage available in cache structure.
- There is enough reclaimable storage in each of the three storage classes to satisfy the request.

System Response
The system reclaims storage from storage class 1 and allocates it to the new data item. It subtracts 1 from the reclaim counter for storage class 1 in the vector.

Vector Counts after the Request
The system can perform subsequent storage reclaims for data items assigned to storage class 1 as follows:

- From storage class 1: 1 reclaim (changed after this request)
- From storage class 2: 0 reclaims
- From storage class 3: 1 reclaim

Scenario 2 - Second Request

The user issues IXLCACHE REQUEST=WRITE_DATA to write a new and changed data item and assigns it to storage class 1.

Environment
Environment at the time of the user action:

- No free storage available in cache structure.
- No reclaimable storage available in storage class 1.
- There is enough reclaimable storage in storage classes 2 and 3 to satisfy the request.

System Response
The system fails the request because there is no free storage and no reclaimable storage in storage class 1. It does not subtract 1 from the reclaim counter for storage class 1.

Vector Counts after the Request
On the next request, the system can perform reclaims for data items assigned to storage class 1 as follows:

- From storage class 1: 1 reclaim. (unchanged after this request)
- From storage class 2: 0 reclaims
- From storage class 3: 1 reclaim

Scenario 3 - Third Request

The user issues IXLCACHE REQUEST=WRITE_DATA to write a new and changed data item and assigns it to storage class 1.

Environment
Environment at the time of the user action:

- No free storage available in cache structure.
- There is enough reclaimable storage in each of the three storage classes to satisfy the request.

System Response
The system can reclaim storage from storage class 1 and allocates it to the new data item. It subtracts 1 from the reclaim counter for storage class 1 in the vector.

Vector Counts after the Request
On the next request, the system can perform storage reclaims for data items assigned to storage class 1 as follows:

- From storage class 1: 0 reclaims (Changed after this request)
- From storage class 2: 0 reclaims
- From storage class 3: 1 reclaims

Scenario 4 - Fourth Request

The user issues IXLCACHE REQUEST=WRITE_DATA to write a new and changed data item and assigns it to storage class 1.

Environment
Environment at the time of the user action:

- No free storage available in cache structure.
- There is enough reclaimable storage in each of the three storage classes to satisfy the request.

System Response
The system cannot reclaim storage from storage classes 1 or 2 because the vector counter indicates that the entries are 0. The system reclaims storage from storage class 3 and allocates it to the new data item. It subtracts 1 from the reclaim counter for storage class 3 in the vector.

Vector Counts after the Request
On the next request, the system can perform storage reclaims for data items assigned to storage class 1 as follows:

- From storage class 1: 0 reclaims
- From storage class 2: 0 reclaims
- From storage class 3: 0 reclaims (Changed after this request)

The system has now made one iteration through the reclaim vector and subtracts 1 from the repeat factor of 2 specified on IXLCACHE. The system resets the vector to the original values for each storage class as follows:

- From storage class 1: 2 reclaims
- From storage class 2: 0 reclaims
- From storage class 3: 1 reclaims

It can make another iteration through the vector and repeat the reclaim process based on the values. When it completes a second time, it subtracts 1 from the current repeat counter value (1) for a value of zero. When the counter equals 0, the system deactivates the vector and uses the default reclaim algorithm for storage class 1.

Activating a Reclaim Vector

To activate and begin using a reclaim vector, code the REPEAT keyword specifying a non-zero value. The value determines the number of times the system uses the vector before it begins to uses the default algorithm for the specified storage class.

The vector that is activated must be specified on the RECLVCTR keyword. The storage class to which the vector applies must be specified on the STGCLASS keyword.

Deactivating a Reclaim Vector

The system automatically deactivates a reclaim vector and resumes use of the default algorithm after using the vector the number of times specified on the REPEAT keyword. To deactivate a vector and resume use of the default algorithm sooner, issue IXLCACHE REQUEST=SET_RECLVCTR and specify a value of 0 for the REPEAT keyword. The storage class whose vector is deactivated must be specified on the STGCLASS keyword. The RECLVCTR keyword can be omitted.

Effect of Structure Alter on Reclaim Vectors

The IXLALTER function provides for the expansion or contraction of the size of a structure and/or for the reapportionment of the entry-to-element ratio of the structure. When the system receives an IXLALTER request for a cache structure, all active reclaim vectors associated with all storage classes for the structure are deactivated. The system resumes using the default reclaim algorithm for all storage classes for the structure.

While the alter process continues, the system rejects any attempt to activate a reclaim vector with non-zero return and reason codes.

At the completion of structure alter processing, you can again activate one or more reclaim vectors. Ensure that when doing so, you take into consideration any changes that were made to the structure's entry and element counts during the alter process. Also, be aware that any reclaim vectors that were deactivated when the structure alter process was initiated are not automatically reinstated at the completion of alter processing.

Receiving Answer Area Information

On most IXLCACHE requests, the system returns information related to the request in the answer area. You specify the answer area on the ANSLEN and ANSAREA keywords. With certain events, the information in the answer area might not be valid. See “Determining Valid Information in the Answer Area” on page 397.

When the request completes, the system returns information to the answer area. When the request is not valid, the system returns non-zero return and reason codes.

For the mapping of the answer area, see the IXLYCAA mapping macro described in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary). For a description of answer area fields and return and reason codes for the request, see *z/OS MVS Programming: Sysplex Services Reference*.

Overriding or Restoring the Default Reclaim Algorithm: Summary

You use the SET_RECLVCTR request to define and activate or deactivate a reclaim vector that you provide for a specified storage class. When you deactivate the reclaim vector, the system resumes using the default reclaim algorithm.

Each request must specify the storage class to which the request applies. To specify the storage class, use the STGCLASS keyword.

- To define and activate a reclaim vector, the request must include the RECLVCTR, REPEAT, and STGCLASS keywords. RECLVCTR defines the vector for the specified storage class. REPEAT, which must specify a non-zero value, defines the number of iterations the system can make through the vector before automatically resuming use of the default algorithm.
- To deactivate a vector before the system automatically resumes using the default algorithm, the request must include the REPEAT, and STGCLASS keywords. REPEAT must specify a value of 0. STGCLASS specifies the storage class whose vector you are deactivating. You can omit the RECLVCTR keyword.

There are other keywords that are required and some that are optional. For a description of keywords that are applicable to all IXLCACHE requests, see:

- “Understanding Synchronous and Asynchronous Cache Operations” on page 386
- “Accessing and Managing Data Within a Cache System” on page 369
- “Requesting Return and Reason Codes” on page 395
- “Defining an Answer Area (ANSAREA)” on page 395

PROCESS_REFLIST: Marking Data Items as Referenced

As part of managing cache structure resources, you can mark specified data items as recently referenced so that the system moves the data entry to the “recently referenced” end of the storage class queue.

When you issue PROCESS_REFLIST for a data item, the system can consider the resources allocated to the data item for reclaim depending on the position of the data entry for the data item on the storage class queue. The system considers data entries that are less recently referenced as more likely candidates for reclaim than data entries that are marked as more recently referenced. (Of course, any data item that is marked as changed is NOT considered for reclaim.) For more information on how the referenced/unreferenced state of a data item affects reclaim, see "Managing Storage Reclaim for Specific Data Items" on page 383.

You can use PROCESS_REFLIST as follows. As you reference data items in your local buffer over time, you can include them in a list of data items that you want to mark as recently referenced. Then, you can periodically issue PROCESS_REFLIST to allow the system to mark the data items in the list as recently referenced and move them to the end of the recently referenced storage queue so that the cache resources for these data items are less likely to be reclaimed.

To mark one or more data items as recently referenced, issue an IXLCACHE REQUEST=PROCESS_REFLIST request.

Guide to the Topic

"PROCESS_REFLIST: Marking Data Items as Referenced" on page 457 is divided into two sections .

The first section , "IXLCACHE Functions for REQUEST=PROCESS_REFLIST" on page 458, applies to all PROCESS_REFLIST requests, and includes the following major topics:

- "Identifying Data Items to Mark as Referenced" on page 458
- "Selecting the Buffering Method" on page 458
- "Specifying the Storage Class" on page 459
- "Receiving Answer Area Information" on page 459

The second section , "Marking a Data Item as Referenced: Summary" on page 459 summarizes a procedure for marking data items as referenced.

IXLCACHE Functions for REQUEST=PROCESS_REFLIST

The following functions apply when you specify REQUEST=PROCESS_REFLIST.

Identifying Data Items to Mark as Referenced

To identify the data items that you want to mark as referenced, build a list of data item names in a buffer. Each name must be 16 bytes long. The names must occupy buffer storage with the first name beginning at buffer offset 0. Your connection must have a registered interest in each name and each name must belong to the storage class that you specify on the STGCLASS keyword.

The system processes only the data items in the list that follow these guidelines. If you include a data item that does not follow these guidelines, the system ignores the data item in the list, but does not indicate which of the data items are ignored when the request completes.

The request must also include the NUMNAMES keyword that specifies the number of names contained in the list of names that you build.

Selecting the Buffering Method

When you issue a REQUEST=PROCESS_REFLIST request, you must identify the buffer that contains the list of data item names. You can use a single buffer (the BUFFER keyword) or multiple buffers (the BUFLIST keyword). Either method enables you to build a list that contains a maximum of 4096 names.

For information about whether to use a single buffer or multiple buffers and for information on selecting buffer attributes, see "Design Considerations for Choosing the Buffer Format" on page 391.

Specifying the Storage Class

The request must specify the storage class to which the data items are assigned. To specify the storage class, code the STGCLASS keyword.

Receiving Answer Area Information

On most IXLCACHE requests, the system returns information related to the request in the answer area. You specify the answer area on the ANSLEN and ANSAREA keywords. With certain events, the information in the answer area might not be valid. See "Determining Valid Information in the Answer Area" on page 397.

When the request completes, the system returns information to the answer area. When the request is not valid, the system returns non-zero return and reason codes.

For the mapping of the answer area, see the IXLYCAA mapping macro described in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary). For a description of answer area fields and return and reason codes for the request, see *z/OS MVS Programming: Sysplex Services Reference*.

Marking a Data Item as Referenced: Summary

You use the PROCESS_REFLIST request to mark one or more data items as recently referenced. You build a list of the data item names that are to be marked. The list must be in a buffer that you identify on the BUFFER or BUFLIST keywords. Your connection must have registered interest in all of the data items in the list, and each data item must belong to the same storage class specified on the STGCLASS keyword. The system processes data items that meet these criteria but does not indicate which entries are in error when the request completes.

There are other keywords that are required and some that are optional. For a description of keywords that are applicable to all IXLCACHE requests, see:

- "Understanding Synchronous and Asynchronous Cache Operations" on page 386
- "Accessing and Managing Data Within a Cache System" on page 369
- "Requesting Return and Reason Codes" on page 395
- "Defining an Answer Area (ANSAREA)" on page 395

RESET_REFBIT: Marking Data Items as Unreferenced

As part of managing cache structure resources, you can mark specified data items as unreferenced. When you issue RESET_REFBIT, the system returns a count of the entries in the list that it has processed and the number of those processed entries with the reference bit set on. For entries in the list with the reference bit set on, the system resets the reference bit so that the data item appears as unreferenced. (The system does not change the order of the entries on the storage class queue as a result of the RESET_REFBIT request.) For more information on how the referenced/unreferenced state of a data item affects reclaim, see "Managing Storage Reclaim for Specific Data Items" on page 383.

To mark a data item as unreferenced, issue an IXLCACHE REQUEST=RESET_REFBIT request. You can tailor the request to mark any of the following:

- A specific named data item (NAME) or a selection of data items based on filtering through a namemask (NAMEMASK)
- Any data item that is indicated as changed or locked for cast out (CRITERIA=CHANGED) based on the name (NAME) or filtering through a namemask (NAMEMASK)

If your protocol relies on external serialization, you need to hold a lock to serialize access to any data items. For serialization recommendations and sample scenarios that show how to establish serialization, see "Serializing and Managing Access to Shared Data" on page 377.

Timing and RESET_REFBIT Requests

When you issue the RESET_REFBIT request, consider the impact of timing issues on serialization. If another user creates a new data item in the cache after you have issued a RESET_REFBIT request, but before the request completes, the request might not mark the new data item as unreferenced. If you want to ensure that when your request completes, all data items matching your criteria have been marked as unreferenced, you must hold serialization throughout the request processing. Serialization needs to remain in effect for both the initial request, and any subsequent request restarts that might be required as a result of a timeout, and the scope of the serialization must prevent any other user from creating a new entry that matches the criteria on the RESET_REFBIT request.

Guide to the Topic

"RESET_REFBIT: Marking Data Items as Unreferenced" on page 459 is divided into two sections .

The firstsection , "IXLCACHE Functions for REQUEST=RESET_REFBIT" on page 460, applies to all RESET_REFBIT requests, and includes the following major topics:

- "Identifying Data Items to Mark as Unreferenced" on page 460
- "Restarting a Request that Ends Prematurely" on page 460
- "Receiving Answer Area Information" on page 461

The second section , "Marking a Data Item as Unreferenced: Summary" on page 461 summarizes a procedure for marking a data item as referenced.

IXLCACHE Functions for REQUEST=RESET_REFBIT

The following functions apply when you specify REQUEST=RESET_REFBIT.

Identifying Data Items to Mark as Unreferenced

To identify the data items that you want to mark as unreferenced, use the following combinations of the NAME, NAMEMASK, and CRITERIA keywords. Table 30 on page 460 describes which of the three keywords to code in order to mark the desired data items as unreferenced.

Table 30. Identifying Data Items to Mark as Unreferenced

To Mark as Unreferenced:	**Code:**
A single data item	NAME
A changed data item	NAME CRITERIA=CHANGED
All data items	CRITERIA=ALL (the default)
Changed data items	CRITERIA=CHANGED
Data items whose names satisfy a character selection pattern.	NAME NAMEMASK
Changed data items whose names satisfy a character selection pattern.	NAME NAMEMASK CRITERIA=CHANGED

Coding both NAME and NAMEMASK defines a character selection pattern that the system uses to select names. For a general description of NAMEMASK and the character selection pattern, see "Using Filters for Names on Requests" on page 398.

Restarting a Request that Ends Prematurely

The IXLCACHE REQUEST=RESET_REFBIT request can complete prematurely if the request exceeds the time-out criteria for the coupling facility. (Time-out criteria is model-dependent.) When a request completes prematurely, the system might not have marked as unreferenced all the data items specified

on the request. To mark the remaining data items, you must restart the request. For general information about restarting requests, see Restarting a Request that Ends Prematurely.

Note that you do not specify a buffer on the IXLCACHE REQUEST=RESET_REFBIT. If you want to keep track of the count for data entries that are processed on the request and the count of data entries for which the system resets the reference bit, you need to ensure that you include an answer area (ANSAREA). Before you restart the prematurely completed request, check the appropriate fields (CAADIRCOUNT for the total count and CAAREFCOUNT for the count of entries that have been reset).

Receiving Answer Area Information

On most IXLCACHE requests, the system returns information related to the request in the answer area. You specify the answer area on the ANSLEN and ANSAREA keywords. With certain events, the information in the answer area might not be valid. See "Determining Valid Information in the Answer Area" on page 397.

When the request completes, the system returns information to the answer area. When the request is not valid, the system returns non-zero return and reason codes.

For the mapping of the answer area, see the IXLYCAA mapping macro described in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary). For a description of answer area fields and return and reason codes for the request, see *z/OS MVS Programming: Sysplex Services Reference*.

Marking a Data Item as Unreferenced: Summary

You use the RESET_REFBIT request to mark as unreferenced specified data items.

- To identify the data items that the system is to mark as unreferenced, use the NAME, NAMEMASK, and CRITERIA keywords.
- Use these keywords together to identify a single data item or a group of data items whose names match a specified character selection pattern or match the criteria specified on the request.

If time-out criteria for the coupling facility are exceeded, the RESET_REFBIT request can complete prematurely. When a request completes prematurely, the system returns a token in the CAARESTOKEN field of the answer area. You can use this token to restart the request from the point at which it completed prematurely.

To restart a request, first process the data based on the count information that the RESET_REFBIT returns in the ANSAREA. After processing the data, code the IXLCACHE REQUEST=RESET_REFBIT request as you previously coded it with the exception of the RESTOKEN keyword. The RESTOKEN keyword must specify the token that the system returned.

There are other keywords that are required and some that are optional. For a description of keywords that are applicable to all IXLCACHE requests, see:

- "Understanding Synchronous and Asynchronous Cache Operations" on page 386
- "Accessing and Managing Data Within a Cache System" on page 369
- "Requesting Return and Reason Codes" on page 395
- "Defining an Answer Area (ANSAREA)" on page 395

READ_DIRINFO: Reading Cache Directory Entries

To read directory information for one or more data items, issue an IXLCACHE REQUEST=READ_DIRINFO request. You can read directory information for:

- A specific named data item (NAME) or a selection of data items based on filtering through a namemask (NAMEMASK)
- Any data item that is indicated as changed or locked for cast out (CRITERIA=CHANGED) based on the name (NAME) or filtering through a namemask (NAMEMASK)

For each specified data item that is selected, the system returns directory information to the local cache buffer. You specify whether you want all of the directory information returned for each selected data item (DIRINFOFMT=DIRENTRYLIST) or a subset of the information (DIRINFOFMT=NAMELIST).

Timing and READ_DIRINFO Requests

When you issue the READ_DIRINFO request, consider the impact of timing issues on serialization. If another user creates a new data item in the cache after you have issued a READ_DIRINFO request with criteria that matches the data item but before the request completes, the request might not include the directory entry for the new data item. If you want to ensure that when your request completes it includes all directory entries that match your criteria, you must hold serialization throughout the request processing. Serialization needs to remain in effect for both the initial request, and any subsequent request restarts that might be required as a result of a timeout, and the scope of the serialization must prevent any other user from creating a new entry that matches the criteria on the READ_DIRINFO request.

Guide to the Topic

"READ_DIRINFO: Reading Cache Directory Entries" on page 461 is divided into two sections .

The first section , "IXLCACHE Functions for REQUEST=READ_DIRINFO" on page 462, applies to all READ_DIRINFO requests, and includes the following major topics:

- "Identifying the Directory Entries to Read" on page 462
- "Selecting the Buffering Method" on page 463
- "Format of Returned Directory Information" on page 463
- "Restarting a REQUEST=READ_DIRINFO Request that Ends Prematurely" on page 464
- "Receiving Answer Area Information" on page 467

The second section , "Reading Directory Entry Information: Summary" on page 464 summarizes a procedure for reading directory information.

IXLCACHE Functions for REQUEST=READ_DIRINFO

The following functions apply when you specify REQUEST=READ_DIRINFO.

Identifying the Directory Entries to Read

To identify the data items whose directory information you want returned, use the NAME, NAMEMASK, and CRITERIA keywords. Table 31 on page 462 describes which of the three keywords to code in order to receive directory information from the desired data items.

Table 31. Identifying Directory Entries to Read

To Read Information For:	**Code:**
A specific data item	NAME
All data items	CRITERIA=ALL (the default)
All data items that are either changed or locked for cast-out	CRITERIA=CHANGED
All data items that satisfy a character selection pattern	NAME NAMEMASK
All data items that are either changed or locked for cast-out **and** whose names satisfy a character selection pattern.	NAME NAMEMASK CRITERIA=CHANGED

Coding both NAME and NAMEMASK defines a character selection pattern that the system uses to select names from the cache structure. For a general description of NAMEMASK and the character selection pattern, see "Using Filters for Names on Requests" on page 398.

Selecting the Buffering Method

The system returns the directory information to your local cache buffers. You can receive information in either a single buffer (the BUFFER keyword) or in multiple buffers (the BUFLIST keyword). Both methods enable you to receive up to 65536 (64K) bytes of data. For information about whether to use a single buffer or multiple buffers and for information on selecting buffer attributes, see "Design Considerations for Choosing the Buffer Format" on page 391.

If your local cache buffer is not large enough to hold all of the available information, the system fills the buffer, ends the request, and returns a specific return and reason code that indicates that the buffer has been filled. After you finish processing the information that is in the buffer, you can restart the request to have the system return the remaining information to the buffer. For information on restarting a request, see Restarting a Request that Ends Prematurely.

Format of Returned Directory Information

The READ_DIRINFO request specifies whether you want the system to return all of the directory information for each selected data item or a subset of the directory information. The information, which the system returns to the local cache buffer, occupies buffer storage starting at offset 0.

Reading All Directory Information

To read all directory information for each selected data item, code DIRINFOFMT=DIRENTRYLIST.

For each data item, the system returns a 128-byte block of directory information to your local cache buffer. Each block consists of the following information:

- Data item name
- Contents of the user-data field for the data entry
- The number of the storage class to which the data item is assigned
- An indication of whether the data item is marked changed or unchanged
- An indication of whether there is data stored in the cache for the data item
- The parity assigned to the data item
- The state of the cast-out lock
- The contents of the cast-out lock
- The number of the cast-out class to which the data item is assigned (valid only if the data entry is marked changed or locked for cast out)
- The number of cache structure elements allocated to the data item
- A bitstring that indicates registration of interest in the data item for all users.

The mapping macro IXLYDEIB maps the 128-byte directory block. For a description of IXLYDEIB, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

In the answer area field CAADIRCOUNT, the system also provides a count of the number of directory blocks returned to the local cache buffer.

Reading a Subset of Directory Information

To obtain a subset of directory information for each selected data item, code DIRINFOFMT=NAMELIST.

For each data item, the system returns a 32-byte block of directory information to your local cache buffer. Each block consists of the following information:

- Data item name
- Contents of the user-data field
- The number of cache structure elements allocated to the data item

Macro IXLYCANB maps the 32-byte directory block. For a description of IXLYCANB, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

The system also indicates in field CAADIRCOUNT of the answer area a count of the number of directory blocks for data items returned to the local cache buffer.

Restarting a REQUEST=READ_DIRINFO Request that Ends Prematurely

The IXLCACHE REQUEST=READ_DIRINFO request can complete prematurely for the following reasons:

- The local cache buffer cannot hold all of the available information.
- The request exceeds the time-out criteria for the coupling facility (Time-out criteria is model-dependent.)

Be sure to process the information returned from this request before reissuing the request. The data returned from this request will be overwritten if you specify the same buffer address. Continue to reissue the request until the return code indicates that all processing has completed.

For general information about restarting a request, see Restarting a Request that Ends Prematurely.

On most IXLCACHE requests, the system returns information related to the request in the answer area. You specify the answer area on the ANSLEN and ANSAREA keywords. With certain events, the information in the answer area might not be valid. See "Determining Valid Information in the Answer Area" on page 397.

When the request completes, the system returns information to the answer area. When the request is not valid, the system returns non-zero return and reason codes.

For the mapping of the answer area, see the IXLYCAA mapping macro described in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary). For a description of answer area fields and return and reason codes for the request, see *z/OS MVS Programming: Sysplex Services Reference*.

Reading Directory Entry Information: Summary

You use the READ_DIRINFO request to read directory information for one or more data items.

- To identify the data items whose directory entry you want to read, use the NAME, NAMEMASK, and CRITERIA keywords.
- Use these keywords together to identify a single data item or a group of data items whose names match a specified character selection pattern.

You can read all of the directory information for each data item or a subset of the information. To specify how much information you want returned, use the DIRINFOFMT keyword. To map the entire directory block for a data item, use the mapping macro IXLYDEIB. To map a subset of the directory block for a data item, use the mapping macro IXLYCANB.

To identify the buffers where the system is to return the information, code either BUFFER or BUFLIST and their related keywords. The system returns the directory information to contiguous buffer storage starting at offset 0.

The read directory entry request can complete prematurely if the buffer is not large enough to hold all of the data that the system is returning, or if the coupling facility time-out criteria are exceeded. When a request completes prematurely, the system returns a token in the CAARESTOKEN field of the answer area. You can use this token to restart the request from the point at which it completed prematurely.

To restart a request, first process the data that is in the buffer. After processing the data, code the IXLCACHE REQUEST=READ_DIRINFO request as you previously coded it with the exception of the RESTOKEN keyword. The RESTOKEN keyword must specify the token that the system returned.

There are other keywords that are required and some that are optional. For a description of keywords that are applicable to all IXLCACHE requests, see:

- “Understanding Synchronous and Asynchronous Cache Operations” on page 386
- “Accessing and Managing Data Within a Cache System” on page 369
- “Requesting Return and Reason Codes” on page 395
- “Defining an Answer Area (ANSAREA)” on page 395

READ_COCLASS: Reading A Cast-Out Class

To read information for all data items associated with a specific cast-out class, issue an IXLCACHE REQUEST=READ_COCLASS request. You can use a READ_COCLASS request to determine the following for a specified cast-out class:

- The names of all data items that belong to the cast-out class
- Whether a specific data item belongs to the cast-out class
- Which data items, whose names match a specified character selection pattern, belong to the cast-out class.

For each specified data item that belongs to the cast-out class, the system returns the name of the data item, user-defined data, if any, from the directory entry for the data item, and the number of cache structure elements allocated to the data entry.

When it is time to cast out changed data that is associated with a specific cast-out class, use the READ_COCLASS request to determine which data items belong to the cast-out class. For each entry that the READ_COCLASS returns, you can then issue the CASTOUT_DATA request, write the entry to permanent storage, and build a list of names for each of the data items in the cast-out class that you can use as input to the UNLOCK_CASTOUT request. When you have completed building the list of names for the data items, you can issue a single UNLOCK_CASTOUT request to release the cast-out locks for the data items.

If your protocol relies on external serialization, you need to hold a lock to serialize access to any data items. For serialization recommendations and sample scenarios that show how to establish serialization, see “Serializing and Managing Access to Shared Data” on page 377.

Timing and READ_COCLASS Requests

When you issue the READ_COCLASS request, consider the impact of timing issues on serialization. If another user creates a new data item in the cache after you have issued a READ_COCLASS request with criteria that matches the data item but before the request completes, the request might not contain information for the new data item. If you want to ensure that when your request completes, cast-out class information for all data items matching your criteria has been included, you must hold serialization throughout the request processing. Serialization needs to remain in effect for both the initial request, and any subsequent request restarts that might be required as a result of a timeout, and the scope of the serialization must prevent any other user from creating a new entry that matches the criteria on the READ_COCLASS request.

Guide to the Topic

“READ_COCLASS: Reading A Cast-Out Class” on page 465 is divided into two sections .

The first section , “IXLCACHE Functions for REQUEST=READ_COCLASS” on page 466, applies to all READ_COCLASS requests, and includes the following major topics:

- “Specifying the Data Item” on page 466
- “Specifying the Cast-Out Class” on page 466
- “Selecting the Buffering Method” on page 466
- “Format of Returned Cast-Out Class Data” on page 466
- “Restarting a REQUEST=READ_COCLASS Request that Ends Prematurely” on page 467
- “Receiving Answer Area Information” on page 467

The second section , “Reading a Cast-Out Class: Summary” on page 467 summarizes a procedure for reading cast-out class information.

IXLCACHE Functions for REQUEST=READ_COCLASS

The following functions apply when you specify REQUEST=READ_COCLASS.

Specifying the Data Item

The NAME keyword, or the NAME and NAMEMASK keywords together indicate which data items that belong to the specified cast-out class to select.

- To read information about all data items that belong to the specified cast-out class, omit both NAME and NAMEMASK from the request.
- To read information about a specific data item that belongs to the cast-out class, code NAME to provide the data item name. Omit NAMEMASK from the request.
- To read information about all data items that belong to the cast-out class and whose names satisfy a specified character selection pattern, code both NAME and NAMEMASK.

When you code both NAME and NAMEMASK, you define a character selection pattern that the system uses to select names from the specified cast-out class. For a general description of NAMEMASK and the character selection pattern, see “Using Filters for Names on Requests” on page 398.

Specifying the Cast-Out Class

Each request for cast-out class information must include the cast-out class number. Specify the number on the COCLASS keyword. On the READ_COCLASS request, you can only read information for one cast-out class at a time.

Note: The total number of cast-out classes defined for the cache structure is specified on the IXLCONN macro of the first user who connects to the structure. Cast-out classes are numbered consecutively from 1 to *n* where *n* is the number of cast-out classes specified on IXLCONN.

Selecting the Buffering Method

The system returns the data from the read cast-out class request to the local buffers that you specify. You can receive data in either a single buffer (the BUFFER keyword) or in multiple buffers (the BUFLIST keyword). Both methods enable you to receive up to 65536 (64K) bytes of data. For information about whether to use a single buffer or multiple buffers and for information on selecting buffer attributes, see “Design Considerations for Choosing the Buffer Format” on page 391.

If your local cache buffer is not large enough to hold all of the available data, the system fills the buffer, ends the request, and returns a specific return and reason code that indicates that the buffer has been filled. After you finish processing the data that is in the buffer, you can restart the request to have the system return the remaining data to the buffer. For information on restarting a request, see Restarting a Request that Ends Prematurely.

Format of Returned Cast-Out Class Data

For each specified data item that belongs to the cast-out class, the system returns a 32-byte block of information to your local cache buffer. Each block occupies contiguous buffer storage starting at offset 0 and consists of three fields containing:

- The name of a data item belonging to the cast-out class
- Any directory entry user-data associated with the data item
- The number of cache structure elements allocated to the data item

Macro IXLYCANB maps the 32-byte block of information. For a description of IXLYCANB, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

In the answer area field CAADIRCOUNT, the system also provides, a count of the number of blocks that are returned to the buffer.

Restarting a REQUEST=READ_COCLASS Request that Ends Prematurely

The IXLCACHE REQUEST=READ_COCLASS request can complete prematurely for the following reasons:

- The local cache buffer is not large enough to hold all of the available data.
- The request exceeds the time-out criteria for the coupling facility. (Time-out criteria is model-dependent.)

Be sure to process the information returned from this request before reissuing the request. The data returned from this request will be overwritten if you specify the same buffer address. Continue to reissue the request until the return code indicates that all processing has completed.

For general information about restarting requests, see Restarting a Request that Ends Prematurely.

Selecting the restart algorithm type

APAR OA14351 provides support for choosing the type of restart token algorithm the system is to use when restarting a READ_COCLASS request that ends prematurely. The NORMAL restart token algorithm operates as it always has. Because of unrelated processing that may have taken place before the user's restart processing begins, duplicate entries that match the user's filtering criteria may be returned and other entries may be missed entirely. The ENHANCED restart token algorithm eliminates the chance of entries that match the user's filtering criteria being missed, but there is still the possibility that duplicate entries may be returned.

To determine whether the coupling facility in which the cache structure resides is capable of using the enhanced restart token algorithm, you must examine field CaaEnhancedRtAlgPresent in IXLYCAA, the Cache Answer Area. A value of 1 in this field indicates that the enhanced restart token support is present; a value of 0 indicates that the support is not present.

Receiving Answer Area Information

On most IXLCACHE requests, the system returns information related to the request in the answer area. You specify the answer area on the ANSLEN and ANSAREA keywords. With certain events, the information in the answer area might not be valid. See “Determining Valid Information in the Answer Area” on page 397.

When the request completes, the system returns information to the answer area. When the request is not valid, the system returns non-zero return and reason codes.

For the mapping of the answer area, see the IXLYCAA mapping macro described in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary). For a description of answer area fields and return and reason codes for the request, see *z/OS MVS Programming: Sysplex Services Reference*.

Reading a Cast-Out Class: Summary

You use a read cast-out class request to obtain information about the data items that belong to a specified cast-out class.

- To identify the cast-out class, code the COCLASS keyword to provide the number of the cast-out class.
- You must identify the data items in which you are interested:
  - To obtain information on all data items in a specific cast-out class, omit both the NAME and NAMEMASK keywords.
  - To obtain information on a specific data item in the specified cast-out class, code NAME and omit NAMEMASK.
  - To obtain information about all data items whose name matches a specified character pattern and who are in the cast-out class, code both the NAME and NAMEMASK keywords. NAME must specify a

data item name that contains the specified character pattern. NAMEMASK must specify a bit-string where the bits that correspond to the specified character pattern are set to B'1'.

- To identify the buffer where the system is to return the cast-out class information, code either BUFLIST or BUFFER (depending on the buffering method you select) and their related keywords. The system returns the information starting at offset 0. Macro IXLYCANB maps each of the 32-byte elements of information.

The read cast-out class request can complete prematurely if the buffer is not large enough to hold all of the data that the system is returning, or if the coupling facility time-out criteria are exceeded. When a request completes prematurely, the system returns a token in the CAARESTOKEN field of the answer area. You can use this token to restart the request from the point at which it completed prematurely.

To restart a request, first process the data that is in the buffer After processing the data, code the IXLCACHE REQUEST=READ_COCLASS request as you previously coded it with the exception of the RESTOKEN keyword. The RESTOKEN keyword must specify the token that the system returned.

There are other keywords that are required and some that are optional. For a description of keywords that are applicable to all IXLCACHE requests, see:

- “Understanding Synchronous and Asynchronous Cache Operations” on page 386
- “Accessing and Managing Data Within a Cache System” on page 369
- “Requesting Return and Reason Codes” on page 395
- “Defining an Answer Area (ANSAREA)” on page 395

READ_COSTATS: Reading Cast-Out Class Statistics

Periodically, you might want to obtain statistics about your use of cast-out classes. For each specified cast-out class, the system returns the total number of data elements allocated to the data items in the cast-out class. To read cast-out statistics, use the IXLCACHE REQUEST=READ_COSTATS request. The system returns the statistics to the buffer that you specify on the request.

Guide to the Topic

“READ_COSTATS: Reading Cast-Out Class Statistics” on page 468 is divided into two section s.

The first section , “IXLCACHE Functions for REQUEST=READ_COSTATS” on page 468, applies to all READ_COSTATS requests, and includes the following major topics:

- “Specifying the Cast-out Classes” on page 468
- “Selecting a Buffering Method” on page 469
- “Format of Returned Cast-out Statistics” on page 469
- “Restarting A REQUEST=READ_COSTATS Request that Ends Prematurely” on page 470
- “Receiving Answer Area Information” on page 471

The second section , “Reading Cast-out Statistics: Summary” on page 471 summarizes how to use IXLCACHE REQUEST=READ_COSTATS.

IXLCACHE Functions for REQUEST=READ_COSTATS

The following functions apply when you specify REQUEST=READ_COSTATS.

Specifying the Cast-out Classes

The request must identify the range of cast-out classes whose statistics are to be read. To identify the first class in the range, code the COCLASSB keyword. To identify the last class in the range, code the COCLASSE keyword. The system returns statistics for classes starting with COCLASSB through COCLASSE. To read statistics for one class, COCLASSB and COCLASSE must specify the same cast-out class.

Selecting a Buffering Method

You can receive cast-out class statistics in either a single buffer (the BUFFER keyword) or in multiple buffers (the BUFLIST keyword). Both methods enable you to receive up to 65536 (64K) bytes of data. For information about whether to use a single buffer or multiple buffers and for information on selecting buffer attributes, see "Design Considerations for Choosing the Buffer Format" on page 391.

If your local cache buffer is not large enough to hold all of the available information, the system fills the buffer, ends the request, and returns a specific return and reason code that indicates that the buffer has been filled. After you finish processing the information that is in the buffer, you can restart the request to have the system return the remaining information to the buffer. For information on restarting a request, see Restarting a Request that Ends Prematurely and "Restarting A REQUEST=READ_COSTATS Request that Ends Prematurely" on page 470.

Format of Returned Cast-out Statistics

The format of the information returned from the READ_COSTATS request depends on the level of the coupling facility in which the structure is allocated and on the COSTATSFMT specification. For cache structures allocated in a coupling facility with CFLEVEL=5 or higher, you can use the COSTATSFMT keyword to specify the level of detailed information that is to be returned.

For cache structures allocated in a coupling facility with CFLEVEL=4 or lower, the system returns the cast-out class statistics to your buffers as follows:

The First Word

The first word or four bytes of the buffer, beginning at offset 0, contain two cast-out classes:

- The high-order two bytes contain the number of the first cast-out class specified on the COCLASSB keyword (that is, the first cast-out class for which information has been returned).
- The low-order two bytes contain the number of the last cast-out class (that is, the last cast-out class for which information has been returned).

Under certain situations, the value returned in the low-order two bytes might not be the value specified on the COCLASSE keyword. For instance, this value might be:

- The number of the last class read before the buffer became full. In this case, the buffer cannot accommodate all of the requested information and only part of the information is returned. You need to reissue the request to receive the remaining information.
- The number of the last class read when COCLASSE specified a number higher than the maximum number of defined cast-out classes. (The first user who connects to the cache structure defines the maximum number of cast-out classes on the IXLCONN macro.)

Mapping macro IXLYCCIH maps the first word of the buffer. For a description of IXLYCCIH, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

The Remainder of the Buffer

The data in the remainder of the buffer depends on whether you have specified the COSTATSFMT keyword. COSTATSFMT is valid only for structures allocated in a coupling facility with CFLEVEL=5 or higher.

- For structures allocated in a coupling facility with CFLEVEL=4 or lower, or when COSTATSFMT=COCOUNTSLIST is specified or defaulted to, the remainder of the buffer is as follows:

  The remainder of the buffer consists of four-byte entries. There is a one-to-one correspondence between each four-byte entry and a cast-out class in the range of cast-out classes for which the information is returned. Each entry contains the number of cache structure data elements allocated to the corresponding cast-out class. The first buffer entry corresponds to the cast-out class specified on the COCLASSB keyword. The next entry corresponds to the next sequentially numbered cast-out class, and so forth.

The following chart summarizes the buffer format and contents upon return from the request:

Offset	Contents
+0	Number of the first cast-out class reported on
+2	Number of the last cast-out class reported on
+4	Number of data elements allocated for the cast-out class specified by COCLASSB
+8	Number of data elements allocated for the next sequential cast-out class.
+(4 * n)	Number of data elements allocated for the last cast-out class where *n* is the total number of elements returned in the buffer.

Mapping CCIHCOUNTS of IXLYCCIH maps the information returned for structures allocated in a coupling facility with CFLEVEL=4 or lower or by specifying COSTATSFMT=COCOUNTSLIST.

- For structures allocated in a coupling facility with CFLEVEL=5 or higher and for which COSTATSFMT=COSTATSLIST is specified, the remainder of the buffer is as follows:

  The remainder of the buffer, starting at offset 32, consists of 16-byte entries. There is a one-to-one correspondence between each 16-byte entry and a cast-out class in the range of cast-out classes for which the information is returned. Each entry contains the number of cache structure data elements allocated to the corresponding cast-out class and eight bytes of user data. If the structure has been allocated with a UDF order queue for each cast-out class, the eight bytes is the user data of the first entry on the UDF order queue. If the structure has not been allocated with a UDF order queue, the eight bytes is the user data of the first entry on the cast-out class queue. The first buffer entry corresponds to the cast-out class specified on the COCLASSB keyword. The next entry corresponds to the next sequentially numbered cast-out class, and so forth.

  The following chart summarizes the buffer format and contents upon return from the request:

Offset	Contents
+0	Number of the first cast-out class reported on
+2	Number of the last cast-out class reported on
+4	Reserved
+32	Cast-out class entry data for the cast-out class specified by COCLASSB. – Number of data elements allocated – User data
+64	Cast-out class entry data for the next sequential cast-out class.
+(32 * n)	Number of data elements allocated for the last cast-out class where *n* is the total number of elements returned in the buffer.

  Mapping CCIHCCIBS of IXLYCCIH maps the information returned for structures allocated in a coupling facility with CFLEVEL=5 or higher when COSTATSFMT=COSTATSLIST is specified. For a description if IXLYCCIH, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Restarting A REQUEST=READ_COSTATS Request that Ends Prematurely

If the buffer is not large enough to hold all of the data to be read, an IXLCACHE REQUEST=READ_COSTATS can complete prematurely. When a request completes prematurely, the system returns as much data as the buffer can hold. In the first word of the buffer, the system returns the number of the first and last cast-out classes whose statistics were read.

To restart a prematurely completed request, use the following procedure:

1. Process the cast-out statistics that the system returns. You must process these statistics because the restarted request reuses the buffer.
2. Obtain the number of the last cast-out class whose statistics were read. This number is in the low-order two-bytes of the first word in the buffer. Macro IXLYCCIH maps the first word of the buffer and assigns symbolic names to both the low-order two bytes and the high-order two bytes.
3. Add 1 to the number obtained in step "2" on page 471, and specify this value on the COCLASSB keyword.
4. Reissue the IXLCACHE REQUEST=READ_COSTATS request. To ensure that you do not alter the intent of the request, the restarted request should specify the same keywords and values (with the exception of the value specified on COCLASSB) as the request that completed prematurely.

A restarted request can also complete prematurely. Restart the request using the procedure described.

Receiving Answer Area Information

On most IXLCACHE requests, the system returns information related to the request in the answer area. You specify the answer area on the ANSLEN and ANSAREA keywords. With certain events, the information in the answer area might not be valid. See "Determining Valid Information in the Answer Area" on page 397.

When the request completes, the system returns information to the answer area. When the request is not valid, the system returns non-zero return and reason codes.

For the mapping of the answer area, see the IXLYCAA mapping macro described in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary). For a description of answer area fields and return and reason codes for the request, see *z/OS MVS Programming: Sysplex Services Reference*.

Reading Cast-out Statistics: Summary

You read cast-out statistics to determine the total number of data entries that are assigned to cast-out classes.

- To identify the range of cast-out classes whose statistics are to be read, code the COCLASSB and COCLASSE keywords. COCLASSB identifies the number for the cast-out class at the beginning of the range and COCLASSE for the number of the cast-out class at the end of the range.
- To identify the buffers where the system is to return the cast-out statistics, code either BUFLIST or the BUFFER and their related keywords.
- To specify the format of the information returned for structures allocated in a coupling facility with CFLEVEL=5 or higher, code the COSTATSFMT keyword.

The system returns, to buffer offset 0, a fullword: the high-order two-bytes identify the first cast-out class whose statistics were read. The low-order two-bytes identify the last cast-out class whose statistics were read. IXLYCCIH maps the first word of the buffer. Following this fullword are the entries that contain the cast-out class statistics. The format of these entries is dependent on the level of coupling facility in which the structure is allocated and the COSTATSFMT sepcification.

The request can complete prematurely if the buffer is not large enough to hold all of the data to be returned. To restart the request, add 1 to the cast-out class number that the system returned in the low-order two bytes of the first word in the buffer. Specify the increment on the COCLASSB keyword and reissue the IXLCACHE REQUEST=READ_COSTATS request as previously coded (except for the new value of COCLASSB).

There are other keywords that are required and some that are optional. For a description of keywords that are applicable to all IXLCACHE requests, see:

- "Understanding Synchronous and Asynchronous Cache Operations" on page 386
- "Accessing and Managing Data Within a Cache System" on page 369
- "Requesting Return and Reason Codes" on page 395

- “Defining an Answer Area (ANSAREA)” on page 395

READ_STGSTATS: Reading Storage Class Statistics

During processing when you are using a cache structure, you might periodically need to obtain statistics about your use of the storage classes you have defined. For example, you can use the statistics to help analyze how efficiently you are using the cache structure. For a specified storage class, the system can return information as described in Table 32 on page 472.

To read storage class statistics, use the IXLCACHE REQUEST=READ_STGSTATS request. The system returns the statistics to a storage area that you specify. You must issue the request once for each storage class whose statistics you read.

Guide to the Topic

“READ_STGSTATS: Reading Storage Class Statistics” on page 472 is divided into two section s.

The first section , “IXLCACHE Functions for REQUEST=READ_STGSTATS” on page 472, applies to all READ_STGSTATS requests, and includes the following major topics:

- “Specifying the Storage Class” on page 472
- “Providing a Storage Area for Returned Statistics” on page 472
- “Description of Returned Statistics” on page 472
- “Receiving Answer Area Information” on page 474

The second section , “Reading Storage Class Statistics: Summary” on page 474 summarizes how to use IXLCACHE REQUEST=READ_STGSTATS.

IXLCACHE Functions for REQUEST=READ_STGSTATS

The following functions apply when you specify REQUEST=READ_STGSTATS.

Specifying the Storage Class

The request must identify the storage class whose statistics you want to read. To identify the storage class, code the STGCLASS keyword.

Providing a Storage Area for Returned Statistics

The request must identify a 256-byte storage area where the system can return the storage statistics. To identify the storage area, code the STGSTATS keyword.

Description of Returned Statistics

The system returns the storage class statistics described below. Mapping macro IXLYCSCS maps the statistics. See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for a description of IXLYCSCS.

Table 32. IXLCACHE Storage Class Statistics Description

Field Name	Description
CSCSREADHITC	**Read-hit** - Number of times system returned data on a read request.
CSCSRMDIRHITC	**Read-miss directory-hit** - number of times the system found the named data item identified to the structure with no cached data.
CSCSRMASSUPRC	**Read-miss assignment suppressed** - number of times the system found the named data item not identified to the structure and the allocation of the directory entry was intentionally suppressed (as a result of ASSIGN=NO on the READ_DATA request).

Table 32. IXLCACHE Storage Class Statistics Description (continued)	
Field Name	**Description**
CSCSRMNAMEASC	**Read-miss name assigned** - number of times the system found the named data item not identified to the structure and a directory entry was allocated (as a result of ASSIGN=YES on the READ_DATA request).
CSCSRMTSCFULLC	**Read-miss target storage class full** - number of times the system found the named data item not identified to the structure and a directory entry could not be allocated because no storage resources were available.
CSCSWHITCB0C	**Write-hit change bit 0** - number of times unchanged data was written.
CSCSWHITCB1C	**Write-hit change bit 1** - number of times changed data was written.
CSCSWMNOTREGC	**Write-miss not-registered** - number of times a request to write data failed because required connection interest was not previously registered.
CSCSWMINVSTATEC	**Write-miss invalid state** - number of times a request to write unchanged data failed because the named data item already had cached changed data.
CSCSWMTSCFULLC	**Write-miss target storage class full** - number of times a request to write data failed because either the named data item was not identified to the structure and no directory entry resource was obtainable, or no data entry resource could be obtained to contain data element resources.
CSCSDIRENTRYRCLC	**Directory entry reclaim** - number of times a directory entry was reclaimed.
CSCSDAENTRCLC	**Data entry reclaim** - number of times a data entry was reclaimed.
CSCSXIDIRRCLC	**XI for directory reclaim** - number of times cross-invalidate was performed as a result of a directory entry reclaim.
CSCSXIWRITEC	**XI for write** - number of times cross-invalidate was performed as a result of a request to write data.
CSCSXINMINVALC	**XI for name invalidation** - number of times cross-invalidate was performed as a result of a request to delete a named data item.
CSCSXICMINVALC	**XI for complement invalidation** - number of times cross-invalidate was performed as a result of a user request to perform cross-invalidation for the named data item.
CSCSCASTOUTC	**Cast-out** - number of times data has been cast-out.
CSCSREFSIGMISSC	**Reference signal miss** - number of named data items for the storage class that reference list processing specified but could not find in the structure.
CSCSTMCFULLC	**Target storage class full** - number of times that the allocation of the directory entry or data entry failed because resources were unavailable and all named data items for the storage class had changed cached data.
CSCSDIRENTRYC	**Directory entry** - number of directory entries allocated for named data items.
CSCSDATAREAELEC	**Data area element** - number of data elements allocated for named data items.
CSCSTOTCHNGDC	**Total changed** - number of named data items assigned to the specified storage class that have changed or locked-for-cast-out cached data.
CSCSDATAREAC	**Data area** - number of data entries allocated for named data items.
CSCSCMPLREFLSTC	**Completed reference lists** - number of PROCESS_REFLIST requests in the list that were processed.
CSCSPRTCREFLSTC	**Partially completed reference lists** - number of PROCESS_REFLIST requests in the list that were processed incompletely because coupling facility time out criteria was exceeded.
CSCSXILCVIREPL	**XI for local cache vector entry replacement** - number of times cross-invalidate was performed as a result of a request that specified a local cache vector index for a data item to replace an existing index.

Table 32. IXLCACHE Storage Class Statistics Description (continued)	
Field Name	**Description**
CSCSWUXIC	**Write unchanged with XI counter** - The number of times cross-invalidate was performed as a result of a WRITE_DATA request that specified CHANGED=NO and CROSSINVAL=YES.

Receiving Answer Area Information

On most IXLCACHE requests, the system returns information related to the request in the answer area. You specify the answer area on the ANSLEN and ANSAREA keywords. With certain events, the information in the answer area might not be valid. See "Determining Valid Information in the Answer Area" on page 397.

When the request completes, the system returns information to the answer area. When the request is not valid, the system returns non-zero return and reason codes.

For the mapping of the answer area, see the IXLYCAA mapping macro described in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary). For a description of answer area fields and return and reason codes for the request, see *z/OS MVS Programming: Sysplex Services Reference*.

Reading Storage Class Statistics: Summary

You read storage statistics to collect information that characterizes, by storage class, your use of the cache structure.

- Specify the STGCLASS keyword to identify the storage class and the STGSTATS keyword to identify where the system is to store the statistics.
- To map the statistics, use the IXLYCSCS macro.

There are other keywords that are required and some that are optional. For a description of keywords that are applicable to all IXLCACHE requests, see:

- "Understanding Synchronous and Asynchronous Cache Operations" on page 386
- "Accessing and Managing Data Within a Cache System" on page 369
- "Requesting Return and Reason Codes" on page 395
- "Defining an Answer Area (ANSAREA)" on page 395

Coding a Complete Exit for IXLCACHE

Your complete exit provides a mechanism for the system to let you know when your asynchronously processed IXLCACHE request completes. You provide the address of your complete exit using the COMPLETEEXIT parameter when issuing the IXLCONN macro to connect to the structure.

You will be informed of request completion through your complete exit in either of the following situations:

- You specify MODE=ASYNCEXIT.
- You specify MODE=SYNCEXIT and the system processes your request asynchronously.

Information Passed to the Complete Exit

When the complete exit gains control, it receives the following information about the IXLCACHE request and its outcome in the complete exit parameter list (CMPL), mapped by the IXLYCMPL macro:

CMPLCONTOKEN
The IXLCACHE invoker's connect token.

CMPLCONNAME
The IXLCACHE invoker's connect name.

CMPLCONDATA
Connect-time data you specified when you issued the IXLCONN macro to connect to the structure. The use of this optional field is defined by the user. One possibility is to allow a user to contain the address or ALET of a connection-related control block or data structure.

CMPLCACHE
Indicates the complete exit received control as a result of an IXLCACHE request.

CMPLREBUILD
Indicates whether the target structure was being rebuilt. When a structure is being rebuilt, there is an interval in which the new structure and the old structure can both be the target of an IXLCACHE request.

0
The target structure was not being rebuilt or, if so, the target structure was the original structure.

1
The target structure was being rebuilt, and the target structure was the new structure.

CMPLRETCODE
Return code from IXLCACHE request. Return code values are defined in the IXLYCON macro.

CMPLRSNCODE
Reason code from IXLCACHE request. Reason code values are defined in the IXLYCON macro.

CMPLREQDATA
Information provided to the complete exit by the issuer of the IXLCACHE request. The use of this optional field is user defined. One possibility is to store the address of a control block that represents the asynchronously-processed request. When the request makes status information available upon completion, the user can return to the control block and update status.

CMPLANSAREAALET
Answer area ALET.

CMPLANSAREA@
Answer area address.

See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for a description of the IXLYCMPL macro.

Environment

The complete exit receives control in the following environment:

Authorization:	Supervisor state, and PSW key 0
Dispatchable unit mode:	SRB
Cross memory mode:	PASN=HASN=SASN. PASN, HASN, and SASN are equal to the PASN at the time of the connect to the structure.
AMODE:	31-bit
ASC mode:	Primary ASC mode
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held.
Control parameters:	None.

Input Specifications

Cache services pass information to the complete exit in registers and in the CMPL.

Registers at Entry:

When the complete exit receives control, the GPRs contain the following information:

Register
Contents

0
Does not contain any information for use by the complete exit.

1
Address of a full word that contains the address of the CMPL.

2-12
Do not contain any information for use by the complete exit.

13
Address of a 72-byte work area for use by the complete exit routine. The exit routine does not have to save and restore registers in this work area. The exit routine can use this work area in any way it chooses.

14
Return address.

15
Entry point address.

When the complete exit receives control, the ARs contain no information for use by the complete exit.

Return Specifications

Your exit must return control to the system by branching to the address provided on entry in GPR 14.

Programming Considerations

If you have more than one outstanding IXLCACHE request being processed asynchronously, multiple instances of your complete exit might run concurrently as the system processes your requests. Also, the order of execution of the complete exit for asynchronous requests is unpredictable. For example if you specify two requests with MODE=ASYNCEXIT, one to read data item A and another to read data item B, the system might complete the read for data item B before the read for data item A.

The CMPL data area is accessible to you only while your complete exit is running. Once the exit returns to its caller, you can no longer access the CMPL data area.

In certain instances, the system must quiesce the activity of user exits in order to perform cleanup processing. The following illustrates scenarios where this processing occurs:

- Connection Termination

  When a user disconnects or abnormally terminates, the system will force to completion any user exits executing on behalf of that user by issuing a PURGEDQ against the appropriate units of work. Note that if a connector terminates while a rebuild is in progress, any exits pertaining to both the original and the new structures will be forced to completion. In addition to forcing the currently executing user exits to completion, the system will also prevent any new invocations of these exits by cancelling any events that are pending presentation.
- Rebuild Stop

  When a connector provides an event exit response for the Rebuild Stop event, the system will force to completion any exits that are executing on behalf of that user's connection to the new structure by issuing a PURGEDQ against the appropriate units of work. Similar to connector termination processing, the user exits pertaining to the new structure will not be presented with any additional events. Note that any user exits executing on behalf of the original structure are unaffected by rebuild stop processing.
- Completion of a Rebuild

  When a connector provides an event exit response for the Rebuild Cleanup event, the system will force to completion any user exits that are executing on behalf of that user's connection to BOTH the original and the new structures by issuing a PURGEDQ against the appropriate units of work. No new events will be presented to the user exits on behalf of the original structure (as it is being discarded). Normal user exit processing will resume for the rebuilt structure upon completion of the rebuild process.

A user exit must be sensitive to conditions that can occur as a result of actions taken by the system and must be able to handle these as appropriate. For example, if a user exit has suspended itself, when the PURGEDQ is issued the system abends the user exit's unit of work with a retryable X'47B' abend and gives control to the user exit's recovery routine. (Note that although the recovery routine can retry, the user exit can not re-suspend itself because the system will fail any request to suspend a unit of work that has been the target of a PURGEDQ.) If the recovery routine percolates back to the system, its associated connection is terminated.

Managing Cache Structure Utilization

The cache structure is allocated with a fixed amount of storage. This storage can be subdivided into directory entries and data elements. If an IXLCACHE request requires that an object be available but none is, a “structure-full” condition occurs. When the structure becomes full, you will no longer be able to perform a number of IXLCACHE functions. Affected functions could include:

- The ability to create a new cache entry.
- The ability to update an existing cache entry, regardless of whether its size would increase, decrease, or remain the same.

The system returns counts of the objects allocated in the structure in the connect answer area (IXLYCONA). The values reflect the state of the structure at the time of the connect.

- CONACACHECHGDIRENTRYCOUNT — Approximate number of changed directory entries in use
- CONACACHEDIRENTRYCOUNT — Approximate number of directory entries supported by the structure
- CONACACHECHGDIRELEMENTCOUNT — Approximate number of changed data elements in use
- CONACACHEMAXELEMENTCOUNT — Approximate maximum number of data elements supported by the structure.

Taking action to alleviate the storage problem before the structure becomes full is especially critical because the CONACACHEDIRENTRYCOUNT and CONACACHEMAXELEMENTCOUNT values are only approximate. As a result, you could receive a return code indicating that the structure is full even though the answer area counts of in-use entries or elements that are changed (and therefore cannot be reclaimed by the coupling facility) are below the limits indicated in the CONA.

A reason for the CONA counts being approximate is that the coupling facility at times uses some of the structure's objects for its own processing. Those objects are not included in your “in-use” counts.

Another result of the CONA counts being approximate is that the IXLCACHE request of one connector might be rejected due to a structure full condition while a subsequent request by a different connector might succeed. Alternatively, a request by a connector might be rejected while a subsequent request by the same connector might succeed. Furthermore, deleting a cache entry when the structure is full might not result in the immediate availability of the storage for the directory entry or data elements. As a result, your request could fail if you attempt to create an entry of the same size as the one you deleted.

Applications using the cache structure are responsible for managing structure utilization. The system does not prevent the structure from becoming full nor take any automatic action to remedy the condition. Therefore, **IBM recommends** that you take steps to correct a storage shortage before your application is affected. To do so, you need to consider the following:

- How to detect when the structure is becoming full
- How full you will permit the structure to become before you take remedial action
- How the storage shortage will be corrected.

Detecting When a Cache Structure Is Becoming Full

One way to monitor cache structure utilization is to issue the IXLMG macro periodically and check the following fields:

- IXLYAMDSTRC_TDAEC, which returns the approximate maximum number of data elements allowed in the structure

- IXLYAMDSTRC_TDEC, which returns the approximate maximum number of entries allowed in the structure
- IXLYAMDSTRC_TSCC, which returns the approximate number of changed entries in use in the structure
- IXLYAMDSTRC_TCDEC, which returns the approximate number of changed data elements in use in the structure

These values can be used to calculate the structure's approximate percentage fullness in terms of entries and elements.

Responding When the Structure Is Getting Full

When your monitoring indicates that the structure is getting full, you can take several actions. First, until you resolve the storage problem, your application could minimize its issuance of IXLCACHE requests that create or modify cache entries. Your application can also issue a message to the operator to warn that the structure is getting full and to request that the operator perform certain actions.

You can issue IXLCACHE REQUEST=READ_STGSTATS or IXLMG to determine how full each storage class in the structure is becoming. See “Managing Cache Structure Resources” on page 380 for a description of how storage in the structure can be reclaimed.

If the structure is running out of elements but has plenty of entries (or vice versa), you can rebuild or alter the structure with a different ratio of elements to entries without changing the structure's size. Because the structure is not changing size, operator intervention is only required if altering ratios and the SETXCF MODIFY command is used to disable alter processing for the structure.

If the structure needs more directory entries or data elements, you can rebuild or alter the structure with more storage. Rebuilding or altering the structure with more storage might require operator intervention.

Rebuilding the Structure to Increase the Storage Capacity

You can rebuild the structure to increase capacity only if the CFRM policy that defines the structure allows for a larger size. If the structure is already the maximum size allowed by the CFRM policy, you must request that the system programmer modify the CFRM policy to allow a larger structure size and reactivate the modified policy.

If the active CFRM policy allows for a larger cache structure, you can issue the IXLREBLD macro to rebuild the structure with a larger size. If you prefer to involve the operator, your application can issue a message to notify the operator that the structure needs to be rebuilt. The operator must issue the SETXCF START,REBUILD command to initiate structure rebuild.

Note: Duplexed structures cannot be rebuilt while they remain duplexed. If the structure is duplexed, duplexing will need to be stopped before the structure can be rebuilt. This can be done using the IXLREBLD macro or the SETXCF STOP,REBUILD command. If the CFRM active policy specifies DUPLEX(ENABLED) for the structure and IXLREBLD IGNOREDUPLEX=YES is not used, the system might immediately reduplex the structure after the completion of the stop processing. There might be a delay before reduplexing when only two coupling facilities are available for duplexing the structure. Reduplexing will occur immediately in configurations with three or more coupling facilities available for duplexing the structure.

To prevent the system from immediately reduplexing the structure or reduplexing the structure at a later time, change the DUPLEX specification for the structure to DUPLEX(ALLOWED) or DUPLEX(DISABLED). Change the DUPLEX setting for the structure in the CFRM policy to DUPLEX(ALLOWED) before stopping duplexing, or change the DUPLEX setting for the structure to DUPLEX(DISABLED), which will cause XCF to initiate the stop processing. Change the DUPLEX setting back to DUPLEX(ENABLED) when you no longer need to prevent the system from reduplexing.

Altering the Structure to Increase the Storage Capacity

With SP 5.2 and above and a structure allocated in a coupling facility with CFLEVEL=1 or higher, you can alter the size of the structure to increase capacity or the entry-to-element ratio to reapportion the structure's storage. As with the rebuild function, you can alter the structure only if the CFRM policy that

defines the structure allows for a larger size. You can issue the IXLALTER macro or notify the operator to issue the SETXCF START,ALTER command to initiate structure alter.

Chapter 8. Using List Services (IXLLIST)

List services allow database products, subsystems, and authorized applications running in the same sysplex to use a coupling facility to share data organized in a list structure. The list structure consists of a set of lists and an optional lock table. Information is stored on each list as a series of list entries. Lists can be used as arrays, stacks, or queues. List entries can also be kept in sorted order by key value. The lock table can be used to serialize on resources in the list structure, such as a particular list or list entry.

You can use the list structure and its associated services to distribute work requests among members of the sysplex or to maintain shared status information. For instance, different lists in the list structure could represent different classes of data or states of work items. A user could move a list entry from one list to another to reflect a change in the class or the state of the work represented by the list entry.

Interfaces to list services provide the functions necessary to access and manage the list structure. Prior to OS/390 Release 9, IXLLIST was the sole interface to list services. Starting with OS/390 Release 9, three additional interfaces are available — IXLLSTC, IXLLSTE, and IXLLSTM. Functionally, these latter three interfaces provide the same services as IXLLIST, but also provide additional function beyond that provided by IXLLIST. It is IBM's intention to continue to support the IXLLIST service, but to add any new functions only to the IXLLSTC, IXLLSTE, and IXLLSTM services.

IXLLSTC, IXLLSTE, and IXLLSTM contain updated syntax and may contain keyword names changed from IXLLIST. However, each IXLLIST invocation can be replaced by an equivalent IXLLSTC, IXLLSTE, or IXLLSTM invocation. Table 34 on page 487 lists the services available with IXLLIST and names the equivalent IXLLSTC, IXLLSTE, or IXLLSTM invocation. Chapter 9, “Using List Services (IXLLSTE, IXLLSTM, IXLLSTC),” on page 595 discusses the additional functions provided by the IXLLSTC, IXLLSTE, and IXLLSTM services at OS/390 Release 9 and higher. Note that many of the new functions require that the list structure be allocated in a coupling facility of CFLEVLE=9 or higher.

The IXLLIST macro, an interface to list services, allows you to:

- Read, write, move, or delete an entry in the list structure
- Combine operations, such as the following, using a single IXLLIST request:
  - Read an entry and delete it
  - Move an entry and update it
  - Move an entry and read it
- Read or delete multiple entries with a single IXLLIST request
- Perform a serialized update to a list entry by performing a lock operation (such as obtain, release, or test) and a list entry operation as part of the same IXLLIST request. The ability to perform a lock operation together with a list entry operation helps applications protect the integrity of data in the list structure.

The lock table consists of an array of exclusive locks. The purpose and scope of each entry in the lock table is entirely user-defined.

IXLLIST also provides high-performance list transition monitoring that allows you to detect when a list changes from the empty state to the nonempty state (in which it has one or more entries) without having to access the coupling facility to check the list. For instance, if you are using the list structure as a distribution mechanism for work requests, list transition monitoring allows users to detect easily the presence or absence of incoming work requests on their queues.

With a coupling facility of CFLEVEL=3 or higher, IXLLIST also provides two additional monitoring functions for keyed list structures — event queue monitoring and sublist monitoring.

- An event queue exists in the list structure for each connected user. Its purpose is to be a repository for event monitor controls objects (EMCs) that represent **events**. An example of an event is the change in the state of a sublist from the empty state to the nonempty state. Event queue monitoring allows users to determine efficiently whether there are events queued on their event queue.

© Copyright IBM Corp. 1994, 2023

- A sublist is a subset of a list in which each entry in the sublist has the same key. As with list monitoring, the sublist monitoring function allows users to detect when the sublist has changed from the empty state to the nonempty state. However, the system reports the state transition by queueing or withdrawing EMCs from the user's event queue. It is this queueing or withdrawing of EMCs to or from the user's event queue that causes the event queue transitions to nonempty or empty. Event queue monitoring monitors these transitions.

You can choose to be notified of list transitions and/or event queue transitions by having your list transition exit receive control or you can issue the IXLVECTR macro to test whether a list or event queue you are monitoring has changed from empty to nonempty.

The system processes each IXLLIST request **atomically**, that is, a request is processed from start to finish without interruption, ensuring that the list structure data can never be viewed or accessed by other connections while it is being modified. The serialized list structure allows you to serialize multiple IXLLIST requests so that they are performed atomically as seen by other users of the structure who are observing the same serialization protocols.

Guide to the Topics

The following topics help you understand the list structure and the functions provided by the IXLLIST macro:

- "List Structure Concepts" on page 482
- "WRITE: Writing to a List Entry" on page 531
- "READ, READ_MULT, READ_LIST: Reading List Entries" on page 537
- "MOVE: Moving a List Entry" on page 550
- "DELETE, DELETE_MULT, DELETE_ENTRYLIST: Deleting List Entries" on page 557
- "READ_LCONTROLS: Reading List Controls" on page 565
- "WRITE_LCONTROLS: Writing List Controls" on page 567
- "LOCK: Performing a Lock Operation" on page 568
- "MONITOR_LIST: Monitoring List Transitions" on page 570
- "MONITOR_EVENTQ: Monitoring an Event Queue" on page 574
- "MONITOR_SUBLIST, MONITOR_SUBLISTS: Monitoring Sublists" on page 576
- "READ_EMCONTROLS: Reading Event Monitor Controls" on page 580
- "READ_EQCONTROLS: Reading Event Queue Controls" on page 581
- "DEQ_EVENTQ: Retrieving Events from the Event Queue" on page 582
- "Coding a Complete Exit" on page 583
- "Coding a Notify Exit" on page 585
- "Coding a List Transition Exit" on page 588
- "Managing List Structure Utilization" on page 590

List Structure Concepts

This section discusses basic concepts relating to the list structure and the functions it provides, such as:

- What is a list structure?
- How is data maintained in the list structure?
- What functions does the list structure provide?
- How do you reference list entries?
- What are event monitor controls?
- What are the notify, complete, and list transition exits?

- What are synchronous and asynchronous list operations?
- What is a serialized list?

What is a List Structure?

A list structure consists of a set of lists and an optional lock table of exclusive locks, which you can use to serialize the use of lists, list entries, or other resources in the list structure. Each list is pointed to by a header and can contain a number of list entries. A list entry consists of list entry controls and can also include a data entry, an adjunct area, or both. Both data entries and adjunct areas are optional. However, data entries are optional for each list entry while a list structure either has or doesn't have adjunct areas. Figure 38 on page 483 shows a list structure that contains an optional lock table. A list structure that includes a lock table is called a **serialized list structure.**

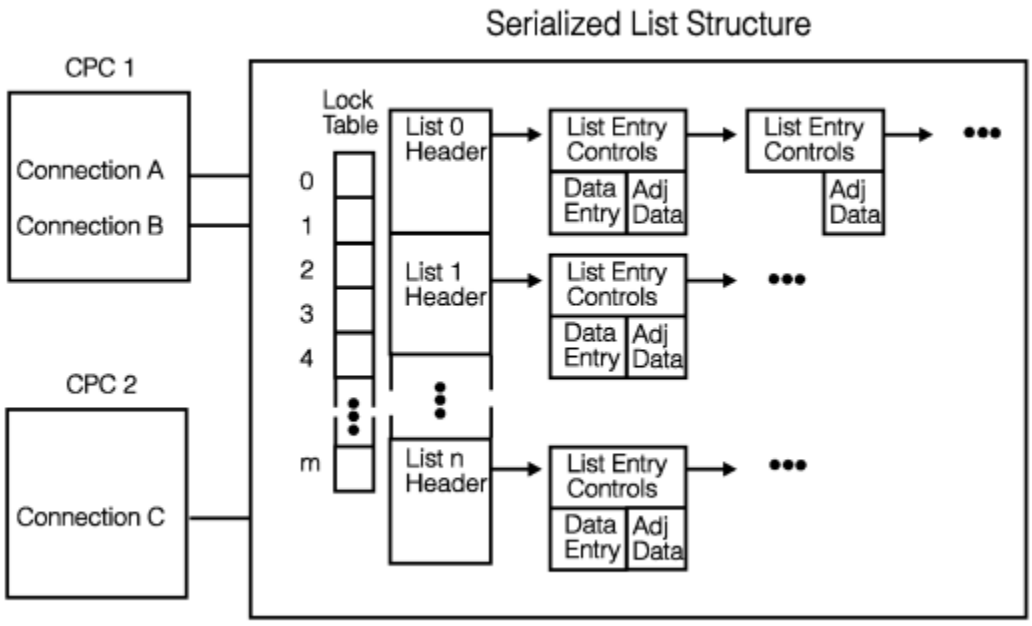


Figure 38. Serialized List Structure

The parts of the coupling facility list structure are:

List header
Anchors the list to the list structure and contains control information associated with the list (list controls). The first user to connect to the list structure designates the number of list headers it is to have, and allocates the list structure.

List entry
An entry on the list. A list entry consists of:

- **List entry controls**, which contain control information associated with the list entry.
- An optional **data entry**, which holds user-specified data. Data entries are composed of units of storage called **data elements**. In a coupling facility of CFLEVEL=0, data entries can be composed of 0 to 16 data elements. In a coupling facility of CFLEVEL=1 or higher, data entries can be composed of 0 to 255 data elements. In either case, a data entry can contain up to 64K (65536 bytes) of data.
- An **adjunct area** used to hold up to 64 bytes of data. You could use the adjunct area to maintain control information about the contents of the data entry. If your data is always 64 bytes or less, you could use adjunct areas to hold your data and omit the use of data entries.

Each list entry can reside on only one list at a time. Unused list entries do not reside on any list.

Lock table
An array of exclusive locks that can be used to serialize access to list structure resources such as lists or list entries. Lock table users create and maintain the association between a lock table entry and its associated resource. The lock table can be used:

- Together with list entry operations such as reading or writing list entry data
- Independently of list entry operations

List Structure Enhancements

With a coupling facility of CFLEVEL=3 or higher, a keyed list structure can optionally support event queues that are associated with sublist monitoring. (A sublist is a subset of a list — see "Understanding List Entry Key Assignment" on page 490 for a description of a sublist.) The system uses the event queues to hold control objects called event monitor controls (EMCs), which contain information about the user and the sublist being monitored. Whenever a monitored sublist transitions from an empty to a nonempty state, an EMC is queued to the user's event queue. The system withdraws the EMC from the user's event queue when the sublist transitions from a nonempty to an empty state.

Figure 39 on page 484 shows the optional parts of a keyed list structure allocated in a coupling facility of CFLEVEL=3 or higher.

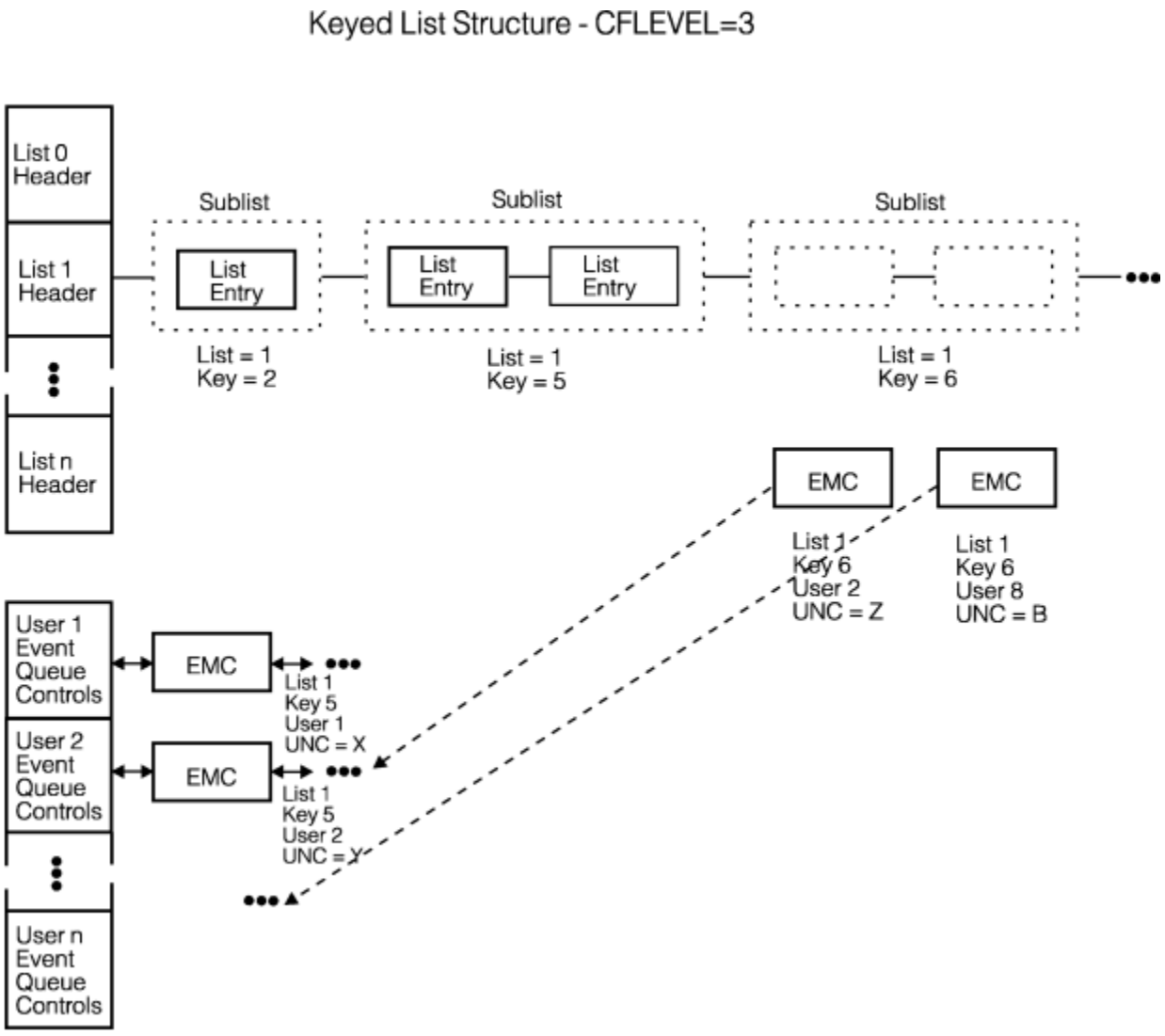


Figure 39. Event Queues in a List Structure

The additional parts of the keyed list structure in a coupling facility of CFLEVEL=3 or higher are:

Event queue controls
Contain control information about the state of the event queue and monitoring data. There is an event queue and event queue controls associated with each user who has connected to the keyed list structure.

Event monitor controls
Contain information about the user and the sublist being monitored. There is an event monitor controls object for each user and sublist combination when the user has registered interest in monitoring a particular sublist. (For example, an EMC exists for User 1 and the sublist specified by List 1 and Key 5; another EMC exists for User 2 and the sublist specified by List 1 and Key 6.) An EMC object can reside in the list structure in association with a particular monitored sublist or on the monitoring user's event queue.

Figure 39 on page 484 shows that for the sublist (List=1, Key=5) there are two EMCs allocated. When the sublist transitioned to a nonempty state, the EMCs were queued to the appropriate users' event queues. If the sublist transitioned to an empty state, the system would withdraw the EMCs from the users' event queues.

The figure also shows that for the sublist (List=1, Key=6), there are as yet no list entries. However, there are two EMCs allocated — for users 2 and 8. When the sublist transitions to a nonempty state, the system will queue these EMCs to the appropriate users' event queues.

How Is Data Maintained in a List Structure?

Data in the list structure is stored in list entries, each of which can consist of a data entry of up to 16 data elements in a CFLEVEL=0 coupling facility (or up to 255 data elements in a CFLEVEL=1 or higher coupling facility) and an optional adjunct data area. Table 33 on page 485 shows the components of a list entry in detail.

Table 33. Components of a List Entry

Component	Size	When Attributes Are Determined	When Attributes Can Be Changed
Data element	256, 512, 1024, 2048, or 4096 bytes	The first connector to the list structure selects the element size	Element size is fixed for the life of the list structure but you can change this attribute when the structure is rebuilt
Data entry		When you perform a write operation, you designate the number of data elements to be allocated to the target data entry	You can change the number of data elements in the target data entry when you perform a write operation
	0 to 16 elements **CFLEVEL=0**	The first connector to the structure specifies the actual maximum number of data elements per data entry (**16 or less**) using the MAXELEMNUM parameter of the IXLCONN macro	
	0 to 255 elements **CFLEVEL=1 or higher**	The first connector to the structure specifies the actual maximum number of data elements per data entry (**255 or less**) using the MAXELEMNUM parameter of the IXLCONN macro	

Table 33. Components of a List Entry (continued)			
Component	**Size**	**When Attributes Are Determined**	**When Attributes Can Be Changed**
Adjunct area	64 bytes	The first connector to the list structure specifies whether the list structure has adjunct areas	The presence or absence of adjunct areas is fixed for the life of the list structure but you can change this attribute when the structure is rebuilt.

The number of data element sizes and range in number of elements per data entry provides a tremendous choice of data entry sizes. The maximum data entry size is 64K bytes, except in a structure that has an element size of 256 bytes. Because the maximum number of elements per data entry is 255, the maximum data entry size with 256-byte data elements is 65280 bytes (255 x 256). All other combinations of element size and entry size allow a maximum of 64K (65536 bytes). Although a data entry is composed of a number of data elements, list operations treat the data entry as a single entity; **data elements cannot be read or written individually**. The adjunct area can be used to hold additional, user-specified information about the data entry.

Figure 40 on page 486 shows a list containing list entries with various numbers of data elements.

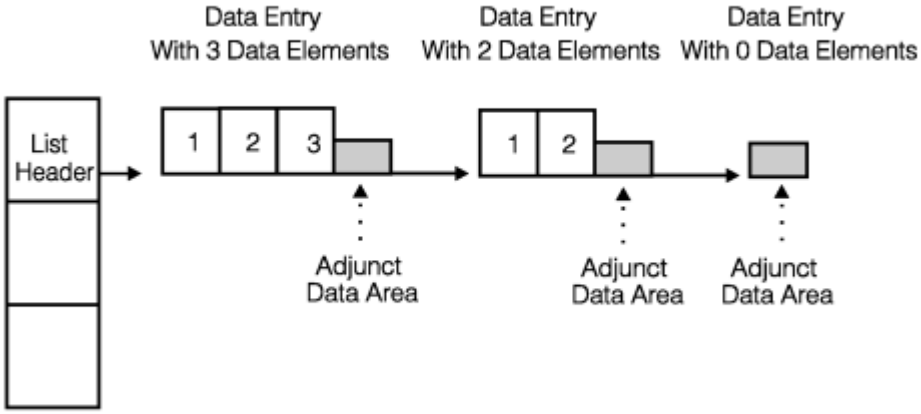


Figure 40. List Containing Entries with Various Numbers of Data Elements

What Functions Does the List Structure Provide?

Table 34 on page 487 summarizes the functions you can perform on a list structure using the IXLLIST macro. This table is intended to give you a brief overview of the functions, each of which is discussed in detail later in this chapter . Functions that are available only for structures allocated in a certain level of coupling facility are noted.

Also included in the table is the equivalent macro statement using one of the OS/390 Release 9 and higher list services macros — IXLLSTC, IXLLSTE, and IXLLSTM.

Table 34. Summary of IXLLIST Macro Functions	
Function	**Action**
WRITE	Update an existing list entry or create a new one CFLEVEL=1 or higher: • Assign a list entry key from a list control value. • Write a list entry based on the success of a list authority comparison or enhanced version number comparison. **IXLLSTE ENTRYTYPE=OLD/ANY,REQUEST=WRITE**
READ	Read the contents of a list entry CFLEVEL=1 or higher: • Read the contents of a list entry based on the success of a list authority comparison or enhanced version number comparison. **IXLLSTE ENTRYTYPE=OLD,REQUEST=READ,ENTRYDISP=KEEP**
READ_LIST	Read the contents of multiple list entries on a particular list or list entries on a particular list with a certain version number CFLEVEL=1 or higher: • Select for processing by extended filtering by entry key value, version number, and/or list authority value. **IXLLSTM REQUEST=READ_LIST**
READ_MULT	Read the contents of all list entries in the structure or only those: • With a certain version number • On a certain list • On a certain list with a certain version number. CFLEVEL=1 or higher: • Select for processing by extended filtering by entry key value, version number, and/or list authority value. **IXLLSTM REQUEST=READ_MULT**
MOVE	Move a list entry to another list or to a different position on the same list. Options are: • Move a list entry • Move a list entry and read its contents • Move a list entry and update its contents • Create a new list entry if it does not already exist. CFLEVEL=1 or higher: • Assign a list entry key from a list control value. • Move a list entry based on the success of a list authority comparison or enhanced version number comparison.: **IXLLSTE ENTRYTYPE=OLD,REQUEST=MOVE,ACTION=NONE**

Table 34. Summary of IXLLIST Macro Functions (continued)	
Function	**Action**
DELETE	Delete a list entry CFLEVEL=1 or higher: • Delete a list entry based on the success of a list authority comparison or enhanced version number comparison. **IXLLSTE ENTRYTYPE=OLD,REQUEST=DELETE**
DELETE_MULT	Delete all list entries in the structure or only those: • With a certain version number • On a certain list • On a certain list with a certain version number. CFLEVEL=1 or higher: • Select for processing by extended filtering by entry key value, version number, and/or list authority value. **IXLLSTM REQUEST=DELETE_MULT**
DELETE_ENTRYLIST	Delete the list entries you identify in a list of entries passed as input. CFLEVEL=1 or higher: • Select for processing by extended filtering by entry key value, version number, and/or list authority value. **IXLLSTM REQUEST=DELETE_ENTRYLIST**
READ_LCONTROLS	Read a list's control information CFLEVEL=1 or higher: • Read additional control information. **IXLLSTC REQUEST=READ_LCONTROLS**
WRITE_LCONTROLS	Alter a list's control information CFLEVEL=1 or higher: • Initialize additional control information. **IXLLSTC REQUEST=WRITE_LCONTROLS**
READ_EMCONTROLS	CFLEVEL=3 or higher: • Read control information about your registered monitoring interest in a particular sublist. **IXLLSTC REQUEST=READ_EMCONTROLS**
READ_EQCONTROLS	CFLEVEL=3 or higher: • Read control information about your event queue. **IXLLSTC REQUEST=READ_EQCONTROLS**
DEQ_EVENTQ	CFLEVEL=3 or higher: • Read and dequeue event monitor controls from your event queue. **IXLLSTC REQUEST=DEQ_EVENTQ**

Table 34. Summary of IXLLIST Macro Functions (continued)	
Function	**Action**
LOCK	Perform a lock operation on a lock table entry without performing any associated list entry operation. **IXLLSTC REQUEST=LOCK**
MONITOR_LIST	Start or stop monitoring the list transitions of a particular list. **IXLLSTC REQUEST=MONITOR_LIST**
MONITOR_SUBLIST	CFLEVEL=3 or higher: • Start or stop monitoring the transitions of a particular sublist. **IXLLSTC REQUEST=MONITOR_SUBLIST**
MONITOR_SUBLISTS	CFLEVEL=3 or higher: • Start monitoring the transitions of a set of sublists. **IXLLSTC REQUEST=MONITOR_SUBLISTS**
MONITOR_EVENTQ	CFLEVEL=3 or higher: • Start or stop monitoring your event queue for the presence of event monitor controls. **IXLLSTC REQUEST=MONITOR_EVENTQ**

Referencing List Entries

All list entries have list entry IDs, which are assigned by list services when list entries are created. In addition, list structures can support list entry names or list entry keys. The use of names or keys is optional but all list entries in a particular list structure must have entry names, entry keys, or neither. The terms are defined as follows:

Entry ID (ENTRYID)
An identifier permanently assigned to each list entry by the system. Each list entry ID is:

- Unique within the structure
- Used only once during the life of the structure

Entry name (ENTRYNAME)
A unique name permanently assigned to a list entry by its creator. List entry names:

- Must be unique within the structure
- Can be re-assigned to a different list entry when a list entry is deleted.

Entry key (ENTRYKEY)
A key value assigned to a list entry. Key values:

- Need not be unique within the structure
- Can be changed when the list entry is moved
- Can be assigned automatically when requested when the list structure is allocated in a coupling facility with CFLEVEL=1 or higher.

Within each list, keyed entries are ordered in hexadecimal collating sequence by key. Keys can be any 16-byte value. Because entries with the same key are maintained consecutively on a list, you could create a sublist (two or more contiguous list entries on a particular list) of list entries with the same key. List entries in a sublist of entries with the same key have special referencing requirements which are covered later.

Entry version number
A field associated with each list entry which you can use to maintain the list entry's version number. You can use the version number to:

- Indicate a change to a list entry's contents
- Select target list entries on some types of IXLLIST requests
- Implement a serialization mechanism (similar to compare and swap) that operates on a single list entry basis.

For list operations involving a single list entry (WRITE, READ, MOVE, DELETE) you can specify the target list entry in one or more of the following ways, depending on the request:

1. By list position on a specific list ("Specifying a List Entry by List Position" on page 491).
2. By list position and entry key on a specific list ("Specifying a List Entry by List Position and Key" on page 491).
3. By list cursor on a specific list ("Using the List Cursor" on page 500).
4. By list entry ID ("Specifying a List Entry by Entry ID" on page 506).
5. By list entry name ("Specifying a Named List Entry by Entry Name" on page 506).

Note: All methods of referencing list entries provide comparable performance.

For list operations involving multiple list entries (READ_LIST, READ_MULT, DELETE_ENTRYLIST, DELETE_MULT), you can specify the target list entries in one or a combination of the following ways, depending on the request:

1. By version number (targets all list entries in the structure that successfully complete a version number comparison operation)
2. By list number (targets all list entries on a particular list)
3. By providing a list of entry names or entry IDs as input to the IXLLIST request (targets the list entries you identify specifically).
4. By entry key (targets all list entries in the structure with a certain entry key value. Valid only for structures allocated in a CFLEVEL=1 or higher coupling facility.)

Combining different criteria, such as version number, list number, and key, gives you many ways to select list entries. These options are explained further under the requests to which they pertain.

Understanding List Entry Key Assignment

List services assign entry key values to list entries when an entry is created (WRITE) or moved (MOVE). You can have list services assign the list entry key on a WRITE or MOVE request, you can explicitly specify an entry key on a WRITE or MOVE request, or you can leave an existing entry key value unchanged. See "Creating a New List Entry" on page 533 and Figure 66 on page 552.

For structures that are allocated in a coupling facility with CFLEVEL=1 or higher, you can have a list entry key assigned automatically. The value of the key is derived from a list control associated with the list — LISTKEY, the list key value, and is limited by another list control — MAXLISTKEY, the maximum list key value, an upper boundary for the value. You can set these two values with a WRITE_LCONTROLS request and read them with a READ_LCONTROLS request.

List services use the list key value and the maximum list key value when automatically assigning an entry key. You specify on your WRITE or MOVE request with the LISTKEYTYPE keyword whether you want the entry key to be set only if, as a result of the request, an entry is created, is moved, or is either created or moved. Optionally, you also can specify that an increment (LISTKEYINC) is to be applied to the list key value after the entry key has been automatically assigned. If adding the increment to the list key value would result in a value that exceeds the maximum list key value, the list operation is suppressed and you receive notification of the failure with reason code IXLRSNCODEMAXLISTKEY. When such a failure occurs, you should first determine whether you specified an incorrect list key increment. Depending on the protocol you are using for assigning list key values, you might want to issue a WRITE_LCONTROLS request to update either the list key value to a lower value or the maximum list key value to a higher value.

Available Options for Automatic List Entry Key Assignment

For a structure allocated in a CFLEVEL=1 or higher coupling facility, you can use the LISTKEYTYPE keyword to specify how to use the list control list key value when assigning the list entry key. The LISTKEYTYPE values are:

NOLISTKEY
Do not use automatic entry key assignment from the list control value. If a key is to be used, explicitly assign it in the WRITE or MOVE request.

CREATE
Set the entry key to the list control list key value only if the entry is created as a result of the request.

MOVE
Set the entry key to the list control list key value only if the entry is moved as a result of the request.

ANY
Set the entry key to the list control list key value if the entry is either created or moved as a result of the request.

Specifying a List Entry by List Position

To designate a list entry by list position, specify the list number (LISTNUM) and the list position (LISTPOS).

List number (LISTNUM)
Designates a specific list in the list structure. The first connector to the list structure uses the LISTHEADERS parameter on the IXLCONN macro to specify the number of lists to be allocated in the list structure. List numbers in a list structure range from 0 to the LISTHEADERS value minus one.

List position (LISTPOS)
Designates the head or tail of list position. Possible values are HEAD or TAIL.

You can create a LIFO (last in, first out) or FIFO (first in, first out) queue or a stack by using LISTPOS to control how list entries are added to and removed from the list.

If there is only one list entry on the list, it is selected whether you specify LISTPOS=HEAD or LISTPOS=TAIL. Figure 41 on page 491 illustrates the use of LISTPOS to reference list entries.

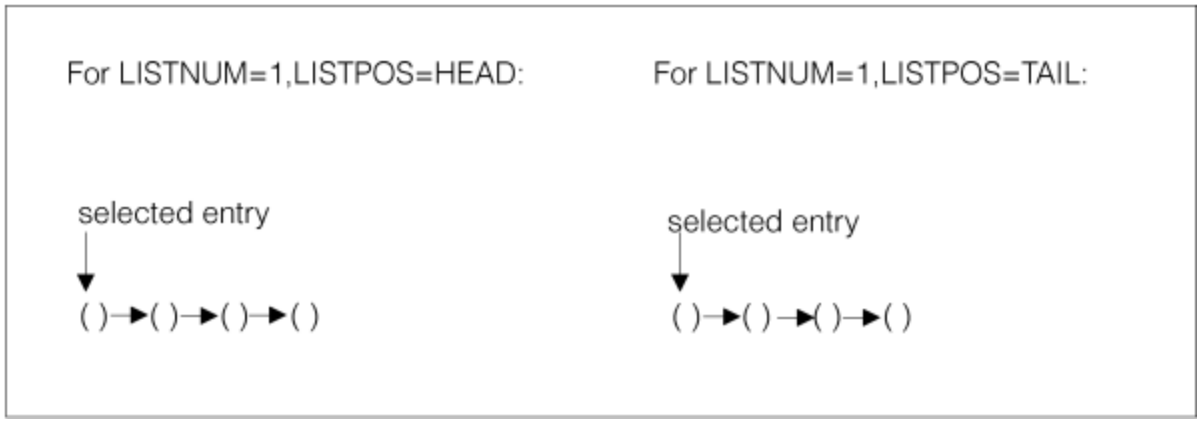


Figure 41. Use of List Position

Specifying a List Entry by List Position and Key

To designate a list entry by list position and key, specify the list number (LISTNUM), the list position (LISTPOS), and the entry key value (ENTRYKEY).

The optional KEYREQTYPE parameter allows you to indicate a range of acceptable key values for the target list entry. If a list entry with the key specified by ENTRYKEY does not exist, the value of the KEYREQTYPE parameter determines which list entry is selected. The KEYREQTYPE values are:

EQUAL
The target list entry's key must match the ENTRYKEY key. Specifying KEYREQTYPE=EQUAL is the same as specifying ENTRYKEY without KEYREQTYPE.

LESSOREQUAL
The target list entry's key must be less than or equal to the ENTRYKEY key.

GREATEROREQUAL
The target list entry's key must be greater than or equal to the ENTRYKEY key.

If you specify LESSOREQUAL or GREATEROREQUAL and there is no list entry whose key matches the ENTRYKEY key, the list entry is selected whose key is closest to the ENTRYKEY key.

For instance, if list entries represent work items and entry keys represent their priority (lowest=1, highest=5), you could select the list entry on the list with the highest priority by specifying KEYREQTYPE=LESSOREQUAL and ENTRYKEY=5:

- If there is a list entry with an entry key of 5, it will be selected
- If there are no list entries with an entry key of 5 but a list entry with an entry key of 4 is present, the list entry with entry key 4 would be selected.

If there is more than one list entry with the specified key, the value of the LISTPOS parameter determines whether the entry selected is at the head or tail of the sublist of list entries with the same key. Figure 42 on page 492 shows a sublist and illustrates how the LISTPOS parameter determines the list entry selected.

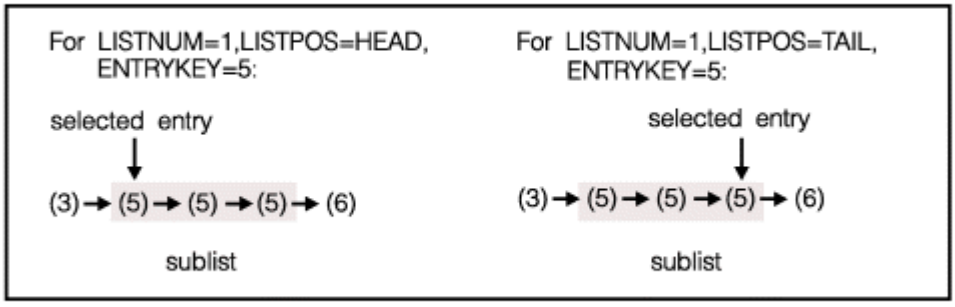


Figure 42. Use of List Position with Entry Key

If there is only one list entry with the specified key, it is selected whether you specify LISTPOS=HEAD or LISTPOS=TAIL.

In a sublist of list entries with the same key, only the first and last entries are accessible by list entry key because you can only request that an operation be performed on the head or tail list entry. A keyed list entry that is neither at the head nor the tail of the sublist cannot be referenced by entry key. Instead, you must use the list cursor or the list entry ID to reference it. Figure 43 on page 493 shows keyed list entries that cannot be referenced by list entry key:

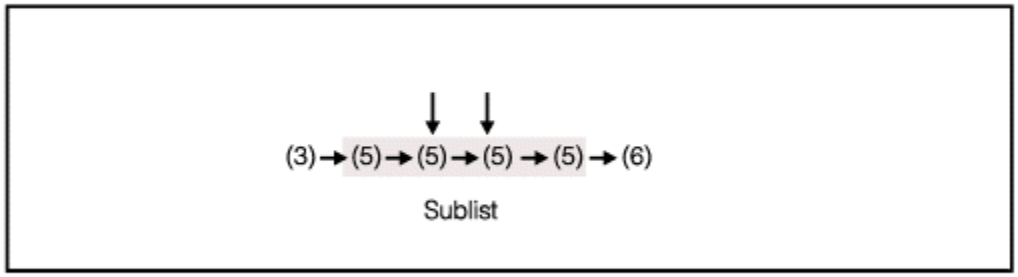


Figure 43. Example of Keyed List Entries that Cannot Be Referenced by Entry Key

If multiple list entries share the entry key specified by KEYREQTYPE, list services use the value of the LISTPOS parameter to determine whether to select the first or last list entry with that entry key:

- If LISTPOS=HEAD, list services select the first list entry with that entry key.
- If LISTPOS=TAIL, list services select the last list entry with that entry key.

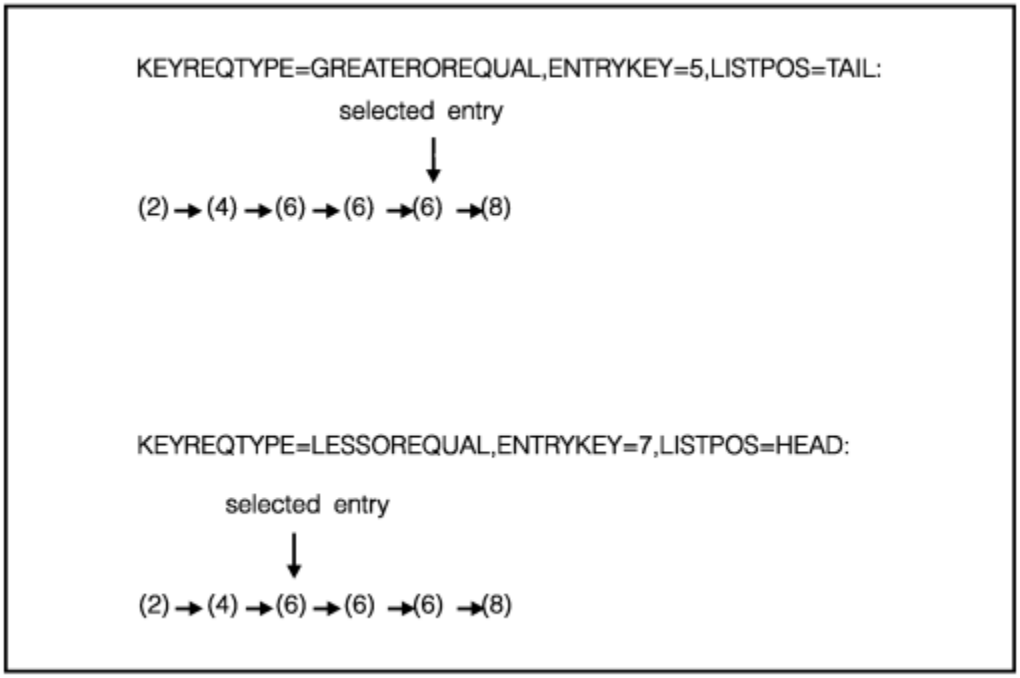


Figure 44. Use of KEYREQTYPE and LISTPOS Parameters

Using the Entry Key in Multiple List Operations

For a structure allocated in a CFLEVEL=1 or higher coupling facility, you can use the entry key to select entries for processing. The KEYCOMP keyword allows you to specify an entry key value with which the current list entry is to be compared. If the comparison fails (that is, the current list entry key does not equal the KEYCOMP value), then no processing is performed for the current entry and processing continues with the next entry to be considered.

Understanding the List Cursor

A list cursor is associated with each list. It acts as a pointer that you can move back and forth on the list. Its most natural use is to enable a set of users to cooperate in the processing of a list. For instance, if the list represented units of work to be performed, the list cursor could be used as follows. After initializing the list cursor to point to a list entry, users seeking work would:

- Read the entry pointed to by the list cursor (the next entry that needs processing) and move the list cursor to the next entry. (The list service performs these two actions atomically on a READ request with UPDATECURSOR=YES and CURSORUPDTYPE=NEXT or NEXTCOND.)
- Process the entry just read.

In this example, once the list cursor reaches the end of the list, the list service resets the list cursor to zero. When a list cursor points to a list entry, it contains the entry ID of that list entry.

For list structures allocated in a coupling facility with CFLEVEL=1 or higher, when you are running on an MVS SP 5.2 system with version one of the IXLLIST macro:

- There are additional cursor update options
- The list cursor update can be conditional, that is, the update occurs only if another condition is true at the time.
- The cursor can be made to point to the current entry instead of the previous or next entry.

Initializing the List Cursor

When you allocate the list structure, the list cursors are all set to zero. A list cursor must be initialized to point to a list entry before you can use it to designate an entry with LOCBYCURSOR. There are two ways to set the list cursor to point to a list entry before processing. One way is to use a WRITE_LCONTROLS request to initialize the cursor and set its direction. The second way to initialize a list cursor is to designate an entry and then use CURSORUPDTYPE to set the cursor to some location relative to the designated entry (either the entry itself or the previous or next entry).

- For list structures that are allocated in a coupling facility with CFLEVEL=1 (and SP 5.2 and IXLLIST version one), you can use the the WRITE_LCONTROLS request to initialize the list cursor. The SETCURSOR parameter of WRITE_LCONTROLS allows you to set both the cursor location and the cursor direction. The options available are:
  - Set the cursor to the first list entry on the list, and set the cursor direction to proceed in a head-to-tail direction.
  - Set the cursor to the last list entry on the list, and set the cursor direction to proceed in a tail-to-head direction.
- To initialize the list cursor to an identified list entry, issue an IXLLIST request, referencing the target list entry using another means such as ENTRYID, ENTRYNAME, or list position. Code UPDATECURSOR=YES on this request to initialize the list cursor for use on future requests. UPDATECURSOR=YES will set the list cursor to the list entry before or after the target entry or to the current entry depending on the value of CURSORUPDTYPE and the direction in which the cursor is progressing. The CURSORUPDTYPE parameter controls how the list cursor is to be updated. See “Controlling How the List Cursor Is Updated” on page 496.

Figure 45 on page 495 and Figure 46 on page 495 illustrate list cursor initialization. The IXLLIST invocation shown in step 2 in Figure 46 on page 495 is intended only as an example of an IXLLIST request that could be used to initialize the list cursor. You can initialize a list cursor using any of the IXLLIST requests involving a list entry operation.

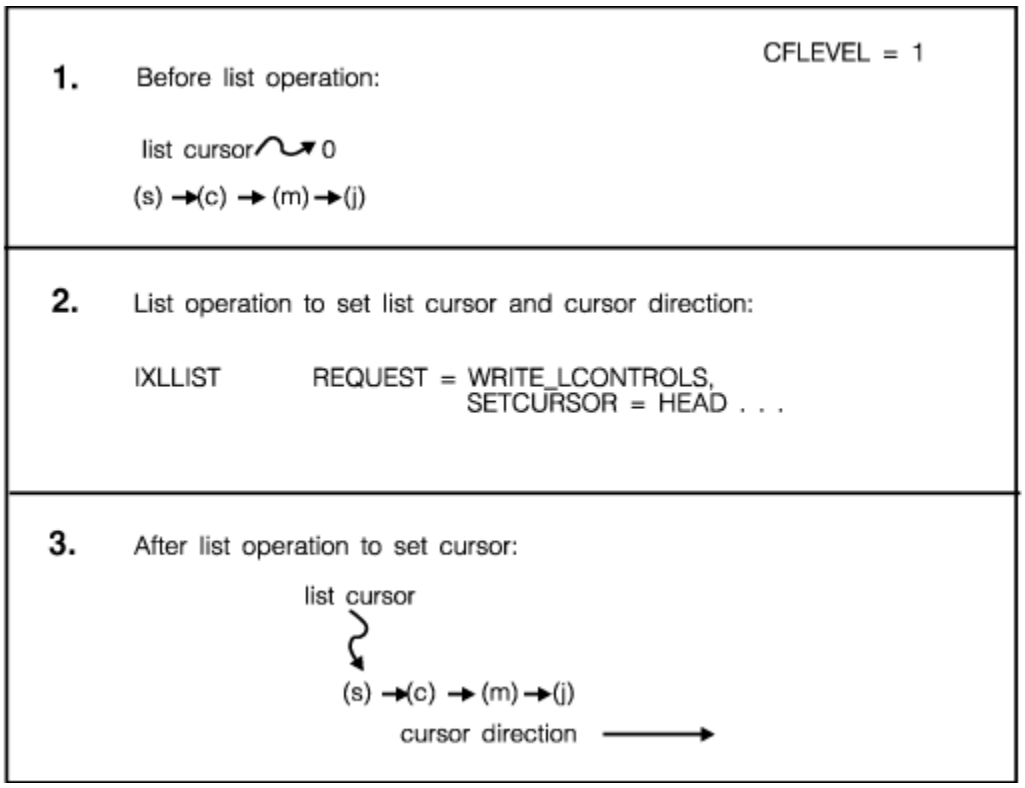


Figure 45. Initializing a List Cursor with an IXLLIST WRITE_CONTROLS Request

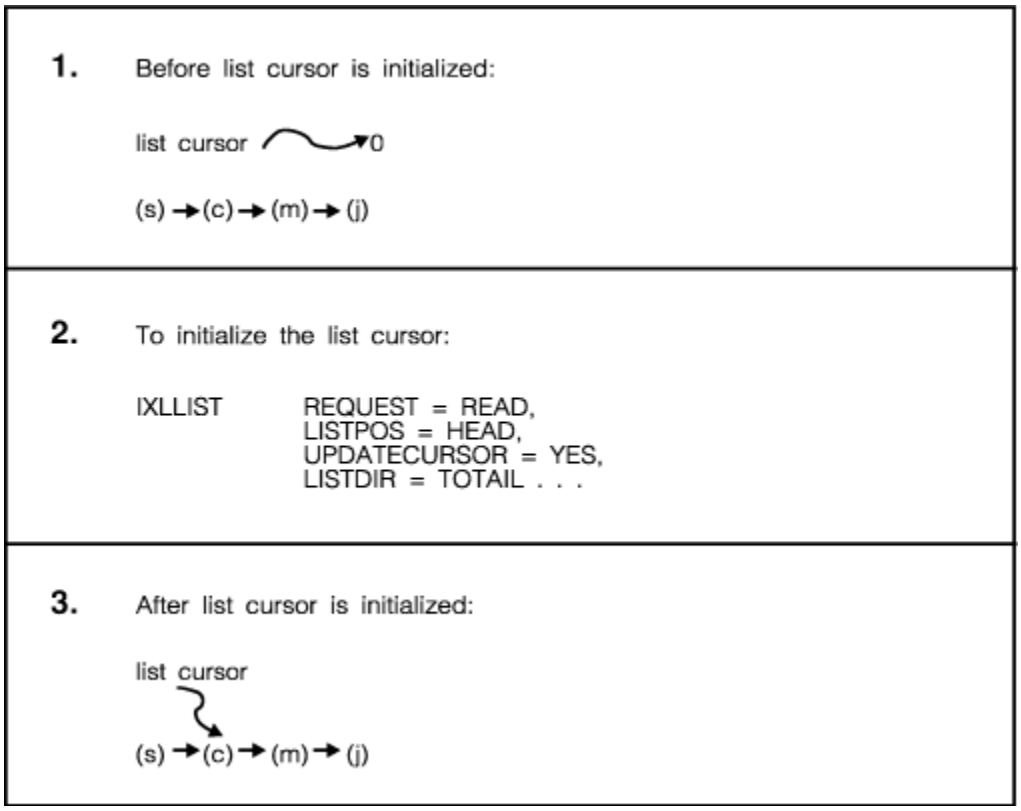


Figure 46. Initializing a List Cursor with Another IXLLIST Request

Controlling How the List Cursor Is Updated

The CURSORUPDTYPE parameter controls how the list cursor is to be updated when UPDATECURSOR=YES is specified. You have several options for controlling the cursor location, depending on the version of the IXLLIST macro you are using, the release of MVS on which your application is running, and the CFLEVEL of the coupling facility in which the structure is allocated. The options are:

NEXT

Update the list cursor to point to the list entry before or after the target entry. The direction of the cursor update depends on the cursor direction for the list, as specified by LISTDIR or LISTPOS, if LISTDIR is not specified. If the request is to create a new entry with a MOVE request, the cursor for the list identified by MOVETOLIST is updated in the direction indicated by the value of MOVETOPOS.

CURSORUPDTYPE=NEXT is the default; its processing is identical to the UPDATECURSOR=YES processing in version zero of the IXLLIST macro. You can use CURSORUPDTYPE=NEXT for structures allocated in a coupling facility of any CFLEVEL.

Figure 47 on page 496 illustrates the use of CURSORUPDTYPE=NEXT.

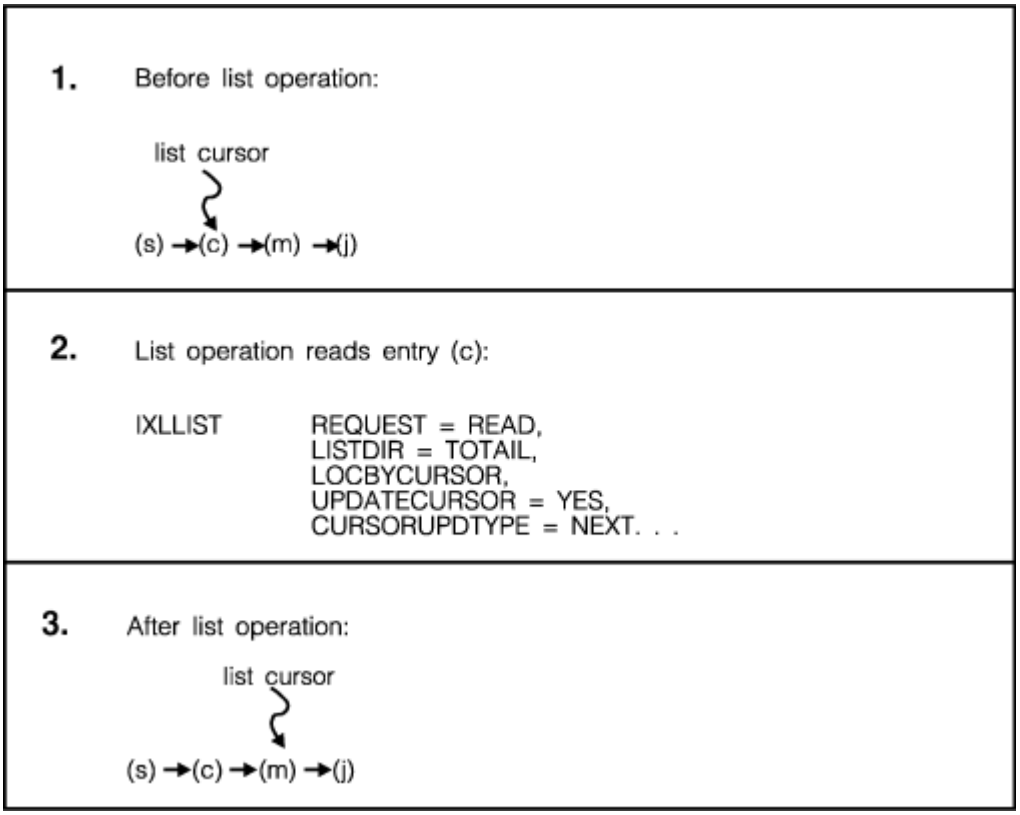


Figure 47. Updating the List Cursor to the Next Entry

NEXTCOND

Update the list cursor to point to the list entry before or after the target entry only if the list cursor points to the target entry and the entry is deleted or moved to another list. Otherwise, the list cursor is not updated. The direction of the cursor update depends on the list cursor direction (which can be set with the SETCURSOR parameter on a WRITE_LCONTROLS request).

If the entry is the last entry on the list and the list cursor direction is set in a head-to-tail direction, or if the entry is the first entry on the list and the list cursor direction is set in a tail-to-head direction, then list services reset the list cursor to binary zeros.

You can use CURSORUPDTYPE=NEXTCOND only for structures allocated in a coupling facility of CFLEVEL=1 or higher with MVS SP 5.2 or above.

Figure 48 on page 497 and Figure 49 on page 497 illustrate the use of CURSORUPDTYPE=NEXTCOND.

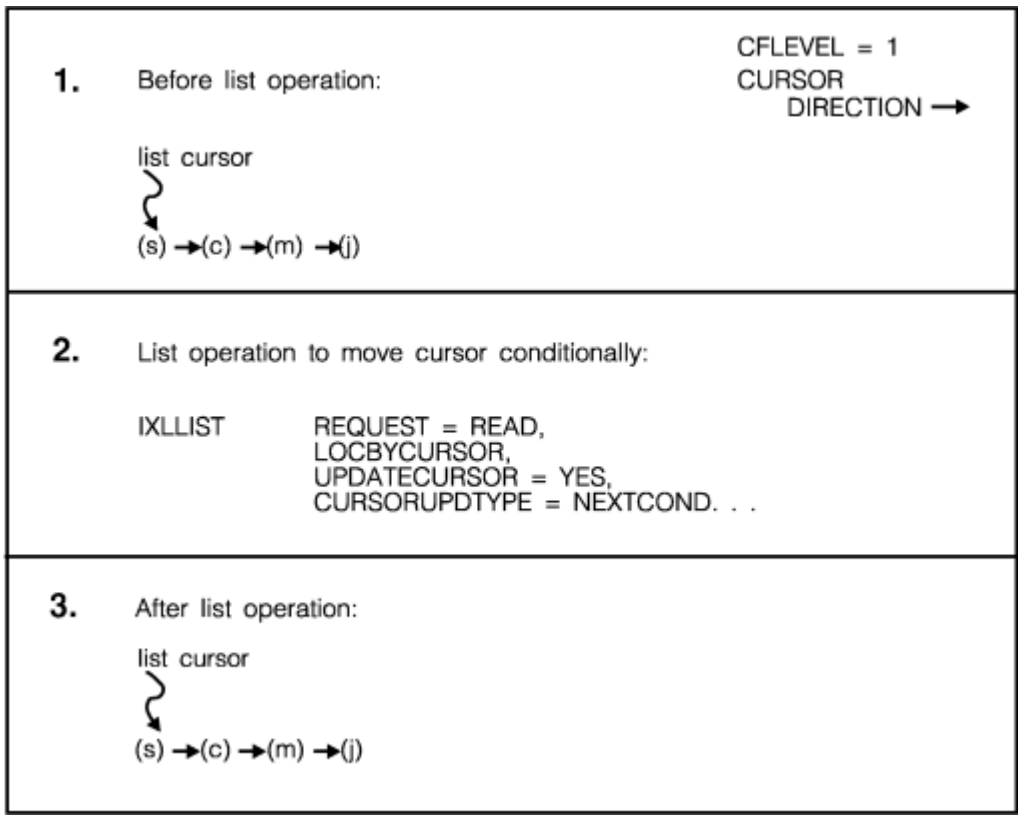


Figure 48. Updating the List Cursor Conditionally — Example 1

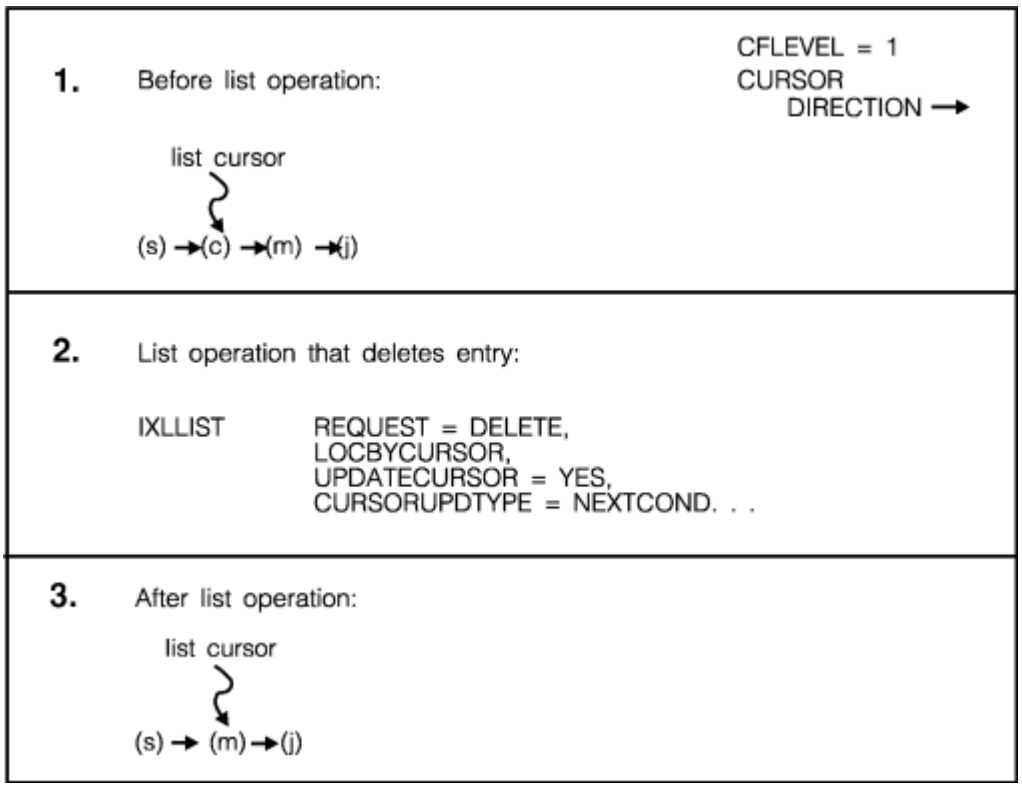


Figure 49. Updating the List Cursor Conditionally — Example 2

CURRENT
Update the list cursor to point to the target entry. If this request deletes the list entry or moves it to another list, the list cursor for the list is reset to zero.

You can use CURSORUPDTYPE=CURRENT only for structures allocated in a coupling facility of CFLEVEL=1 or higher with MVS SP 5.2 or above.

Figure 50 on page 498 and Figure 51 on page 498 illustrate the use of CURSORUPDTYPE=CURRENT.

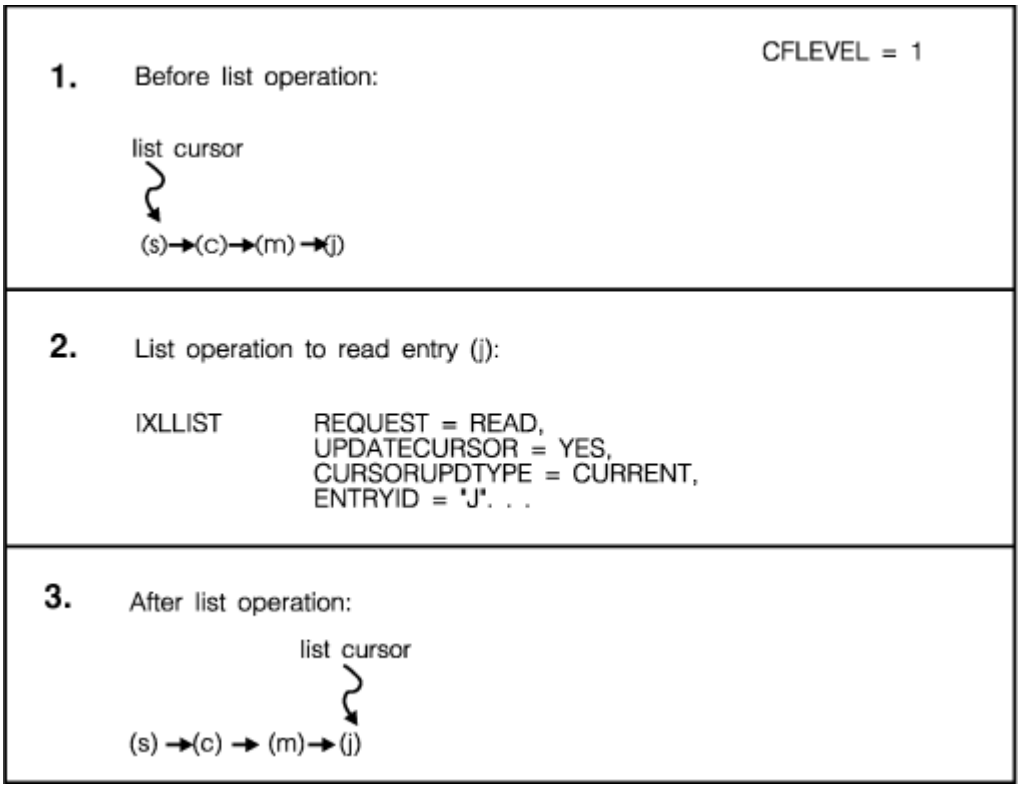


Figure 50. Updating the List Cursor to the Current Entry — Example 1

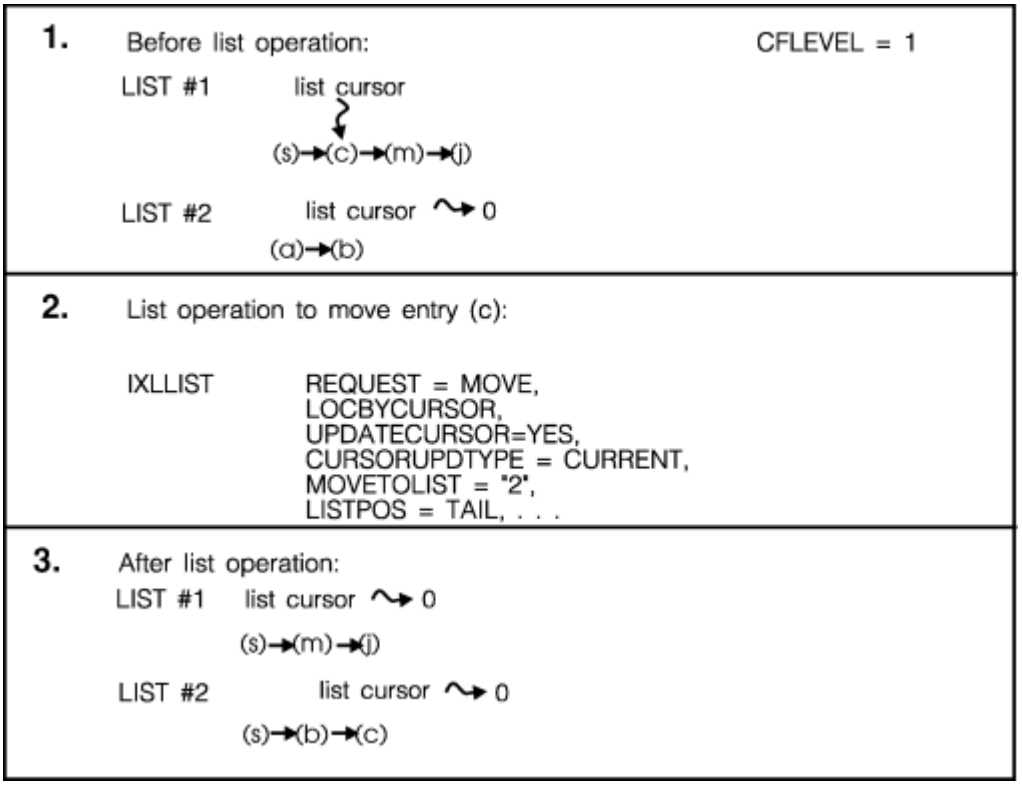


Figure 51. Updating the List Cursor to the Current Entry — Example 2

CURRENTCOND

Update the list cursor to point to the target entry only if the list cursor value currently is zero and this request is not deleting the target list entry or moving it to another list. If the request deletes the list entry or moves it to another list, the list cursor remains zero.

You can use CURSORUPDTYPE=CURRENTCOND only for structures allocated in a coupling facility of CFLEVEL=1 or higher with MVS SP 5.2 or above.

Figure 52 on page 499 and Figure 53 on page 500 illustrate the use of CURSORUPDTYPE=CURRENTCOND.

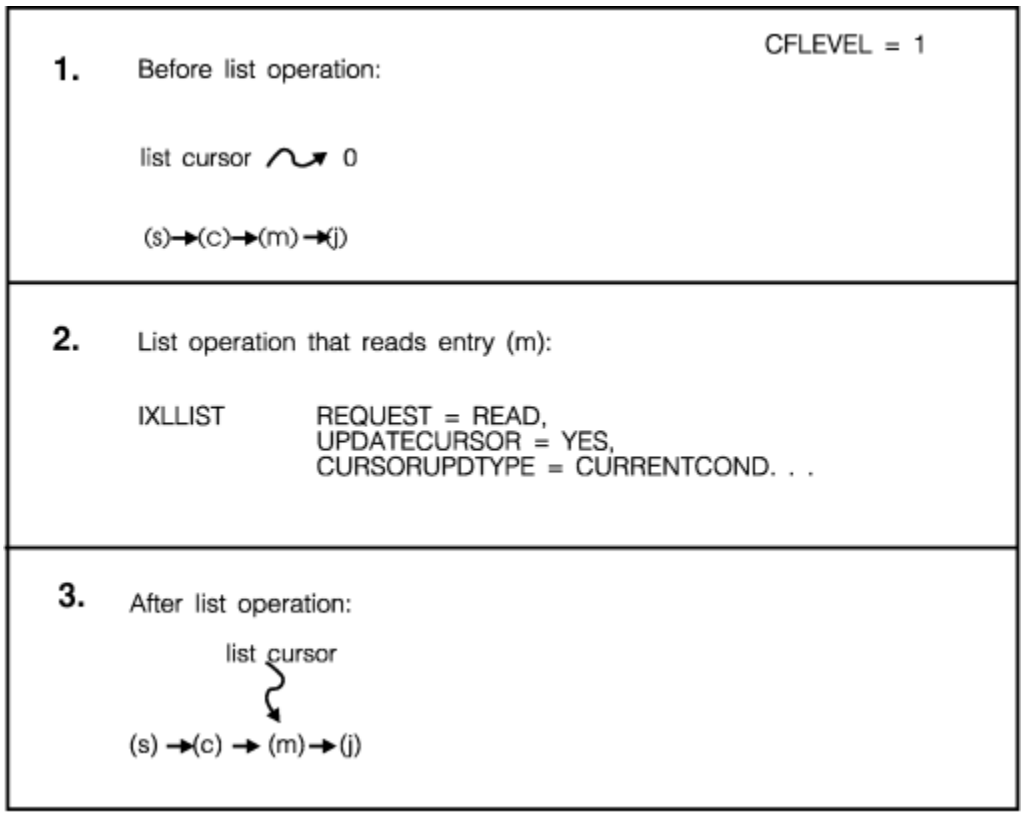


Figure 52. Conditionally Updating the List Cursor to the Current Entry — Example 1

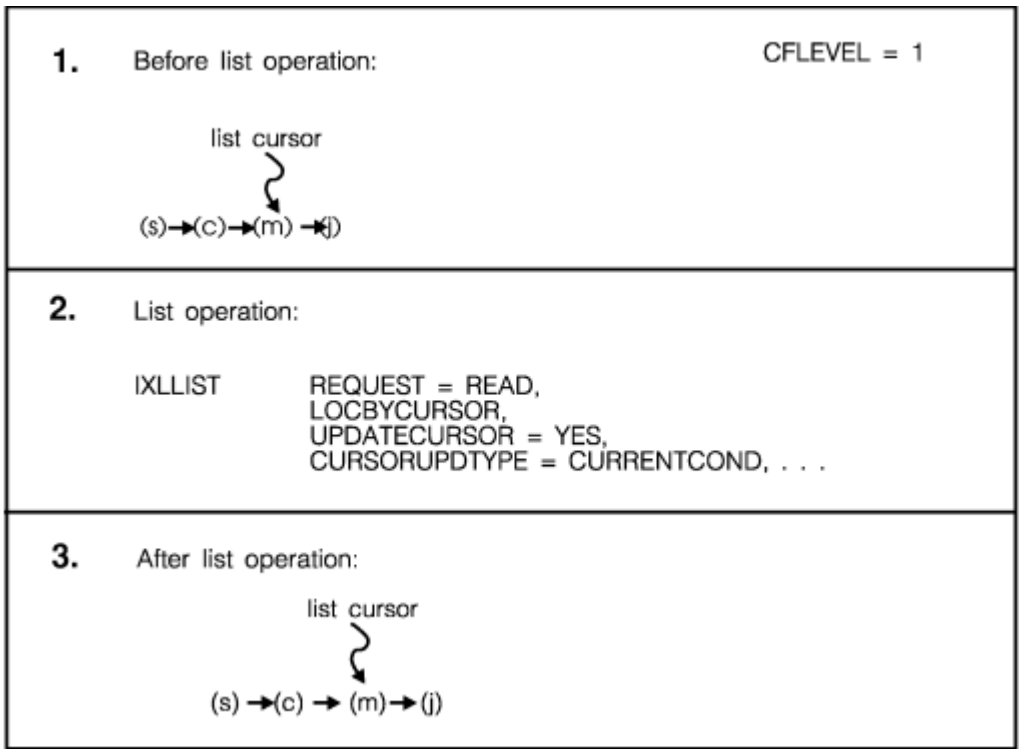


Figure 53. Conditionally Updating the List Cursor to the Current Entry — Example 2

Using the List Cursor

Once the list cursor is initialized, code the LOCBYCURSOR parameter to specify a target list entry using a list cursor. Issue the READ_LCONTROLS request for a particular list to determine the value to which its list cursor is set. The value of the list cursor is returned in the LAALISTCURSOR field of the answer area, an output area returned by IXLLIST.

For a structure allocated in a CFLEVEL=1 or higher coupling facility, for which you have set the cursor with a WRITE_LCONTROLS SETCURSOR request, you can use the READ_LCONTROLS request to determine the current value of the list cursor direction indicator. The value of the list cursor direction is returned in the LAACURSORDIR field of the list answer area. See "READ_LCONTROLS: Reading List Controls" on page 565 for more information.

- For version zero of IXLLIST (SP 5.1 and above), code the UPDATECURSOR=YES parameter to move the list cursor to the entry before or after the target entry, depending on the value of LISTDIR or LISTPOS.

  You can code the UPDATECURSOR=YES parameter without coding the LOCBYCURSOR parameter, so you can move the list cursor even if you don't specify the target list entry by list cursor. Figure 54 on page 501 illustrates this scenario.

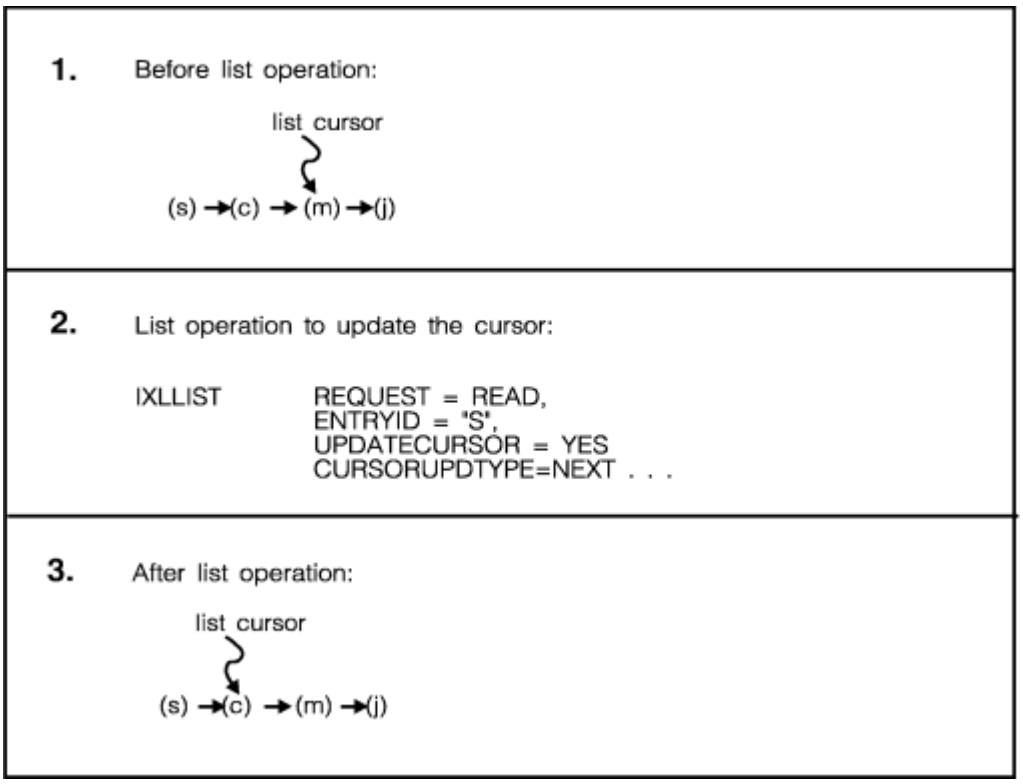


Figure 54. Updating the List Cursor without Using LOCBYCURSOR

- For version one of IXLLIST (SP 5.2 and above), code the CURSORUPDTYPE=NEXT parameter to move the list cursor to point to the list entry before or after the target entry, depending on the value of LISTDIR or LISTPOS. The CURSORUPDTYPE=NEXT option is identical to the UPDATECURSOR processing in IXLLIST version zero.

List services always move the list cursor **before** performing the list entry operation except when you request an operation that causes a new entry to be created. For example, if a WRITE request causes a new entry to be created, list services update the cursor for the list on which the new entry is created **after** the entry has been created. See Figure 55 on page 502. If a MOVE request causes a new entry to be created, list services updates the cursor for the list on which the new entry is created **after** the entry has been created. See Figure 56 on page 502.

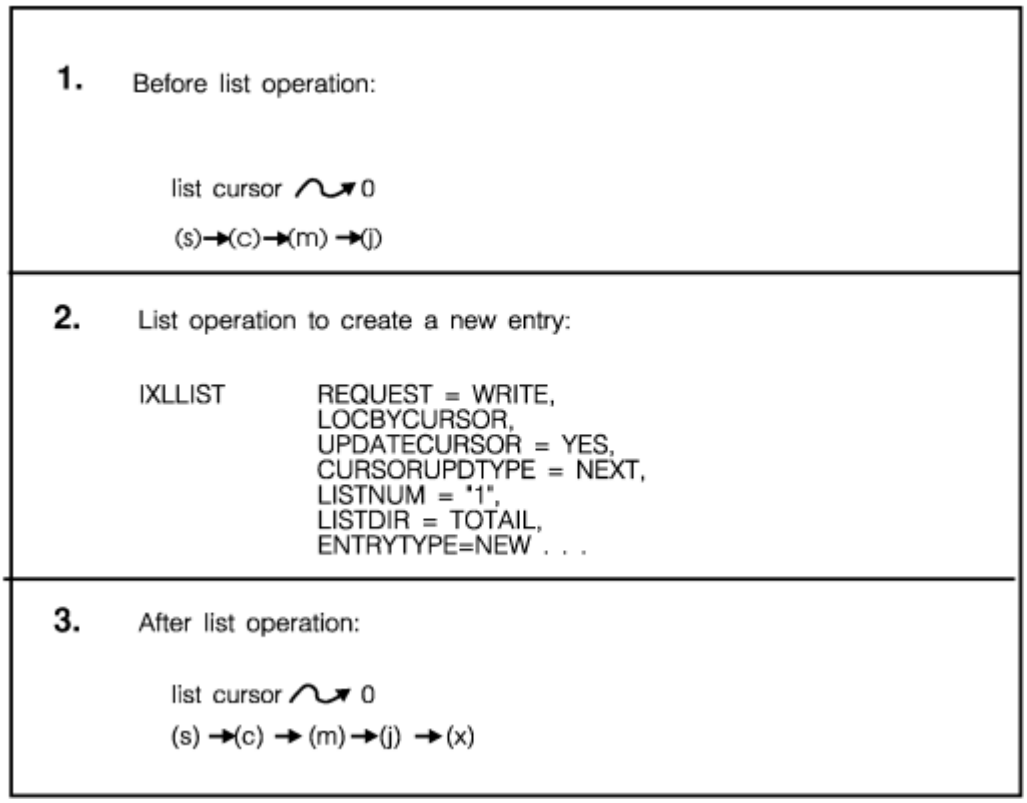


Figure 55. Updating the List Cursor when Creating an Entry with WRITE

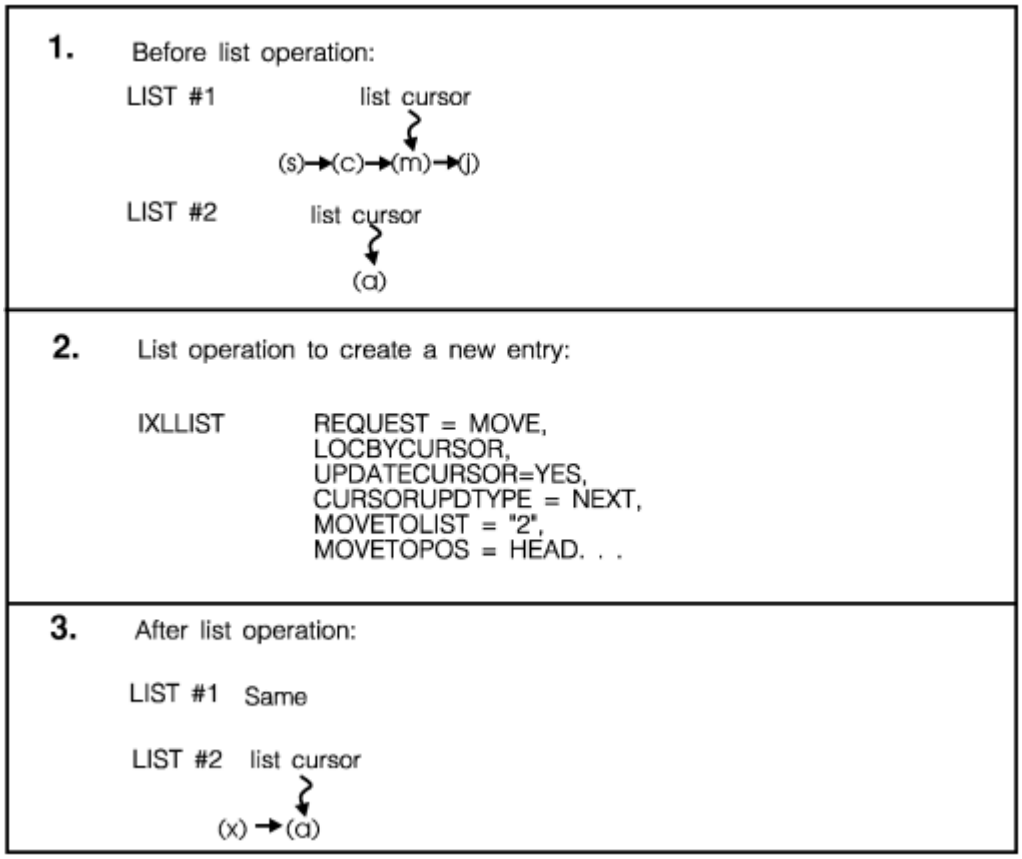


Figure 56. Updating the List Cursor when Creating an Entry with MOVE

If the list cursor is set to zero and you specify LOCBYCURSOR on your IXLLIST request, the result is an entry-not-found condition. If the request mandates that the target entry must exist, then your request fails. If the request indicates that an entry be created if an existing entry is not found, then a new entry is created.

Resetting the List Cursor to Zero

The following circumstances cause the list cursor to stop pointing to a valid list entry and get reset to zero:

- If you specify UPDATECURSOR=NO and the entry to which the list cursor points is deleted or moved to another list. Since the list cursor no longer points to a list entry on that list, the list cursor is reset to zero. Figure 57 on page 504 illustrates this scenario.
- If you specify UPDATECURSOR=YES with LISTDIR=TOHEAD and CURSORUPDTYPE=NEXT and the list cursor is already pointing to the head entry on the list. Since there is no entry before the head entry, the list cursor is reset to zero. Figure 58 on page 504 illustrates this scenario.
- If you specify UPDATECURSOR=YES with LISTDIR=TOTAIL and CURSORUPDTYPE=NEXT and the list cursor is already pointing to the tail entry on the list. Since there is no entry after the tail entry, the list cursor is reset to zero. Figure 59 on page 505 illustrates this scenario.
- For a structure allocated in a coupling facility of CFLEVEL=1 or higher,
  - If you specify CURSORUPDTYPE=NEXTCOND and the cursor direction is set in a tail-to-head direction and the list cursor is already pointing to the head entry on the list
  - If you specify CURSORUPDTYPE=NEXTCOND and the cursor direction is set in a head-to-tail direction and the list cursor is already pointing to the tail entry on the list.
- For a structure allocated in a coupling facility of CFLEVEL=1 or higher, if you specify CURSORUPDTYPE=CURRENT and the entry to which the list cursor points is deleted or moved to another list.

When the list cursor is reset to zero, you must re-initialize it as described above before using it again to designate an entry with LOCBYCURSOR.

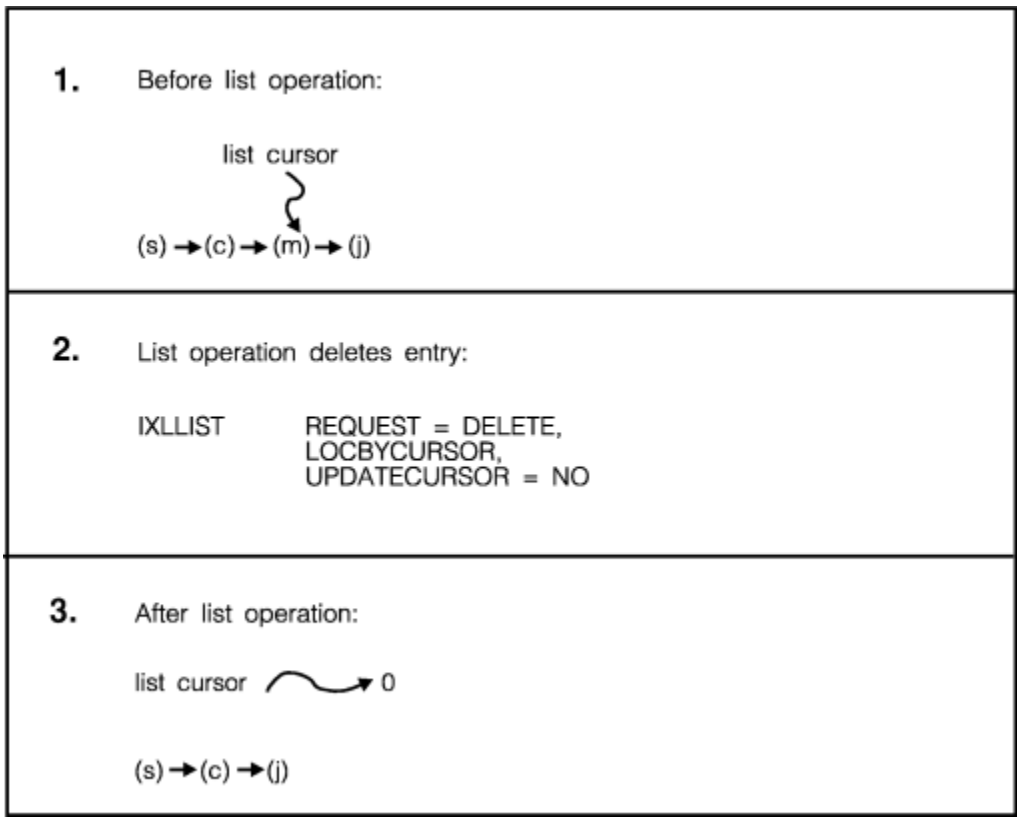


Figure 57. List Cursor After the List Entry is Deleted

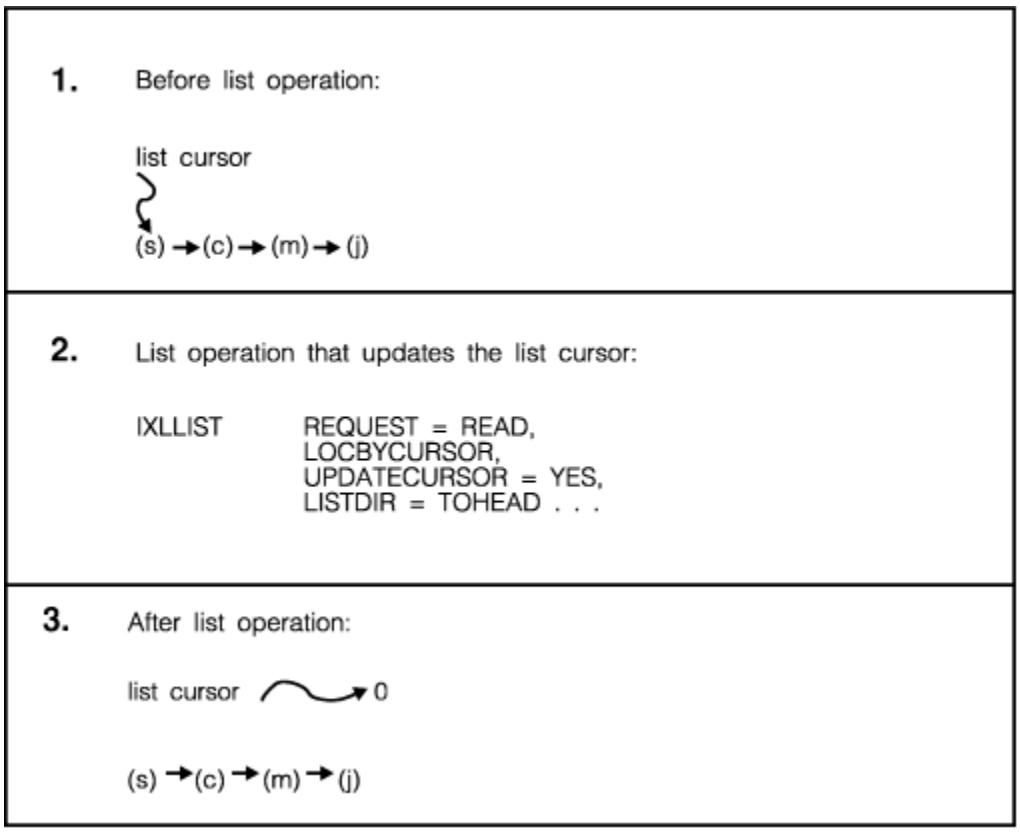


Figure 58. List Cursor When Moved Before the First List Entry

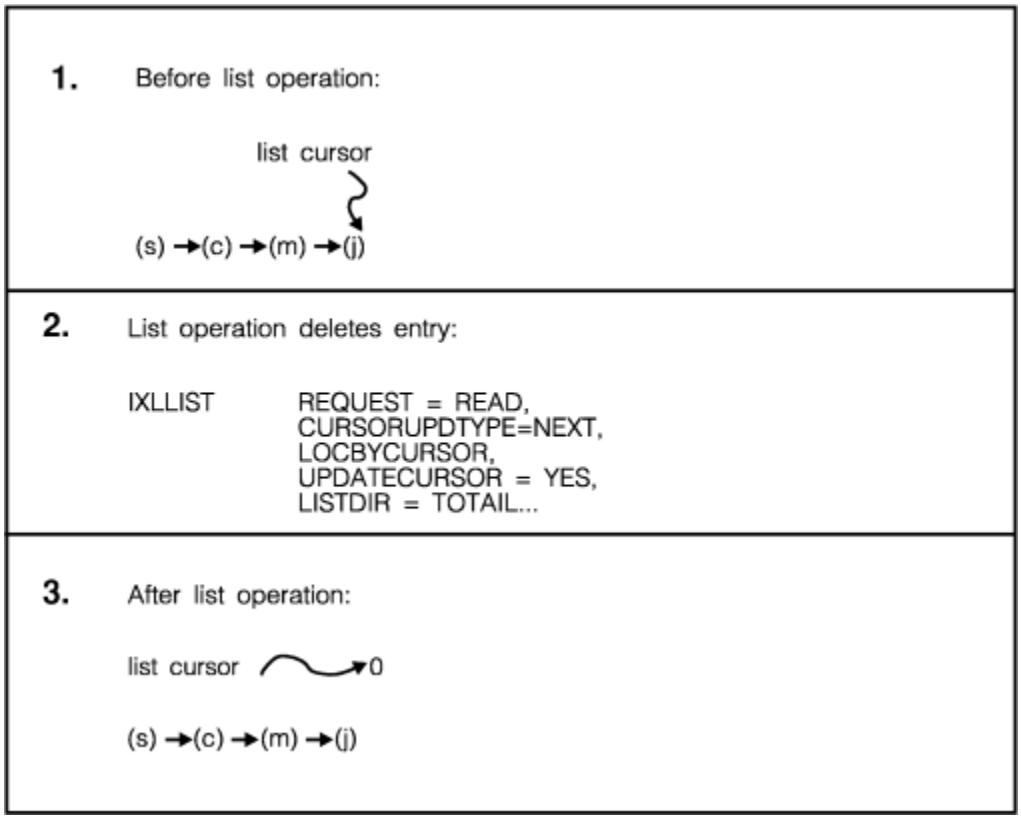


Figure 59. List Cursor When Moved After the Last List Entry

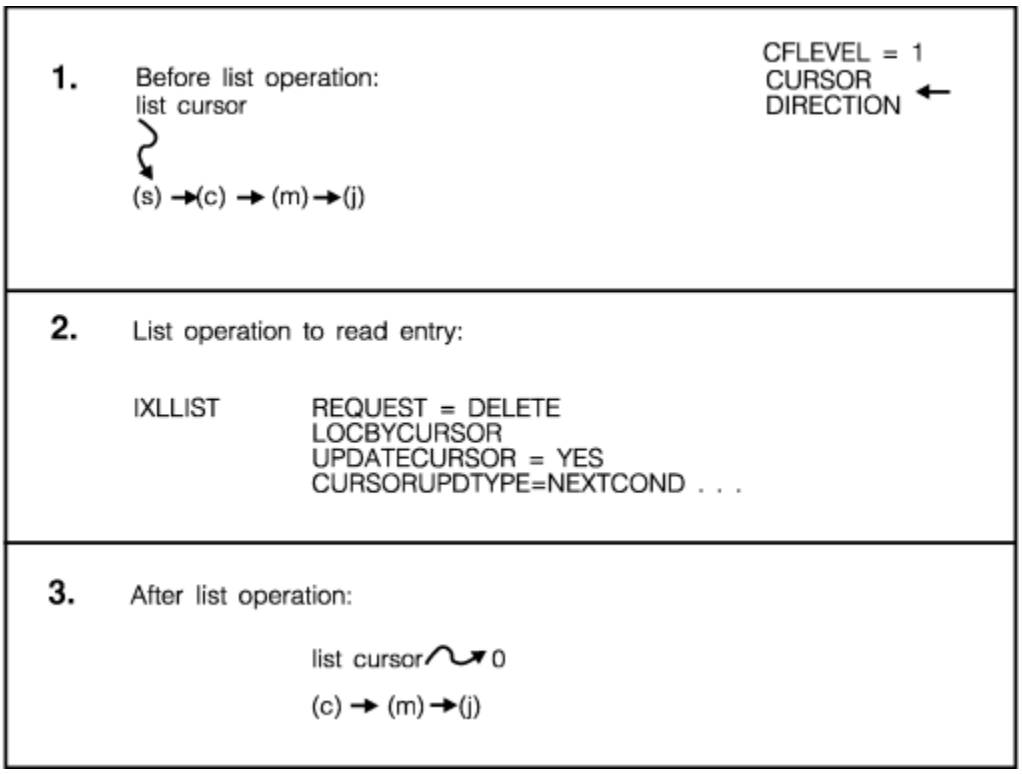


Figure 60. List Cursor When Moved Conditionally Before First Entry

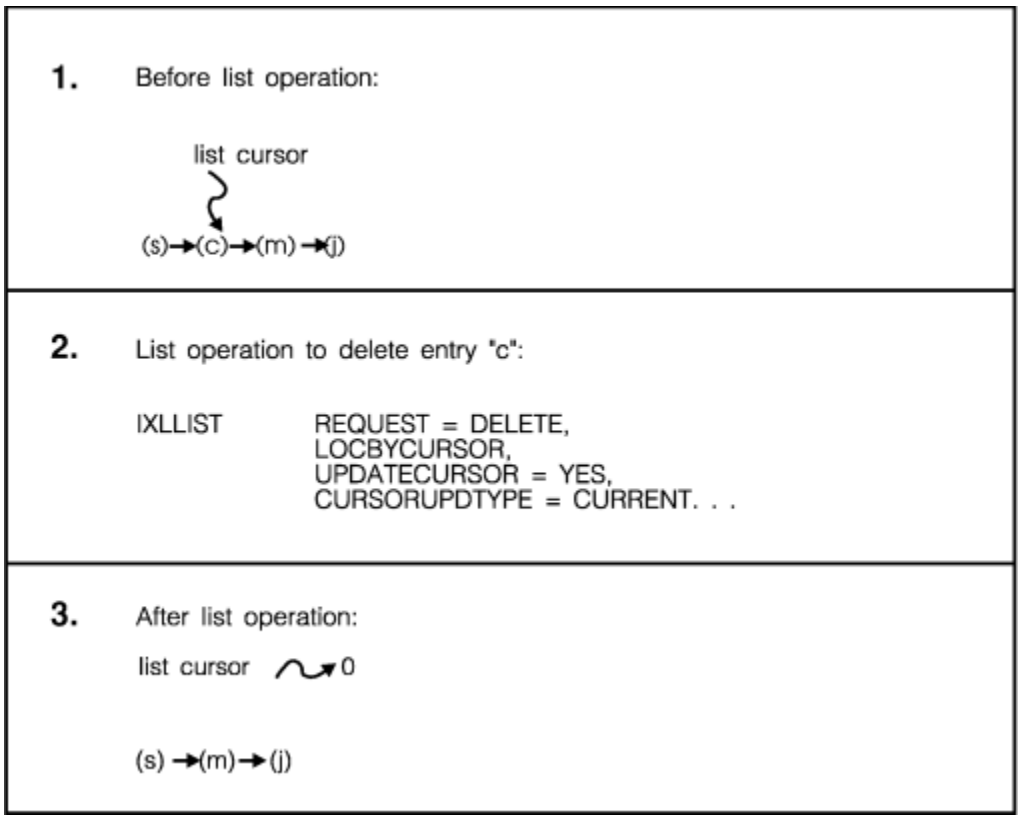


Figure 61. List Cursor When List Entry Is Deleted

Specifying a List Entry by Entry ID

To designate a list entry by entry ID, specify the entry ID (ENTRYID). The entry ID, which is assigned by the system when a list entry is created, is one of the list entry controls returned in the answer area for certain requests such as READ, WRITE, MOVE, and DELETE. The description of each request contains a section describing the answer area information returned for that request. Refer to the answer area information for each request to determine whether a list entry ID is returned.

Specifying a Named List Entry by Entry Name

To designate a named list entry by entry name, specify the entry name (ENTRYNAME). The entry name, which is assigned by the creator of the list entry, is one of the list entry controls returned in the answer area for certain requests such as READ, WRITE, MOVE, and DELETE. The description of each request contains a section describing the answer area information returned for that request. Refer to the answer area information for each request to determine whether a list entry name is returned.

Understanding List Structure Monitoring

Depending on the CFLEVEL of the coupling facility in which the list structure is allocated, the list structure monitoring functions allow you to determine whether a particular list or event queue is **empty** (contains no entries) or **nonempty** (contains one or more entries). The monitoring functions do not incur the overhead of accessing the coupling facility. Instead, the system maintains list or event queue information in a list notification vector allocated in high-speed processor storage on your own system.

A change from empty to nonempty in a list or event queue within the list structure is called a list or event queue transition. Not only does the list structure monitoring function offer you a faster way to determine the state of a list or event queue, but it also offers the option of being informed of list and event queue transitions by means of a list transition exit.

- With a coupling facility of any CFLEVEL, you can monitor the transition of a list from empty to nonempty.

- With a coupling facility of CFLEVEL=3 or higher, you also can monitor the transition of an event queue from empty to nonempty. Monitoring an event queue is the method by which you can, indirectly, monitor sublists within a keyed list.

The List Notification Vector

When you connect to the list structure and indicate your interest in using the list structure monitoring function, the system allocates a list notification vector for your use and returns a token to you representing this vector. The list notification vector shows the state (empty or nonempty) of each list or event queue you are monitoring. Each connector to the list structure that indicates interest in list monitoring (by coding the VECTORLEN parameter on the IXLCONN macro) or event queue monitoring (by coding the VECTORLEN and EMCSTGPCT parameters on the IXLCONN macro) is allocated a list notification vector.

A list notification vector consists of an array of entries, each of which can be associated with a particular list header or with the user's event queue. The number of entries must be a multiple of 32. The assignment of particular vector entries to monitor particular lists or to monitor the user's event queue is under the user's control with the IXLLIST MONITOR_LIST and MONITOR_EVENTQ request types. Note that the user can change this monitoring assignment dynamically over time (so that at any given point in time, none, some, or all of the allocated vector entries might be actively in use for monitoring purposes). However, the user should take care to manage the assignment of monitoring to particular vector entries such that any given vector entry is never monitoring more than one thing at a time. In such a case, the results are unpredictable.

When a transition occurs for a monitored list or event queue, the system automatically updates the associated entry in the list notification vector to reflect the empty or nonempty state of the list or event queue. The IXLVECTR macro provides the interface to the list notification vector. To determine whether a list or event queue you are monitoring is empty or non-empty, invoke the IXLVECTR macro with either the TESTLISTSTATE or LTVECENTRIES parameter. You can use the IXLVECTR macro with the MODIFYVECTORSIZE parameter to change the size of your list notification vector, so you can, for instance, monitor more lists.

Note: An IXLVECTR with either the TESTLISTSTATE or LTVECENTRIES parameter can show as non-empty due to an over-indication of bits in a local cache vector or list notification vector. Over-indication is set by the hardware if communication with the coupling facility (CF) is interrupted. When communication with the coupling facility is interrupted, the system may not receive notification of a transition that has occurred for a monitored list or event queue. In this case, the system may over-indicate a transition for all monitored objects by adjusting bits in the list notification vector so that a subsequent IXLVECTR TESTLISTSTATE or LTVECENTRIES invocation returns a non-empty result. This ensures that connectors do not lose initiative to process any transitions that may have occurred during the interruption of connectivity.

See “Using the IXLVECTR Macro” on page 680 for more information.

Options for Detecting a List or Event Queue Transition

You can detect list or event queue transitions two different ways:

- By having your list notification exit receive control when the list or event queue changes from empty to nonempty. Your list notification exit then invokes the IXLVECTR macro to check the state (empty or nonempty) of each list or event queue you are monitoring.
- By coding a polling routine to invoke the IXLVECTR macro periodically to check the state of each list or event queue you are monitoring.

For each list or event queue you monitor, you can choose how you want to detect list or event queue transition. You can monitor some using a list notification exit and others by whatever method you choose, such as polling the list notification vector.

Understanding the Event Queue

Within a keyed list structure allocated in a coupling facility of CFLEVEL=3 or higher, the system creates an event queue and an event queue controls object associated with each user. The event queue is created when the list structure is allocated with keyed list entries and is deleted when the list structure is deallocated. When you are monitoring an event (such as the state change of a sublist), the system queues or withdraws an event monitor controls (EMC) object to or from your associated event queue. For example, an EMC can be queued to your event queue when:

- An empty to nonempty state transition occurs for a monitored sublist.
- You register monitoring interest in a sublist at a time that the sublist is nonempty.

An EMC can be withdrawn from your event queue when:

- A nonempty to empty state transition occurs for a monitored sublist. In this case, the system returns the EMC to association with its sublist.

An EMC can be dequeued from your event queue when:

- You specifically request that the EMCs be retrieved and dequeued from the event queue. The EMC remains associated with its sublist.

An EMC can be deleted from the list structure when:

- You deregister monitoring interest in a sublist. In this case, the system discards the EMC.
- You disconnect from the structure or your connection terminates. The system deletes all EMCs associated with the connector.

The list services function uses the event queue for notifying a user that a state transition has occurred in one or more sublists that the user is monitoring. When a user registers interest in monitoring a sublist, list services creates an event monitor controls object (EMC) that associates and identifies both the user and the particular sublist. When the sublist transitions to a nonempty state, (or if the user registers interest in a sublist that is already in the nonempty state), the EMC is queued to the user's event queue. When the sublist transitions to an empty state, the EMC is withdrawn from the user's event queue but continues to be associated with the user and the monitored sublist.

By monitoring his event queue for the presence or absence of EMCs, the user is able to monitor one or more sublists in the structure. Each EMC uniquely identifies the sublist for which a transition has occurred.

Monitoring the Event Queue

The IXLLIST REQUEST=MONITOR_EVENTQ request allows you to start and stop monitoring your event queue for the presence of event monitor controls objects. To start monitoring the event queue, you must provide the list notification vector index that is associated with the event queue. List services uses the vector index to indicate whether the event queue is in the empty or nonempty state.

As with list monitoring, you can be notified about the transition of the event queue from empty to nonempty by having the system drive your list transition exit. You can also create your own polling protocol to poll the list notification vector to determine when a change has occurred.

Understanding Event Queue Controls

Event queue controls contain information about each event queue. Each user's event queue has its own set of event queue controls. The IXLLIST REQUEST=READ_EQCONTROLS request allows you to read your event queue's controls. Event queue control information includes the following:

- Vector index associated with the monitored event queue
- Number of event monitor controls that are currently queued to the event queue
- Approximate number of empty to nonempty event queue transitions that have occurred
- Indicator as to whether the user wants the list transition exit (identified by the LISTTRANEXIT keyword on IXLCONN) given control when the event queue transitions from empty to nonempty

- Indicator as to whether the user is currently monitoring the event queue

Understanding Event Monitor Controls

Information about a user and a designated sublist being monitored by the user is stored in an event monitor controls (EMC) object for the user. There can be at most one EMC per user per sublist being monitored. The information in an event monitor controls object includes the following:

- List number of the list with which the EMC is associated.
- List entry key of the sublist with which the EMC is associated.
- User notification control data either supplied by the connector when this EMC was established to monitor the sublist or updated by a subsequent MONITOR_SUBLIST or MONITOR_SUBLISTS request.
- Connection identifier of the user with which the EMC is associated.

The IXLLIST REQUEST=MONITOR_SUBLIST and REQUEST=MONITOR_SUBLISTS request types allow you to create EMCs and update their user notification control data.

There are two additional IXLLIST request types that allow you to reference EMCs:

- The IXLLIST REQUEST=READ_EMCONTROLS request allows you to determine if an EMC for a specific sublist is queued to your event queue. If the EMC exists, the system returns the EMC information, including the user notification controls data, in an answer area that you specify on the request. If the EMC does not exist, the system returns reason code IXLRSNCODENOENTRY.
- The IXLLIST REQUEST=DEQ_EVENTQ request allows you to atomically read the EMCs and dequeue them from the event queue with a single command. The system removes the EMCs from your event queue but maintains their association with the sublist(s) you are monitoring. The system returns the EMC information in a buffer area that you specify on the request. The system also returns a count of how many EMCs were read and dequeued from the event queue and a count of how many EMCs remain queued on the event queue.

Understanding Sublist Monitoring

Sublist monitoring differs from list or event queue monitoring in the way in which the user is notified of a change in the state of the sublist. While sublist monitoring is in effect, the system will queue or withdraw EMCs to or from your event queue to indicate the empty or nonempty state of the sublist. An event queue is present for each structure user when the structure is a keyed list structure that resides in a coupling facility with CFLEVEL=3 or higher. To determine whether a sublist transition has occurred, monitor your event queue for the presence or absence of EMCs.

With a keyed list structure allocated in a coupling facility with CFLEVEL=3 or higher, you can register interest in monitoring a single sublist within a list or multiple sublists within one or more lists.

- The IXLLIST REQUEST=MONITOR_SUBLIST request allows you to register or deregister interest in monitoring a **single sublist**. You identify the sublist to be monitored by list number and entry key. You can also specify 16 bytes of user data, called the user notification controls, to reside in the EMC.
- The IXLLIST REQUEST=MONITOR_SUBLISTS request allows you to register interest in monitoring **multiple sublists** (from 1 to 1024) with a single command. Information about the sublists in which you wish to register interest is stored in a buffer area that you specify. The information about each sublist is mapped by the macro IXLYMSRI and includes the following:
  - List number of the sublist to be monitored
  - List entry key of the sublist to be monitored
  - 16 bytes of user-defined data, called user notification controls, to reside in the EMC.

  On a MONITOR_SUBLISTS request you also must provide a storage area, called a MOSVECTOR, which is used to return the “monitored object state” for each sublist that was processed by the request. The monitored object state indicates whether a sublist was empty or nonempty at the time you registered monitoring interest; thus the MOSVECTOR provides you with information on the initial state of the sublists in which you've registered a monitoring interest.

Each bit in the MOSVECTOR area corresponds one-to-one with an IXLYMSRI entry in the input buffer for the request. Only those bits in the MOSVECTOR that correspond to IXLYMSRI entries that were actually processed by the current request are valid; all other bits in the MOSVECTOR are unpredictable.

A MONITOR_SUBLISTS request can complete prematurely for a variety of reasons, such as a model dependent timeout, an incorrectly-specified list number, or a lack of available event monitor controls. When this occurs, the user should handle the set of registrations that were performed on the current request (including observing the monitored object states in the MOSVECTOR area) before reissuing the MONITOR_SUBLISTS request to continue processing additional IXLYMSRI entries, because the second request will not return valid information for any entries other than those that are actually processed by the second request. On completion of the second request, the state of the MOSVECTOR bits corresponding to IXLYMSRI entries that were processed by the first request is unpredictable.

See "MONITOR_SUBLIST, MONITOR_SUBLISTS: Monitoring Sublists" on page 576 for a description of the MONITOR_SUBLIST and MONITOR_SUBLISTS functions of IXLLIST.

Once you have determined that one or more EMCs are queued to your event queue, you can issue an IXLLIST REQUEST=DEQ_EVENTQ request to read the information in the EMCs to identify the sublists that have transitioned from empty to nonempty.

Reviewing Sublist and Event Queue Monitoring

The following points outline the use of an event queue to accomplish sublist monitoring:

- A keyed list structure is allocated in a coupling facility with CFLEVEL=3 or higher. The connector specifies both a local vector and a percentage of storage for event monitor control objects.
- Connectors to the structure register interest in monitoring their event queues and specify the vector index to be associated with the event queue. (IXLLIST REQUEST=MONITOR_EVENTQ)
- When a connector registers interest in monitoring one or more sublists, the system creates an EMC to uniquely associate the user with each sublist. (IXLLIST REQUEST=MONITOR_SUBLIST, IXLLIST REQUEST=MONITOR_SUBLISTS)
- When a monitored sublist transitions to a nonempty state, an EMC is queued to the user's event queue. Users are notified either through their list transition exit or their own polling protocol.
- Users read EMCs from their event queues and examine the EMC contents to identify a monitored sublist that has transitioned. The operation that reads the EMCs also dequeues them from the event queue. (IXLLIST REQUEST=DEQ_EVENTQ)

You must be aware of certain timing considerations when monitoring state transitions of both a sublist and an event queue. Some examples are:

1. The queueing of an EMC to an event queue occurs asynchronously with respect to the command that caused the queueing to be performed. For example,
   - You add the first entry to a sublist that you are monitoring.
   - You read and dequeue the EMCs from your event queue.

   Result: The EMC for the sublist that just transitioned to a nonempty state might or might NOT have been queued to your event queue by the time the DEQ_EVENTQ command is processed. Thus the EMC representing the now-nonempty sublist might or might not be read by the DEQ_EVENTQ command.
2. The withdrawal of an EMC from an event queue occurs asynchronously with respect to the command that caused the withdrawal to be performed. For example,
   - You delete the last entry from a sublist that you are monitoring.
   - You read and dequeue the EMCs from your event queue.

   Result: The EMC for the sublist for which the last entry was deleted might or might NOT have been withdrawn from your event queue by the time the DEQ_EVENTQ command is processed. Thus the EMC representing the now-empty sublist might or might not be read by the DEQ_EVENTQ command.

3. The list notification signal that sets the state of the local vector entry that represents the empty or nonempty state of the event queue occurs asynchronously with respect to the state change of the event queue. For example,
   - You read and dequeue all EMCs from your event queue so that it is now empty.
   - You test the local vector entry with which you are monitoring your event queue.

   Result: The local vector entry might or might NOT indicate that the event queue is now empty.

In all cases, the coupling facility preserves the ordering of individual EMCs on the event queue and list notification signals for setting a given vector index.

- For any particular EMC, the coupling facility preserves the ordering of queueing and withdrawal processes, so that the final location of the EMC — either on or off the event queue — is always correct.
- For any given vector entry, the coupling facility preserves the ordering of list notification signals to set the vector entry, so that the final state of the vector entry — either empty or nonempty — is always correct.

Understanding List Entry Controls

Information relating to the list entry is stored in the list entry controls of each entry. List entry control information can include the following:

- List number
- List entry ID
- List entry name or list entry key, if applicable
- List entry version number
- List entry size

Most of the list entry controls listed above have already been discussed in “Referencing List Entries” on page 489. List entry size indicates the number of data elements that comprise the data entry.

Understanding List Controls

List controls, not to be confused with list entry controls, contain information relating to each list. Each list has its own set of list controls. The READ_LCONTROLS request allows you to read a list's controls. The WRITE_LCONTROLS request allows you to change the values of certain list controls. The remaining list controls are under the exclusive control of list services. Their values are updated as part of IXLLIST request processing.

List Controls That Can Be Updated Using WRITE_LCONTROLS

The following list controls can be updated using WRITE_LCONTROLS:

- The list limit, which can be either of the following:
  - The maximum number of list entries allowed on the list
  - The maximum number of data elements allowed on the list.

  The choice of limit type is specified using the LISTCNTLTYPE parameter on the IXLCONN macro when the structure is allocated. The initial value of the list limit for each list is the maximum number of list entries or data elements for the entire structure. So, in effect, you could place all list entries in the list structure on a single list.
- The list description. An optional, user-defined description of the list. The list description for each list is initialized to zeros when the structure is allocated.
- The list authority. Applications optionally can define a list authority value that must be specified when users update list controls. The list authority value for each list is initialized to zeros when the structure is allocated.

For a list structure allocated in a CFLEVEL=1 or higher coupling facility, several IXLLIST requests can be made conditional upon the success of a list authority comparison. Some IXLLIST requests can also update the list authority.

- The list key. LISTKEY specifies an optional list key value that is associated with the list. The list key can be assigned to a list entry automatically when a list entry is created or moved. Some IXLLIST requests can also update the list key value by specifying a list key increment. LISTKEY is valid only for structures in a coupling facility with CFLEVEL=1 or higher. See "Understanding List Entry Key Assignment" on page 490.
- The maximum list key. MAXLISTKEY specifies an optional list key value that provides an upper boundary for the list key. IXLLIST commands that specify automatic list key assignment can also increment the current list key value. When the maximum list key value is exceeded, the system will not automatically assign a list key to a list entry. MAXLISTKEY is valid only for structures in a coupling facility with CFLEVEL=1 or higher.
- The location of the list cursor and the list cursor direction. SETCURSOR is an optional method (for structures allocated in a CFLEVEL=1 or higher coupling facility) of setting the list cursor to the first list entry on the list with a list cursor direction of head-to-tail or to the last list entry on the list with a list cursor direction of tail-to-head.

List Controls That Cannot be Updated Using WRITE_LCONTROLS

The following list controls cannot be updated using WRITE_LCONTROLS. They are updated automatically by list services:

- The current number of list entries **or** data elements on the list (choice is determined by the value of LISTCNTLTYPE as described above). This field is initialized to zero when the structure is allocated.
- The approximate number of times the list has changed from empty to nonempty. This field is initialized to zero when the structure is allocated.
- The number of list monitoring information entries associated with the list. See "Obtaining List Monitoring Information" on page 565 for additional information about list monitoring information entries. This field is initialized to the model-dependent maximum number of connectors to the structure.
- For structures allocated in a CFLEVEL=0 coupling facility, the value of the list cursor (entry ID to which it points or zero). This field is initialized to zero when the structure is allocated.

List Controls That Can Be Updated Using READ, WRITE, MOVE, and DELETE

You can update the list authority using a READ, WRITE, MOVE, or DELETE request. Your update occurs only if you explicitly specify a list number (LISTNUM) and the request completes successfully.

You also can update the list key using a WRITE or MOVE request that specifies automatic key assignment.

Understanding List Counts

As discussed in "Deciding How to Limit the Storage Used by Each List", "Deciding How to Limit the Storage Used by Each List" on page 238 the LISTCNTLTYPE parameter allows you to choose how storage use is to be managed for individual lists. In addition to determining list related storage management, LISTCNTLTYPE determines how list counts are expressed when returned by list services.

The value specified for LISTCNTLTYPE on the IXLCONN macro when the list structure is allocated determines whether the LAALISTCNT, LAARLCLISTCNT and LAAMNL_LISTCNT fields in the list services answer area represent a count of list entries or data elements for the target list of the list service request. Additionally, list services may return opposite list counts in the list services answer area.

For example, if the value specified in LAALISTCNT represents the count of list entries on the list, the count of data elements on the list is returned in LAALISTOPPCNT. Note that opposite list counts are only valid for use when an opposite list count valid flag in the list services answer area is set.

The following tables summarizes the list counts returned by list services. Table 35 on page 513 shows the list counts returned for LISTCNTLTYPE = ENTRY. Table 36 on page 513 shows the list counts returned for LISTCNTLTYPE = ELEMENT.

Table 35. LISTCNTLTYPE = ENTRY when a list structure is allocated

List Services Request	Number of Entries on a list	Number of Elements on a list
READ, WRITE, MOVE or DELETE	LaaListCnt	LaaRlcListOppCnt when LaaRlcListOppCntValid is set ON
READ_LCONTROLS	LaaRlcListCnt	LaaRlcListOppCnt when LaaRlcListOppCntValid is set ON
MONITOR_LIST, MONITOR_KEYRANGE	LaaMnl_ListCnt	LaaMnl_ListOppCnt when LaaMnl_ListOppCntValid is set ON

Table 36. LISTCNTLTYPE = ELEMENT when a list structure is allocated

List Services Request	Number of Elements on a list	Number of Entries on a list
READ, WRITE, MOVE or DELETE	LaaListCnt	LaaRlcListOppCnt when LaaRlcListOppCntValid is set ON
READ_LCONTROLS	LaaRlcListCnt	LaaRlcListOppCnt when LaaRlcListOppCntValid is set ON
MONITOR_LIST, MONITOR_KEYRANGE	LaaMnl_ListCnt	LaaMnl_ListOppCnt when LaaMnl_ListOppCntValid is set ON

Understanding the List Authority Value

The list authority value provides a way of ensuring that only users authorized to do so issue certain requests for list services. You can use the list authority to select entries on a list for processing and for some requests you can update the list authority with a new value when the request completes successfully.

Using the List Authority Value to Select Entries for Processing

For structures allocated in a coupling facility with CFLEVEL=1 or higher, you can use the list authority value to provide conditional processing. For single-entry requests (READ, WRITE, MOVE, DELETE), processing can be made conditional on the success of a comparison between the current a list authority value that you specify in the request itself. You can specify that the comparison is to be either an equal operation or a less-than-or-equal operation. You must explicitly provide the list number as part of the request. If the list authority comparison is successful, the request is processed; if not, the system returns reason code IXLRSNCODEBADLISTAUTH to indicate why the request was not processed. The current list authority value is also returned in the list answer area.

For multiple-entry requests (READ_LIST, READ_MULT, DELETE_MULT, DELETE_ENTRYLIST), the same type of filtering can be used. If the list authority comparison is successful, the request is processed and continues with the next entry to be processed. If the comparison is not successful, the request is not processed and continues with the next entry in the list.

Updating the List Authority Value

For structures allocated in a coupling facility with CFLEVEL=1 or higher, you can update the list authority value for a list associated with an entry. On a single-entry request, you can specify a new list authority value (NEWAUTH) which will be used to update the current list authority value. You must explicitly provide the list number as part of the request. The update to the list authority value only occurs if the request is successful.

By adhering to a protocol of updating the list authority value when you update a list entry's contents, you can avoid corrupting or deleting changes made to the entry by other users. For instance, you could establish the following procedure for updating list entries:

1. Read a list entry.
2. Update its contents.
3. Increment, decrement, or set the list authority value of the updated copy of the list entry.
4. Write the changes back to the list entry using the AUTHCOMP parameter to ensure that the list entry is updated only if its list authority value is still the same as when you read it.

If the list authority comparison fails, the write request is not performed and you must start the update process again after re-reading the current list entry.

Understanding the User Exits

User-written exits play a critical role in the operation of many of the list structure services. Users provide their exit addresses when they issue the IXLCONN macro to connect to the list structure. The following exits are used with a list structure:

Complete exit
Informs users when their asynchronous requests have completed processing. See “Coding a Complete Exit” on page 583.

Notify exit
Informs users when list services detect contention for list structure locks they hold (see “Coding a Notify Exit” on page 585). If the structure includes a lock table, users must provide a notify exit to receive notification when they hold a lock for which there is contention. The notify exit can release the lock, take other actions to speed up the release of the lock, or ignore the notification. See “Coding a Notify Exit” on page 585.

List transition exit
Informs users when monitored lists or the user's monitored event queue changes from the empty state to the nonempty state. This is the only list structure exit that is optional. See “Coding a List Transition Exit” on page 588.

Understanding Synchronous and Asynchronous List Operations

You can request that your IXLLIST request be processed synchronously or asynchronously. In addition, for requests that run asynchronously, you can also choose the way you want to be notified of request completion. You select the type of processing and the method of request completion notification using a single parameter – the MODE Parameter. Table 37 on page 515 lists each MODE parameter option. However, before discussing the MODE parameter, an explanation of synchronous and asynchronous IXLLIST processing is necessary.

- **Synchronous processing**

  Synchronous processing of an IXLLIST request is defined as follows: your program regains control only when the IXLLIST request has completed processing. In certain cases, however, the system cannot process the IXLLIST request synchronously without suspending your program.

  If a synchronous IXLLIST request cannot be processed without suspending your program, IXLLIST either suspends your program or processes the request asynchronously. Your program is suspended only if you explicitly permit it by coding MODE=SYNCSUSPEND. Otherwise, even though you have requested synchronous processing, your request is processed asynchronously.

  The following circumstances cause MODE=SYNCSUSPEND requests to be suspended and other synchronous IXLLIST requests to be processed asynchronously:

  – The necessary resources for the request (such as a subchannel) are not currently available.
  – A dump of the structure is in progress.

– The system might also choose to convert synchronous requests to asynchronous processing, based on performance considerations or other criteria.

The system indicates its intention to process your synchronous request asynchronously by returning a return code of IXLRETCODEWARNING with a reason code of IXLRSNCODEASYNC when you issue the IXLLIST request.

- **Asynchronous processing**

  When the system processes a request asynchronously, your program regains control after issuing the request and the IXLLIST request runs independently. When your request runs asynchronously, you need a way to determine when it has completed processing. **All IXLLIST requests except those coded with MODE=SYNCSUSPEND could be processed asynchronously**. For non-SYNCSUSPEND requests, both synchronous (MODE=SYNCxxx) and asynchronous (MODE=ASYNCxxx), you must specify how you want to be informed of asynchronous request completion.

- **The MODE parameter**

  The MODE parameter options that specify synchronous processing have the format SYNCxxx, where xxx (except for SYNCSUSPEND) indicates the way you want to be informed of request completion if your request is processed asynchronously.

  The MODE parameter options that specify asynchronous processing have the format ASYNCxxx, where xxx indicates the way the system will inform you of request completion if your request is processed asynchronously. You can choose to have the system inform you of asynchronous request completion in any of the following ways:

  – Post an ECB (event control block):

    - MODE=SYNCECB
    - MODE=ASYNCECB

  – Return an asynchronous request token to be specified on the IXLFCOMP macro, which you invoke to obtain the results of the the IXLLIST request:

    - MODE=SYNCTOKEN
    - MODE=ASYNCTOKEN

    Issuing IXLLIST requests with MODE=SYNCTOKEN or MODE=ASYNCTOKEN enables you to issue multiple IXLLIST requests, continue with other work while the requests are being processed, and obtain request results at your convenience using the IXLFCOMP macro. See “Using the IXLFCOMP Macro with MODE=ASYNCTOKEN or MODE=SYNCTOKEN” on page 516 for more information.

  – Give control to your complete exit:

    - MODE=SYNCEXIT
    - MODE=ASYNCEXIT

  In addition, you can choose not to be informed of request completion by coding MODE=ASYNCNORESPONSE.

Table 37 on page 515 presents the options for IXLLIST request processing and asynchronous request completion notification:

Table 37. Options for IXLLIST Request Processing and Completion Notification

MODE Parameter Value	Actions Specified
SYNCECB	Attempt to process the request synchronously but if the request must be processed asynchronously, post an ECB to indicate request completion.
ASYNCECB	Process the request asynchronously and post an ECB to indicate request completion.

Table 37. Options for IXLLIST Request Processing and Completion Notification (continued)	
MODE Parameter Value	**Actions Specified**
SYNCTOKEN	Attempt to process the request synchronously but if the request must be processed asynchronously, return an asynchronous request token representing the request. To obtain request results, invoke the IXLFCOMP macro with the asynchronous request token you received. For more information, see "Using the IXLFCOMP Macro with MODE=ASYNCTOKEN or MODE=SYNCTOKEN" on page 516.
ASYNCTOKEN	Process the request asynchronously and return an asynchronous request token representing the request. To obtain request results, invoke the IXLFCOMP macro with the asynchronous request token you received. For more information, see "Using the IXLFCOMP Macro with MODE=ASYNCTOKEN or MODE=SYNCTOKEN" on page 516.
SYNCEXIT	Attempt to process the request synchronously but if the request must be processed asynchronously, give control to the complete exit when the request completes. For more information about the complete exit, see "Coding a Complete Exit" on page 583.
ASYNCEXIT	Process the request asynchronously and give control to the complete exit when the request completes.
SYNCSUSPEND	Process the request synchronously. If necessary, suspend the program until the request completes processing. Note that this is the only MODE option that could cause your program to be suspended. To use this option, your program must be enabled for I/O and external interrupts.
ASYNCNORESPONSE	Process the request asynchronously. Do not provide notification of request completion.

Using the IXLFCOMP Macro with MODE=ASYNCTOKEN or MODE=SYNCTOKEN

If you specify MODE=ASYNCTOKEN, or if you specify MODE=SYNCTOKEN and your request is processed asynchronously, you must invoke the IXLFCOMP macro to obtain the results of your IXLLIST request. You can use the IXLFCOMP macro either to determine whether your request has completed or to have your unit of work suspended until the request completes.

If the return code from IXLFCOMP indicates that your request has completed, the results are available in the output areas you have specified on the IXLLIST macro invocation.

For more information about the IXLFCOMP macro, see "Using the IXLFCOMP Macro" on page 679.

Understanding the Serialized List Structure

A serialized list structure is a list structure that contains a lock table. The lock table is an array of exclusive locks, whose purpose and scope are application-defined. Lock table locks can provide a serialization mechanism for lists, list entries, or any other list structure entity you designate. The first connector to the list structure specifies whether it is to be a serialized list structure, and if so, the number of lock entries to be allocated in the lock table. Figure 38 on page 483 shows a serialized list structure.

This topic will help you understand how to use the serialized list structure and how to design protocols to handle lock contention, recovery, and cleanup. Some of this information, as well as additional detail about the lock-related parameters is provided in "LOCK: Performing a Lock Operation" on page 568.

Overview of Locking Functions

IXLLIST offers a variety of specialized locking operations beyond the usual obtain, release, or test. Some of the unique locking functions include:

- Obtaining a lock only if it is held by a certain connection
- Releasing a lock only if it is held by a certain connection
- Performing a list entry operation only if the specified lock is not held
- Performing a list entry operation only if the specified lock is held by a certain connection ID
- Determining whether a lock is held by a specified connection ID
- Determining the lock table index of the next lock that is held, or held by a specified connection ID

Another key aspect of IXLLIST lock operations is that they can be performed together with or independently of list entry operations. For instance, in a single operation, you can obtain a lock for a list and update a list entry on that list. Alternatively, you can obtain the lock without performing the list entry operation. When you request a lock operation together with a list entry operation, the list entry operation is not performed unless the lock operation is successful.

Applications can use the locking functions provided by the serialized list structure in many different ways. Some examples:

- To serialize accesses to each list, an application can define a lock table with one lock per list. Having a lock for each list also enables users to serialize list operations involving multiple list entries.

  For instance, a user could obtain the lock, perform a READ_LIST while holding the lock, then release the lock. Having a lock for each list also allows an application to perform recovery actions on a single list basis.

  Users that perform operations on single list entries on a list can use the NOTHELD option to avoid interfering with users performing a series of list entry operations on the list while holding the lock. The NOTHELD option requests that a list entry operation be performed only if the specified lock is not held.
- To deny access to a list structure when performing recovery processing, an application can define a lock to serialize access to the list structure. List structure users can use the NOTHELD option to allow them to perform list operations only if the lock for the list structure is not held (and therefore recovery is not in progress.)

A lock can be in any of the following states:

- Held by a single user
- Held by the system
- Not held

When the system is transferring lock ownership from one user to another or performing other internal lock-related processing, the lock state is defined as **held by the system.**

Locks that are held by the system cannot be obtained or stolen. A reason code of IXLRSNCODELOCKHELDBYSYS is returned on any request you issue for a lock in this state except: unconditional SET or NOTHELD requests, which are just queued by the system until the lock operation can be processed (see “Understanding Lock Contention and the Notify Exit” on page 518 for more information.)

Table 38 on page 518 shows the IXLLIST locking functions. The lock operations (specified by the LOCKOPER parameter) perform different functions depending on whether you specify a **comparative lock value** using the LOCKCOMP parameter. The comparative lock value is a connection ID — your own or that of another connection. Users receive a connection ID when they issue the IXLCONN macro to connect to the list structure.

Table 38. List Structure Lock Operations		
Lock Operation	**With LOCKCOMP**	**Without LOCKCOMP**
SET	Transfer ownership of the lock to the requesting connection if the lock is currently held by the connection identified by LOCKCOMP (also known as lock stealing)	Obtain ownership of the specified lock
RESET	Free the specified lock if it is held by the connection identified by LOCKCOMP (another form of lock stealing)	Release ownership of the specified lock
NOTHELD	Not applicable.	Perform the specified list operation (such as a read or write operation) only if the specified lock is free
HELDBY	Perform the specified list operation (such as a read or write operation) only if the lock is held by the connection identified by LOCKCOMP	Perform the specified list operation (such as a read or write operation) only if the specified lock is held by the requesting connection
TEST	Determine whether the specified lock is held by the connection identified by LOCKCOMP	Determine whether the requesting connection holds the specified lock
READNEXT	Return the lock table index of the next lock held by the connection identified by LOCKCOMP	Return the lock table index and connection ID associated with the next lock in the lock table that is held.

Conditional and Unconditional Lock Requests

Lock requests can be conditional or unconditional. When you issue a conditional lock request (LOCKMODE=COND), your program regains control either with or without the lock request being satisfied. If the lock request could not be processed, your request simply fails.

When you issue an unconditional lock request (LOCKMODE=UNCOND), your program regains control only when the lock request has been processed successfully. If the request cannot be satisfied immediately, it is queued until it can be satisfied. Note that only the SET and NOTHELD requests give you an explicit choice of conditional or unconditional processing (using the LOCKMODE parameter.) Other requests might always be conditional, always unconditional, or either depending on the other parameters specified (for example, the RESET request is always conditional when LOCKCOMP is specified and always unconditional when LOCKCOMP is omitted.)

Understanding Lock Contention and the Notify Exit

An unconditional request for a lock that is held by another connection or by the system, causes a condition known as **contention**. A conditional request does not cause contention; the system simply fails the request and returns control to the calling program.

In a serialized list structure, there are only two cases where contention is created:

- A lock is held by a connection (or by the system) and another connection issues an unconditional SET request (without LOCKCOMP) for the lock.
- A lock is held by a connection (or by the system) and another connection issues an unconditional NOTHELD request for the lock.

Contention Processing

When your IXLLIST request causes contention, the system does the following:

1. Suspends your unit of work or processes your IXLLIST request asynchronously.
   - If you specified MODE=SYNCSUSPEND, the system suspends your unit of work until the lock is available and your request can be processed.
   - If you specified any other MODE value, the system processes your request asynchronously. You are informed of request completion by the method specified on the MODE parameter.
2. Queues your request on a sysplex-wide queue for the lock. Requests on each queue usually are processed in FIFO order. However, lock operations such as those issued in recovery processing for a failed connection, preempt the lock requests on the queue.
3. Gives control to the lock owner's notify exit to inform the connection of the lock contention (described in detail below).

The Notify Exit

A lock owner's notify exit receives control each time a new lock request is queued for the lock. A lock owner's notify exit also receives control when the lock owner has just obtained a lock and there are existing requests queued for that lock. In this case, the new owner's notify exit is immediately given control once for each of the pending lock requests. The intent in both cases is to give the lock owner information about the number of pending lock requests and the identity of each connection requesting the lock.

The notify exit can use the information provided to decide whether to release the lock, ignore the pending request, or take some other application-specific action. The notify exit can compare the current owner's importance to that of the pending request and respond accordingly.

The system supplies the notify exit with the following information each time it receives control:

- The index of the lock for which there is contention
- The current state of the lock
- The connection ID and connection name associated with the lock request causing the contention
- The lock request (SET or NOTHELD) causing the contention

Information presented to the notify exit is described in more detail under “Coding a Notify Exit” on page 585.

If the notify exit releases the lock, the lock becomes available to satisfy the first eligible lock request, which might not be the lock request that caused the notify exit to be given control. For instance, suppose there are five outstanding lock requests for a lock. The lock owner's notify exit receives control five times. On the fifth time, the notify exit releases the lock. If the lock request at the head of the queue were eligible to be processed, the lock would go to that connection.

Designing Protocols for Using the Serialized List Structure

The use of a serialized list structure requires a set of protocols for sharing locks. You should consider issues such as the following when you design your locking protocols:

- What list structure resource does each lock represent?
- How will I maintain information about the lock requests so my notify exit can make decisions to resolve lock contention?
- How will I manage multiple, asynchronous lock requests – both my own and those of other serialized list structure users?
- How will my notify exit decide how to handle lock contention?
- How will a lock be released if the owning connection cannot free it?

- How will my application handle recovery for work that was being performed by a failed connection?

Maintaining Information about the Lock Request

To help manage lock contention and facilitate the recovery of lock resources, you need to maintain lock ownership information such as:

- The identity of the lock owner
- The function being performed with the lock
- When the lock was obtained

Identifying the Lock Owner

When your program issues the IXLCONN macro to connect to the list structure, it becomes a list structure connector and acquires several types of identification:

Connection token (CONTOKEN)
A system-assigned token to be used on all subsequent list structure operations. You receive a new one each time you connect or reconnect.

Connection ID (CONID)
A system-assigned ID to identify your connection to other list structure connectors. You receive a connection ID each time you connect but you receive the same connection ID as you had last time if you reconnect.

Connection name (CONNAME)
A connection name to describe your connection. You can choose the name yourself using the IXLCONN macro, or have the system assign your connection a name.

While you receive a new connection token and a new connection ID every time you connect to the list structure, you can use the same connection name each time. Your connection name allows you to be recognized by other connectors as the same entity with a different connection token and connection ID.

If multiple programs in the same address space issue IXLLIST requests, they share the same connect token, connection ID, and connection name. In this case, you will need to use additional, non-connection-related identifiers to indicate the lock-owning program.

Distinguishing One Lock Request from Another

You need the ability to distinguish one lock request from another. For instance, there could be two lock requests, issued by the same connector, and specifying the same list operation and lock function. This might happen if your program issues IXLLIST requests on behalf of other programs. You need a way to distinguish between identical lock requests for the following reasons:

- To determine which lock request caused your complete exit to receive control (if you are using a complete exit):

  You can use the REQDATA parameter to pass information to identify the specific IXLLIST request to your complete exit.
- To allow your notify exit, when it receives control, to identify the owner of the lock for which there is contention:

  You can use the LOCKDATA parameter, specified with LOCKOPER=SET (obtain a lock), to associate 16 bytes of user-defined information with the lock when you obtain it. This information is presented to your notify exit when it receives control due to contention for a lock you hold. You could use the LOCKDATA parameter to:

  – Identify the program that owns the lock
  – Identify the lock request (possibly including a time stamp) that caused the lock to be obtained
  – Identify the user on whose behalf you are obtaining the lock

  - Pass the address of a shared control block containing information about each connection using the list structure, including information about each lock owner or lock requestor that could be used to resolve lock contention
- To provide recovery for a lock that is held by a failed or failing connection.

Managing Multiple, Asynchronous Lock Requests

Your program can issue multiple, asynchronous requests to obtain the same lock. Each request is processed independently. **The order in which requests are processed might differ from the order in which they are submitted**. Furthermore, due to the nature of asynchronous processing, the order of certain events could deviate from what you would expect. For instance:

- Your notify exit could receive control to inform you of contention for a lock you requested before you are informed that you have obtained the lock.

  Note: You own a lock you have requested only when you are informed (in the manner specified on your IXLLIST invocation) that your request has completed successfully. Unless you have received this confirmation, you cannot assume you hold the lock.
- Your notify exit could receive control to handle contention for a lock you no longer own. If you have, in the recent past, obtained and released the same lock you own currently, your notify exit could receive control due to contention arising from your previous instance of lock ownership.

To handle situations like these correctly, you should use the LOCKDATA and REQDATA parameters to pass any information your exits will require to determine if they need to take action. Your exits must also be prepared to handle cases, such as those listed above, where they receive control but need not take any action.

If you request a lock that you already hold (perhaps on behalf of a different user), your request is treated like any other user's request for that lock; it is placed on a queue behind any existing requests for that lock.

Important: A deadlock will occur if the unit of work responsible for releasing a lock is suspended while waiting to obtain the same lock.

Recovering Locks Held by Failed or Failing Connections

Confiscating a lock held by another user is called **lock stealing**. It is usually reserved for situations in which the owner is perceived to have failed or to be failing. **When a lock is stolen, its owner is not notified**.

You can steal a lock for either of the following reasons:

- To obtain it for yourself (LOCKOPER=SET with LOCKCOMP specifying the CONID of the current lock owner).
- To make it available again (LOCKOPER=RESET with LOCKCOMP specifying the CONID of the current lock owner).

When a lock is stolen, outstanding lock requests are unaffected; if you steal a lock and have other outstanding requests for the same lock, those requests remain queued and waiting to be processed even if you now have the lock. To cancel these requests, you must issue the IXLPURGE macro specifying the REQID of the request.

If you obtain a lock to serialize multiple IXLLIST requests and your protocol includes lock stealing, you should use LOCKOPER=HELDBY on each IXLLIST request once you hold the lock to ensure that the request is performed only if the lock is still yours.

Recovering Persistent Locks

When a persistent connector to a serialized list structure fails while holding locks, the system leaves the locks as **persistent locks** until they are cleaned up either by surviving peer connectors or by a new instance of the failed connector that reconnects.

Because persistent locks could be unavailable for a considerable amount of time, all requests for locks held by failed persistent connectors are automatically failed with a return code of IXLRETCODEPARMERROR and a reason code of IXLRSNCODEPERSISTENTLOCK. To obtain persistent locks, they must be stolen.

When your event exit receives control to inform you of a connection failure, you can determine whether the failed connection was persistent (specified CONDISP=KEEP on the IXLCONN macro) by checking the EEPLSUBJDISPOSITIONKEEP bit in the Event Exit Parameter List (EEPL). If you have a peer recovery protocol, you should clean up the locks held by the failed connection as follows:

1. Determine why the failed connector was holding the locks.
2. Perform any required clean up for the failed connection.
3. Steal locks from the failed connector as appropriate.
4. Provide an event exit response for the failure.

Note: After you and your peer connections have finished processing required by your own protocol, the system performs cleanup based on whether the failing connection is to be made failed-persistent or undefined. If you want to save or restore information or obtain locks, you must do so before the system begins its own recovery processing.

The system's lock recovery actions depend on:

- Whether the failed connection is persistent
- Whether RELEASECONN=YES (release the connection) was specified as an event exit response by any surviving peer connectors. See "Deleting Failed-Persistent Connections" on page 262 for more information about event exit responses relating to failed connections.

If RELEASECONN=NO is specified by all peer connections for a failed persistent connector, the system releases only locks associated with the failed connector that are held by the system. Locks held by the failed connector are considered persistent locks and are not released. All requests by surviving connectors for persistent locks are failed with a return code of IXLRETCODEPARMERROR and a reason code of IXLRSNCODEPERSISTENTLOCK. To obtain persistent locks, they must be stolen.

For all other cases (non-persistent connector or RELEASECONN=YES for persistent connector), the system releases both the locks held by the failed connector and the locks associated with the failed connector that are held by the system.

When you issue the IXLFORCE macro to delete a failed persistent connector, the system releases any persistent locks the connector holds at that time. The process of releasing the persistent locks continues after the IXLFORCE request completes; the only guarantee is that the persistent locks will be reset before the connection ID of the failed persistent connector is re-assigned to another connector.

Reconnecting with Persistent Locks

When a failed persistent connector reconnects to a serialized list structure, the locks previously held by the connector, which remained as persistent locks, are reassigned to the connector. When a lock becomes persistent, the system sets its LOCKDATA field to zero. Once the connector is reassigned its persistent locks, the locks are no longer persistent. They are ordinary locks subject to normal serialized list processing. The LOCKDATA value of zero identifies a lock as having been persistent.

When you reconnect to a serialized list structure and you might own persistent locks, you should perform recovery processing for the work you were doing at the time of the failure. When you are finished with this recovery processing, you should reset any locks you no longer need. To identify the locks you own, scan the lock table using LOCKOPER=READNEXT with a LOCKCOMP containing your connection ID.

Once a failed persistent connector reconnects to the list structure, the connector's notify exit will begin receiving control when contention occurs for locks held by that connector. When the notify exit receives control for contention involving a formerly persistent lock, the NEPLOWNERPERSISTENTLOCK bit in the notify exit parameter list (NEPL) is set to indicate that the LOCKDATA associated with the lock is not valid (the LOCKDATA field is set to zero because the lock became persistent).

Summary of Recovery Steps for Failed Connector to a Serialized List Structure

The previous sections described in detail the considerations involved in planning recovery actions for a failed connector to a serialized list structure. This section presents the key steps in time order to help you understand the sequence of events associated with the failure of a persistent connector:

1. A persistent connector fails while holding locks.
2. Peer connectors are notified of the failure through their event exits.
3. Peer connectors respond to this failure by performing recovery processing for the failed connector's work in progress and for locks held by the failed connector. Recovery could involve stealing locks held by the failed connector. Locks that are stolen from the failed connector by peer connections will not become persistent locks.
4. Peer connections provide an event exit response.
5. When all event exit responses are received by the system, it cleans up the failed connection.
6. If any peer connector indicated RELEASECONN=YES on its event exit response, the failed connector becomes undefined.

If the failed connector becomes failed persistent:

- The system releases all locks associated with the connector that are held by the system.
- All locks still held by the failed connector become persistent locks and have their LOCKDATA fields reset to zero.
- The connector becomes failed persistent.
- The system fails requests by surviving connectors to obtain the failed connector's persistent locks.
- The system honors requests by surviving connectors to steal the failed connector's persistent locks.
- When a new instance of the failed connector reconnects to the structure:
  - It is reassigned its persistent locks, which now have their LOCKDATA fields set to 0. The reassigned locks are no longer considered persistent when they are reassigned to the connector; they are now ordinary locks held by the connector, subject to normal serialized list processing.
  - Its notify exit can begin receiving control at once if there are requests for a lock held by the connector.
  - The connector should issue the IXLLIST macro with LOCKOPER=READNEXT to identify any reassigned (previously persistent) locks it holds, and take appropriate recovery actions to handle the work that was in progress at the time of the failure.
  - The connector should release the reassigned (previously persistent) locks once it has performed the recovery actions since the persistent locks and their resources have been cleaned up.

If the failed connector becomes undefined:

- The system releases all locks associated with the connector — those that are owned by the connector and those that are held by the system on the connector's behalf.
- The connector becomes undefined.
- There are no persistent locks since the connector is no longer persistent.

Understanding the List Entry Version Number

You can use the version number field associated with each list entry to indicate when the contents of the list entry have changed, to select list entries for certain types of IXLLIST requests, or to implement a serialization mechanism (similar to compare and swap) on a single list entry basis.

Setting the List Entry Version Number

The READ, WRITE, and MOVE requests allow you to set or change the version number of the target list entry by specifying the VERSUPDATE parameter. The version number can be:

- Assigned a particular value (VERSUPDATE=SET,NEWVERS=*newvers*)
- Incremented by one (VERSUPDATE=INC)
- Decremented by one (VERSUPDATE=DEC).

Note: When a list entry is created, its version number is set to zero. If you specify VERSUPDATE=INC or VERSUPDATE=DEC when you create a new list entry, the system uses zero as the value to be incremented or decremented.

Using the Version Number to Select List Entries for Processing

On READ, WRITE, MOVE, and DELETE requests, you can require the target list entry to compare successfully with a version number and type of comparison that you specify in order to be selected for processing. With structures allocated in a coupling facility with CFLEVEL=1 or higher, you can specify that a version number be equal or less-than-equal to a designated version number with the VERSCOMPTYPE keyword. If the version number for the target list entry does not meet the version comparison criteria you specify, the IXLLIST request fails.

On READ_LIST, READ_MULT, DELETE_MULT, and DELETE_ENTRYLIST requests, you can require that all selected list entries have a version number which compares successfully with a version number and type of comparison you specify. If the comparison fails, no processing is performed for the current list entry and processing continues with the next entry to be considered.

Using the Version Number to Serialize List Entry Operations

By adhering to a protocol of updating the version number when you update a list entry's contents, you can avoid corrupting or deleting changes made to the entry by other users. For instance, you could establish the following procedure for updating list entries:

1. Read a list entry
2. Update its contents
3. Increment, decrement, or set the version number of the updated copy of the list entry
4. Write the changes back to the list entry using the VERSCOMP parameter to ensure that the list entry is updated only if its version number is still the same as when you read it.

If the version number comparison fails, the write request is not performed and you must start the update process again after re-reading the current list entry.

Selecting the Buffer Format

Most IXLLIST requests require that you provide a buffer for one of the following reasons:

- To receive information read from list entries or list controls
- To hold information to be written to list entries or list controls
- To hold the names or IDs of list entries to be deleted

You can pass data or receive data using either a single buffer (BUFFER parameter) or multiple buffers (BUFLIST parameter). Both the BUFFER and BUFLIST parameters enable you to pass or receive up to 65536 (64K) bytes of data.

The parameters used to specify the buffers are discussed below . These include BUFFER, BUFLIST, and their associated parameters. Unless otherwise noted, this information applies to all IXLLIST requests. This topic provides an overview of the buffer formatting requirements and options. Additional information is presented in *z/OS MVS Programming: Sysplex Services Reference* under the parameter descriptions for each IXLLIST request.

There are also performance considerations for choosing the format of your buffers. These are discussed after the buffer options and parameters are presented.

BUFFER and Its Associated Parameters

BUFFER

The BUFFER parameter specifies a single contiguous buffer. It consists of a virtual storage area containing information to be passed to the request or received from a request. Only 31-bit addressable (below 2GB) virtual storage areas are supported for the BUFFER specification. The requirements for the storage area depend on the request.

- For READ, WRITE, MOVE, DELETE, and MONITOR_SUBLISTS requests, the storage area must meet the following requirements:

  For a buffer up to 4096 bytes in size, the buffer must:

  – Be 256, 512, 1024, 2048, or 4096 bytes
  – Start on a 256-byte boundary
  – Not cross a 4096-byte (page) boundary
  – Not start below storage address 512

  For a buffer greater than 4096 bytes in size, the buffer must:

  – Be a maximum of 65536 bytes
  – Be a multiple of 4096 bytes
  – Start on a 4096-byte boundary
  – Not start below storage address 512
- For DELETE_ENTRYLIST, READ_LIST, and READ_MULT requests, the buffer must:
  – Be between 4096 and 65536 bytes in size
  – Be a multiple of 4096 bytes
  – Start on a 4096-byte boundary
  – Not start below storage address 512
- For READ_LCONTROLS and DEQ_EVENTQ requests, the buffer must:
  – Be 4096 bytes
  – Start on a 4096-byte boundary
  – Not start below storage address 512

BUFSIZE

The BUFSIZE parameter, to be coded with BUFFER, specifies the size of the data buffer, in bytes.

Note that even though the BUFFER format does not support the BUFALET keyword, the BUFFER can still be ALET-qualified. If the caller is in AR mode, the IXLLIST macro extracts the AR associated with the BUFFER area and passes it on the request.

BUFLIST and Its Associated Parameters

BUFLIST

The BUFLIST parameter specifies the address of a storage area (the buffer list) that contains the addresses of up to 16 buffers. These buffers do not have to be contiguous, however, the system treats them as if they form a single buffer. Data is transferred to or from the set of buffers in order of ascending buffer number. The buffer list, shown in Figure 62 on page 526 and Figure 63 on page 526, has the following characteristics:

- The buffer list consists of a 128-byte storage area containing a list of 0 to 16 buffer addresses.

- Each entry in the buffer list consists of an 8-byte field in which either the high-order (left-most) 4 bytes are reserved and the low-order (right-most) 4 bytes contain the address of a buffer or the entire 8 bytes contain the address of a buffer.

  Note: Only the number of buffer list entries that you specify with the BUFNUM parameter must be formatted in this manner. For instance, if you specify a BUFNUM value of 5, all buffers beyond the fifth are ignored.

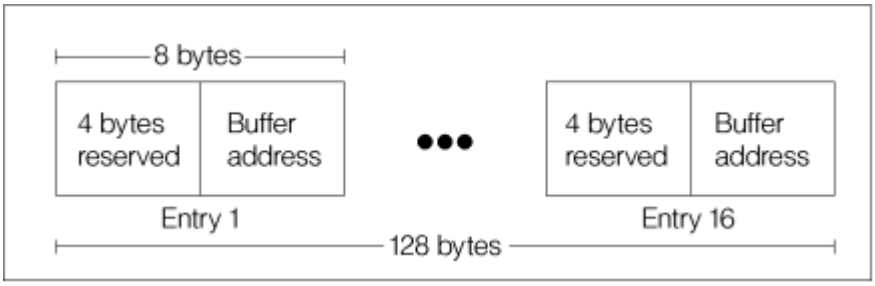


Figure 62. Format of Buffer List Specified by the BUFLIST Parameter

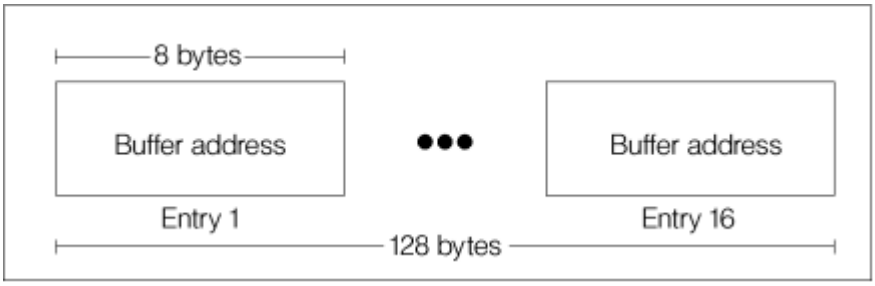


Figure 63. Format of Buffer List - 64-bit Addresses

All buffers in the buffer list must be the same size. Other requirements depend on the request:

- For READ, WRITE, MOVE, DELETE, and MONITOR_SUBLISTS requests, the buffers must:
  - Be 256, 512, 1024, 2048, or 4096 bytes
  - Start on a 256-byte boundary
  - Not cross a 4096-byte boundary
  - Not start below storage address 512
- For DELETE_ENTRYLIST, READ_LIST, READ_MULT, DEQ_EVENTQ, and READ_LCONTROLS requests, the buffers must:
  - Be 4096 bytes
  - Start on a 4096-byte boundary
  - Not start below storage address 512

Note: For READ_LCONTROLS and DEQ_EVENTQ requests, requests, only one buffer can be specified.

BUFALET
The BUFALET parameter specifies the ALET of each buffer in the buffer list. All the buffers must be in the same address or data space.

BUFNUM
The BUFNUM parameter indicates the number of buffers defined in the BUFLIST list. For READ_LCONTROLS and DEQ_EVENTQ requests, because the system allows only one buffer to be passed, you cannot specify the BUFNUM parameter. For all other requests, when BUFLIST is specified, BUFNUM is required.

BUFINCRNUM
The BUFINCRNUM parameter specifies the size of each BUFLIST buffer in 256-byte increments. Valid values are 1, 2, 4, 8, and 16. For example, a BUFINCRNUM value of 4 indicates that each buffer in the buffer list is 1024 bytes (4 * 256).

For DELETE_ENTRYLIST, READ_LIST, READ_MULT, READ_LCONTROLS, and DEQ_EVENTQ requests, the system requires your buffers to consist of sixteen 256-byte increments and you cannot specify the BUFINCRNUM parameter.

BUFADDRTYPE
The BUFADDRTYPE parameter specifies whether the buffer addresses are real addresses (BUFADDRTYPE=REAL) or virtual addresses (BUFADDRTYPE=VIRTUAL).

BUFADDRSIZE
The BUFADDRSIZE parameter specifies whether the BUFLIST entry address is a 31-bit (BUFADDRSIZE=31) or a 64-bit (BUFADDRSIZE=64) address.

Design Considerations for Choosing the Buffer Format

Choosing the buffer format and attributes involves a number of considerations. This topic helps you evaluate the available options and decide which ones are most suitable for you. The questions addressed are the following:

- How much buffer storage should I use?
- Should I use BUFFER or BUFLIST?
- If I use BUFLIST, how many buffers should I use?

Your buffer storage should be just sufficient to hold the data you are passing or receiving. If you are writing data to a data entry and you want to create a data entry with extra space for use later on, specify a greater number of data elements (ELEMNUM parameter) than necessary to hold your data. Specifying more data elements than your data requires does not affect performance.

The choice of whether to use a single buffer or multiple buffers depends on:

- Whether you are issuing IXLLIST multiple times
- Whether (if are performing a write operation) the data resides in contiguous storage
- Whether (if are performing a read operation) the data is to be placed in contiguous storage
- Whether your buffer addresses are real addresses or virtual addresses
- How concerned you are about performance.

When you specify a single buffer, IXLLIST creates a buffer list for that buffer in the same manner as if you specified BUFLIST. If you invoke IXLLIST multiple times, you obtain better performance if you create the buffer list yourself and use BUFLIST as opposed to using BUFFER and having IXLLIST build the buffer list on each invocation. On WRITE requests, using BUFLIST prevents you from having to move data from multiple storage areas into a single buffer before passing it to IXLLIST.

A single buffer less than or equal to 4096 bytes in size provides the best performance because if you specify more than 4096 bytes of buffer storage, or specify BUFLIST with more than one buffer, your request will always be processed asynchronously. (Note that requests are also processed asynchronously for other reasons such as unavailability of a required resource.)

If you choose to use multiple buffers, you must determine how many to use and what their size should be. You can achieve the best performance with multiple buffers if you use the fewest, largest buffers possible.

The buffer size need not equal the data element size, but if you find it useful, you can set it up that way. To create a buffer size equal to a structure element, specify the same value for BUFINCRNUM as you specified on the IXLCONN macro's ELEMINCRNUM parameter.

Design Considerations for Defining Buffer Storage Areas

The IXLLIST request types that allow you to specify buffer storage areas generally result in data being transferred directly between the data buffer storage and the coupling facility storage. The coupling facility

transfers data using real storage addresses; therefore, the data buffer storage must be fixed in a specific, known real storage location and remain so until the coupling facility has transferred all data for the request.

When defining the buffer storage areas for an IXLLIST request, consider the following:

- The cross-memory mode of your application
- The use of real versus virtual storage

The data buffers for an IXLLIST request can be addressable in the caller's primary, secondary, or home address space, from the PASN access list, or from the DU access list. The system assigns ownership of a data buffer to the address space either in which the buffer storage resides or that has an associated data space in which the buffer storage resides.

Determining Buffer Storage Ownership

XES always assumes that the storage for the data buffers is owned by the home address space (the “requestor's” or “client's” address space) at the time of the IXLLIST request. However, XES also allows the buffers to be owned by the primary address space (the “connector's” or “server's address space”) at the time of the request when the following conditions both exist:

- The connector's space is not equal to the requestor's home space
- The connector's space is non-swappable

Thus, the possible address space environments for your application are:

- Requestor (Home) equals Connector (Primary)
- Requestor (Home) does not equal Connector (Primary) with buffer storage owned by Connector's address space
- Requestor (Home) does not equal Connector (Primary) with buffer storage owned by Requestor's address space

In general, the IXLLIST service allows you to designate your data buffer storage using real or virtual storage addresses. However, it is of the utmost importance that XES is aware of the specific location of the data buffer storage and that the location remains so until all data transfer is complete.

Using Real Versus Virtual Storage

The IXLLIST service allows you to designate the data buffer storage in three different ways:

- By **real** storage address
- By **pageable** virtual storage address (including pageable subpools, disabled-reference (DREF) subpools, and page-fixed storage that might not remain page-fixed in a particular real storage location until the completion of the request)
- By **nonpageable** virtual storage address (including fixed subpools and storage that might not remain page-fixed in a particular real storage location until the completion of the request)

(For information about whether a subpool is pageable, fixed, or DREF storage, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

Specifying the PAGEABLE parameter with BUFFER and BUFLIST is a way to identify to the system whether the storage area you pass is in pageable or potentially pageable storage.

Real storage address

When data buffer storage is designated by real address, XES takes no responsibility for its ownership or its attributes. The IXLLIST invoker is entirely responsible for management of the storage binds.

For example, suppose a swappable connector

Obtains a pageable virtual storage buffer in storage associated with the connector's address space
Pagefixes the storage
Loads the real address of the buffer storage

Passes those real storage addressses to XES on a request

If the connector's address space were to be swapped out at some point after loading the real addresses, the system could free and then reassign the real storage frames backing the data buffer. (Page-fixed storage does not remain fixed in real storage when the owning address space is swapped out.) Then, if those real addresses were subsequently used to transfer data to or from the coupling facility, the results would be unpredictable because XES is unaware that the bind between the real addresses and the data buffer virtual storage has been broken.

To summarize: When data buffer storage is passed by real address, it is the caller's responsibility to manage the binds between the data buffer virtual storage and the real storage addresses provided to the coupling facility. The caller must ensure that the data buffer virtual storage remains bound to the real storage addresses provided until the request completes.

Pageable virtual storage address

When data buffer storage is designated by pageable virtual storage address (PAGEABLE=YES on the IXLLIST request), XES takes full responsibility for the ownership and its attributes regardless of what address space owns the storage. XES performs the required page fixing to fix the buffer in real storage while the IXLLIST request transfers data to or from the coupling facility. XES establishes the storage binds between the data buffer virtual storage and the real storage backing it and then releases those binds when the data transfer is complete.

If the storage-owning address space were to be swapped out while the XES-established storage binds exist, XES does not allow the swap-out to complete until those storage binds have been broken. The following three scenarios describe actions taken by XES at the time of the swap-out:

1. Coupling facility data transfer has not yet been initiated.

   XES breaks the real storage binds associated with the request. When the address space is swapped-in again, XES re-establishes the storage binds for the request by once again fixing the data buffer virtual storage in real storage (which most likely is a different real storage location than the data buffer previously occupied). XES subsequently uses these real storage addresses for the coupling facility data transfer.
2. Coupling facility data transfer is actively in progress.

   XES delays the swap-out until the coupling facility data transfer completes. When the address space is swapped-in again, the data transfer for the request is complete and there is no need to re-establish the storage binds for the request.
3. Coupling facility data transfer has completed.

   XES breaks the real storage binds associated with the request (or, the storage binds might already have been broken, depending on when the swap-out occurred). When the address space is swapped-in again, the data transfer for the request is complete and there is no need to re-establish storage binds for the request.

To summarize: When data buffer storage is passed by pageable virtual storage address, XES is responsible for managing the binds between the data buffer virtual storage and the real storage used to transfer data to or from the coupling facility.

Nonpageable virtual storage address

When data buffer storage is designated by non-pageable virtual storage address (PAGEABLE=NO on the IXLLIST request), XES takes full responsibility for the ownership and its attributes if and only if the storage is owned by the requestor's or connector's address space. XES establishes the storage binds between the data buffer virtual storage and the real storage backing it and then releases those binds when the data transfer associated with the request is complete.

If the storage-owning address space (the requestor's or connector's address space) were to be swapped out while the XES-established storage binds exist, XES does not allow the swap-out to complete until those storage binds have been broken. The following three scenarios describe actions taken by XES at the time of the swap-out:

1. Coupling facility data transfer has not yet been initiated.

XES breaks the real storage binds associated with the request. When the address space is swapped-in again, XES re-establishes the storage binds for the request (which most likely is a different real storage location than the data buffer previously occupied). XES subsequently uses these real storage addresses for the coupling facility data transfer.

2. Coupling facility data transfer is actively in progress.

   XES delays the swap-out until the coupling facility data transfer completes. When the address space is swapped-in again, the data transfer for the request is complete and there is no need to re-establish the storage binds for the request.

3. Coupling facility data transfer has completed.

   XES breaks the real storage binds associated with the request (or, the storage binds might already have been broken, depending on when the swap-out occurred). When the address space is swapped-in again, the data transfer for the request is complete and there is no need to re-establish storage binds for the request.

To summarize: When data buffer storage is passed by nonpageable virtual storage address, XES is responsible for managing the binds between the data buffer virtual storage and the real storage used to transfer data to or from the coupling facility if and only if the storage is owned by the requestor's or connector's address space.

Note:

1. If you specify PAGEABLE=NO and your request is processed synchronously, you can free storage when you receive control back from IXLLIST and check the return code to verify that your request was performed synchronously.
2. Table 39 on page 530 shows how long you must keep storage areas fixed for each processing mode if you specify PAGEABLE=NO **and the system processes the request asynchronously.**

Table 39. When Storage Areas Passed to IXLLIST Can Be Made Pageable

MODE Value	**When Storage Can Be Made Pageable**
ASYNCECB or SYNCECB	After ECB is posted
ASYNCTOKEN or SYNCTOKEN	When your program regains control from the IXLFCOMP service **and the request has completed**
SYNCEXIT	When your completion exit receives control

Deciding Whether to Provide Page-Fixed Storage

The system can page-fix and page-free the storage (if you specify PAGEABLE=NO) much faster than you can using PGSER services. However, if you are issuing IXLLIST multiple times and reusing the same storage areas to pass information, you still might obtain better performance if you page-fix the buffers once and specify PAGEABLE=NO rather than having the system page-fix storage for you on each IXLLIST invocation. The choice for best performance depends on the number of times you are invoking IXLLIST.

Another consideration in choosing whether to page-fix the storage or have the system do it is that IXLLIST page-fixes the storage only until the request completes. If you need fixed storage for other reasons than to meet IXLLIST requirements, you should fix the storage yourself and specify PAGEABLE=NO.

See “Using Real Versus Virtual Storage” on page 528 for more information about specifying pageable and nonpageable virtual storage.

Specifying the Buffer Storage Key

The BUFSTGKEY parameter, specified with BUFFER or BUFLIST, and PAGEABLE=YES, identifies the storage key associated with the buffers.

Specifying a storage key helps provide data integrity by allowing list services to check that the buffer is accessible in the key intended by the caller. This is particularly important when the buffer is owned by a client address space and is passed by the server address space to IXLLIST.

IXLLIST performs the storage key check, allowing the server address space to avoid having to transfer the data into its own storage before passing it to IXLLIST.

If you omit BUFSTGKEY with PAGEABLE=YES, the system uses the PSW key of the IXLLIST requestor as the default storage key and performs key checking using the caller's PSW key.

You cannot specify the BUFSTGKEY parameter with PAGEABLE=NO. The system does not do any storage key checking when non-pageable buffers are used. It is the IXLLIST invoker's responsibility to do any storage key checking that might be required for non-pageable buffer storage.

WRITE: Writing to a List Entry

Use the WRITE request to update an existing list entry or create a new one.

See also IXLLSTE for information about updating or creating a list entry.

Understanding the Write Operation

Assuming the list structure has been allocated to contain both data entries and adjunct areas, you can write data to any of the following with the WRITE operation:

- The data entry only
- The adjunct area only
- Both the data entry and the adjunct area.

The only exception is when you create a new list entry in a structure that has adjunct areas; if you don't specify data to be written to the adjunct area, the adjunct area of the new list entry is initialized to zeros.

If the list structure contains adjunct areas, each list entry always contains an adjunct area. For list structures with both data entries and adjunct areas, it is possible to have list entries with either of the following:

- An adjunct area but no data entry
- A data entry and an adjunct area

Guide to the Topic

"WRITE: Writing to a List Entry" on page 531 is divided into three section s:

- The first section provides information applicable to all WRITE requests:
  - "Specifying the Type of Write Operation" on page 531
  - "Specifying the Size of the Data Entry to Hold the Data" on page 532
  - "Selecting the Buffer Format" on page 524
  - "Specifying the Buffer Storage Key" on page 530
  - "Requesting a Lock Operation as Part of a WRITE Request" on page 533
- The second section , "Updating an Existing List Entry" on page 533, explains how to update an existing entry.
- The third section , "Creating a New List Entry" on page 533, explains how to create a new entry.

Specifying the Type of Write Operation

You specify the ENTRYTYPE parameter to indicate whether you want to update an existing list entry, create a new one, or to indicate that a new list entry is to be created only if an existing one (with the attributes you have specified) cannot be found:

ENTRYTYPE=OLD
Indicates that the write request is to be performed **only** if the specified target list entry already exists.

ENTRYTYPE=NEW
Indicates that a new list entry is to be created.

ENTRYTYPE=ANY
Indicates that the WRITE request should be performed as follows:

- If a list entry with the specified attributes exists already, it is to be updated.
- If a list entry with the specified attributes does not exist, a new list entry is to be created.

Specifying the Size of the Data Entry to Hold the Data

If you are writing data entry information to the list entry you are updating or creating, you use the ELEMNUM parameter to specify the size of the data entry (number of data elements) needed for the data. When you write to a data entry, the size of the data entry is changed to the number of elements indicated by ELEMNUM. Table 40 on page 532 shows the result of specifying zero, too few, too many, and the correct number of data elements to hold the contents of a given amount of buffer storage.

Table 40. Results of Specifying the Number of Data Elements on a WRITE Request

Number of Data Elements Specified	Result
Enough to hold data	Specified number of data elements is allocated.
More than number needed to hold data	Specified number of data elements is allocated. Extra space is padded with binary zeros.
Fewer than number needed to hold data	The data is truncated to fit the allotted space.
Zero	Existing data entry deleted, if there is one. No data elements are allocated.

Specifying the List Entry Version Number on a WRITE Request

For information about:

- Using the list entry version number to maintain data integrity on a WRITE request
- Updating the version number on a WRITE request

see “Understanding the List Entry Version Number” on page 523.

Specifying the List Authority Value on a WRITE Request

For information about:

- Using the list authority value to select an entry for processing
- Updating the list authority value

see “Understanding the List Authority Value” on page 513.

Requesting Automatic Key Assignment on a WRITE Request

For information about requesting automatic key assignment on a write request, see “Understanding List Entry Key Assignment” on page 490.

Passing Data for a WRITE Request

You can write data to a list entry's data entry, adjunct area, or both. You pass data to be written to the data entry in a single buffer or multiple buffers. Both methods enable you to pass up to 65536

(64K) bytes of data. You pass data to be written to the adjunct area in a single 64-byte storage area. See "Selecting the Buffer Format" on page 524 for a description of the buffer format options and their performance considerations.

Requesting a Lock Operation as Part of a WRITE Request

To perform a serialized write operation, one in which a lock operation is performed together with a write operation, specify the LOCKOPER parameter on the IXLLIST macro. If the list service cannot perform both the lock operation and the write operation, it performs neither and fails the request.

You can specify the following LOCKOPER values on a WRITE request:

- SET
- RESET
- NOTHELD
- HELDBY

See "LOCK: Performing a Lock Operation" on page 568 for detailed information about the LOCKOPER parameter.

Updating an Existing List Entry

When you update an existing list entry (ENTRYTYPE=OLD) or might do so (ENTRYTYPE=ANY), you can designate the target list entry in several ways:

- To specify the head or tail entry on a particular list, code the list position (LISTPOS) and the list number (LISTNUM).
- To specify a particular entry regardless of where it resides in the list structure, code one of the following:
  - The entry name (ENTRYNAME), for named entries only.
  - The entry ID (ENTRYID).

  You can code the LISTNUM parameter to stipulate that the selected entry must reside on a certain list.
- To specify a keyed list entry at the head or tail of a sublist of list entries with the same key, code the list entry's key (ENTRYKEY), the position on the sublist (LISTPOS), and the list number (LISTNUM).
- To specify the list entry associated with the list cursor for a certain list, code the LOCBYCURSOR and LISTNUM parameters.

  See "Understanding the List Cursor" on page 494 for information about using the list cursor with a WRITE request.

If you omit the LISTPOS parameter, the default value, is HEAD. So in effect, there is always a value for LISTPOS if one is needed.

Creating a New List Entry

If your write request will cause (ENTRYTYPE=NEW) or might cause (ENTRYTYPE=ANY) a new list entry to be created, you can either provide the information necessary to create and position the new list entry or have the list service position the new list entry according to the defaults for the parameters you omit. In addition to specifying the list number (LISTNUM) of the list to receive the new entry, you need to provide the following information.

For a list structure with entry names: You must provide a list entry name (ENTRYNAME) for the list service to use if it creates a new list entry for you.

The list service uses the value of LISTPOS to determine whether to place the new list entry at the head or tail of the target list. If you specify LISTPOS=HEAD, the list service places the list entry at the head of the list. If you specify LISTPOS=TAIL, the list service places the list entry at the tail of the list.

- For ENTRYTYPE=ANY:

  - If you specify both the ENTRYID and ENTRYNAME parameters, the list service uses the value of ENTRYID to check for an existing list entry, and the value of ENTRYNAME to assign a name to a new entry.
  - If a list entry already exists with the specified name, the list entry is updated.
- For ENTRYTYPE=NEW: if a list entry already exists with the name you have specified for the new entry, the WRITE request fails.

For a list structure with entry keys: You can provide a list entry key (ENTRYKEY), have the list service assign a list entry key as shown in Table 41 on page 534, or have the list service automatically assign a list entry key from the list control value. List entries in each list are maintained by key in ascending order. The list service places a new list entry on the target list as follows:

Table 41. Rules for Placement of Keyed List Entry for REQUEST=WRITE

ENTRYKEY Specified?	LISTPOS Value	Existing Entries With Same Key?	Position for New List Entry
Yes	HEAD or TAIL	No	Positioned to maintain ascending order of keys.
Yes	HEAD	Yes	Before the first list entry with the same key.
Yes	TAIL	Yes	After the last list entry with the same key.
No	HEAD	Not applicable	At head of list. List entry key initialized to binary zeros.
No	TAIL	Not applicable	At tail of list. List entry key initialized to binary ones.

When you specify automatic list entry key assignment with LISTKEYTYPE, the list service uses the key value that was automatically assigned to follow the same placement rules as if you had explicitly specified ENTRYKEY.

If you specify a locking operation (LOCKOPER) to be performed with a write operation, the list service performs the locking operation as described under Table 42 on page 569.

Creating a New Keyed List Entry in a CFLEVEL=3 or Higher Coupling Facility

If your write request will cause a new keyed list entry to be created on an empty sublist and the target sublist is being monitored, then the system queues all EMCs associated with the transitioning sublist to the respective users' event queues.

Creating a List Entry With No Data

You can create a list entry with no data entry, by specifying an ELEMNUM value of 0. This option might be useful under circumstances such as the following:

- You want to establish list entries as place holders to receive data during later processing and you don't know what size to make the data entries.
- You are using adjunct areas to hold data instead of data entries.

Receiving Answer Area Information from a WRITE Request

When you invoke IXLLIST, list services return information related to your request in the answer area specified using the ANSLEN and ANSAREA parameters.

Determining if the Answer Area is Valid

Under certain conditions, the system will not be able to return answer area information. For example, if you issue an IXLLIST request and specify asynchronous processing, or your synchronous request is run asynchronously, the answer area will not be valid when your program regains control.

To determine whether the answer area is valid for an IXLLIST request you have issued, check the explanation for the return code and reason code combination you have received in *z/OS MVS Programming: Sysplex Services Reference*.

The location of the return code and reason code to be checked depends on how your program is notified of request completion:

- If you issue IXLLIST and receive control back directly (not through a complete exit) or you issue the IXLFCOMP macro to handle request completion, check:
  - GPR 15 or the RETCODE field for the return code
  - GPR 0 or the RSNCODE field for the reason code
- If you issue IXLLIST and specify a complete exit to receive control, check:
  - The CMPLRETCODE field for the return code
  - The CMPLRSNCODE field for the reason code

  These fields are mapped by the IXLYCMPL macro, shown in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

The following list describes the answer area information returned when the answer area is valid. The answer area is mapped by the IXLYLAA macro, which is shown in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

LAARETCODE
The return code from the IXLLIST service. Return code values are defined in the IXLYCON macro.

LAARSNCODE
The reason code associated with the return code from the IXLLIST service. Reason code values are defined in the IXLYCON macro.

LAALCTL
The list entry controls for the list entry that was updated or created. Returned for successful WRITE requests. The area is mapped by IXLYLCTL.

For a WRITE request that failed because the target list entry did not meet the list number or version number criteria specified by LISTNUM or VERSCOMP, the list entry controls for the list entry that failed to meet the selection criterion.

For a WRITE request that failed because the entry name (ENTRYNAME) specified for the new list entry already existed, the list entry controls for the list entry already having the specified name.

LAALISTDESC
The user-specified description of the list. Returned for WRITE requests that failed because of an authority mismatch. Returned only for structures allocated in a coupling facility with CFLEVEL=1 or higher.

LAALISTAUTH
The list authority for the list. Returned for WRITE requests that failed because of an authority mismatch. Returned only for structures allocated in a coupling facility with CFLEVEL=1 or higher.

LAATOTALCNT
The total number of list entries in use in the structure. Returned for requests that completed successfully.

LAATOTALELECNT
The total number of data elements in use in the structure. Returned for requests that completed successfully.

LAALISTCNT
The number of list entries or data elements on the list that was the target of the write operation. This count is returned for requests that completed successfully. This count includes the number of list entries or data elements on the list that currently reside in coupling facility real and storage class memory. The value specified for LISTCNTLTYPE on the IXLCONN macro when the list structure was allocated determines whether this field represents a count of list entries or data elements. LAALISTCNTLTYPE found in the answer area can also be used to determine whether LAALISTCNT represents a count of list entries or data elements.

LAALISTOPPCNT
The number of list entries or data elements on the list that was the target of the write operation. This count is valid when LAALISTOPPCNTVALID is ON. When LaaListCntlType is set to 1, LaaListOppCnt is expressed as the number of data elements on the list. When LAALISTCNTLTYPE is set to 2, LaaListOppCnt is expressed as the number of list entries on the list. Like LAALISTCNT, this count includes the number of list entries or elements for the processed list that currently reside in coupling facility real and storage class memory.

LAALISTOPPCNTVALID
Opposite List Count Valid indicator. This is a flag that can be returned for requests that completed successfully. The flag indicates when LAALISTOPPCNT is valid for use.

LAALISTCNTLTYPE
List Control Type. This is a value returned for requests that completed successfully. The value is also returned when the list designated as the target of a write request cannot accommodate more entries or elements (LAARSNCODE=IxlRsnCodeListFull). The value is set to 1 (IXLLISTCNTLTYPEENTRY) when ENTRY is specified for LISTCNTLTYPE on the IXLCONN macro when the list structure was allocated. LAALISTCNT is expressed in number of list entries. When LAALISTOPPCNTVALID is ON, LAALISTOPPCNT is expressed in number of data elements.

The value is set to 2 (IXLLISTCNTLTYPEELEMENT) when ELEMENT is specified for LISTCNTLTYPE on the IXLCONN macro when the list structure was allocated. LAALISTCNT is expressed in number of data elements. When LAALISTOPPCNTVALID is ON, LAALISTOPPCNT is expressed in number of list entries.

A value of zero (0) indicates that this field is not provided by list services for the request.

LAACONID
For a WRITE request specifying the LOCKOPER parameter, LAACONID contains, under the following circumstances:

- HELDBY parameter specified and the lock was not held by the connection specified by LOCKCOMP or taken as the default.
- NOTHELD parameter specified with LOCKMODE=COND and the request failed because the lock is held by another connection.
- NOTHELD parameter specified with LOCKMODE=UNCOND and the request failed because the lock is held by a failed persistent connection.
- SET parameter specified with LOCKMODE=COND and the request failed because the lock is held by another connection.
- SET parameter specified with LOCKCOMP and the lock was not held by the specified connection.
- SET parameter specified with LOCKMODE=UNCOND and the request failed because the lock is held by a failed persistent connection.
- RESET parameter specified without LOCKCOMP and the request failed because you do not hold the lock.
- RESET parameter specified with LOCKCOMP and the lock was not held by the specified connection.

Either of the following:

- The connection identifier of the lock owner, if the lock specified by the LOCKINDEX parameter is held
- Zeros, if the lock specified by the LOCKINDEX parameter is free

LAALISTKEY
The current value of the list key from the list controls. Returned for WRITE requests that failed because the maximum list key value would be exceeded. Returned only for structures allocated in a coupling facility with CFLEVEL=1 or higher.

LAALISTFULLLIMIT
The maximum number of entries or elements that can be placed on the designated target list of a write request. It is returned when the list designated as the target of a write request cannot accommodate more entries or elements (LAARSNCODE=IxlRsnCodeListFull). LAALISTFULLLIMIT is valid when the structure is allocated in a CFLEVEL=22 or higher coupling facility.

LAALISTFULLCNT
The count of in-use entries or elements that reside on a list designated as the target of a write request and the target list cannot accommodate more entries or elements (LAARSNCODE=IxlRsnCodeListFull). This count includes the number of list entries or elements on the target list that currently reside in coupling facility real and storage class memory. LAALISTFULLCNT is valid when the structure is allocated in a CFLEVEL=22 or higher coupling facility.

LAAMAXLISTKEY
The current value of the maximum list key from the list controls. Returned for WRITE requests that failed because the maximum list key value would be exceeded. Returned only for structures allocated in a coupling facility with CFLEVEL=1 or higher.

LAAENTRYCREATED
Flag to indicate that the request created a new entry. Returned for successful WRITE requests. Returned only for structures allocated in a coupling facility with CFLEVEL=1 or higher.

READ, READ_MULT, READ_LIST: Reading List Entries

You can perform three types of read operations on list entries in a list structure:

REQUEST=READ
Reads the contents of a single list entry.

See also IXLLSTE for information about reading a single list entry.

REQUEST=READ_LIST
Reads the contents of multiple list entries on a specified list or read the contents of list entries on a specified list or only those:

- With a version number that satisfies a comparison criteria, using VERSCOMP and VERSCOMPTYPE
- With a list authority value that satisfies a comparison criteria, using AUTHCOMP and AUTHCOMPTYPE
- With a list entry key that satisfies a comparison criteria, using KEYCOMP
- With any combination of the above

See also IXLLSTM for information about reading the contents of multiple list entries.

REQUEST=READ_MULT
Read the contents of all list entries in the structure or only those:

- On a certain list
- With a version number that satisfies a comparison criteria, using VERSCOMP and VERSCOMPTYPE
- With a list authority value that satisfies a comparison criteria, using AUTHCOMP and AUTHCOMPTYPE
- With a list entry key that satisfies a comparison criteria, using KEYCOMP
- With any combination of the above

See also IXLLSTM for information about reading list entries from multiple lists.

Guide to the Topic

"READ, READ_MULT, READ_LIST: Reading List Entries" on page 537 is divided into the following section s.

- "READ: Reading a Single List Entry" on page 538 presents information about the READ request.
- "READ_LIST: Reading Multiple List Entries from a List" on page 541 presents information about the READ_LIST request.
- "READ_MULT: Reading Multiple List Entries from One or More Lists" on page 547 presents information about the READ_MULT request.

READ: Reading a Single List Entry

Use the READ request to read information from a specific list entry. You can use any of the following to identify the target list entry:

- To specify the head or tail entry on a particular list, code the list position (LISTPOS) and the list number (LISTNUM).
- To specify a particular entry regardless of where it resides in the list structure, code one of the following:
  - The entry name (ENTRYNAME), for named entries only.
  - The entry ID (ENTRYID)

  You can code the LISTNUM parameter to stipulate that the selected entry must reside on a certain list.
- To specify a keyed list entry at the head or tail of a sublist of list entries with the same key, code the list entry's key (ENTRYKEY), position on the sublist (LISTPOS), and list number (LISTNUM).
- To specify the list entry associated with the list cursor for a certain list, code the LOCBYCURSOR and LISTNUM parameters.

  See "Understanding the List Cursor" on page 494 for information about using the list cursor on a READ request.

If you omit the LISTPOS parameter, the default value is HEAD. So in effect, there is always a value for LISTPOS if one is needed.

You indicate the types of information you want read by coding the parameter for the storage area to receive the information. To receive data entry information, code the BUFFER parameter or the BUFLIST parameter. To receive adjunct area information, code the ADJAREA parameter.

Specifying the List Entry Version Number on a READ Request

For information about:

- Using the list entry version number to select the list entry to be read
- Updating the version number on a READ request

see "Understanding the List Entry Version Number" on page 523.

Specifying the List Authority Value on a READ Request

For information about:

- Using the list authority value to select an entry for processing
- Updating the list authority value

see "Understanding the List Authority Value" on page 513.

Requesting a Lock Operation as Part of a READ Request

To perform a serialized read operation, one in which a lock operation is performed along with a read request, specify the LOCKOPER parameter on the IXLLIST macro. If the list service cannot perform both the lock operation and the read operation, it performs neither and fails the request.

You can specify the following LOCKOPER values on a READ request:

- SET
- RESET
- NOTHELD
- HELDBY

See "LOCK: Performing a Lock Operation" on page 568 for detailed information about the LOCKOPER parameter.

Receiving Data from a READ Request

See "Selecting the Buffer Format" on page 524 for a description of the buffer format options and their performance considerations.

Obtaining the List Entry Information from the Output Areas

The READ request returns the list entry information as follows:

- **Data entry information**: In the storage areas specified by BUFFER or BUFLIST.
- **Adjunct area information**: In the storage area specified by ADJAREA.
- **List entry controls**: In the LAALCTL field of the answer area.

Receiving Answer Area Information from a READ Request

When you invoke IXLLIST, list services return information related to your request in the answer area specified using the ANSLEN and ANSAREA parameters.

Under certain circumstances, answer area information is not valid. See "Determining if the Answer Area is Valid" on page 535 for information on how to determine whether the answer area information is valid.

The following list describes the information returned when the answer area is valid. The answer area is mapped by the IXLYLAA macro, which is presented in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

LAARETCODE
The return code from the IXLLIST service. Return code values are defined in the IXLYCON macro.

LAARSNCODE
The reason code associated with the return code from the IXLLIST service. Reason code values are defined in the IXLYCON macro.

LAALCTL
The list entry controls for the list entry that was read. Returned for a successful READ request. The area is mapped by IXLYLCTL.

For a READ request that failed because insufficient buffer storage was provided to hold the data, the list entry controls for the entry whose data couldn't fit in the buffer.

For a READ request that failed because the target list entry did not meet the list number or version number criteria specified by LISTNUM or VERSCOMP, the list entry controls of the failing entry.

LAALISTDESC
The user-specified description of the list. Returned for READ requests that failed because of an authority mismatch. Returned only for structures allocated in a coupling facility with CFLEVEL=1 or higher.

LAALISTAUTH
The list authority for the list. Returned for READ requests that failed because of an authority mismatch. Returned only for structures allocated in a coupling facility with CFLEVEL=1 or higher.

LAATOTALCNT
The total number of list entries in use in the structure. Returned for requests that completed successfully.

LAATOTALELECNT
The total number of data elements in use in the structure. Returned for requests that completed successfully.

LAALISTCNT
The number of list entries or data elements on the list that was the target of the read operation. Returned for requests that completed successfully. This count includes the number of list entries or data elements on the list that currently reside in coupling facility real and storage class memory. The value specified for LISTCNTLTYPE on the IXLCONN macro when the list structure was allocated determines whether this field represents a count of list entries or data elements. LAALISTCNTLTYPE found in the answer area can also be used to determine whether LAALISTCNT represents a count of list entries or data elements.

LAALISTOPPCNT
The number of list entries or data elements on the list that was the target of the read operation. Valid when LAALISTOPPCNTVALID is ON. When LaaListCntlType is set to 1, LaaListOppCnt is expressed as the number of data elements on the list. When LAALISTCNTLTYPE is set to 2, LaaListOppCnt is expressed as the number of list entries on the list. Like LAALISTCNT, this count includes the number of list entries or elements for the processed list that currently reside in coupling facility real and storage class memory.

LAALISTOPPCNTVALID
Opposite List Count Valid indicator. A flag that can be returned for requests that completed successfully. The flag indicates when LAALISTOPPCNT is valid for use.

LAALISTCNTLTYPE
List Control Type. A value returned for requests that completed successfully. The value is set to 1 (IXLLISTCNTLTYPEENTRY) when ENTRY is specified for LISTCNTLTYPE on the IXLCONN macro when the list structure was allocated. LAALISTCNT is expressed in number of list entries. When LAALISTOPPCNTVALID is ON, LAALISTOPPCNT is expressed in number of data elements.

The value is set to 2 (IXLLISTCNTLTYPEELEMENT) when ELEMENT is specified for LISTCNTLTYPE on the IXLCONN macro when the list structure was allocated. LAALISTCNT is expressed in number of data elements. When LAALISTOPPCNTVALID is ON, LAALISTOPPCNT is expressed in number of list entries.

A value of zero (0) indicates that this field is not provided by list services for the request.

LAACONID
For a READ request specifying the LOCKOPER parameter, LAACONID contains, under the following circumstances:

- HELDBY parameter specified and the lock was not held by the connection specified by LOCKCOMP or taken as the default.
- NOTHELD parameter specified with LOCKMODE=COND and the request failed because the lock is held by another connection.
- NOTHELD parameter specified with LOCKMODE=UNCOND and the request failed because the lock is held by a failed persistent connection.
- SET parameter specified with LOCKMODE=COND and the request failed because the lock is held by another connection.
- SET parameter specified with LOCKCOMP and the lock was not held by the specified connection.
- SET parameter specified with LOCKMODE=UNCOND and the request failed because the lock is held by a failed persistent connection.
- RESET parameter specified without LOCKCOMP and the request failed because you do not hold the lock.
- RESET parameter specified with LOCKCOMP and the lock was not held by the specified connection.

Either of the following:

- The connection identifier of the lock owner, if the lock specified by the LOCKINDEX parameter is held

- Zeros, if the lock specified by the LOCKINDEX parameter is free

READ_LIST: Reading Multiple List Entries from a List

Use the READ_LIST request to read multiple list entries from a single list. Specifying READ_LIST causes the list service to read all list entries (or all list entries which succeed at a version number comparison that you specify with VERSCOMP and VERSCOMPTYPE and/or all list entries which succeed at an entry key comparison that you specify with KEYCOMP) beginning at the specified entry, traversing the list in the direction specified by LISTDIR. Note that if another user adds or updates a list entry while your request is being processed, these changes will not be included in the output you receive unless they occur in a location on the list that has not yet been scanned. You cannot assume that your READ_LIST request has read every list entry that meets your selection criteria unless you serialize access to the list before issuing the request and prevent other users from changing the list until your READ_LIST processing is finished.

The READ_LIST request has the following features:

- List entries are read and returned in the buffer in the order they occur on the list (or in reverse order, if you specified LISTDIR=TOHEAD).
- You needn't know in advance which list the starting entry is on. You can either omit LISTNUM and process whatever list contains the particular entry you specify or specify LISTNUM and process the list only if the starting entry is on the list you have specified.
- For reading multiple list entries from a single list, READ_LIST offers substantially better performance than the READ_MULT request
- If a list entry is added after the scan has begun, it will be found if it occurs in a location on the list that has not yet been scanned.
- If a READ_LIST request completes prematurely, the list service returns the list entry controls of the entry at which processing is to resume. If the returned list entry controls represent a list entry that is moved or deleted before the request is reissued, entries could be read twice or skipped, or the reissued request could fail. See “Handling an Incompletely Processed READ_LIST Request” on page 545 for more information.

Specifying the Starting List Entry and the Processing Direction

You can designate the starting list entry in several ways:

- To specify the head or tail entry on a particular list, code the list position (LISTPOS) and the list number (LISTNUM).
- To specify a particular entry regardless of where it resides in the list structure, code one of the following:
  - The entry name (ENTRYNAME), for named entries only
  - The entry ID (ENTRYID)

  You can code the LISTNUM parameter to stipulate that the selected entry must reside on a certain list.
- To specify a keyed list entry at the head or tail of a sublist of list entries with the same key, code the list entry's key (ENTRYKEY), position on the sublist (LISTPOS), and list number (LISTNUM).
- To specify the list entry associated with the list cursor for a certain list, code the LOCBYCURSOR and LISTNUM parameters. Note that you cannot specify the UPDATECURSOR parameter on the READ_LIST request, so the list cursor remains pointed to the target list entry after processing completes.

  See “Understanding the List Cursor” on page 494 for information about using the list cursor.

If you omit the LISTPOS parameter, the default value is HEAD. So in effect, there is always a value for LISTPOS if one is needed.

Use the LISTDIR parameter to designate the direction in which processing is to proceed from the starting list entry.

Specifying the Types of List Entry Information to be Read

The TYPE parameter specifies the types of information you want read from each list entry. You can request any combination of the following types:

ECONTROLS
List entry control information

ENTDATA
Data entry information

ADJDATA
Adjunct area information

Important

Be careful to request entry data or adjunct data only if the structure supports each type of data. If the list structure does not support entry data and you request ENTDATA or if the list structure does not support adjunct data and you request ADJDATA, the list service fails the request with reason code IXLRSNCODEBADREADTYPE.

Specifying the List Entry Version Number on a READ_LIST Request

For information about using the list entry version number to select the list entries to be read, see "Understanding the List Entry Version Number" on page 523.

Specifying the List Authority Value on a READ_LIST Request

For information about using the list authority value to select entries for processing, see "Understanding the List Authority Value" on page 513.

Requesting Entry Key Comparison on a READ_LIST Request

For information about using the entry key to select entries for processing, see "Using the Entry Key in Multiple List Operations" on page 493.

Requesting a Lock Operation as Part of a READ_LIST Request

To perform a serialized READ_LIST operation, one in which a lock operation is performed before performing a READ_LIST request, specify the LOCKOPER parameter on the IXLLIST macro.

You can specify the following LOCKOPER values on a READ_LIST request:

- NOTHELD
- HELDBY

If your serialization protocol permits lock stealing, you can use LOCKOPER=HELDBY to ensure that your read request is performed only if the specified lock is in the state you expect.

See "LOCK: Performing a Lock Operation" on page 568 for detailed information about the LOCKOPER parameter.

Receiving Data from a READ_LIST Request

See "Selecting the Buffer Format" on page 524 for a description of the buffer format options and their performance considerations.

The requested list entry information returns to you in the following output areas:

- Buffers specified by BUFFER or BUFLIST
- The storage area specified by ADJAREA
- The LAARLRMLCTLS field in the answer area specified by ANSAREA.

The particular layout of the returned list entry information depends on the types of information you have requested:

- Data entry information
- Adjunct area information
- List entry control information
- Some combination of these.

To access list entries with data entries of different sizes, you must read the list entry controls for each list entry to determine the size of the associated data entry before accessing it. Therefore, if there are different sized data entries, **you should also request list entry controls (TYPE=ECONTROLS) if you request data entry information (TYPE=ENTDATA).**

The answer area is mapped by the IXLYLAA macro. The contents of the IXLYLAA field, LAARLRMLCTLS, and the list entry controls returned in output buffers are mapped by the IXLYLCTL macro. See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for a complete listing of the IXLYLAA and IXLYLCTL macros.

For each list entry read, requested types of information are arranged in the output buffer in the following order:

1. List entry controls
2. Data entry data (if requested and there is a data entry)
3. Adjunct area data (if requested and adjunct areas exist).

The order of the returned information in the output buffer is maintained even if you request only two of the three types of information.

The order in which you specify ENTDATA, ADJDATA, or ECONTROLS on the TYPE parameter has no bearing on the order in which information is arranged in the output buffers.

Figure 64 on page 544 illustrates how list entry information is returned by the READ_LIST request if you provide a single buffer.

As shown in Figure 64 on page 544:

- If adjunct area information is returned, the adjunct area information for the first entry is returned in the ADJAREA field, not in a buffer
- If list entry controls are returned, the list entry controls for the first entry are returned in the answer area field mapped by LAARLRMLCTLS, not in a buffer.

If you provide multiple buffers, the information is copied into the buffers in order of ascending buffer number. List entry information that cannot fit in the current continues into the next buffer. As a result, the information for a single list entry could be split between buffers. However, all of the data for a particular list entry is returned in the same READ_LIST invocation. You won't get the list entry controls and adjunct for an entry in one invocation and the data entry information for the entry on another invocation.

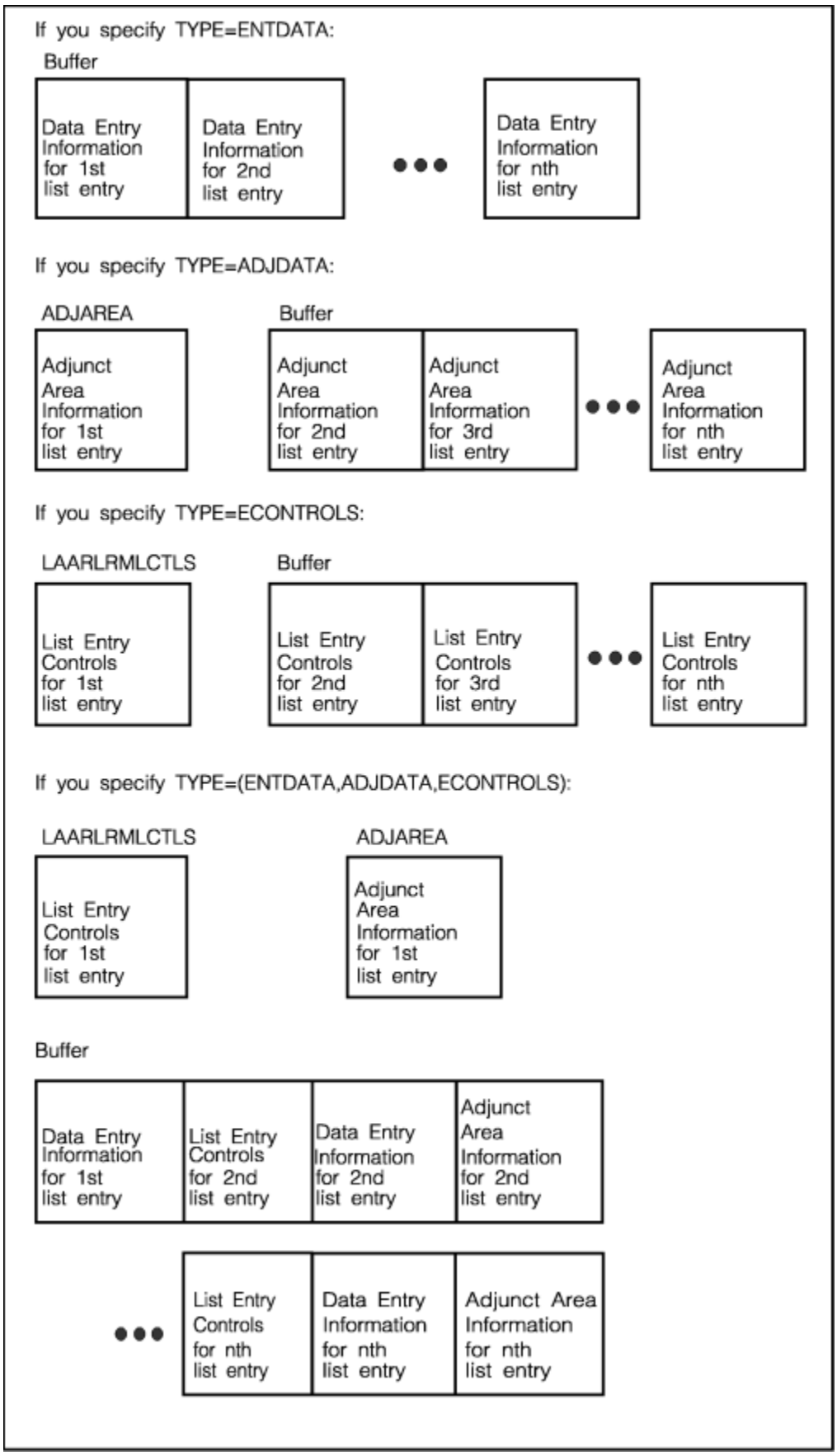


Figure 64. Layout of List Entry Information Returned by READ_LIST Request

Handling an Incompletely Processed READ_LIST Request

A READ_LIST request can complete prematurely for either of the following reasons:

- A request could time out before completion.
- A request could require more buffer space than you provided.

When a READ_LIST request ends before returning all the information, IXLLIST:

- Sets the IXLLIST return code to IXLRETCODEWARNING and the reason code to:
  - IXLRSNCODETIMEOUT if the processing timed out
  - IXLRSNCODEBUFFERFULL if the buffer was too small to hold all the output
  - IXLRSNCODEBADBUFSIZE if the buffer was empty but still too small to hold the first list entry being read.
- Returns in the LAALCTL field of the answer area, the complete set of list entry controls for the next list entry to be scanned.

To continue scanning the list, reissue the READ_LIST request with the ENTRYID keyword specifying the entry ID from the returned list entry controls for the next list entry to be scanned.

Be sure to process the data you received on the last request before reissuing the request. Continue reissuing the request until the return code indicates that all processing has completed.

If the request ended prematurely because the buffer was too small to hold the first entry to be read (for instance, your buffer is 4096 bytes but the data entry information is 65536 bytes), determine the size of the data entry for the list entry that caused the failure by checking the list entry control information returned in LAALCTL (mapped by the IXLYLCTL macro.) Note that the LAALCTL field is valid only when the request ended prematurely because the buffer could not hold all the requested information.

You must know the data element size to make this calculation because the list entry controls only indicate the number of data elements, not their size. If the buffer is too small to hold the information associated with the failing list entry, reissue the READ_LIST request with a buffer at least the size of the failing list entry's data entry.

The Effect of List Changes on Request Resumption

To ensure that the list does not change while your READ_LIST request is being performed, you must have serialized access to the to the list for the entire duration and other users must not be able to modify the list while you hold the lock.

If you don't have this kind of serialization, the list structure might change between the time you first issue a READ_LIST request and the time you reissue it to finish processing the request. If you don't have serialized access to the list, the list entry at which scanning is to resume could have been:

- Deleted by another user. The list entry controls (returned in the answer area) for that entry now specify a list entry that does not exist on the list and the reissued request will fail.
- Moved to another position on the same list. Depending on the direction in which the entry moved and the direction of specified for the READ_LIST scan, the reissued request might either read some entries again, or skip some entries. Figure 65 on page 546 illustrates the problem for a READ_LIST request with LISTDIR=TOTAIL.
- Moved to another list. When you reissue the request, scanning will resume on the new list. To prevent this, specify LISTNUM when you reissue the READ_LIST request to ensure that the request is processed only if the starting list entry is on the correct list.

To handle a problem like those just described, you must restart your READ_LIST request scanning where it began in the initial request.

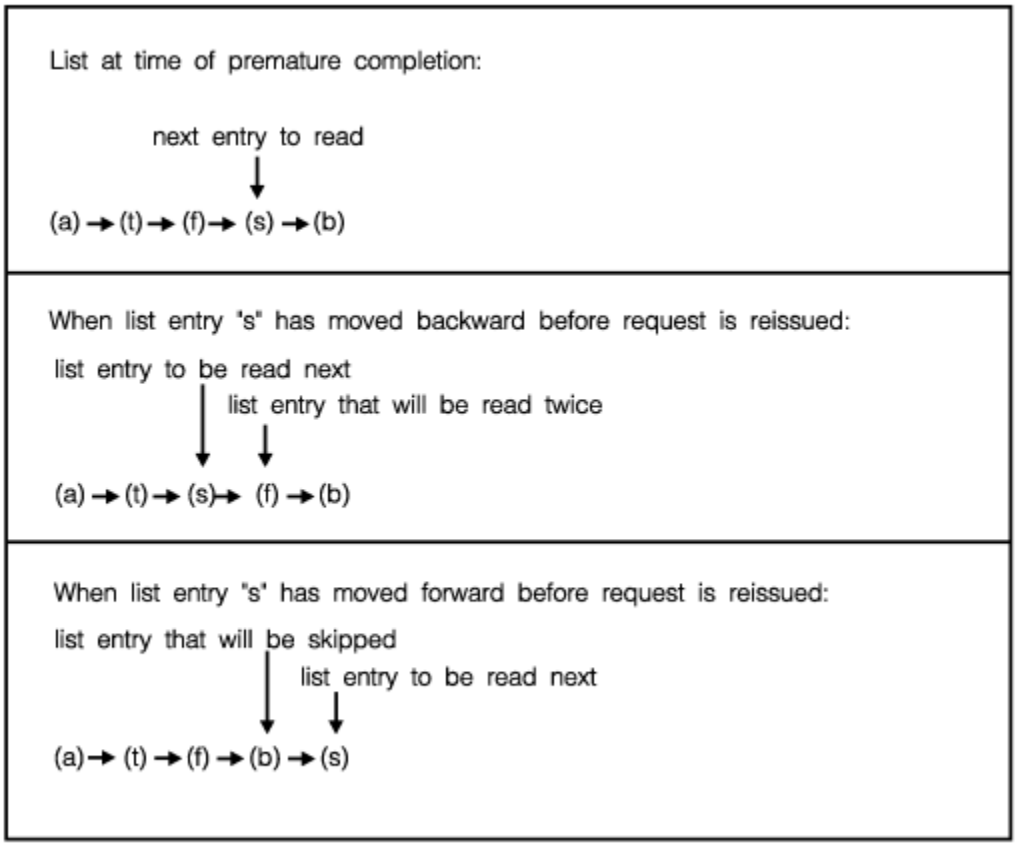


Figure 65. Possible Errors Resulting from Reissue of READ_LIST Request

Receiving Answer Area Information from a READ_LIST Request

When you invoke IXLLIST, list services return information related to your request in the answer area specified using the ANSLEN and ANSAREA parameters.

Under certain circumstances, answer area information is not valid. See "Determining if the Answer Area is Valid" on page 535 for information on how to determine whether the answer area information is valid.

The following list describes the information returned when the answer area is valid. The answer area is mapped by the IXLYLAA macro, which is presented in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

LAARETCODE
The return code from the IXLLIST service. Return code values are defined in the IXLYCON macro.

LAARSNCODE
The reason code associated with the return code from the IXLLIST service. Reason code values are defined in the IXLYCON macro.

LAALCTL
The list entry controls. The area is mapped by IXLYLCTL.

- For a request that completes prematurely, the list entry controls for the list entry to be scanned next. When you reissue the READ_LIST request to continue the scan, designate this list entry as the starting entry by specifying the entry ID returned here with the ENTRYID parameter.
- For a request that fails because the first list entry to be read is larger than the entire buffer, the list entry controls for the list entry that is too large.

- For a request that fails because the starting entry specified was not on the specified list, returns the list entry controls of the specified starting list entry. Use the list controls to determine the correct list to specify for that starting entry.

LAALISTDESC
The user-specified description of the list. Returned for READ_LIST requests that failed because of an authority mismatch. Returned only for structures allocated in a coupling facility with CFLEVEL=1 or higher.

LAALISTAUTH
The list authority for the list. Returned for READ_LIST requests that failed because of an authority mismatch. Returned only for structures allocated in a coupling facility with CFLEVEL=1 or higher.

LAAREADCNT
The count of entries read by READ_LIST. This information is returned for both successful and premature request completion. If no scanned list entries met the criteria specified for selection to be read, the count is set to zero. If the request completed prematurely, there might be additional list entries that meet the selection criteria that have not yet been scanned.

LAACONID
For a READ_LIST request specifying the LOCKOPER parameter, LAACONID contains, under the following circumstances:

- NOTHELD parameter specified and the request failed because the lock is held by another connection.
- HELDBY parameter specified with or without the LOCKCOMP parameter and the lock was not held by the specified connection.

Either of the following:

- The connection identifier of the lock owner, if the lock specified by the LOCKINDEX parameter is held
- Zeros, if the lock specified by the LOCKINDEX parameter is free.

LAARLRMLCTLS
For a READ_LIST request specifying ECONTROLS on the TYPE parameter, the list entry controls of the first list entry read. This field is mapped by the IXLYLCTL macro and is valid whether the request completes successfully or prematurely.

READ_MULT: Reading Multiple List Entries from One or More Lists

With the READ_MULT request you can read multiple list entries in the structure and filter their selection by list number, version number, list authority value, entry key value, or any combination of these filters. Note that if another user adds or updates a list entry while your request is being processed, these changes might not be included in the output you receive. You cannot assume that your READ_MULT request has read every list entry that meets your selection criteria unless you serialize access to the list structure before issuing the request and prevent other users from changing the list structure while your request is being performed.

The READ_MULT request has the following features:

- You can read list entries from multiple lists
- The order in which a list entry is read and returned in the buffer has no relationship to its position in the list structure.
- Since the order in which list entries are read is unpredictable, it is also unpredictable whether an entry, added after the scan has begun, will be found.
- The list service returns a restart token in the answer area when a READ_MULT request ends prematurely. Unlike a READ_LIST request, a prematurely completed READ_MULT request can be resumed successfully regardless of whether list entries have been moved or deleted. Entries will not be read twice or skipped.

Note: For reading list entries from a single list, the READ_LIST request provides better performance.

Specifying Selection Filters on a READ_MULT Request

For information about using the list entry version number to select the list entries to be read, see "Understanding the List Entry Version Number" on page 523.

For information about using the list authority value to select list entries for processing, see "Understanding the List Authority Value" on page 513.

For information about using the entry key value to select entries for processing, see "Using the Entry Key in Multiple List Operations" on page 493.

Requesting a Lock Operation as Part of a READ_MULT Request

To perform a serialized READ_MULT operation, one in which a lock operation is performed before performing a READ_MULT request, specify the LOCKOPER parameter on the IXLLIST macro. If the list service cannot perform both the lock operation and the READ_MULT operation, it performs neither and fails the request.

You can specify the following LOCKOPER values on a READ_MULT request:

- NOTHELD
- HELDBY

See "LOCK: Performing a Lock Operation" on page 568 for detailed information about the LOCKOPER parameter.

Receiving Data from a READ_MULT Request

See "Selecting the Buffer Format" on page 524 for a description of the buffer format options and their performance considerations.

See Figure 64 on page 544 for the layout of the list entry information returned from the READ_MULT request. The layout of the returned information is the same for READ_MULT and READ_LIST requests.

Handling an Incompletely Processed READ_MULT Request

A READ_MULT request can end prematurely for either of the following reasons:

- A request could time out before completion
- A request could require more buffer space than you provided.

When a READ_MULT request ends before returning all the information, IXLLIST:

- Sets the IXLLIST return code to IXLRETCODEWARNING and the reason code to:
  - IXLRSNCODETIMEOUT if the processing timed out
  - IXLRSNCODEBUFFERFULL if the buffer was too small to hold all the output
  - IXLRSNCODEBADBUFSIZE if the buffer was empty but still too small to hold the first list entry being read.
- Returns in the LAARESTOKEN or LAAEXTRESTOKEN field of the answer area, a restart token to be provided when you reissue the request to continue the scan.

To reissue the READ_MULT request, access the restart token returned in the LAARESTOKEN or LAAEXTRESTOKEN field of the answer area and specify the token with the RESTOKEN or EXTRESTOKEN parameter on the next READ_MULT invocation. Be sure to process the information returned from the last request before reissuing the request. Continue to reissue the request until the return code indicates that all processing has completed.

If you do not have exclusive access to the list structure, it could be modified by other users between the time you issue the READ_MULT request and the time you reissue it. Since the READ_MULT request uses a restart token instead of the next entry's list controls to indicate where scanning should resume, scanning can resume successfully even if a particular list entry is moved or deleted.

You can avoid coding a separate IXLLIST invocation with the RESTOKEN or EXTRESTOKEN parameter to handle incomplete processing, by coding a single IXLLIST invocation with the restart token initialized to zero for the first time through. A restart token of zero causes IXLLIST to treat the invocation as a new request. When reissuing the request due to premature completion, be sure to first set the restart token to the value from the LAARESTOKEN or LAAEXTRESTOKEN field.

If the request ended prematurely because the buffer was too small to hold the first entry to be read (for instance, your buffer is 4096 bytes but the data entry information is 65536 bytes), determine the size of the data entry for the list entry that caused the failure by checking the list entry control information returned in LAALCTL. You must know the data element size to make this calculation because the list entry controls only indicate the number of data elements, not their size. Reissue the READ_MULT request with a buffer at least the size of the failing list entry's data entry.

Receiving Answer Area Information from a READ_MULT Request

When you invoke IXLLIST, list services return information related to your request in the answer area specified using the ANSLEN and ANSAREA parameters.

Under certain circumstances, answer area information is not valid. See "Determining if the Answer Area is Valid" on page 535 for information on how to determine whether the answer area information is valid.

The following list describes the information returned when the answer area is valid. The answer area is mapped by the IXLYLAA macro, which is presented in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

LAARETCODE
The return code from the IXLLIST service. Return code values are defined in the IXLYCON macro.

LAARSNCODE
The reason code associated with the return code from the IXLLIST service. Reason code values are defined in the IXLYCON macro.

LAALCTL
The list entry controls.

For a request that fails because the first list entry to be read is larger than the entire buffer, the list entry controls for the list entry that is too large.

LAALISTDESC
The user-specified description of the list. Returned for READ_MULT requests that fail because of an authority mismatch. Returned only for structures allocated in a coupling facility with CFLEVEL=1 or higher.

LAALISTAUTH
The list authority for the list. Returned for READ_MULT requests that fail because of an authority mismatch. Returned only for structures allocated in a coupling facility with CFLEVEL=1 or higher.

LAAREADCNT
The count of entries read by READ_MULT. This information is returned for both successful and premature request completion. If no scanned list entries met the criteria specified for selection to be read, the count is set to zero. If the request completed prematurely, there might be additional list entries that meet the selection criteria that have not yet been scanned.

LAARESTOKEN
The restart token for the request. Returned if ALLOWAUTO=NO was specified or defaulted to on IXLCONN to connect to the structure and the READ_MULT request completed prematurely.

LAAEXTRESTOKEN
The extended restart token for the request. Returned if ALLOWAUTO=YES was specified on IXLCONN to connect to the structure and the READ_MULT request completed prematurely.

LAACONID
For a READ_MULT request specifying the LOCKOPER parameter, LAACONID contains, under the following circumstances:

- NOTHELD parameter specified and the request failed because the lock is held by another connection.
- HELDBY parameter specified with or without the LOCKCOMP parameter and the lock was not held by the specified connection.

Either of the following:

- The connection identifier of the lock owner, if the lock specified by the LOCKINDEX parameter is held
- Zeros, if the lock specified by the LOCKINDEX parameter is free.

LAARLRMLCTLS
For a READ_MULT request specifying ECONTROLS on the TYPE parameter, the list entry controls of the first list entry read. This field is mapped by the IXLYLCTL macro and is valid whether the request completes successfully or prematurely.

MOVE: Moving a List Entry

You can move a list entry to another list or to another position on the same list.

See also IXLLSTE for information about moving a list entry.

Understanding the MOVE Operations

There are four types of move operations:

- **DATAOPER=NONE**

  Move a list entry.
- **DATAOPER=READ**

  Move a list entry and read data from it.
- **DATAOPER=WRITE with ENTRYTYPE=OLD**

  Move a list entry and write data to it.
- **DATAOPER=WRITE with ENTRYTYPE=ANY**

  Move a list entry and write data to it. If no existing list entry meets the specified selection criteria, create a new list entry at the target list position.

Note: List entries can be moved only within the same list structure.

Guide to the Topic

"MOVE: Moving a List Entry" on page 550 is divided into five section s:

1. The first section provides information about input parameters applicable to all MOVE requests.
   - "Specifying the List Entry to be Moved" on page 551
   - "Specifying the Target List and List Position" on page 551
   - "Receiving or Passing Data on a MOVE Request" on page 553
   - "Requesting a Lock Operation as Part of a Move Request" on page 553
2. The second section , "Moving a List Entry Without Performing a Read or Write Operation" on page 553, describes how to perform a MOVE request without a read or write operation.
3. The third section , "Performing a Write Operation as part of a MOVE Request" on page 554, describes how to perform a write operation as part of a MOVE request.
4. The fourth section , "Performing a Read Operation as Part of a Move Request" on page 554, describes how to perform a read operation as part of a MOVE request.

5. The fifth section , “Receiving Answer Area Information from a MOVE Request” on page 555, lists the answer area information returned from all MOVE requests.

A Note about Terminology:

- The **source list** is the list containing the target list entry before the list service performs the MOVE operation. The list that is searched for the target list entry before creating a new one, is also considered the source list.
- The **target list** is the list that receives the moved list entry or the newly created one. A single list can be both the source and target list, as is the case when a list entry is created as part of a MOVE request.
- The **target list entry** is the list entry to be moved or created.

Specifying the List Entry to be Moved

You can use any of the following to identify the target list entry:

- To specify the head or tail entry on a particular list, code the list position (LISTPOS) and the list number (LISTNUM).
- To specify a particular entry regardless of where it resides in the list structure, code one of the following:
  - The entry name (ENTRYNAME), for named entries only.
  - The entry ID (ENTRYID).

  You can code the LISTNUM parameter to stipulate that the selected entry must reside on a certain list.
- To specify a keyed list entry at the head or tail of a sublist of list entries with the same key, code the list entry's key (ENTRYKEY), the position on the sublist (LISTPOS), and the list number (LISTNUM).
- To specify the list entry associated with the list cursor for a certain list, code the LOCBYCURSOR and LISTNUM parameters.

If you omit the LISTPOS parameter, the default value is HEAD. So in effect, there is always a value for LISTPOS if one is needed.

If you specify both the ENTRYID parameter and the ENTRYNAME parameter, the list service uses the value of ENTRYID to locate the list entry to be moved, and the value of ENTRYNAME to assign the entry name if a new list entry is created.

List Cursor Placement on a MOVE Request

See “Understanding the List Cursor” on page 494 for information about using the list cursor.

Specifying the Target List and List Position

The MOVETOLIST parameter identifies the list number of the target list.

For a list structure with entry names

The list service places the list entry at the head or tail of the target list as specified by the MOVETOPOS parameter. If you specify MOVETOPOS=HEAD, the list service places the list entry at the head of the list. If you specify MOVETOPOS=TAIL, the list service places the list entry at the tail of the list.

For a list structure with entry keys

You can assign an entry key to the entry being moved or created in one of two ways:

1. Use the MOVETOKEY parameter to specify the key to be assigned. The ENTRYKEY parameter specifies the key value the target list entry must currently have to be selected for processing.
2. For list structures allocated in a CFLEVEL=1 or higher coupling facility, you can use the list key value associated with the list to set the list entry key. See “Understanding List Entry Key Assignment” on page 490.

The flowchart in Figure 66 on page 552 shows how the key value is assigned when a keyed entry is moved or created using REQUEST=MOVE and automatic list key assignment is not being used (LISTKEYTYPE=NOLISTKEY). Once the list entry key is assigned, the list service places the list entry on the target list as follows:

- If the list entry's key is unique within the list, the list entry is positioned to maintain the ascending order of the entry keys.
- If other list entries on the list have the same key, the value of the MOVETOPOS parameter determines whether to add the list entry to the head or tail of the sublist of list entries with matching keys.

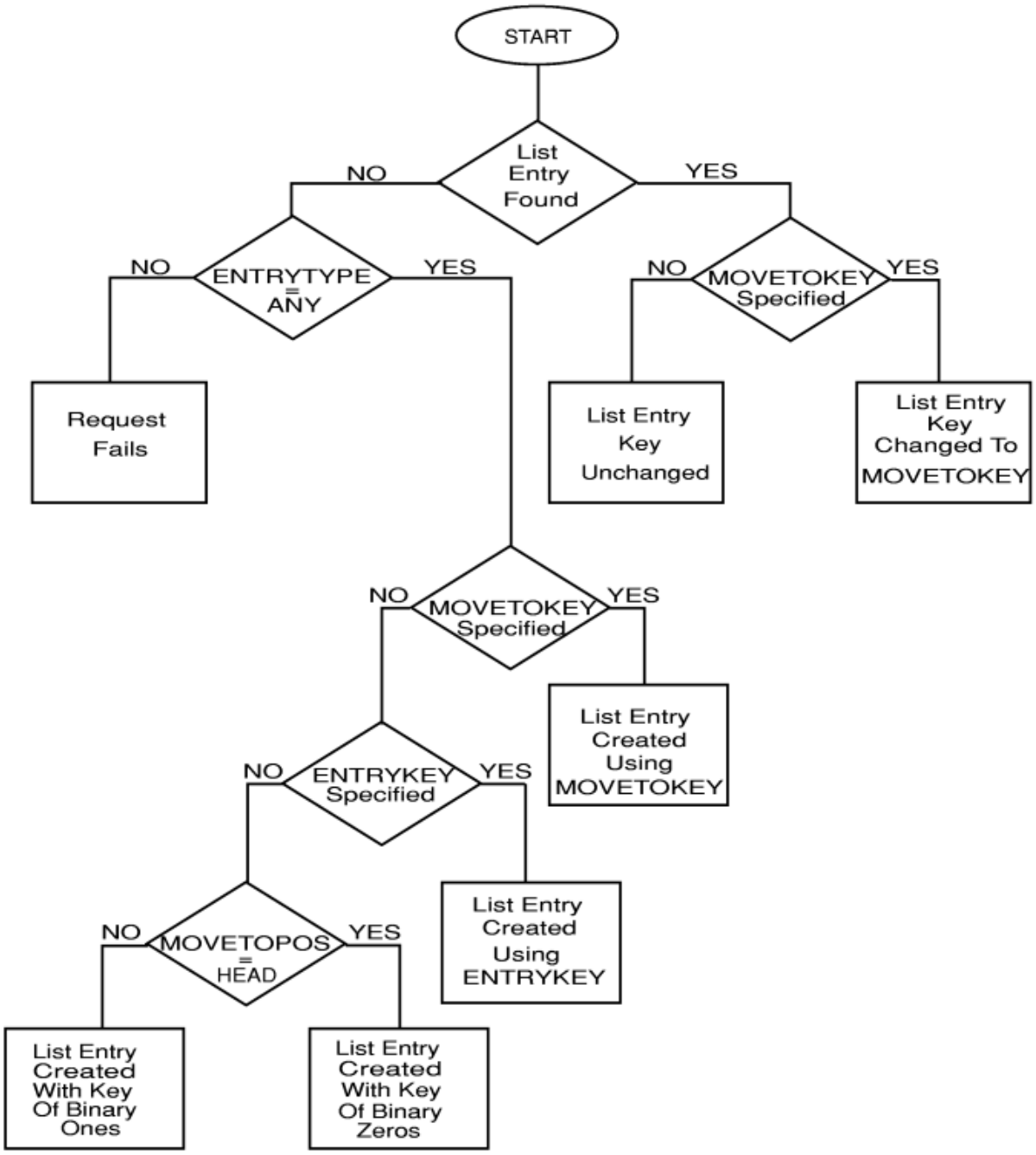


Figure 66. List Entry Key Resulting from a MOVE Request

For a list structure with neither entry names nor entry keys

The list service places the list entry at the head or tail of the target list as described for a list structure with entry names.

Receiving or Passing Data on a MOVE Request

See "Selecting the Buffer Format" on page 524 for a description of the buffer format options and their performance considerations.

Specifying the List Entry Version Number on a MOVE Request

See "Understanding the List Entry Version Number" on page 523 for information about:

- Using the list entry version number to select the list entry to be moved
- Updating the list entry version number on a MOVE request.

Specifying the List Authority Value on a MOVE Request

See "Understanding the List Authority Value" on page 513 for information about:

- Using the list authority value to select an entry for processing
- Updating the list authority value.

Requesting Automatic Key Assignment on a MOVE Request

For information about requesting automatic key assignment when moving a list entry, see "Understanding List Entry Key Assignment" on page 490.

Requesting a Lock Operation as Part of a Move Request

To perform a serialized MOVE operation, one in which the list service performs a lock operation together with a MOVE request, specify the LOCKOPER parameter on the IXLLIST macro. If the list service cannot perform both the lock operation and the move operation, it performs neither and fails the request.

You can specify the following LOCKOPER values on a MOVE request:

- SET
- RESET
- NOTHELD
- HELDBY

See "LOCK: Performing a Lock Operation" on page 568 for detailed information about the LOCKOPER parameter.

Moving a Keyed List Entry in a CFLEVEL=3 or Higher Coupling Facility

If your request is to move a keyed list entry in a coupling facility of CFLEVEL=3 or higher from one sublist to another sublist, the following scenarios apply:

- The request is to move the keyed list entry to an empty sublist and the target sublist is being monitored.
  - The system queues all EMCs associated with the target sublist to the respective users' event queues.
- The request is to move the keyed list entry from a sublist thus causing the source sublist to become empty.
  - The system withdraws all EMCs assocated with the source sublist from the event queues.

Moving a List Entry Without Performing a Read or Write Operation

For a MOVE request without a read or write operation, specify DATAOPER=NONE, which is the default.

Performing a Read Operation as Part of a Move Request

To perform a read operation as part of the move request, specify DATAOPER=READ. You specify the type of list entry information you want read by coding the parameter for the storage area to receive the information. Specify:

- BUFFER or BUFLIST to read data entry data.
- ADJAREA to read adjunct area data.

Performing a Write Operation as part of a MOVE Request

To perform a write operation as part of the move request, specify DATAOPER=WRITE with ENTRYTYPE=OLD or ENTRYTYPE=ANY.

- Specify ENTRYTYPE=OLD to update the list entry's data entry with the data in the buffer specified by BUFFER or BUFLIST, the adjunct data with the data in the storage area specified by ADJAREA, or both.
- Specify ENTRYTYPE=ANY to request the following:
  - If the target list entry exists, move it and update it as described for ENTRYTYPE=OLD.
  - If the target list entry does not exist, create a new list entry at the target position, containing the data entry data in the buffer specified by BUFFER or BUFLIST, the adjunct data in the storage specified by ADJAREA, or both.

Creating a New List Entry as Part of a MOVE Request

If you specify ENTRYTYPE=ANY, your request might cause a new list entry to be created. You can either provide the information necessary to create and position the new list entry or have the list service position the new list entry according to its default values.

For a list structure with entry names: You must provide a list entry name (ENTRYNAME) for the list service to use if it creates a new list entry for you.

The list service uses the value of LISTPOS to determine whether to place the new list entry at the head or tail of the target list.

For a list structure with entry keys: If you specify the MOVETOKEY parameter, the new list entry is assigned the key specified by MOVETOKEY. If you omit the MOVETOKEY parameter but specify the ENTRYKEY parameter, the new list entry is assigned the key specified by ENTRYKEY. If you omit both MOVETOKEY and ENTRYKEY, the new list entry is assigned a key based on the value of MOVETOPOS. If MOVETOPOS=HEAD, the key is set to binary zeros. If MOVETOPOS=TAIL, the key is set to binary ones.

For a keyed list structure in a CFLEVEL=3 or higher coupling facility: If your MOVE request specifies DATAOPER=WRITE and ENTRYTYPE=ANY for a keyed list structure in a CFLEVEL=3 or higher coupling facility and the target sublist is empty,

- The target sublist transitions to nonempty
- The system queues all EMCs associated with the target sublist to the respective users' event queues.

For list structures allocated in a CFLEVEL=1 or higher coupling facility, you can use the list key value associated with the list to set the list entry key. See “Understanding List Entry Key Assignment” on page 490.

See Figure 66 on page 552 for a flowchart illustrating some key assignment rules.

For More Information

The following information discussed under “WRITE: Writing to a List Entry” on page 531 also applies to REQUEST=MOVE with DATAOPER=WRITE:

- “Specifying the Size of the Data Entry to Hold the Data” on page 532

Receiving Answer Area Information from a MOVE Request

When you invoke IXLLIST, list services return information related to your request in the answer area specified using the ANSLEN and ANSAREA parameters.

Under certain circumstances, answer area information is not valid. See "Determining if the Answer Area is Valid" on page 535 for information on how to determine whether the answer area information is valid.

The following list describes the information returned when the answer area is valid. The answer area is mapped by the IXLYLAA macro, which is presented in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

LAARETCODE
The return code from the IXLLIST service. Return code values are defined in the IXLYCON macro.

LAARSNCODE
The reason code associated with the return code from the IXLLIST service. Reason code values are defined in the IXLYCON macro.

LAALCTL
The list entry controls for the list entry. The area is mapped by IXLYLCTL.

For a successful MOVE request, the list entry controls for the list entry that was moved or created. This area is mapped by IXLYLCTL.

For a MOVE request that failed because the target list entry did not meet the list number or version number criteria specified by LISTNUM or VERSCOMP, the list entry controls for the list entry that failed to meet the selection criterion.

For a MOVE request that failed because the entry name (ENTRYNAME) specified for the new list entry already existed, the list entry controls for the list entry already having the specified name.

For a MOVE request with DATAOPER=READ that failed because the buffer was too small to hold the data to be read, the list entry controls for the list entry that caused the operation to fail.

LAALISTDESC
The user-specified description of the list. Returned for MOVE requests that failed because of an authority mismatch. Returned only for structures allocated in a coupling facility with CFLEVEL=1 or higher.

LAALISTAUTH
The list authority for the list. Returned for MOVE requests that failed because of an authority mismatch. Returned only for structures allocated in a coupling facility with CFLEVEL=1 or higher.

LAALISTKEY
The current value of the list key from the list controls. Returned for MOVE requests that failed because the maximum list key value would be exceeded. Returned only for structures allocated in a coupling facility with CFLEVEL=1 or higher.

LAAMAXLISTKEY
The current value of the maximum list key from the list controls. Returned for MOVE requests that failed because the maximum list key value would be exceeded. Returned only for structures allocated in a coupling facility with CFLEVEL=1 or higher.

LAATOTALCNT
The total number of list entries in use in the structure. Returned for requests that completed successfully.

LAATOTALELECNT
The total number of data elements in use in the structure. Returned for requests that completed successfully.

LAALISTCNT
The number of list entries or data elements on the list to which the list entry was moved or the list which received the new list entry if DATAOPER=WRITE was specified with ENTRYTYPE=ANY and an entry was created. Returned for requests that completed successfully. This count includes the number of list entries or data elements on the list that currently reside in coupling facility real and storage

class memory. The value specified for LISTCNTLTYPE on the IXLCONN macro when the list structure was allocated determines whether this field represents a count of list entries or data elements. LAALISTCNTLTYPE found in the answer area can also be used to determine whether LAALISTCNT represents a count of list entries or data elements.

LAALISTOPPCNT
The number of list entries or data elements on the list that was the target of the move operation. Valid when LAALISTOPPCNTVALID is ON. When LaaListCntlType is set to 1, LaaListOppCnt is expressed as the number of data elements on the list. When LAALISTCNTLTYPE is set to 2, LaaListOppCnt is expressed as the number of list entries on the list. Like LAALISTCNT, this count includes the number of list entries or elements for the processed list that currently reside in coupling facility real and storage class memory.

LAALISTOPPCNTVALID
Opposite List Count Valid indicator. A flag that can be returned for requests that completed successfully. The flag indicates when LAALISTOPPCNT is valid for use.

LAALISTCNTLTYPE
List Control Type. A value returned for requests that completed successfully. The value is also returned when the list that is designated as the target of a single list entry move or the move of a list of entries (IXLLSTM REQUEST=MOVE_ENTRYLIST) request cannot accommodate more entries or elements (LAARSNCODE=IxlRsnCodeListFull). The value is set to 1 (IXLLISTCNTLTYPEENTRY) when ENTRY is specified for LISTCNTLTYPE on the IXLCONN macro when the list structure was allocated. LAALISTCNT is expressed in number of list entries. When LAALISTOPPCNTVALID is ON, LAALISTOPPCNT is expressed in number of data elements.

The value is set to 2 (IXLLISTCNTLTYPEELEMENT) when ELEMENT is specified for LISTCNTLTYPE on the IXLCONN macro when the list structure was allocated. LAALISTCNT is expressed in number of data elements. When LAALISTOPPCNTVALID is ON, LAALISTOPPCNT is expressed in number of list entries.

A value of zero (0) indicates that this field is not provided by list services for the request.

LAALISTFULLLIMIT
The maximum number of entries or elements that can be placed on the designated target list of a move request. It is returned when the list designated as the target of a move or a move of a list of entries request cannot accommodate more entries or elements (LAARSNCODE=IxlRsnCodeListFull). LAALISTFULLLIMIT is valid when the structure is allocated in a CFLEVEL=22 or higher coupling facility.

LAALISTFULLCNT
The count of in-use entries or elements that reside on a list designated as the target of a move or a move of a list of entries and the target list cannot accommodate more entries or elements (LAARSNCODE=IxlRsnCodeListFull). This count includes the number of list entries or elements on the target list that currently reside in coupling facility real and storage class memory. LAALISTFULLCNT is valid when the structure is allocated in a CFLEVEL=22 or higher coupling facility.

LAACONID
For a MOVE request specifying the LOCKOPER parameter, LAACONID contains, under the following circumstances:

- HELDBY parameter specified and the lock was not held by the connection specified by LOCKCOMP or taken as the default.
- NOTHELD parameter specified with LOCKMODE=COND and the request failed because the lock is held by another connection.
- NOTHELD parameter specified with LOCKMODE=UNCOND and the request failed because the lock is held by a failed persistent connection.
- SET parameter specified with LOCKMODE=COND and the request failed because the lock is held by another connection.
- SET parameter specified with LOCKCOMP and the lock was not held by the specified connection.

- SET parameter specified with LOCKMODE=UNCOND and the request failed because the lock is held by a failed persistent connection.
- RESET parameter specified without LOCKCOMP and the request failed because you do not hold the lock.
- RESET parameter specified with LOCKCOMP and the lock was not held by the specified connection.

Either of the following:

- The connection identifier of the lock owner, if the lock specified by the LOCKINDEX parameter is held
- Zeros, if the lock specified by the LOCKINDEX parameter is free.

LAAENTRYCREATED
Flag to indicate that the request created a new entry. Returned for successful MOVE requests when DATAOPER=WRITE is specified. Returned only for structures allocated in a coupling facility with CFLEVEL=1 or higher.

DELETE, DELETE_MULT, DELETE_ENTRYLIST: Deleting List Entries

You can perform three types of delete operations on entries in a list structure:

REQUEST=DELETE
Delete a single list entry.

See also IXLLSTE for information about deleting a single list entry.

REQUEST=DELETE_MULT
Delete all list entries in the structure or only those:

- On a certain list
- With a version number that succeeds on a version number comparison, using VERSCOMP and VERSCOMPTYPE
- With a list authority value that succeeds on a list authority comparison, using AUTHCOMP and AUTHCOMPTYPE
- Any combination of the above.

See also IXLLSTM for information about deleting list entries from multiple lists.

REQUEST=DELETE_ENTRYLIST
Delete the list entries specified in a list of entry names or entry IDs passed as input.

See also IXLLSTM for information about deleting list entries passed in a list of entry names or entry IDs.

If the list structure uses list entry names, you can reuse the entry names of deleted list entries. List entry IDs, which are assigned to list entries by list services, are unique for the life of the list structure and are not reused.

Guide to the Topic

"DELETE, DELETE_MULT, DELETE_ENTRYLIST: Deleting List Entries" on page 557 is divided into three section s:

- "DELETE: Deleting a Single List Entry" on page 557
- "DELETE_MULT: Deleting Multiple List Entries" on page 560
- "DELETE_ENTRYLIST: Deleting a List of Entries" on page 562

DELETE: Deleting a Single List Entry

The DELETE request allows you to delete a single list entry or read a list entry and delete it.

DATAOPER=NONE
Indicates that the list entry is not to be read.

DATAOPER=READ
Reads the list entry and deletes it. You can read either or both of the following:

- Data entry data into the buffer specified by BUFFER or BUFLIST
- Adjunct area data into the storage specified by ADJAREA.

Requesting a Lock Operation as Part of a DELETE Request

To perform a serialized DELETE operation, one in which the list service performs a lock operation together with a DELETE request, specify the LOCKOPER parameter on the IXLLIST macro. If the list service cannot perform both the lock operation and the delete operation, it performs neither and fails the request.

You can specify the following LOCKOPER values on a DELETE request:

- SET
- RESET
- NOTHELD
- HELDBY

See “LOCK: Performing a Lock Operation” on page 568 for detailed information about the LOCKOPER parameter.

Specifying the List Entry to be Deleted

You can designate the list entry to be deleted in several ways:

- To specify the head or tail entry on a particular list, code the list position (LISTPOS) and the list number (LISTNUM).
- To specify a particular entry regardless of where it resides in the list structure, code one of the following:
  - The entry name (ENTRYNAME), for named entries only.
  - The entry ID (ENTRYID).

  You can code the LISTNUM parameter to stipulate that the selected entry must reside on a certain list.
- To specify a keyed list entry at the head or tail of a sublist of list entries with the same key, code the list entry's key (ENTRYKEY), the position on the sublist (LISTPOS), and the list number (LISTNUM).
- To specify the list entry associated with the list cursor for a certain list, code the LOCBYCURSOR and LISTNUM parameters.

  See “Understanding the List Cursor” on page 494 for information about using the list cursor.

If you omit the LISTPOS parameter, the default value is HEAD. So in effect, there is always a value for LISTPOS if one is needed.

Specifying the List Entry Version Number on a DELETE Request

For information about using the list entry version number to select the list entry to be deleted, see “Understanding the List Entry Version Number” on page 523.

Specifying the List Authority Value on a DELETE Request

For information about using the list authority value to select an entry for processing, see “Understanding the List Authority Value” on page 513.

Deleting a Keyed List Entry in a CFLEVEL=3 or Higher Coupling Facility

If your delete request is to delete the last keyed list entry from one or more monitored sublists in a coupling facility of CFLEVEL=3 or higher, the following occurs:

- The sublist(s) transition from nonempty to empty.
- The system withdraws all EMCs associated with the sublist(s) from the event queues.

Receiving Data on a DELETE Request

See "Selecting the Buffer Format" on page 524 for a description of the buffer format options and their performance considerations.

Receiving Answer Area Information from a DELETE Request

When you invoke IXLLIST, list services return information related to your request in the answer area specified using the ANSLEN and ANSAREA parameters.

Under certain circumstances, answer area information is not valid. See "Determining if the Answer Area is Valid" on page 535 for information on how to determine whether the answer area information is valid.

The following list describes the information returned when the answer area is valid. The answer area is mapped by the IXLYLAA macro, which is presented in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

LAARETCODE
The return code from the IXLLIST service. Return code values are defined in the IXLYCON macro.

LAARSNCODE
The reason code associated with the return code from the IXLLIST service. Reason code values are defined in the IXLYCON macro.

LAALCTL
The list entry controls for the list entry. The area is mapped by IXLYLCTL.

For a successful DELETE request, the list entry controls for the list entry that was deleted. This area is mapped by IXLYLCTL.

For a DELETE request that failed because the target list entry did not meet the list number or version number criteria specified by LISTNUM or VERSCOMP, the list entry controls for the list entry that failed to meet the selection criterion.

For a DELETE request with DATAOPER=READ that failed because the buffer was too small to hold the data to be read, the list entry controls for the list entry that caused the operation to fail.

LAALISTDESC
The user-specified description of the list. Returned for DELETE requests that failed because of a list authority mismatch. Returned only for structures allocated in a coupling facility with CFLEVEL=1 or higher.

LAALISTAUTH
The list authority for the list. Returned for DELETE requests that failed because of a list authority mismatch. Returned only for structures allocated in a coupling facility with CFLEVEL=1 or higher.

LAATOTALCNT
The total number of list entries in use in the structure. Returned for requests that completed successfully.

LAATOTALELECNT
The total number of data elements in use in the structure. Returned for requests that completed successfully.

LAALISTCNT
The number of list entries or data elements on the list that was the target of the DELETE operation. Returned for requests that completed successfully. This count includes the number of list entries or data elements on the list that currently reside in coupling facility real and storage class memory. The value specified for LISTCNTLTYPE on the IXLCONN macro when the list structure was allocated determines whether this field represents a count of list entries or data elements. LAALISTCNTLTYPE found in the answer area can also be used to determine whether LAALISTCNT represents a count of list entries or data elements.

LAALISTOPPCNT
The number of list entries or data elements on the list that was the target of the DELETE operation. Valid when LAALISTOPPCNTVALID is ON. When LaaListCntlType is set to 1, LaaListOppCnt is expressed as the number of data elements on the list. When LAALISTCNTLTYPE is set to 2, LaaListOppCnt is expressed as the number of list entries on the list. Like LAALISTCNT, this count includes the number of list entries or elements for the processed list that currently reside in coupling facility real and storage class memory.

LAALISTOPPCNTVALID
Opposite List Count Valid indicator. A flag that can be returned for requests that completed successfully. The flag indicates when LAALISTOPPCNT is valid for use.

LAALISTCNTLTYPE
List Control Type. A value returned for requests that completed successfully. The value is set to 1 (IXLLISTCNTLTYPEENTRY) when ENTRY is specified for LISTCNTLTYPE on the IXLCONN macro when the list structure was allocated. LAALISTCNT is expressed in number of list entries. When LAALISTOPPCNTVALID is ON, LAALISTOPPCNT is expressed in number of data elements.

The value is set to 2 (IXLLISTCNTLTYPEELEMENT) when ELEMENT is specified for LISTCNTLTYPE on the IXLCONN macro when the list structure was allocated. LAALISTCNT is expressed in number of data elements. When LAALISTOPPCNTVALID is ON, LAALISTOPPCNT is expressed in number of list entries.

A value of zero (0) indicates that this field is not provided by list services for the request.

LAACONID
For a DELETE request specifying the LOCKOPER parameter, LAACONID contains, under the following circumstances:

- HELDBY parameter specified and the lock was not held by the connection specified by LOCKCOMP or taken as the default.
- NOTHELD parameter specified with LOCKMODE=COND and the request failed because the lock is held by another connection.
- NOTHELD parameter specified with LOCKMODE=UNCOND and the request failed because the lock is held by a failed persistent connection.
- SET parameter specified with LOCKMODE=COND and the request failed because the lock is held by another connection.
- SET parameter specified with LOCKCOMP and the lock was not held by the specified connection.
- SET parameter specified with LOCKMODE=UNCOND and the request failed because the lock is held by a failed persistent connection.
- RESET parameter specified without LOCKCOMP and the request failed because you do not hold the lock.
- RESET parameter specified with LOCKCOMP and the lock was not held by the specified connection.

Either of the following:

- The connection identifier of the lock owner, if the lock specified by the LOCKINDEX parameter is held
- Zeros, if the lock specified by the LOCKINDEX parameter is free

DELETE_MULT: Deleting Multiple List Entries

With the DELETE_MULT request you can delete multiple list entries in the structure and filter their selection by list number, version number, list authority value, entry key value, or any combination of these filters.

The order in which list entries are deleted is unrelated to the order of the list entries in the structure. Because of this, you can't tell whether an entry, added after the scan has begun, will be deleted.

For more information about the list entry version number, see “Understanding the List Entry Version Number” on page 523.

For more information about the list authority value, see “Understanding the List Authority Value” on page 513.

For more information about using the entry key to select entries for processing, see “Using the Entry Key in Multiple List Operations” on page 493.

Requesting a Lock Operation as Part of a DELETE_MULT Request

To perform a serialized DELETE_MULT operation, one in which a lock operation is performed before performing a DELETE_MULT request, specify the LOCKOPER parameter on the IXLLIST macro. If the list service cannot perform both the lock operation and the DELETE_MULT operation, it performs neither and fails the request.

You can specify the following LOCKOPER values on a DELETE_MULT request:

- NOTHELD
- HELDBY

See “LOCK: Performing a Lock Operation” on page 568 for detailed information about the LOCKOPER parameter.

If your serialization protocol permits lock stealing, you can use either of these lock operations to ensure that your delete request is performed only if the specified lock is in the state you expect.

Handling an Incompletely Processed DELETE_MULT Request

A DELETE_MULT request can time out before finishing all its processing. If this happens, IXLLIST:

- Sets the IXLLIST return code to IXLRETCODEWARNING and the reason code to IXLRSNCODETIMEOUT to indicate that processing did not complete.
- Returns in the LAADELCNT field of the answer area, the count of list entries that have been deleted on that invocation.
- Returns in the LAARESTOKEN or LAAEXTRESTOKEN field of the answer area, a restart token to be provided when you reissue the request to continue the scan.

To reissue the DELETE_MULT request, access the restart token returned in the LAARESTOKEN or LAAEXTRESTOKEN field of the answer area and specify the token with the RESTOKEN or EXTRESTOKEN parameter on the next DELETE_MULT invocation. Continue to reissue the request until the return code indicates that all processing has completed.

If you do not have exclusive access to the list structure, it could be modified by other users between the time you issue the DELETE_MULT request and the time you reissue it. Since the DELETE_MULT request uses a restart token instead of the next entry's list controls to indicate where scanning should resume, scanning can resume successfully even if a particular list entry is moved or deleted.

You can avoid coding a separate IXLLIST invocation with the RESTOKEN or EXTRESTOKEN parameter to handle incomplete processing, by coding a single IXLLIST invocation with the restart token initialized to zero for the first time through. A restart token of zero causes IXLLIST to treat the invocation as a new request. When reissuing the request due to premature completion, be sure to first set the restart token to the value from the LAARESTOKEN or LAAEXTRESTOKEN field.

Receiving Answer Area Information from a DELETE_MULT Request

When you invoke IXLLIST, list services return information related to your request in the answer area specified using the ANSLEN and ANSAREA parameters.

Under certain circumstances, answer area information is not valid. See “Determining if the Answer Area is Valid” on page 535 for information on how to determine whether the answer area information is valid.

The following list describes the information returned when the answer area is valid. The answer area is mapped by the IXLYLAA macro, which is presented in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

LAARETCODE
The return code from the IXLLIST service. Return code values are defined in the IXLYCON macro.

LAARSNCODE
The reason code associated with the return code from the IXLLIST service. Reason code values are defined in the IXLYCON macro.

LAALISTDESC
The user-specified description of the list. Returned for DELETE_MULT requests that failed because of a list authority mismatch. Returned only for structures allocated in a coupling facility with CFLEVEL=1 or higher.

LAALISTAUTH
The list authority for the list. Returned for DELETE_MULT requests that failed because of a list authority mismatch. Returned only for structures allocated in a coupling facility with CFLEVEL=1 or higher.

LAADELCNT
The number of list entries deleted on the invocation that just completed. Returned for request that completed successfully or prematurely. If no scanned list entries met the criteria specified for selection to be deleted, the count is set to zero. If the request completed prematurely, there might be additional list entries that meet the selection criteria that have not yet been scanned.

LAACONID
For a DELETE_MULT request specifying the LOCKOPER parameter, LAACONID contains, under the following circumstances:

- NOTHELD parameter specified and the request failed because the lock is held by another connection.
- HELDBY parameter specified with or without the LOCKCOMP parameter and the lock was not held by the specified connection.

Either of the following:

- The connection identifier of the lock owner, if the lock specified by the LOCKINDEX parameter is held
- Zeros, if the lock specified by the LOCKINDEX parameter is free.

LAARESTOKEN
The restart token for the request. Returned if ALLOWAUTO=NO was specified or defaulted to on IXLCONN and the DELETE_MULT request completed prematurely.

LAAEXTRESTOKEN
The extended restart token for the request. Returned if ALLOWAUTO=YES was specified on IXLCONN to connect to the structure and the DELETE_MULT request completed prematurely.

DELETE_ENTRYLIST: Deleting a List of Entries

Use the DELETE_ENTRYLIST request to delete the list entries identified in a list of entry names or entry IDs you provide as input. A list of entry names is only valid if the list structure uses entry names.

The LISTTYPE parameter indicates whether you are providing a list of entry names or entry IDs:

LISTTYPE=IDLIST
Indicates a list of entry IDs.

LISTTYPE=NAMELIST
Indicates a list of entry names.

The FIRSTELEM and LASTELEM parameters specify the index of the first and last entry in the list, respectively, which are to be processed for the request.

You can use the LISTNUM parameter to specify that the list entries should be deleted only if they reside on a certain list. You can use additional filtering of entries by version number and/or list authority value. For a list of entry IDs, you also can filter entries by entry key.

- For information about using the list entry version number to select the list entries to be deleted, see "Understanding the List Entry Version Number" on page 523.
- For information about selecting entries for processing by list authority value, see "Understanding the List Authority Value" on page 513.
- For information about using the entry key to select entries for processing, see "Using the Entry Key in Multiple List Operations" on page 493.

Passing the List of Entries to be Deleted

See "Selecting the Buffer Format" on page 524 for a description of the buffer format options and their performance considerations.

For LISTTYPE=IDLIST, the buffer(s) containing the list of entry IDs should be formatted as an array of 12-byte fields, each containing an entry ID. Note that IDLIST entries can be split across buffers if necessary.

For LISTTYPE=NAMELIST, the buffer(s) containing the list of entry names should be formatted as an array of 16-byte fields, each containing an entry name.

Requesting a Lock Operation as Part of a DELETE_ENTRYLIST Reques

To perform a serialized DELETE_ENTRYLIST operation, one in which a lock operation is performed before performing a DELETE_ENTRYLIST request, specify the LOCKOPER parameter on the IXLLIST macro. If the list service cannot perform both the lock operation and the DELETE_ENTRYLIST operation, it performs neither and fails the request.

You can specify the following LOCKOPER values on a DELETE_ENTRYLIST request:

- NOTHELD
- HELDBY

See "LOCK: Performing a Lock Operation" on page 568 for detailed information about the LOCKOPER parameter.

If your serialization protocol permits lock stealing, you can use either of these lock operations to ensure that your delete request is performed only if the specified lock is in the state you expect.

Handling an Incompletely Processed DELETE_ENTRYLIST Request

If IXLLIST cannot find a list entry list you have specified in your entry list, processing ends prematurely and the entry list index of the entry that couldn't be found is returned in the LAAFAILINDEX field of the answer area. To continue processing, reissue the request starting with the entry after the one that couldn't be found (LAAFAILINDEX+1).

A DELETE_ENTRYLIST request can also time out before finishing all its processing. If this happens, IXLLIST:

- Sets the IXLLIST return code to IXLRETCODEWARNING and the reason code to IXLRSNCODETIMEOUT to indicate that processing did not complete.
- Returns in the LAADELCNT field of the answer area, the count of list entries that have been deleted on that invocation.
- Returns in the LAAFAILINDEX field of the answer area, the index of the first unprocessed entry in the list of entry names or entry IDs you have specified for deletion.

All entries preceding the failing list entry will have been deleted and all entries beginning with the failed entry will still remain to be processed. To continue processing, reissue the DELETE_ENTRYLIST request

using the index value returned in LAAFAILINDEX as the value of the FIRSTELEM parameter. Continue reissuing the request until the return code indicates that all the processing has completed.

Receiving Answer Area Information from a DELETE_ENTRYLIST Request

When you invoke IXLLIST, list services return information related to your request in the answer area specified using the ANSLEN and ANSAREA parameters.

Under certain circumstances, answer area information is not valid. See "Determining if the Answer Area is Valid" on page 535 for information on how to determine whether the answer area information is valid.

The following list describes the information returned when the answer area is valid. The answer area is mapped by the IXLYLAA macro, which is presented in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

LAARETCODE
The return code from the IXLLIST service. Return code values are defined in the IXLYCON macro.

LAARSNCODE
The reason code associated with the return code from the IXLLIST service. Reason code values are defined in the IXLYCON macro.

LAALISTDESC
The user-specified description of the list. Returned for DELETE_ENTRYLIST requests that failed because of a list authority mismatch. Returned only for structures allocated in a coupling facility with CFLEVEL=1 or higher.

LAALISTAUTH
The list authority for the list. Returned for DELETE_ENTRYLIST requests that failed because of a list authority mismatch. Returned only for structures allocated in a coupling facility with CFLEVEL=1 or higher.

LAADELCNT
The number of list entries deleted on the invocation that just completed. Returned for request that completed successfully or prematurely.

LAACONID
For a DELETE_ENTRYLIST request specifying the LOCKOPER parameter, LAACONID contains, under the following circumstances:

- NOTHELD parameter specified and the request failed because the lock is held by another connection.
- HELDBY parameter specified with or without the LOCKCOMP parameter and the lock was not held by the specified connection.

either of the following:

- The connection identifier of the lock owner, if the lock specified by the LOCKINDEX parameter is held
- Zeros, if the lock specified by the LOCKINDEX parameter is free.

LAAFAILINDEX
Under the circumstances listed below, contains an index into the entry list specified by IDLIST or NAMELIST:

- If the DELETE_ENTRYLIST request failed because one of the entries could not be found, contains the index of the list entry that could not be found.
- If the DELETE_ENTRYLIST request ended prematurely, contains the index of the list entry to be processed next.

READ_LCONTROLS: Reading List Controls

Use the READ_LCONTROLS request to obtain the list control information for a specific list. The list service returns the following list control information in the answer area specified using the ANSLEN and ANSAREA parameters.

- The list limit (maximum allowable number of list entries or data elements on the list)
- The current number of list entries or data elements on the list
- The approximate number of times the list has changed from empty to nonempty
- The user-specified list description
- The user-defined list authority value
- The number of entries in the array of list monitoring information associated with the list. The list monitoring information itself is returned in the buffer specified by BUFFER or BUFLIST
- The value of the list cursor (entry ID to which it points or zero)
- The list key value (when CFLEVEL=1 or higher)
- The maximum list key value (when CFLEVEL=1 or higher)
- The list cursor direction (when CFLEVEL=1 or higher).

See also IXLLSTC for information about reading list controls.

Obtaining List Monitoring Information

Each list has associated with it an array of list monitoring entries. Each list monitoring entry describes the list monitoring activities associated with that list for a particular connection ID. A list monitoring entry is returned for each connection ID regardless of whether that connection ID represents an active user of the structure. List monitoring entries for unused connection IDs are set to zeros. Each list monitoring information entry is mapped by the IXLYLMI macro.

The array of list monitoring information entries is returned in the buffer specified by BUFFER or BUFLIST. When control returns from a READ_LCONTROLS request, the LAALMICNT field in the answer area contains the number of list monitoring information entries in the array. The entries are numbered from 0 to LAALMICNT-1 but **entry 0 does not contain list monitoring information** and should not be accessed. The list monitoring information, indexed by connection ID number, begins with entry 1.

To access information for a particular connection ID, use the ID number as an index into the array or scan the array from entry 1 to entry LAALMICNT-1. Each list monitoring information entry contains the following information about the associated connection ID:

- Whether the connection ID is monitoring the list
- Whether the connection ID is using a list transition exit to receive notification of this list's transitions.
- The vector index in the connection ID's list notification vector representing the list.

Receiving Answer Area Information from a READ_LCONTROLS Request

Under certain circumstances, answer area information is not valid. See "Determining if the Answer Area is Valid" on page 535 for information on how to determine whether the answer area information is valid.

The following list describes the information returned when the answer area is valid. The answer area is mapped by the IXLYLAA macro, which is presented in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

LAARETCODE
The return code from the IXLLIST service. Return code values are defined in the IXLYCON macro.

LAARSNCODE
The reason code associated with the return code from the IXLLIST service. Reason code values are defined in the IXLYCON macro.

LAARLCKEYRADDTALLY
The count of the total number of entries added to an established key-range for the list since the structure was allocated. LAARLCKEYRADDTALLY is valid when the structure is allocated in a CFLEVEL=22 or higher coupling facility and supports the use of entry keys.

LAARLCLISTADDTALLY
The count of the total number of entries added to the list since the structure was allocated. LAARLCLISTADDTALLY is valid when the structure is allocated in a CFLEVEL=22 or higher coupling facility.

LAARLCLISTCNT
The number of list entries or data elements on the list. Returned for requests that completed successfully. This count includes the number of list entries or data elements on the list that currently reside in coupling facility real and storage class memory. The value specified for LISTCNTLTYPE on the IXLCONN macro when the list structure was allocated determines whether this field represents a count of list entries or data elements. LAARLCLISTCNTLTYPE found in the answer area can also be used to determine whether LAALISTCNT represents a count of list entries or data elements.

LAARLCLISTOPPCNT
The number of list entries or data elements on the list. Returned for requests that completed successfully. Valid when LAARLCLISTOPPCNTVALID is ON. When LAARLCLISTCNTLTYPE is set to 1, LAARLCLISTLOPPCNT represents the number of data elements on the list. When LAARLCLISTCNTLTYPE is set to 2, LAARLCLISTOPPCNT represents the number of list entries on the list. Like LAARLCLISTCNT, this count includes the number of list entries or elements for the processed list that currently reside in coupling facility real and storage class memory.

LAARLCLISTOPPCNTVALID
Opposite List Count Valid indicator. A flag that can be returned for requests that completed successfully. The flag indicates when LAARLCLISTOPPCNT is valid for use.

LAARLCLISTCNTLTYPE
List Control Type. A value that is returned for requests that completed successfully. The value is set to 1 (IXLLISTCNTLTYPEENTRY) when ENTRY is specified for LISTCNTLTYPE on the IXLCONN macro when the list structure was allocated. LAARLCLISTCNT is expressed in number of list entries. When LAARLCLISTOPPCNTVALID is ON, LAARLCLISTOPPCNT is expressed in number of data elements.

The value is set to 2 (IXLLISTCNTLTYPEELEMENT) when ELEMENT is specified for LISTCNTLTYPE on the IXLCONN macro when the list structure was allocated. LAARLCLISTCNT is expressed in number of data elements. When LAARLCLISTOPPCNTVALID is ON, LAARLCLISTOPPCNT is expressed in number of list entries.

A value of zero (0) indicates that this field is not provided by list services for the request.

LAALMICNT
The count of list monitoring information entries returned. Returned for requests that completed successfully. List monitoring information entries are mapped by the IXLYLMI macro. The list monitoring entries are numbered from 0 to LAALMICNT-1.

LAALISTLIMIT
The maximum number of list entries permitted on the list or the maximum number of data elements permitted in the list structure (also called the list limit.) The choice of which type of limit to use is made using the LISTCNTLTYPE parameter on the IXLCONN macro when the structure is allocated. Returned for requests that completed successfully.

LAALISTDESC
The user-defined list description. Returned for requests that completed successfully.

LAALISTTRAN
The approximate number of transitions from empty to nonempty.

LAALISTAUTH
The user-defined list authority value.

LAALISTKEY
The list controls list key value. Returned for successful requests for structures allocated in a coupling facility with a CFLEVEL=1 or higher.

LAAMAXLISTKEY
The list controls maximum list key value. Returned for successful requests for structures allocated in a coupling facility with a CFLEVEL=1 or higher.

LAALISTCURSOR
The value of the list cursor (an entry ID or zero, if the list cursor is not set).

LAACURSORDIR
The list cursor direction. Returned for successful requests for structures allocated in a coupling facility with a CFLEVEL=1 or higher.

WRITE_LCONTROLS: Writing List Controls

Use the WRITE_LCONTROLS request with the following parameters to alter the list control information associated with a list:

NEWAUTH
Specifies a new list authority value.

LISTLIMIT
Specifies a new list limit.

LISTDESC
Specifies a new list description.

SETCURSOR
Sets the list cursor location and specifies the cursor direction (with CFLEVEL=1 or higher).

LISTKEY
Specifies the list key associated with the list (with CFLEVEL=1 or higher).

MAXLISTKEY
Specifies the maximum value for the list key associated with the list (with CFLEVEL=1 or higher).

Note: Your application can use the list authority value to implement a serialization mechanism (similar to compare and swap) for updating list controls. For structures allocated in a coupling facility with CFLEVEL=1 or higher, your application can also use the list authority value to serialize updates to entries on a list. See “Understanding the List Authority Value” on page 513.

If you are using the list authority value as a serialization mechanism, specify the current list authority value on the AUTHCOMP parameter when you issue WRITE_LCONTROLS to change a list control. If the AUTHCOMP value does not match the current list authority value, the request fails. You can obtain the current authority value using the READ_LCONTROLS request.

For more information about list controls, including their initial values when the list structure is allocated, see “Understanding List Controls” on page 511.

The list service returns information about the outcome of the request in the answer area specified by the ANSAREA and ANSLEN parameters.

See also IXLLSTC for information about altering list controls.

Changing the List Limit

You can change the list limit for a list at any time. If your new list limit allows fewer list entries or data elements (depending on which type of limit you have) than currently exist on the list, list services does not alter the list to conform to the new list limit you have set. However, any IXLLIST requests that try to increase the number of list entries or data elements on the list will fail until the request can be satisfied without exceeding the new limit.

Effect of Structure Alter on the List Limit

For structures that are allocated in a coupling facility with CFLEVEL=1 or higher, the IXLALTER function provides for the expansion or contraction of the size of a structure and/or for the reapportionment of the entry-to-element ratio of the structure. When the system processes an IXLALTER request for a list

structure, the list limit for each list in the structure is automatically adjusted only if one of the following conditions exist:

- You never set a list limit for the list
- You set a list limit for the list that is equal to the total number of list entries or data elements (depending on which type of limit you have).

In both of these cases, when the structure alter is initiated the list limit for the list is equal to the total number of list entries or data elements in the structure. As the alter process changes the total number of entries and elements, structure alter automatically adjusts the list limit to correspond to the changes made to the structure.

If neither condition exists (that is, you have explicitly set a list limit not equal to the total number of list entries or data elements in the structure), then structure alter does not automatically adjust the list limit. It is your responsibility, at the completion of structure alter processing, to set any new list limits that you require. Ensure that when doing so, you take into consideration any changes that were made to the structure's entry and element counts during the structure alter process.

Receiving Answer Area Information from a WRITE_LCONTROLS Request

Under certain circumstances, answer area information is not valid. See "Determining if the Answer Area is Valid" on page 535 for information on how to determine whether the answer area information is valid.

The following list describes the information returned when the answer area is valid. The answer area is mapped by the IXLYLAA macro, which is presented in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

LAARETCODE
The return code from the IXLLIST service. Return code values are defined in the IXLYCON macro.

LAARSNCODE
The reason code associated with the return code from the IXLLIST service. Reason code values are defined in the IXLYCON macro.

LAALISTDESC
The user-defined list description of the target list. Returned for requests that failed because the specified list authority value did not match that of the target list.

LAALISTAUTH
The user-defined list authority value of the target list. Returned for requests that failed because the specified list authority value did not match that of the target list.

LOCK: Performing a Lock Operation

Use the LOCK request with the LOCKOPER parameter to perform a lock operation on a lock table entry without performing any associated list entry operation. Lock operations are valid only for list structures that contain a lock table.

See also IXLLSTC for information about performing lock operations.

Selecting the Lock Operation

The LOCKOPER parameter specifies the lock operation to be performed. You can also code the LOCKOPER parameter on an IXLLIST request such as WRITE or READ, to perform a list entry operation together with a lock operation. The lock operations specified by the LOCKOPER parameter perform different functions depending on whether you specify a **comparative lock value** using the LOCKCOMP parameter.

Table 42 on page 569 lists the LOCKOPER functions with and without the LOCKCOMP parameter:

Table 42. List Structure Lock Operations		
Lock Operation	**With LOCKCOMP**	**Without LOCKCOMP**
SET	Transfer ownership of the lock to the requesting connection if the lock is currently held by the connection identified by LOCKCOMP (also known as lock stealing)	Obtain ownership of the specified lock
RESET	Free the specified lock if it is held by the connection identified by LOCKCOMP (another form of lock stealing)	Release ownership of the specified lock
NOTHELD	Not applicable	Perform the specified list operation (such as a read or write operation) only if the specified lock is free
HELDBY	Perform the specified list operation (such as a read or write operation) only if the lock is held by the connection identified by LOCKCOMP	Perform the specified list operation (such as a read or write operation) only if the specified lock is held by the requesting connection
TEST	Determine whether the specified lock is held by the connection identified by LOCKCOMP	Determine whether the requesting connection holds the specified lock
READNEXT	Return the lock table index of the next lock held by the connection identified by LOCKCOMP	Return the lock table index and connection ID associated with the next lock in the lock table that is held.

Lock Stealing: Specifying SET or RESET with the LOCKCOMP parameter allows you to steal a lock that is owned by connection. A lock steal request preempts any other outstanding requests for a particular lock. Lock stealing is intended for use primarily as part of connection failure recovery — to obtain a lock held by a failed connection.

Note: If you obtain a lock to serialize multiple IXLLIST requests and your protocol includes lock stealing, you should use LOCKOPER=HELDBY on each IXLLIST request once you hold the lock to ensure that the request is performed only if the lock is still yours.

Handling an Incompletely Processed LOCK Request that Specifies LOCKOPER=READNEXT

A LOCK request specifying LOCKOPER=READNEXT can time out before finishing all its processing. If this happens, IXLLIST:

- Sets the IXLLIST return code to IXLRETCODEWARNING and the reason code to IXLRSNCODETIMEOUT to indicate that processing did not complete.
- Returns in the LAALOCKINDEX field of the answer area, the index of the next lock table entry to be processed when the next LOCK request that specifies LOCKOPER=READNEXT is issued.

To continue processing, reissue the LOCK request specifying LOCKOPER=READNEXT using the index value returned in LAALOCKINDEX as the value of the LOCKINDEX parameter. Continue reissuing the request until the return code indicates that all the processing has completed.

Receiving Answer Area Information from a LOCK Request

When you invoke IXLLIST, list services return information related to your request in the answer area specified using the ANSLEN and ANSAREA parameters.

Under certain circumstances, answer area information is not valid. See "Determining if the Answer Area is Valid" on page 535 for information on how to determine whether the answer area information is valid.

The following list describes the information returned when the answer area is valid. The answer area is mapped by the IXLYLAA macro, which is presented in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

LAARETCODE
The return code from the IXLLIST service. Return code values are defined in the IXLYCON macro.

LAARSNCODE
The reason code associated with the return code from the IXLLIST service. Reason code values are defined in the IXLYCON macro.

LAACONID
For a LOCK request specifying the LOCKOPER parameter, LAACONID contains, under the following circumstances:

- SET parameter specified with LOCKMODE=COND and the request failed because the lock is held by another connection.
- SET parameter specified with LOCKCOMP and the request failed because the lock is not held by the specified connection.
- SET parameter specified with LOCKMODE=UNCOND and the request failed because the lock is held by a failed persistent connection.
- RESET parameter specified without LOCKCOMP and the request failed because you do not hold the lock.
- RESET parameter specified with LOCKCOMP and the request failed because the lock is not held by the specified connection.
- TEST parameter specified with LOCKCOMP and the request failed because the lock is not held by the specified connection.
- TEST parameter specified without LOCKCOMP and the request failed because you do not hold the lock. specified.
- READNEXT parameter specified without LOCKCOMP and a lock was found belonging to your connection.
- READNEXT parameter specified with LOCKCOMP and a lock was found belonging to the specified connection.

Either of the following:

- The connection identifier of the lock owner, if the lock specified by the LOCKINDEX parameter is held
- Zeros, if the lock specified by the LOCKINDEX parameter is free.

LAALOCKINDEX
For a LOCK request with the READNEXT parameter, this field contains one of the following:

- If you specified a connection ID using the LOCKCOMP parameter, this field contains the lock index of the next lock held by that connection ID
- If you omitted the LOCKCOMP parameter, this field contains the lock index of the next lock held by any connection.
- If the request timed out, this field contains the next lock table entry to be processed when the next LOCK request is issued.

MONITOR_LIST: Monitoring List Transitions

The list monitoring function allows you to determine whether a list in the structure is **empty** (contains no list entries) or **nonempty** (contains one or more list entries) without incurring the overhead of accessing the coupling facility. Instead, with the list monitoring function, the system maintains list state information in a list notification vector allocated in high-speed processor storage on your own system. A list's change

from empty to nonempty is called a **list transition**. Not only does the list monitoring function offer you a faster way to determine the state of a list, it also offers the option of being informed of list transitions by means of a list transition exit.

You could use this function when implementing a set of message queues using the list structure. When an empty message queue receives a message, the system notifies the interested user. The user removes the message from the queue, processes it, and waits for notification of the arrival of the next message.

See also IXLLSTC for information about monitoring list transitions.

The List Notification Vector

When you connect to the list structure and indicate your interest in using the list transition monitoring function, the system allocates a list notification vector for your use and returns a token to you representing this vector. The list notification vector shows the state (empty or non-empty) of each list you are monitoring. Each connector to the list structure that indicates interest in list monitoring (by coding the VECTORLEN parameter on the IXLCONN macro) is allocated a list notification vector.

A list notification vector consists of an array of entries in which each entry is logically associated with a list in the structure. The number of entries must be a multiple of 32.

When a list transition occurs for a monitored list, the system automatically updates the associated entry in the list notification vector to reflect the empty or nonempty state of the list. The IXLVECTR macro provides the interface to the list notification vector. To determine whether a list you are monitoring is empty or non-empty, invoke the IXLVECTR macro with either the TESTLISTSTATE or LTVECENTRIES parameter. You can use the IXLVECTR macro with the MODIFYVECTORSIZE parameter to change the size of your list notification vector, so you can monitor more lists, for instance. See "Using the IXLVECTR Macro" on page 680 for more information.

See "IXLLSTC: List Structure Control Services" on page 615 for additional list monitoring functions such as monitoring lists for full and not full state transitions, aggressive list monitoring notification, and list notification delays.

Options for Detecting a List Transition

You can detect list transitions two different ways:

- By having your list transition exit receive control when the list changes from empty to non-empty or from full to non-full. Your list transition exit then invokes the IXLVECTR macro to check the state (empty, non-empty, full, or non-full) of each list you are monitoring. The reported states of a list are determined by the type of list monitoring in effect for a list. When a structure is allocated in a coupling facility with CFLEVEL=22 or higher, your list transition exit can be used to inform you when a list that you are monitoring is changed from the full state to the non-full state.
- By coding a polling routine to invoke the IXLVECTR macro periodically to check the state of each list you are monitoring.

For each list you monitor, you can choose how you want to detect list transition. You can monitor some lists using a list notification exit and others by whatever method you choose, such as polling the list notification vector.

Steps to Set Up List Transition Monitoring

Setting up list monitoring involves the following steps. You must:

1. Indicate when you connect to the list structure using the IXLCONN macro that you are interested in using list monitoring
2. Establish list monitoring for specific lists in the list structure by invoking the IXLLIST macro with REQUEST=MONITOR_LIST.

Indicating Your Interest in List Transition Monitoring

To establish your interest in list transition monitoring, specify the VECTORLEN parameter on the IXLCONN macro invocation when you connect to the list structure. Specifying VECTORLEN will cause a list notification vector to be created for your connection's use. If you want to be informed of list transitions using an exit, you must specify the LISTTRANEXIT parameter along with the VECTORLEN parameter.

Starting Transition Monitoring of a List

To begin monitoring a particular list, you invoke the IXLLIST macro with REQUEST=MONITOR_LIST and ACTION=START. You also specify the list number (LISTNUM) of the list to be monitored and the index of the list notification vector entry (VECTORINDEX) to be associated with the monitored list.

If you want to have your list transition exit receive control when the list changes from empty to nonempty, specify DRIVEEXIT=YES. If you omit the DRIVEEXIT parameter or code DRIVEEXIT=NO, you indicate your intention to monitor the list's transitions by a different means.

When you start transition monitoring of a list, the system initializes the associated vector entry to indicate the current state of the list: empty or nonempty.

See "IXLLSTC: List Structure Control Services" on page 615 for additional list monitoring functions such as monitoring lists for full and not full state transitions, aggressive list monitoring notification, and list notification delays.

Stopping Transition Monitoring of a List

To stop monitoring a particular list, you invoke the IXLLIST macro with REQUEST=MONITOR_LIST and ACTION=STOP, specifying the list number of the list you no longer wish to monitor.

To reassign a list notification vector entry to monitor a different list, you must first stop monitoring the list currently associated with that entry.

To assign a different list notification vector entry to represent a list you are currently monitoring, you need only issue another start monitoring request for the same list using a different list notification vector entry. The second start monitoring request automatically cancels the use of the first entry for monitoring that list.

Design Considerations for Using the List Transition Exit

You can use the list transition exit to monitor lists and/or your event queue. See "MONITOR_EVENTQ: Monitoring an Event Queue" on page 574 for information about event queue monitoring. In either case, whether you use the list transition exit to monitor multiple lists, your single event queue, or both, it is important to understand the relationship between your list transition exit and the object(s) it is monitoring.

If you use a list notification exit to monitor multiple objects, note that the exit is given control whenever any object you monitor this way changes from empty to nonempty. To determine which monitored object triggered the notification, use the IXLVECTR macro with either the TESTLISTSTATE or LTVECENTRIES parameter to check the vector entry for each candidate object.

During structure rebuild, the list transition exit is not given control. When the rebuild completes, either normally or because of a rebuild stop, the list transition exit is given control once to inform you of any list transition transitions that might have occurred during the rebuild. **This is done whether or not you are currently monitoring any lists or your event queue.**

Timing Considerations

The time span involved in detecting and responding to a transition of an object you are monitoring introduces several timing considerations, particularly if multiple connections are monitoring the same object. (Note however, that only your connection will be monitoring your event queue.)

If multiple connections are monitoring the same list, the first connection to respond to a list transition could empty the list before other connections test the list notification vector or check the list. Depending on when the other connection emptied the list, either of the following would occur:

- Your list transition exit could receive control, test the list notification vector, and find no non-empty lists.
- You could test the list notification vector, find the list had become non-empty, then attempt to read a list entry from the list and find the list empty.
- Your list transition exit does not receive control if list monitoring notification delays are enabled for the list and the list is emptied within the list notification delay time. For more information on the list monitoring notification delay function, see "Understanding List Monitoring Notification Delay" on page 619.

Another timing consideration is the possible delay between the time a monitored object changes from empty to non-empty or full to non-full or when the type of monitoring is changed (for example, NOTEMPTY to NOTFULL) and the time its list notification vector entry is updated. All list transitions for monitored objects are reflected in the list notification vector in the order in which they occur, but the timing of the updates is not guaranteed.

Under certain circumstances, there might be a sizable delay between the time a transition of a monitored object occurs and the time the list transition exit is given control. List transition exits are not given control while a structure is being rebuilt. Once the rebuild processing has finished the list transition exits for the connections that participated in the structure rebuild are given control to inform the connections of any transitions that might have occurred during the rebuild process.

Another circumstance of which you should be aware is that if connectivity to the coupling facility is interrupted, the list transition exit might be given control even though the monitored object has not changed.

Best Circumstances for Using List Monitoring

If a list you are monitoring is constantly receiving new entries and having them removed, list monitoring would be of little value because the list would be constantly changing back and forth between empty and non-empty or full and non-full. List monitoring is best suited for situations in which a list receives new entries and removes existing entries on a less frequent basis.

If you are monitoring a large number of lists, determining which list changed states could be a lengthy process. List monitoring using a list transition exit is more appropriate when used on a small number of lists.

Receiving Answer Area Information from a MONITOR_LIST Request

When you invoke IXLLIST, list services return information related to your request in the answer area specified using the ANSLEN and ANSAREA parameters.

Under certain circumstances, answer area information is not valid. See "Determining if the Answer Area is Valid" on page 535 for information on how to determine whether the answer area information is valid.

The following list describes the information returned when the answer area is valid. The same information is returned in the answer area for a IXLLSTC REQUEST=MONITOR_KEYRANGE service. The answer area is mapped by the IXLYLAA macro, which is presented in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

LAARETCODE
The return code from the IXLLIST service. Return code values are defined in the IXLYCON macro.

LAARSNCODE
The reason code associated with the return code from the IXLLIST service. Reason code values are defined in the IXLYCON macro.

LAAMNL_LISTCNT
Count of the in-use entries or elements residing on the processed list at the time monitoring was started. Returned for successful MONITOR_LIST requests that specify ACTION=START. This count includes the number of list entries or data elements on the list that currently reside in coupling facility

real and storage class memory. The value specified for LISTCNTLTYPE on the IXLCONN macro when the list structure was allocated determines whether this field represents a count of list entries or data elements. LAAMNL_LISTCNTLTYPE found in the answer area can also be used to determine whether LAALISTCNT represents a count of list entries or data elements.

LAAMNL_LISTOPPCNT
The number of list entries or data elements on the list. Returned for requests that completed successfully. Valid when LAAMNL_LISTOPPCNTVALID is ON. When LAAMNL_LISTCNTLTYPE is set to 1, LAAMNL_LISTLOPPCNT is expressed as the number of data elements on the list. When LAAMNL_LISTCNTLTYPE is set to 2, LAAMNL_LISTOPPCNT is expressed as the number of list entries on the list. Like LAAMNL_LISTCNT, this count includes the number of list entries or elements for the processed list that currently reside in coupling facility real and storage class memory.

LAAMNL_LISTOPPCNTVALID
Opposite List Count Valid indicator. A flag that can be returned for requests that completed successfully. The flag indicates when LAAMNL_LISTOPPCNT is valid for use.

LAAMNL_LISTCNTLTYPE
List Control Type. A value returned for requests that completed successfully. The value is set to 1 (IXLLISTCNTLTYPEENTRY) when ENTRY is specified for LISTCNTLTYPE on the IXLCONN macro when the list structure was allocated. LAAMNL_LISTCNT is expressed in number of list entries. When LAAMNL_LISTOPPCNTVALID is ON, LAAMNL_LISTOPPCNT is expressed in number of data elements.

The value is set to 2 (IXLLISTCNTLTYPEELEMENT) when ELEMENT is specified for LISTCNTLTYPE on the IXLCONN macro when the list structure was allocated. LAAMNL_LISTCNT is expressed in number of data elements. When LAAMNL_LISTOPPCNTVALID is ON, LAAMNL_LISTOPPCNT is expressed in number of list entries.

A value of zero (0) indicates that this field is not provided by list services for the request.

LAAMNL_ENTRYQUEUED
Indication of whether the list was in the empty or non-empty state at the time a IXLLIST MONITOR_LIST ACTION=START or IXLLSTC MONITOR_LIST ACTION=START MONITORTYPE=NOTEMPTY request was processed.

Indication of whether the key-range was in the empty state or non-empty state at the time a IXLLSTC MONITOR_KEYRANGE ACTION=START request was processed.

LAAMNL_LISTNOTFULL
Indication of whether the list was in the full state or not full state at the time a IXLLSTC MONITOR_LIST ACTION=START MONITORTYPE=NOTFULL request was processed for a structure allocated in a coupling facility with CFLEVEL=22 or higher.

MONITOR_EVENTQ: Monitoring an Event Queue

The event queue monitoring function allows you to determine whether your event queue in the structure is **empty** (contains no event monitor controls objects (EMCs)) or **non-empty** (contains one or more EMCs) without incurring the overhead of accessing the coupling facility. As with list monitoring, the event queue monitoring function uses a list notification vector allocated in high-speed processor storage. The system maintains event queue state information in the list notification vector. As with the list monitoring function, you can choose to be informed of the event queue's transitions to a non-empty state by means of a list transition exit or through your own polling mechanism.

The MONITOR_EVENTQ request type is valid only for a keyed list structure allocated in a coupling facility with CFLEVEL=3 or higher.

See also IXLLSTC for information about event queue monitoring.

Steps to Set Up Event Queue Transition Monitoring

Setting up event queue monitoring involves the following steps. You must:

1. Indicate when you connect to the list structure using the IXLCONN macro that you are interested in using event queue monitoring by specifying a vector (VECTORLEN), event monitor controls (EMCSTGPCT), and optionally, a list transition exit for notification purposes (LISTTRANEXIT).
2. Establish event queue monitoring by invoking the IXLLIST macro with REQUEST=MONITOR_EVENTQ ACTION=START.

Indicating Your Interest in Event Queue Transition Monitoring

There are four factors to consider when establishing your interest in event queue transition monitoring.

- The list must support keyed entries, specified by the REFOPTION=KEY parameter on the initial connect to the list structure.
- You must specify the VECTORLEN parameter on the IXLCONN macro to cause the system to create a list notification vector for your connection's use.
- You must specify a non-zero value for the EMCSTGPCT parameter on the IXLCONN macro to allow the system to set aside a percentage of the list structure's storage for use as event monitor controls objects that will be queued to the structure's event queues.
- You have the option of having a list transition exit for notification purposes.

Starting Transition Monitoring of an Event Queue

To begin monitoring your event queue, invoke the IXLLIST macro with REQUEST=MONITOR_EVENTQ and ACTION=START. You also specify the index of the list notification vector entry (VECTORINDEX) to be associated with the event queue.

If you want to have your list transition exit receive control when the event queue changes from empty to non-empty, specify DRIVEEXIT=YES, If you omit the DRIVEEXIT parameter or code DRIVEEXIT=NO, you indicate your intention to monitor the event queue's transitions by a different means.

When you start transition monitoring of an event queue, the system initializes the associated vector entry to indicate the current state of the event queue: empty or non-empty.

Stopping Transition Monitoring of an Event Queue

To stop monitoring your event queue, invoke the IXLLIST macro with REQUEST=MONITOR_EVENTQ and ACTION=STOP.

See “Design Considerations for Using the List Transition Exit” on page 572 for information describing the relationship between your list transition exit and the event queue it is monitoring.

Receiving Answer Area Information from a MONITOR_EVENTQ Request

When you invoke IXLLIST, list services return information related to your request in the answer area specified used the ANSLEN and ANSAREA parameters.

Under certain circumstances, answer area information is not valid. See “Determining if the Answer Area is Valid” on page 535 for information on how to determine whether the answer area information is valid.

The following list describes the information returned when the answer area is valid. The answer area is mapped by the IXLYLAA macro, which is presented in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

LAARETCODE
The return code from the IXLLIST service. Return code values are defined in the IXLYCON macro.

LAARSNCODE
The reason code associated with the return code from the IXLLIST service. Reason code values are defined in the IXLYCON macro.

LAAMNEQ_EVENTQUEUED
A flag to indicate whether your event queue was not empty. Returned for successful MONITOR_EVENTQ ACTION=START requests.

LAAMNEQ_EVENTCNT
Count of the number of events (event monitor control objects) that were queued to your event queue when monitoring was established. Returned for successful MONITOR_EVENTQ REQUEST=START requests.

MONITOR_SUBLIST, MONITOR_SUBLISTS: Monitoring Sublists

The sublist monitoring function, which is available only with a keyed list structure allocated in a coupling facility of CFLEVEL=3 or higher, allows you to determine whether a sublist in the structure is **empty** (contains no list entries) or **non-empty** (contains one or more list entries) without incurring the overhead of accessing the coupling facility. Instead, with the sublist monitoring function, the system queues or withdraws event monitor controls objects on your event queue when a monitored sublist transitions from empty to non-empty or vice versa.

Understanding the Event Queue

An event queue is established for each list structure user that specifies an interest in sublist monitoring when the list structure is allocated. While sublist monitoring is in effect, the system will queue or withdraw event monitor controls objects (EMCs) to or from your event queue to indicate the empty or nonempty state of the sublists you are monitoring. Using event queue monitoring in conjunction with sublist monitoring allows you to determine whether the set of sublists that you are monitoring is empty or nonempty, and if one or more sublists is nonempty, to determine efficiently which sublist(s) those are.

Indicating Your Interest in Sublist Transition Monitoring

To establish your interest in sublist transition monitoring, specify the EMCSTGPCT and the VECTORLEN parameters on the IXLCONN macro invocation when you connect to the list structure. Specifying EMCSTGPCT indicates the amount of space in the list structure's available storage that you want to allocate to event monitor controls. Specifying VECTORLEN will cause a list notification vector to be created for your connection's use. The system will update the associated entry in the list notification vector when the your event queue transitions from empty to nonempty.

Specifying User Notification Controls

When you register interest in monitoring a designated sublist, you can specify 16 bytes of user data (called user notification controls) to be associated with the sublist. The use of the user notification controls (UNCs) depends on your application requirements. For example, the UNCs might contain information about the meaning of the sublist. If a sublist transition occurs (and an EMC is queued to your event queue), the EMC will contain the 16 bytes of user notification controls. The system returns this information to you when you read and dequeue the EMCs by issuing the IXLLIST REQUEST=DEQ_EVENTQ macro.

Guide to the Topic

"MONITOR_SUBLIST, MONITOR_SUBLISTS: Monitoring Sublists" on page 576 is divided into the following section s.

- "MONITOR_SUBLIST: Monitoring a Single Sublist" on page 577 presents information about the MONITOR_SUBLIST request.
- "MONITOR_SUBLISTS: Monitoring Multiple Sublists" on page 578 presents information about the MONITOR_SUBLISTS request.

MONITOR_SUBLIST: Monitoring a Single Sublist

The IXLLIST REQUEST=MONITOR_SUBLIST allows you to start and stop monitoring interest in a single sublist. The sublist must be part of a keyed list structure allocated in a coupling facility of CFLEVEL=3 or higher. You can issue IXLLIST REQUEST=MONITOR_SUBLIST multiple times, either to request monitoring of a set of different sublists or to update the monitoring information (such as the user notification controls) that is associated with a sublist you are currently monitoring.

See also IXLLSTC for information about monitoring a single sublist.

Starting Transition Monitoring of a Sublist

To begin monitoring a particular sublist, you invoke the IXLLIST macro with REQUEST=MONITOR_SUBLIST and ACTION=START. You also specify the list number (LISTNUM), the entry key of the sublist (ENTRYKEY), the connect token (CONTOKEN) that was returned when you connected to the structure, and any user notification control information that your processing might require (UNC). This information is associated with the EMC for this user/sublist combination.

To update your registered interest in monitoring a particular sublist, you can reissue the IXLLIST REQUEST=MONITOR_SUBLIST specifying the same sublist but with different user notification control information. The system replaces the UNC information in the existing EMC that is associated with the user/sublist combination.

Stopping Transition Monitoring of a Sublist

To stop monitoring a particular sublist, you invoke the IXLLIST macro with REQUEST=MONITOR_SUBLIST and ACTION=STOP. You also must specify the list number of the list you no longer want to monitor, the list entry key of the sublist, and your connect token.

Scenario for Monitoring a Sublist

Issue the macro requests in the following order:

- Connect to the keyed list structure with a non-zero EMCSTGPCT value and a local vector to specify sublist monitoring.
- Issue IXLLIST REQUEST=MONITOR_EVENTQ to monitor your event queue. The reason for registering interest in monitoring your event queue **before** specifying the sublist(s) you want to monitor is to ensure that your notification that a sublist has transitioned from an empty to a nonempty state is not deferred. For example, as soon as you register interest in a sublist, it is possible for the EMC that represents the registration of that sublist to get queued to your event queue. If you have not previously registered interest in monitoring your event queue, the system cannot notify you that an EMC is queued there.
- Issue IXLLIST REQUEST=MONITOR_SUBLIST to monitor the sublist, or issue the macro multiple times to monitor multiple sublists. If a sublist transition occurs, an EMC will be queued or withdrawn from your event queue and you will be notified, either through your list transition exit or through your own vector polling protocol.
- When you are notified that a sublist transition has occured, you can issue IXLLIST REQUEST=DEQ_EVENTQ to read the EMC into a storage area that you specify. The EMC will contain any user notification controls that you initially specified when registering to monitor the sublist, or as updated by a subsequent MONITOR_SUBLIST request against the same sublist.

Receiving Answer Area Information from a MONITOR_SUBLIST Request

When you invoke IXLLIST, list services return information related to your request in the answer area specified used the ANSLEN and ANSAREA parameters.

Under certain circumstances, answer area information is not valid. See “Determining if the Answer Area is Valid” on page 535 for information on how to determine whether the answer area information is valid.

The following list describes the information returned when the answer area is valid. The answer area is mapped by the IXLYLAA macro, which is presented in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

LAARETCODE
The return code from the IXLLIST service. Return code values are defined in the IXLYCON macro.

LAARSNCODE
The reason code associated with the return code from the IXLLIST service. Reason code values are defined in the IXLYCON macro.

LAAMNSL_ENTRYQUEUED
A flag to indicate that the sublist is not empty. Returned for successful MONITOR_SUBLIST ACTION=START requests.

LAAMNSL_EMCCCNT
Count of event monitor control objects in use by the structure when sublist monitoring was established. Returned for successful MONITOR_SUBLIST ACTION=START requests, or for requests that fail because the structure has no more EMCs (reason code IXLRSNCODESTRFULL).

LAAMNSL_MAXEMCCNT
Maximum number of EMCs for the structure. Returned for successful MONITOR_SUBLIST ACTION=START requests, or for requests that fail because the structure has no more EMCs (reason code IXLRSNCODESTRFULL).

MONITOR_SUBLISTS: Monitoring Multiple Sublists

The IXLLIST REQUEST=MONITOR_SUBLISTS request allows you to register interest in monitoring multiple sublists with a single command. Each sublist must be part of a keyed list structure allocated in a coupling facility of CFLEVEL=3 or higher. You can only **start** sublist monitoring with the MONITOR_SUBLISTS request; to stop sublist monitoring you must issue a MONITOR_SUBLIST request to stop monitoring each individual sublist.

See also IXLLSTC for information about monitoring multiple sublists.

Identifying the Sublists to be Monitored

The IXLLIST REQUEST=MONITOR_SUBLISTS request allows you to specify from 1 to 1024 sublists. To identify the sublists to be monitored, you build a record for each sublist in a buffer area, designated by BUFFER or BUFLIST on the macro invocation. The record is mapped by the IXLYMSRI macro and, for each sublist, contains the same information that you would have provided for a single request — the list number, the entry key, and any user notification control information.

Passing Buffered Data on a MONITOR_SUBLISTS Request

See "Selecting the Buffer Format" on page 524 for a description of the buffer format options and their performance considerations.

Using the Monitored Object State Vector

When you issue an IXLLIST REQUEST=MONITOR_SUBLISTS request, you must provide a 128-byte storage area, (the MOSVECTOR), in which the system will indicate the monitored object state (empty or non-empty) of each sublist in which you tried to register interest. The storage area will contain a bit string, with bit 1 as the origin, where each bit corresponds one-to-one with the IXLYMSRI entries passed as input in the BUFFER or BUFLIST. Only the bits corresponding to the IXLYMSRI entries that were actually processed on the current request will contain valid monitored object state information for the sublists designated by the corresponding IXLYMSRI entries. Bits in the MOSVECTOR that lie outside the valid range are not meaningful. A bit value of ON in the monitored object state vector indicates that the corresponding sublist is non-empty; a bit value of OFF indicates that the corresponding sublist is empty.

Handling an Incompletely Processed MONITOR_SUBLISTS Request

An IXLLIST REQUEST=MONITOR_SUBLISTS can complete prematurely for one of the following reasons:

- A request could time out before completion.
- The structure has no more event monitor control objects left and creation of an EMC was required by the request.
- A request specifies an IXLYMSRI entry that contains a list number that is not valid.

When a MONITOR_SUBLISTS request ends before processing all the IXLYMSRI entries, list services sets the IXLLIST return and reason codes as follows:

- If the processing timed out, IXLRETCODEWARNING and IXLRSNCODETIMEOUT.
- If the structure had no more EMCs, IXLRETCODEENVERROR and IXLRSNCODESTRFULL.
- If an IXLYMSRI entry contained a list number that was not valid, IXLRETCODEPARMERROR and IXLRSNCODEBADLISTNUMBER.

List services also returns in the LAAMNSLS_FAILINDEX field of the answer area, the index of the first unprocessed IXLYMSRI entry when the request completed prematurely. The MOSVECTOR bits that correspond to the IXLYMSRI entries between STARTINDEX and LAAMNSLS_FAILINDEX minus one contain valid monitored object state information.

To continue processing the IXLYMSRI entries after processing the entries that were successfully completed, reissue the MONITOR_SUBLISTS request with the STARTINDEX keyword specifying the index of the first unprocessed IXLYMSRI entry to be processed. If the premature completion was caused by a lack of EMCs in the structure, you must first either release some EMCs or rebuild the structure allowing for more EMCs. You can reissue the request to continue processing the IXLYMSRI entries when either of the corrective actions is complete. Note that if the corrective action was to rebuild the structure, you must first start monitoring for all the sublists you were monitoring prior to the rebuild before reissuing the request to process the IXLYMSRI entries.

If the premature completion was caused by a list number that was not valid in an IXLYMSRI entry, either correct the list number value and reissue the request with STARTINDEX updated to the value in LAAMNSLS_FAILINDEX or update STARTINDEX to skip over the IXLYMSRI entry containing the list number that was not valid.

Receiving Answer Area Information from a MONITOR_SUBLISTS Request

When you invoke IXLLIST, list services return information related to your request in the answer area specified used the ANSLEN and ANSAREA parameters.

Under certain circumstances, answer area information is not valid. See "Determining if the Answer Area is Valid" on page 535 for information on how to determine whether the answer area information is valid.

The following list describes the information returned when the answer area is valid. The answer area is mapped by the IXLYLAA macro, which is presented in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

LAARETCODE
The return code from the IXLLIST service. Return code values are defined in the IXLYCON macro.

LAARSNCODE
The reason code associated with the return code from the IXLLIST service. Reason code values are defined in the IXLYCON macro.

LAAMNSLS_FAILINDEX
Index of the first unprocessed IXLYMSRI entry when the IXLLIST REQUEST=MONITOR_SUBLISTS request completed prematurely. Premature completion can occur when the request times out (reason code IXLRSNCODETIMEOUT), when the structure has no more EMCs left (reason code IXLRSNCODESTRFULL), or when an IXLYMSRI entry specifies a list number that is not valid (reason code IXLRSNCODEBADLISTNUMBER).

LAAMNSLS_EMCCCNT
Count of event monitor control (EMC) objects in use by the structure when the MONITOR_SUBLISTS request completed. Returned when the request completes successfully or prematurely.

LAAMNSLS_MAXEMCCNT
Maximum number of EMCs for the structure. Returned when the request completes successfully or prematurely.

READ_EMCONTROLS: Reading Event Monitor Controls

Use the READ_EMCONTROLS request to obtain the event monitor control (EMC) information associated with the user and a monitored sublist. The list containing the sublist must be a keyed list structure allocated in a coupling facility of CFLEVEL=3 or higher. At most one such unique EMC can exist per user per sublist. If the EMC exists, the list service returns the following event monitor control information in the answer area specified using the ANSLEN and ANSAREA parameters.

- The connection identifier of the connector
- The list number
- The list entry key of the sublist
- The user-supplied user notification control data
- Flag to indicate whether the EMC is queued to the user's event queue

If the EMC does not exist, the list service returns the IXLRSNCODENOENTRY reason code to the requestor.

The READ_EMCONTROLS request type is valid only for a keyed list structure allocated in a coupling facility with CFLEVEL=3 or higher.

See also IXLLSTC for information about reading the event monitor controls associated with a user.

Receiving Answer Area Information from a READ_EMCONTROLS Request

When you invoke IXLLIST, list services return information related to your request in the answer area specified used the ANSLEN and ANSAREA parameters.

Under certain circumstances, answer area information is not valid. See “Determining if the Answer Area is Valid” on page 535 for information on how to determine whether the answer area information is valid.

The following list describes the information returned when the answer area is valid. The answer area is mapped by the IXLYLAA macro, which is presented in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

LAARETCODE
The return code from the IXLLIST service. Return code values are defined in the IXLYCON macro.

LAARSNCODE
The reason code associated with the return code from the IXLLIST service. Reason code values are defined in the IXLYCON macro.

LAAREMC_CONID
The connection identifier of the connector associated with the event monitor control (EMC) object.

LAAREMC_EMCQUEUED
A flag to indicate whether an EMC is queued to the event queue of the connector identified by LAAREMC_CONID.

LAAREMC_LISTNUM
The list number of the list with which this EMC is associated.

LAAREMC_LISTENTRYKEY
The list entry key of the sublist with which this EMC is associated.

LAAREMC_UNC
The user notification control data supplied by the connector when this EMC was established to monitor the sublist identified by the list number and entry key, or when modified by a subsequent MONITOR_SUBLIST or MONITOR_SUBLISTS request.

READ_EQCONTROLS: Reading Event Queue Controls

Use the READ_EQCONTROLS request to obtain the event queue control information associated with the connector's event queue. The list service returns the following event queue control information in the answer area specified using the ANSLEN and ANSAREA parameters.

- Flag to indicate whether the list transition exit is to be driven when the user's event queue changes from empty to non-empty
- Flag to indicate whether the user is currently monitoring the event queue
- The vector index associated with the event queue being monitored
- The number of event monitor control (EMC) objects that are currently queued to the event queue
- The approximate number of times the event queue has changed from empty to non-empty

The READ_EQCONTROLS request type is valid only for a keyed list structure allocated in a coupling facility with CFLEVEL=3 or higher.

See also IXLLSTC for information about reading the event queue controls associated with the connector's event queue.

Obtaining Event Queue Monitoring Information

There is an event queue associated with every list structure user intending to do sublist monitoring. For every event queue there is an event queue control object that contains information about the state of the queue and associated monitoring information. A user can monitor the state (empty or non-empty) of an event queue with the IXLLIST REQUEST=MONITOR_EVENTQ request.

Receiving Answer Area Information from a READ_EQCONTROLS Request

When you invoke IXLLIST, list services return information related to your request in the answer area specified used the ANSLEN and ANSAREA parameters.

Under certain circumstances, answer area information is not valid. See "Determining if the Answer Area is Valid" on page 535 for information on how to determine whether the answer area information is valid.

The following list describes the information returned when the answer area is valid. The answer area is mapped by the IXLYLAA macro, which is presented in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

LAARETCODE
The return code from the IXLLIST service. Return code values are defined in the IXLYCON macro.

LAARSNCODE
The reason code associated with the return code from the IXLLIST service. Reason code values are defined in the IXLYCON macro.

LAAREQC_MONITORINGACTIVE
A flag to indicate whether the user is currently monitoring the event queue for which the system is returning information.

LAAREQC_DRIVEEXIT
A flag to indicate whether XES is to drive the connection list transition exit when the user's event queue changes from empty to non-empty.

LAAREQC_VECTORINDEX
The vector index associated with the event queue being monitored.

LAAREQC_EMCQUEUEDCNT
The number of event monitor control (EMC) objects that are queued to the event queue.

LAAREQC_EVENTTRAN
A count of the approximate number of empty to non-empty event queue transitions that have occurred.

DEQ_EVENTQ: Retrieving Events from the Event Queue

Use the DEQ_EVENTQ request to read and dequeue queued events from a user's event queue. You can read and dequeue multiple EMCs from your event queue with a single invocation of the DEQ_EVENTQ command. Each set of read and dequeue operations is done atomically. Once dequeued from the event queue, an EMC is not deleted. The EMC remains associated with the user and the sublist for which it was created until the user deregisters its interest in monitoring the sublist or the user disconnects or fails.

List services return the event monitor controls (EMC) objects in a storage area you specify with either the BUFFER or BUFLIST parameter. Each of the EMCs returned in the BUFFER or BUFLIST area is mapped by the IXLYEMC macro and contains the following information:

- The connection identifier
- The list number of the list header containing the sublist
- The list entry key of the sublist
- The user notification controls — 16 bytes of user-defined data

List services returns the EMCs in the BUFFER or BUFLIST storage area in the order in which they are queued to the event queue, with the oldest transitions first and the most recent transitions last.

The DEQ_EVENTQ request type is valid only for a keyed list structure allocated in a coupling facility with CFLEVEL=3 or higher.

See also IXLLSTC for information about retrieving events from an event queue.

Handling an Incompletely Processed DEQ_EVENTQ Request

An IXLLIST REQUEST=DEQ_EVENTQ request might complete prematurely before all the EMCs have been read from the event queue. Be sure to process the information returned from the last request before reissuing the request. The data returned from this request will be overwritten if you specify the same buffer address. Continue to reissue the request until the return code indicates that all processing has completed.

When all EMCs have been read and dequeued from the user's event queue, the system returns a zero return code and a zero count of how many EMCs remain queued (IXLYLAA field LAADEQ_EMCQUEUEDCNT).

Receiving Answer Area Information from a DEQ_EVENTQ Request

When you invoke IXLLIST, list services return information related to your request in the answer area specified used the ANSLEN and ANSAREA parameters.

Under certain circumstances, answer area information is not valid. See “Determining if the Answer Area is Valid” on page 535.

The following list describes the information returned when the answer area is valid. The answer area is mapped by the IXLYLAA macro, which is presented in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

LAARETCODE
The return code from the IXLLIST service. Return code values are defined in the IXLYCON macro.

LAARSNCODE
The reason code associated with the return code from the IXLLIST service. Reason code values are defined in the IXLYCON macro.

LAADEQ_EMCQUEUEDCNT
A count of the number of event monitor control (EMC) objects that remain queued to the event queue after the current invocation has returned the EMCs that were read and dequeued. Returned for successful DEQ_EVENTQ requests and for DEQ_EVENTQ requests that end prematurely.

LAADEQ_NUMEMCREAD
Count of the EMCs that were read and dequeued by the current request. The storage area identified by BUFFER or BUFLIST on the IXLLIST request contains the EMCs, which are numbered from one to this count. The EMCs in the storage area are mapped by the macro IXLYEMC. Returned for successful DEQ_EVENTQ requests and for DEQ_EVENTQ requests that end prematurely.

Coding a Complete Exit

Your complete exit provides a mechanism for list services to let you know when your asynchronously-processed IXLLIST request completes. You provide the address of your complete exit using the COMPLETEEXIT parameter when you issue the IXLCONN macro to connect to the list structure.

You will be informed of request completion through your complete exit in either of the following situations:

- You specify MODE=ASYNCEXIT
- You specify MODE=SYNCEXIT and the system processes your request asynchronously.

Information Passed to the Complete Exit

When the complete exit gains control, it receives the following information about the IXLLIST request and its outcome in the complete exit parameter list (CMPL), mapped by the IXLYCMPL macro:

CMPLCONTOKEN
The IXLLIST invoker's connect token.

CMPLCONNAME
The IXLLIST invoker's connect name.

CMPLCONDATA
Connect-time data you specified when you issued the IXLCONN macro to connect to the list structure. The use of this optional field is user defined. One possibility is to store a pointer to your connection's control structure.

CMPLLIST
Indicates the complete exit received control as a result of an IXLLIST request.

CMPLREBUILD
Indicates whether the target list structure was being rebuilt. When a list structure is being rebuilt, there is an interval in which the new structure and the old structure can both be the target of an IXLLIST request.

0
The target list structure was not being rebuilt or, if so, the target list structure was the original structure.

1
The target list structure was being rebuilt, and the target list structure was the new list structure.

CMPLRETCODE
Return code from IXLLIST request. Return code values are defined in the IXLYCON macro.

CMPLRSNCODE
Reason code from IXLLIST request. Reason code values are defined in the IXLYCON macro.

CMPLREQDATA
Information provided to the complete exit by the issuer of the IXLLIST request. The use of this optional field is user defined. It is intended to allow you to identify the particular request that has completed processing. One possibility is to store the address and ALET of an area containing the parameters specified on the IXLLIST request or other information that identifies the request.

CMPLANSAREAALET
Answer area ALET.

CMPLANSAREA@
Answer area address. The answer area is mapped by the IXLYLAA macro.

See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for listings of the IXLYCMPL, IXLYLAA, and IXLYCON mapping macros.

Environment

The complete exit receives control in the following environment:

Authorization:	Supervisor state, and PSW key 0
Dispatchable unit mode:	SRB
Cross memory mode:	PASN=HASN=SASN. PASN, HASN, and SASN are equal to the PASN at the time of the connect to the list structure.
AMODE:	31-bit
ASC mode:	Primary ASC mode
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held.
Control parameters:	None.

Input Specifications

List services pass information to the complete exit in registers and in the CMPL.

Registers at Entry

When the complete exit receives control, the GPRs contain the following information:

Register
Contents

0
Does not contain any information for use by the complete exit.

1
Address of a fullword containing the address of the CMPL.

2-12
Do not contain any information for use by the complete exit.

13
Address of a 72-byte work area for use by the complete exit routine. The exit routine does not have to save and restore registers in this work area. The exit routine can use this work area in any way it chooses.

14
Return address of list services

15
Entry point address.

When the complete exit receives control, the ARs contain no information for use by the complete exit.

Return Specifications

Your exit must return control to the system by branching to the address provided on entry in register 14. There are no requirements for the GPRs or ARs to contain any particular value.

Programming Considerations

If you have more than one outstanding IXLLIST request being processed asynchronously, multiple instances of your complete exit might run concurrently as list services process your requests. Note that you can access the CMPL data area only while your complete exit is running. If you want to save the CMPL information for later processing, make a copy of it before your complete exit returns control to the system.

Circumstances a User Exit Should Be Prepared to Handle

In certain instances, the system must quiesce the activity of user exits in order to perform cleanup processing. The following illustrates scenarios where this processing occurs:

- Connection Termination

  When a user disconnects or abnormally terminates, the system will force to completion any user exits executing on behalf of that user by issuing a PURGEDQ against the appropriate units of work. Note that if a connector terminates while a rebuild is in progress, any exits pertaining to both the original and the new structures will be forced to completion. In addition to forcing the currently executing user exits to completion, the system will also prevent any new invocations of these exits by cancelling any events that are pending presentation.

- Rebuild Stop

  When a connector provides an event exit response for the Rebuild Stop event, the system will force to completion any exits that are executing on behalf of that user's connection to the new structure by issuing a PURGEDQ against the appropriate units of work. Similar to connector termination processing, the user exits pertaining to the new structure will not be presented with any additional events. Note that any user exits executing on behalf of the original structure are unaffected by rebuild stop processing.

- Completion of a Rebuild

  When a connector provides an event exit response for the Rebuild Cleanup event, the system will force to completion any user exits that are executing on behalf of that user's connection to BOTH the original and the new structures by issuing a PURGEDQ against the appropriate units of work. No new events will be presented to the user exits on behalf of the original structure (as it is being discarded). Normal user exit processing will resume for the rebuilt structure upon completion of the rebuild process.

A user exit must be sensitive to conditions that can occur as a result of actions taken by the system and must be able to handle these as appropriate. For example, if a user exit has suspended itself, when the PURGEDQ is issued the system abends the user exit's unit of work with a retryable X'47B' abend and gives control to the user exit's recovery routine. (Note that although the recovery routine can retry, the user exit can not re-suspend itself because the system will fail any request to suspend a unit of work that has been the target of a PURGEDQ.) If the recovery routine percolates back to the system, its associated connection is terminated.

Coding a Notify Exit

Your notify exit provides a mechanism for list services to inform you that contention exists for a lock you hold. When you issue the IXLCONN macro to connect to a serialized list structure, you must specify the address of a user-written notify exit using the NOTIFYEXIT parameter. It is possible that your notify exit might receive control before you receive control back from IXLCONN. Therefore, ensure that before you issue IXLCONN, you have the notify exit established along with any control structures necessary to complete the exit's processing.

"Understanding Lock Contention and the Notify Exit" on page 518 explains in detail the role of the notify exit, the circumstances under which it receives control, and the actions it can take. This topic is limited to reference information for coding the exit.

Information Passed to the Notify Exit

When the notify exit gains control, it receives the following information:

- The lock index for which there is contention

- The LOCKDATA information you specified when you obtained the lock. This information can help your notify exit decide how to handle the lock contention. For instance, you might be able to determine why you obtained the lock and whether you can release it.
- Whether the lock is a persistent lock, which is indicated by a LOCKDATA field of zero.
- The connection ID (CONID) and connection name (CONNAME) associated with the request causing the contention
- The type of lock request (LOCKOPER=SET or LOCKOPER=NOTHELD) causing the contention.

This information is passed to the notify exit in the notify exit parameter list (NEPL), mapped by the IXLYNEPL macro:

NEPLCONTOKEN
Your connection token.

NEPLCONNAME
Your connection name.

NEPLCONDATA
Connect-time data you specified when you issued the IXLCONN macro to connect to the list structure. The use of this optional field is user defined. One possibility is to store the address and ALET of an area containing information used by your connection.

NEPLLIST
Indicates that the notify exit received control as a result of an IXLLIST request.

NEPLREBUILD
Indicates whether the target list structure was being rebuilt. When a list structure is being rebuilt, there is an interval in which the new structure and the old structure can both be the target of an IXLLIST request.

0
The target list structure was not being rebuilt or, if so, the target list structure was the original structure.

1
The target list structure was being rebuilt, and the target list structure was the new list structure.

NEPLLOCKINDEX
The lock table index for the lock for which there is contention.

NEPLOWNERLOCKDATA
The lock data specified with the LOCKDATA parameter when the lock was obtained

NEPLOWNERPERSISTENTLOCK
The lock was previously a persistent lock and the LOCKDATA field is now set to zero. See “Recovering Persistent Locks” on page 522 and “Reconnecting with Persistent Locks” on page 522 for more information about persistent locks.

NEPLPENDINGCONID
The connection ID associated with the pending request

NEPLPENDINGREQUESTTYPE
The pending request type

0
The pending request is LOCKOPER=NOTHELD

1
The pending request is LOCKOPER=SET

NEPLPENDINGCONNAME
The connection name of the pending request

See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for a listing of the IXLYNEPL macro.

Environment

The notify exit receives control in the following environment:

Authorization:	Supervisor state, and PSW key 0
Dispatchable unit mode:	SRB
Cross memory mode:	PASN=HASN=SASN. PASN, HASN, and SASN are equal to the PASN at the time of the connect to the list structure.
AMODE:	31-bit
ASC mode:	Primary ASC mode
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held.
Control parameters:	None.

Input Specifications

List services pass information to the notify exit in registers and in the NEPL.

Registers at Entry

When the notify exit receives control, the GPRs contain the following information:

Register
Contents

0
Does not contain any information for use by the notify exit.

1
Address of a fullword containing the address of the NEPL.

2-12
Do not contain any information for use by the notify exit.

13
Address of a 72-byte work area for use by the notify exit routine. The exit routine does not have to save and restore registers in this work area. The exit routine can use this workarea in any way it chooses.

14
Return address of list services

15
Entry point address.

When the notify exit receives control, the ARs contain no information for use by the notify exit.

Return Specifications

Your exit must return control to the system by branching to the address provided on entry in register 14. There are no requirements for the GPRs or ARs to contain any particular value.

Programming Considerations

If your lock request is processed asynchronously, your notify exit might receive control to inform you of contention for the lock you have requested even before you are informed that you obtained the lock. If your lock request is processed synchronously, your notify exit might receive control before you receive control back from the IXLLIST request.

Note

You own a lock you have requested **only** when you are informed (in the manner specified on your IXLLIST invocation) that your lock request has completed successfully. Unless you have received this confirmation, you cannot assume you hold the lock.

If you are a failed persistent connector that reconnects to a list structure, your notify exit receives control when contention occurs for any locks that you held. “Reconnecting with Persistent Locks” on page 522 describes the processing associated with these persistent locks.

See “Managing Multiple, Asynchronous Lock Requests” on page 521 for additional information about situations your notify exit should be prepared to handle.

Multiple instances of your notify exit might run concurrently if contention arises for more than one lock you hold. Note that you can access the NEPL data area only while your notify exit is running. If you want to save the NEPL information for later processing, make a copy of it before your notify exit returns control to the system.

See “Circumstances a User Exit Should Be Prepared to Handle” on page 585 for important information regarding additional situations user exits must anticipate.

Coding a List Transition Exit

Your list transition exit provides a mechanism for list services to inform you that one or more lists and/or the event queue you are monitoring changed from empty to nonempty. When a structure is allocated in a coupling facility with CFLEVEL=22 or higher, your list transition exit can also be used as a mechanism to inform you when a list that you are monitoring is changed from the full state to the not full state. For more information about monitoring lists for full and not full state transitions, see “IXLLSTC: List Structure Control Services” on page 615. The list transition exit parameter list (IXLYLEPL) does not specifically identify the affected monitored object, nor does it indicate how many monitored objects have transitioned. Your list transition exit must invoke the IXLVECTR macro to determine which monitored object(s) have changed from empty to non-empty or from full to not full.

You provide the address of your list transition exit using the LISTTRANEXIT parameter when you issue the IXLCONN macro to connect to the list structure. It is possible that your list transition exit might receive control before you receive control back from IXLCONN. Therefore, ensure that before you issue IXLCONN, you have the list transition exit established along with any control structures necessary to complete the exit's processing.

“Design Considerations for Using the List Transition Exit” on page 572 discusses the list transition exit in more detail. This topic is limited to reference information for coding the exit.

Information Passed to the List Transition Exit

When the list transition exit gains control, it receives the following information in the list transition exit parameter list (LEPL), mapped by the IXLYLEPL macro:

LEPLCONTOKEN
The connect token returned from the IXLCONN invocation that established the list transition exit.

LEPLCONDATA
Connect-time data you specified when you issued the IXLCONN macro to connect to the list structure. The use of this optional field is user defined. One possibility is to store a pointer to your connection's control structure.

LEPLEVENT
Event code indicating a list transition, event queue transition, or both occurred.

LEPLVECTORTOKEN
Token representing the user's list notification vector.

See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for a complete listing of the IXLYLEPL macro.

Environment

The list transition exit receives control in the following environment:

Authorization:	Supervisor state, and PSW key 0
Dispatchable unit mode:	SRB
Cross memory mode:	PASN=HASN=SASN. PASN, HASN, and SASN are equal to the PASN at the time of the connect to the list structure.
AMODE:	31-bit
ASC mode:	Primary ASC mode
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held.
Control parameters:	None.

Input Specifications

List services pass information to the list transition exit in registers and in the LEPL.

Registers at Entry

When the list transition exit receives control, the GPRs contain the following information:

Register
Contents

0
Does not contain any information for use by the list transition exit.

1
Address of a fullword containing the address of the LEPL.

2-12
Do not contain any information for use by the list transition exit.

13
Address of a 72-byte work area for use by the list transition exit routine. The exit routine does not have to save and restore registers in this work area. The exit routine can use this work area in any way it chooses.

14
Return address of list services

15
Entry point address.

When the list transition exit receives control, the ARs contain no information for use by the list transition exit.

Return Specifications

Your exit must return control to the system by branching to the address provided on entry in register 14. There are no requirements for the GPRs or ARs to contain any particular value.

Programming Considerations

Only a single instance of the list transition exit can run at a time for any particular connector to the list structure. If additional monitored lists, or the user's event queue, become nonempty while the list transition exit is running, then the list transition exit will immediately receive control again after it completes its current processing.

Note that you can access the LEPL data area only while your list transition exit is running. If you want to save the LEPL information for later processing, make a copy of it before your list transition exit returns control to the system.

See “Circumstances a User Exit Should Be Prepared to Handle” on page 585 for important information regarding additional situations user exits must anticipate.

Managing List Structure Utilization

The list structure is allocated with a fixed amount of storage. Depending on the CFLEVEL of the coupling facility in which the structure is allocated, this storage can be subdivided into entries, elements, and event monitor controls objects. (See Figure 39 on page 484, which describes the parts of a keyed list structure allocated in a coupling facility of CFLEVEL=3 or higher.) If an IXLLIST request requires that an object be available but none is, a “structure-full” condition occurs. When the structure becomes full, you will no longer be able to perform a number of IXLLIST functions. Affected functions could include:

- The ability to create a new list entry.
- The ability to update an existing list entry, regardless of whether its size would increase, decrease, or remain the same.
- The ability to register sublist monitoring interest (that is, to create an event monitor controls object).

The system returns counts of the objects allocated in the structure in the connect answer area (IXLYCONA). The values reflect the state of the structure at the time of the connect.

- CONALISTENTRYCOUNT — Number of entries in use
- CONALISTMAXENTRYCOUNT — Approximate maximum number of entries supported by the structure
- CONALISTELEMENTCOUNT — Number of data elements in use
- CONALISTMAXELEMENTCOUNT — Approximate maximum number of data elements supported by the structure
- CONALISTEMCCOUNT — Number of EMCs in use (if applicable)
- CONALISTMAXEMCCOUNT — Approximate maximum number of EMCs in the structure (if applicable).

Taking action to alleviate the storage problem before the structure becomes full is especially critical because the CONALISTMAXENTRYCOUNT and CONALISTMAXELEMENTCOUNT values are only approximate. As a result, you could receive a return code indicating that the structure is full even though the IXLLIST answer area counts of entries or elements in use are below the limits indicated in the CONA.

A reason for the CONA counts being approximate is that the coupling facility at times uses some of the structure's objects for its own processing. Those objects are not included in your “in-use” counts.

Another result of the CONA counts being approximate is that the IXLLIST request of one connector might be rejected due to a structure full condition while a subsequent request by a different connector might succeed. Alternatively, a request by a connector might be rejected while a subsequent request by the same connector might succeed. Furthermore, deleting a list entry when the structure is full might not result in the immediate availability of the storage for the list entry or data elements. As a result, your request could fail if you attempt to create a list entry of the same size as the one you deleted.

Applications using the list structure are responsible for managing structure utilization. The system does not prevent the structure from becoming full nor take any automatic action to remedy the condition. Therefore, **IBM recommends** that you take steps to correct a storage shortage before your application is affected. To do so, you need to consider the following:

- How to detect when the structure is becoming full
- How full you will permit the structure to become before you take remedial action
- How the storage shortage will be corrected.

Detecting When a List Structure Is Becoming Full

One way to monitor list structure utilization is to periodically check the fields listed below, which are returned in the answer area by certain successful IXLLIST requests:

- LAATOTALCNT, which returns the number of list entries in use in the structure (compare to the value of CONALISTMAXENTRYCOUNT.)
- LAATOTALELECNT, which returns the number of data elements in use in the structure (compare to the value of CONALISTMAXELEMENTCOUNT.)
- LAALISTCNT, which returns the number of entries or data elements on the list. The value specified for LISTCNTLTYPE on the IXLCONN macro when the list structure was allocated determines whether this field represents a count of list entries or data elements. LAALISTCNTLTYPE, LAARLCLISTCNTLTYPE, or LAAMNL_LISTCNTLTYPE found in the answer area can also be used to determine whether LAALISTCNT represents a count of list entries or data elements. (This value is also returned in LAAMNL_LISTCNT, and LAARLCLISTCNT.)
- LAAMNSL_EMCCNT, which returns the number of EMCs in use in the structure. (LAANMSLS_EMCCNT also contains this value.)
- LAAMNSL_MAXEMCCNT, which returns the approximate maximum number of EMCs in the structure. (LAAMNSLS_MAXEMCCNT also contains this value.)

To determine which IXLLIST requests return this information, see the description of the IXLYLAA data area in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Another way to monitor list structure utilization is to issue the IXLMG macro periodically and check the following fields:

- IXLYAMDSTRL_MLSELC, which returns the approximate maximum number of data elements allowed in the structure
- IXLYAMDSTRL_MLSEC, which returns the approximate maximum number of list entries allowed in the structure
- IXLYAMDSTRL_LSELC, which returns the number of data elements in use in the structure
- IXLYAMDSTRL_LSEC, which returns the number of list entries in use in the structure.

For a keyed list structure allocated in a coupling facility of CFLEVEL=3 or higher, you can also check these additional fields:

- IXLYAMDSTRL_EMCCNT, which returns the number of EMCs in use in the structure.
- IXLYAMDSTRL_MAXEMCCNT, which returns the approximate maximum number of EMCs in the structure.

These values can be used to calculate the structure's percentage fullness in terms of entries, elements, and EMCs.

Responding When the Structure is Getting Full

When your monitoring indicates that the structure is getting full, you can take several actions. First, until you resolve the storage problem, your application could minimize its issuance of IXLLIST requests that create or modify list entries or that request registering new sublist monitoring interest and thus create event monitor controls objects. Your application can also issue a message to the operator to warn that the structure is getting full and to request that the operator perform certain actions.

The easiest approach is to delete unneeded list entries or EMCs. In some cases, however, this might not be possible and the structure might need to be rebuilt to change its attributes.

If the structure is running out of elements but has plenty of entries (or vice versa), you can rebuild or alter the structure with a different ratio of elements to entries without changing the structure's size. Or, if the structure is running out of EMCs, you can rebuild or alter the structure with a different EMCSTGPCT percent value. Because the structure is not changing size, operator intervention is only required if altering ratios |and the SETXCF MODIFY command is used to disable alter processing for the structure.

If the structure needs more list entries, more data elements, and/or more EMCs, you can rebuild or alter the structure with more storage. Rebuilding or altering the structure with more storage might require operator intervention.

Rebuilding the Structure to Increase the Storage Capacity

You can rebuild the structure to increase capacity only if the CFRM policy that defines the structure allows for a larger size. If the structure is already the maximum size allowed by the CFRM policy, you must request that the system programmer modify the CFRM policy to allow a larger structure size and reactivate the modified policy.

If the active CFRM policy allows for a larger list structure, you can issue the IXLREBLD macro to rebuild the structure with a larger size. If you prefer to involve the operator, your application can issue a message to notify the operator that the structure needs to be rebuilt. The operator must issue the SETXCF START,REBUILD command to initiate structure rebuild. Rebuilding a keyed list structure allocated in a CFLEVEL=3 or higher coupling facility with a larger size results in the creation of additional EMCs, entries, and elements, depending on the values specified for the EMCSTGPCT and entry-to-element ratios.

Note: Duplexed structures cannot be rebuilt while they remain duplexed. If the structure is duplexed, duplexing will need to be stopped before the structure can be rebuilt. This can be done using the IXLREBLD macro or the SETXCF STOP,REBUILD command. If the CFRM active policy specifies DUPLEX(ENABLED) for the structure and IXLREBLD IGNOREDUPLEX=YES is not used, the system might immediately reduplex the structure after the completion of the stop processing. There might be a delay before reduplexing when only two coupling facilities are available for duplexing the structure. Reduplexing will occur immediately in configurations with three or more coupling facilities available for duplexing the structure.

To prevent the system from immediately reduplexing the structure or reduplexing the structure at a later time, change the DUPLEX specification for the structure to DUPLEX(ALLOWED) or DUPLEX(DISABLED). Change the DUPLEX setting for the structure in the CFRM policy to DUPLEX(ALLOWED) before stopping duplexing, or change the DUPLEX setting for the structure to DUPLEX(DISABLED), which will cause XCF to initiate the stop processing. Change the DUPLEX setting back to DUPLEX(ENABLED) when you no longer need to prevent the system from reduplexing.

Altering the Structure to Increase the Storage Capacity

With SP 5.2 and above and a structure allocated in a coupling facility with CFLEVEL=1 or higher, you can alter the size of the structure to increase capacity or the entry-to-element ratio to reapportion the structure's storage. As with the rebuild function, you can alter the structure only if the CFRM policy that defines the structure allows for a larger size. You can issue the IXLALTER macro or notify the operator to issue the SETXCF START,ALTER command to initiate structure alter.

For keyed list structures allocated in a coupling facility with CFLEVEL=3, you cannot alter the structure to change the number of EMCs in the structure. However, if the keyed list structure is allocated in a coupling facility of CFLEVEL=4 or higher, you can alter the number of EMCs in the structure.

Enhancements to Sublist Monitoring

Exploiters of sublist monitoring, such as MQ Shared Queues and IMS Shared Message Queue, can transparently reduce their scheduling overhead with the enhanced sublist notification mechanism.

By specifying a non-zero value or taking the default value on the SUBNOTIFYDELAY sub-parameter on the STRUCTURE CFRM parameter in the Administrative Data Utility (IXCMIAPU), users can specify in their policy definitions the amount of time between notification of a system selected message queue exploiter and the notification of the other instances.

A single, selected exploiter instance will receive notifications of sublist transitions before any other exploiter instances are notified, and the other instances may not be notified at all if the initial exploiter processes the sublist in a timely manner. The exploiter instance who receives the initial notification will be selected in a round-robin fashion.

To benefit from this enhancement, the sysplex environment needs to meet the following conditions:

- The monitored list structure is allocated in a coupling facility at CFLEVEL16.
- The z/OS system support for CFLEVEL16 has been installed on the particular system in the sysplex. For full benefit, all z/OS systems in the sysplex need to support CFLEVEL16.

Chapter 9. Using List Services (IXLLSTE, IXLLSTM, IXLLSTC)

IXLLSTE, IXLLSTM, and IXLLSTC are interfaces to list services which are successors to the IXLLIST macro interface and are the suggested list services interfaces to use. The three macros collectively provide the same functions available with the IXLLIST macro. The IXLLIST macro is maintained, however, is not updated with any new support.

The list services macros are:

- IXLLSTE, List Structure Single Entry Services, which contains all requests that manipulate a single list entry.
- IXLLSTM, List Structure Multiple Entry Services, which contains all requests that manipulate multiple list entries.
- IXLLSTC, List Structure Control Services, which contains all requests that modify structure controls.

IXLLSTE, IXLLSTM, and IXLLSTC contain updated syntax and may contain keyword names that are changed from those used in the IXLLIST macro. Applications using IXLLIST can be changed to an equivalent IXLLSTE, IXLLSTM, or IXLLSTC service. For compatibility, however, when running on lower-level systems (OS/390 Release 8 or lower), the application that uses the new services cannot make use of any of the new functions available with the IXLLSTE, IXLLSTM, and IXLLSTC macros that do not have an IXLLIST equivalent function.

Chapter 8, “Using List Services (IXLLIST),” on page 481 describes list services and the functions provided by IXLLIST. None of that information is negated by the IXLLSTE, IXLLSTM, and IXLLSTC list services macros. Therefore, this chapter concentrates on the functions introduced by the IXLLSTE, IXLLSTM, and IXLLSTC macro interfaces, as well as provides guidance in the use of the three successor macros to IXLLIST.

The following topics introduce you to and help you understand the functions available with the IXLLSTE, IXLLSTM, and IXLLSTC list services macros.

- “Additional List Services Provided by IXLLSTE, IXLLSTM, and IXLLSTC” on page 595
- “Comparing IXLLSTC, IXLLSTE, and IXLLSTM with IXLLIST” on page 598
- “IXLLSTE: List Structure Single Entry Services” on page 599
- “IXLLSTM: List Structure Multiple Entry Services” on page 605
- “IXLLSTC: List Structure Control Services” on page 615

Additional List Services Provided by IXLLSTE, IXLLSTM, and IXLLSTC

This section describes additional list services.

IXLLSTE, IXLLSTM, and IXLLSTC list services provide additional functions to the IXLLIST macro. Some, but not all, of these additional functions require that the list structure be allocated in a minimum coupling facility CFLEVEL. The additional list services functions, introduced by the IXLLSTE, IXLLSTM, or IXLLSTC macros, are:

- A list structure user can specify that a program-specified entry identifier will be used to reference a list structure entry as opposed to a system-generated entry identifier.
- New key comparison functions have been added for both single and multiple entry request types.
- List monitoring functions provide more granularity within a range of list entries being monitored.
- Secondary keys can be used to identify a list entry.

© Copyright IBM Corp. 1994, 2023

- New request types provide more efficient methods of deleting entries from a list and for processing an input list of entries either by changing the key value or moving the entry to another list.
- A list structure user can specify that 64-bit storage is to be used for data buffers.

Understanding Program-Specified Entry Identifiers

A list entry can be located by its assigned entry identifier (ENTRYID). Prior to OS/390 Release 9, the system assigned an ENTRYID when the list entry was created and assured that the ENTRYID was unique among the list entries previously and currently allocated in the list structure.

With OS/390 Release 9 and higher, an allocation attribute can specify whether the list structure is to be allocated with system-assigned ENTRYIDs or user-assigned ENTRYIDs. The IXLCONN ENTRYIDTYPE=USER parameter indicates that the user will assign the ENTRYID for each list entry created, and will assure that the ENTRYID is unique among the list entries currently allocated in the list structure. Unlike a system-assigned ENTRYID, which is never reused and is unique for the life of the structure, a user-assigned ENTRYID can be reused over time provided it is unique while assigned to a list entry.

To use user-assigned ENTRYIDs, the list structure must be allocated in a coupling facility of CFLEVEL=8 or higher or the connect request will be rejected. An IXLCONN request to connect to an allocated structure also will be rejected if the ENTRYIDTYPE attribute does not match the attribute currently in effect for the structure. Similarly, an IXLCONN REBUILD request will be rejected if the connector's requested ENTRYIDTYPE attribute does not match the attribute in effect for the original structure.

The type of list entry identifier is consistent for the entire list structure. Either all list entries are referenced by system-assigned ENTRYIDs or all list entries are referenced by user-assigned ENTRYIDs.

Enhancement to List Services Entry Key and Secondary Key Comparison

With list structures allocated in a coupling facility of CFLEVEL=9 or higher, the new list structure services allow entry key (KEYCOMPARE) and secondary key (SKEYCOMPARE) comparison to be performed to determine whether a list entry should be processed. With the KEYREQTYPE and SKEYREQTYPE keywords, the key comparison can specify that the entry key and the secondary key of the designated list entry be equal, less than or equal, or greater than or equal to a specified value.

When processing single list entries, key comparison is allowed on read, write, move, and delete operations. If the designated entry exists but fails to meet the comparison criterion, the IXLLSTE request is ended.

When processing multiple list entries, the key comparators of less than or equal and greater than or equal have been added to the List Structure Multiple Services requests. For list structures allocated in a coupling facility of CFLEVEL=9 or higher, these request types can also use a range of key values to be compared with the existing entry key. To be selectable for processing, the list entry must have a key value within the specified range.

When processing a list of list entries and locating list entries by keyed position, the key comparators of less than or equal and greater than or equal have been added to the READ_LIST request and are also available with the new DELETE_LIST request. These request types can also use a range of key values for selecting list entries for processing when the list structure is allocated in a coupling facility of CFLEVEL=9 or higher.

Enhancements to List Monitoring

In addition to the list and sublist monitoring functions that are provided by the IXLLIST macro, the IXLLSTC service provides the following additional monitoring functions:

- The capability to set up monitoring for a range of keys on a particular list, to start or stop the keyrange monitoring and obtain the keyrange monitoring information.
- The threshold values that the user can interrogate and modify. These threshold types are:

  - The list or keyrange empty threshold, which is the number of list entries that must remain in the list or keyrange to suppress a not-empty to empty list notification.
  - The list or keyrange not-empty threshold, which is the number of list entries that must be included in the list or keyrange before an empty to not-empty list notification is generated.
- For sublist monitoring, the function provides the option of specifying that EMCs should be queued to the user's events queue for only the first entry that is added to the sublist or for each entry that is added.
- For list and keyrange monitoring, the function provides the option of specifying the frequency in which the coupling facility updates the list notification vector for empty and not-empty list state transitions. For more information about aggressive list and keyrange monitoring and notification, as well as the affect it has on the frequency a connector's list transition exist receives control, see “Monitoring a List” on page 618 and “Monitoring a List by Keyrange Values” on page 620.
- The capability to override list and keyrange notification delays on a list and keyrange basis. For more information about list and keyrange notification delay function, see “Updating Control Information for a List” on page 618.
- A list monitoring function that allows a list structure connector to determine whether a list in the structure is full or not-full. If the list is full, the list count limit for entries or elements has been reached. If the list is not-full, the list count limit for entries or elements has not been reached. For more information about list full and not-full monitoring and how to have a list transition exit driven when a list transitions from a full to not-full state, see “Monitoring a List” on page 618.

Understanding Secondary Keys

A secondary key is a new type of key that can be used when creating, moving, or locating list entries, or when comparing the entry keys of list entries. A secondary list entry key is a 32-byte value that exists in parallel with the 16-byte list entry key and that represents a second key ordering for each list in the list structure.

Secondary keys are stored in the first half of the list entry adjunct data. Therefore, to support secondary keys, the list structure must be allocated not only with IXLCONN KEYTYPE=SECONDARY but also with ADJUNCT=YES. The use of secondary keys requires that the list structure be allocated in a coupling facility of CFLEVEL=9 or higher.

Some functions that are available for list entry key processing are also available for secondary key processing, namely:

- List entry location can be performed using either the entry key or the secondary key for all single entry request types. The READ_LIST and DELETE_LIST multiple entry requests can also locate list entries through either the entry key or the secondary key.
- Entry key comparison can be performed using either the entry key, the secondary key, or both entry key and secondary key, for all single entry and multiple entry request types.
- A list structure allocated with secondary keys will also have a secondary event queue. The current requests to monitor single or multiple sublists have been extended to support the monitoring of the secondary event queue as well as the primary event queue.
- The secondary key can be changed or updated using the list structure service, IXLLSTM REQUEST=MOVE_ENTRYLIST. This is the only service capable of updating a secondary key.

Using the New List Services Request Types

New request types are added to list services that provide function that is not available with the IXLLIST service.

- IXLLSTM REQUEST=DELETE_LIST allows the user to sequentially process list entries in the order in which they exist on the specified list.
- IXLLSTM REQUEST=MOVE_ENTRYLIST is used to process an input list of entries. The list entries can be identified by entry identifier or by entry name. For each list entry on the input list, the user can specify a target list number, a target list entry key, and a target secondary key, as appropriate. During processing of the list,

  - Each list entry can be updated or moved to a new position on a list in the structure (by updating the list entry key), or
  - Each list entry can be moved from one list to another list in the structure (by specifying a new target list number).

  IXLLSTM REQUEST=MOVE_ENTRYLIST provides optional version number comparison and the ability to halt processing of the list if a miscompare occurs on a list entry version number, list entry key, list entry secondary key, or list number. For consistency, these options are also added to the IXLLSTM REQUEST=DELETE_ENTRYLIST service.
- IXLLSTC REQUEST=MONITOR_KEYRANGE is used to determine whether the state of a list's keyrange is empty or not-empty, as specified by the keyrange thresholds.

Specifying 64-bit buffers

The IXLLSTC, IXLLSTE, and IXLLSTM services support 31-bit and 64-bit addressable storage. The BUFFER keyword supports only 31-bit addressable storage areas (below 2GB). The BUFLIST keyword supports both 31-bit addressable and 64-bit addressable (above 2GB) virtual storage areas, depending on the specifications for the BUFADDRTYPE and BUFADDRSIZE keywords. However, pageable high shared virtual storage areas (above 2GB) may not be used.

Comparing IXLLSTC, IXLLSTE, and IXLLSTM with IXLLIST

The following section s indicate differences in either the syntax or the keyword specification between the IXLLIST macro and the new list services macros in OS/390 Release 9.

Locating a List Entry

The IXLLSTE and IXLLSTM services provide options for the user to locate the list entry or list entries to be processed. The options are comparable to those used by the IXLLIST service and are described in “Referencing List Entries” on page 489. Some options provide additional capabilities for specifying the list entry, such as by comparing the entry name or key values. See “Enhancement to List Services Entry Key and Secondary Key Comparison” on page 596 for a description of the comparator options available with the IXLLSTE, IXLLSTM, and IXLLSTC services.

LOCATOR=CURSOR
Use the list cursor to designate the list entry. See also “Understanding the List Cursor” on page 494.

LOCATOR=ENTRYID
Use the ENTRYID to designate the list entry. Note that for list structures allocated in a coupling facility of CFLEVEL=8 or higher, the ENTRYID can be either system-assigned or user-assigned.

LOCATOR=ENTRYNAME
Use the ENTRYNAME (if the structure was allocated to use named entries) to designate the list entry.

LOCATOR=UNKEYPOS
Use LISTNUM and DIRECTION to designate the list entry at either the head or the tail of the list. Note that for IXLLIST services, the keyword LISTPOS is analogous to DIRECTION.

LOCATOR=KEYPOS
Use LISTNUM, DIRECTION, and the key specified by KEYTYPE (entry key or secondary key) to designate the list entry. Note that secondary keys are valid only in list structures that have been allocated with the appropriate attributes in coupling facilities of CFLEVEL=9 or higher.

Note that LOCATOR=KEYPOS is the only method of locating an existing list entry by secondary key. Use KEYTYPE=SECONDARY and specify SKEYREQTYPE to designate how a comparison is to be performed to find the list entry. SKEYREQTYPE provides comparison values of EQUAL, LESSOREQUAL, and GREATEROREQUAL, in addition to RANGE for multiple requests.

Specifying a Range of Values

List Control Services use the KEYRANGESTART and KEYRANGEEND keywords to establish the starting and ending values for a range of values when monitoring a keyrange to determine its empty or not-empty state.

List Multiple Entry Services use the KEYRANGEEND or SKEYRANGEEND keywords to specify the end value when locating or comparing keys.

Specifying Entry Keys or Secondary Keys

For structures allocated in a coupling facility of CFLEVEL=9 or higher, entry keys are specified by the keyword ENTRYKEY, as with IXLLIST. Secondary keys are specified by the keyword SECONDARYKEY. To indicate which key is to be used to scan a list, the keyword KEYSCANTYPE specifies either ENTRYKEY or SECONDARYKEY.

Identifying Individual List Entries for IXLLSTM Requests

Two new mapping macros are available for use with requests that reference a list of entry names or entry IDs that are to be moved or deleted.

- IXLYDELI, Delete Entrylist Input, maps the information needed to identify an individual list entry that is to be deleted with either:
  - IXLLSTM REQUEST=DELETE_ENTRYLIST, or
  - IXLLIST REQUEST=DELETE_ENTRYLIST
- IXLYMELI, Move Entrylist Input, maps the information needed to identify an individual list entry to be moved or updated with IXLLSTM REQUEST=MOVE_ENTRYLIST.

IXLLSTE: List Structure Single Entry Services

Use the IXLLSTE service to operate on individual list entries in a list structure. The following services are available:

- Create a new list entry.
- Read a list entry.
- Write a list entry.
- Move a list entry.
- Delete a list entry.

The following table lists the request types for the IXLLSTE macro.

Table 43. Request Types for IXLLSTE

Request Type	Description	Where described
DELETE	Delete a list entry from a coupling facility list structure. You can also read an entry into your buffer and delete it from the coupling facility list structure.	“Deleting a List Entry” on page 605
MOVE	Move a list entry from its current location to another.	“Moving a List Entry” on page 603
READ	Read an existing list entry.	“Reading a List Entry” on page 602
WRITE	Update an existing list entry or create a new list entry.	“Writing a List Entry” on page 603, “Creating a List Entry” on page 601

Selecting an Entry for Processing by IXLLSTE

Prior to the processing of an IXLLSTE request, you can specify comparison criteria that will control whether the entry is processed. In order for the IXLLSTE request to be performed, any and all of the requested comparison criteria must be met.

Requesting a Lock Operation as Part of an IXLLSTE Request

To perform a serialized operation, one in which a lock operation is performed together with an IXLLSTE READ, WRITE, MOVE, or DELETE operation, specify the LOCKOPER parameter on the IXLLSTE request. If list services cannot perform both the lock operation and the IXLLSTE operation, the request fails.

You can specify the following LOCKOPER values on an IXLLSTE request:

- SET
- RESET
- NOTHELD
- HELDBY

See "LOCK: Performing a Lock Operation" on page 568 for detailed information about the LOCKOPER parameter.

Requesting List Authority Checking

The AUTHCOMPARE keyword specifies whether list authority checking is to be performed to determine whether the entry should be processed. List authority comparison, if requested, precedes processing of the IXLLSTE request. If the criterion is not met, the IXLLSTE request is terminated.

List authority checking is meaningful only for list structures allocated in a coupling facility of CFLEVEL=1 or higher.

Requesting List Number Checking

The LISTCOMPARE keyword specifies whether list number comparison is to be performed to determine whether the entry should be processed. List number comparison, if requested, compares the list number on which the entry resides to a specified list number value. If the criterion is not met, the IXLLSTE request is terminated.

Requesting Version Number Checking

The VERSCOMPARE keyword specifies whether version number comparison is to be performed to determine whether the list entry should be processed. Version number comparison, if requested, compares the version number of the designed list entry to a specified version number value. Version number comparison can request that the version number of the designated list entry be equal or less than or equal to a specified version number value. If the criterion is not met, the IXLLSTE request is terminated.

Additionally, when moving or writing a list entry you can specify a version number to be assigned as the initial value for the new list entry and whether the list cursor for the list containing the newly created list entry should be updated. See "Understanding the List Entry Version Number" on page 523.

Version number comparison is meaningful only for list structures allocated in a coupling facility of CFLEVEL=1 or higher.

Requesting Entry Key Comparison Checking

The KEYCOMPARE keyword specifies that key comparison should be performed to determine whether the list entry should be processed. When ENTRYTYPE=OLD or ENTRYTYPE=ANY, key comparison can request that the designated list entry be equal, less than or equal, or greater than or equal to a specified entry key value. If the criterion (KEYREQTYPE) is not met, the IXLLSTE request is terminated.

Entry key comparison is meaningful only for list structures allocated in a coupling facility of CFLEVEL=9 or higher. If the structure was not allocated with keyed list entries, the KEYCOMPARE keyword is ignored.

Requesting Secondary Key Comparison Checking

The SKEYCOMPARE keyword specifies that secondary key comparison should be performed to determine whether the list entry should be processed. Secondary key comparison can request that the designated list entry be equal, less than or equal, or greater than or equal to a specified secondary key value. If the criterion (SKEYREQTYPE) is not met, the IXLLSTE request is terminated.

Secondary key comparison is meaningful only for list structures allocated in a coupling facility of CFLEVEL=9 or higher. If the structure was not allocated with secondary keys, the SKEYCOMPARE keyword is ignored.

Receiving Answer Area Information

When the IXLLSTE operation completes, information is returned in the list answer area, mapped by IXLYLAA.

Creating a List Entry

Use IXLLSTE ENTRYTYPE=ANY with REQUEST=WRITE or IXLLSTE ENTYTYPE=NEW to create a new list entry. ENTRYTYPE=ANY provides the option for creating a new list entry if the designated entry does not currently exist.

At structure allocation by IXLCONN, the following attributes of the list structure are established:

- Whether list entry identifiers are system-assigned or user-assigned (IXLCONN ENTRYIDTYPE parameter)
- Whether named entries or keyed entries (entry keys or secondary keys) are used to reference list entries.

When creating a new list entry, specify whether the list entry is to be assigned an entry name, entry key, or list key with the ASSIGN keyword. The initial allocation of the structure determines which ASSIGN value can be used.

- ASSIGN=NAME

  Assign the name specified for ENTRYNAME to the list entry. If a list entry already exists with the specified ENTRYNAME, a new list entry is not created and the IXLLSTE request is terminated. The newly created list entry is placed on the list specified by LISTNUM at the head or the tail specified by DIRECTION.
- ASSIGN=KEY

  Assign the entry key specified for ENTRYKEY to the list entry. If the structure was allocated to use secondary keys, assign the secondary key value specified by SECONDARYKEY to the list entry. If SECONDARYKEY is not supplied, the secondary key value is set to all binary zeros.

  If a keyed list entry already exists with either the specified ENTRYKEY or SECONDARYKEY, a new list entry is created. The list entry is placed on the list as follows:

  – If there exists a sublist of one or more entries with a matching key on the list, the target position is at the head or the tail of the sublist, as specified by DIRECTION.
  – If all existing list entries have a key greater than that specified by ENTRYKEY, the target position is at the head of the list.
  – If all existing list entries have a key less than that specified by ENTRYKEY, the target position is at the tail of the list.

  The list entry is placed on the list as follows:

  – If there exists a sublist of one or more entries with a matching secondary key on the list, the target position is at the head or tail of the sublist as specified by SKEYTARGETDIR.

- If all existing list entries have a secondary key greater than that specified by SECONDARYKEY, the target position relative to secondary key ordering is at the head of the list.
- If all existing list entries have a secondary key less than that specified by SECONDARYKEY, the target position is at the tail of the list.

If no matching entry key exists or if ENTRYKEY or SECONDARYKEY is neither the greatest nor least among the entry keys or secondary key, the target position for the newly created list entry is determined according to the entry key and secondary key sequence for the list.

- ASSIGN=LISTKEY

  Assign the current list key value to the newly created list entry. The list key and maximum list key values can be set with IXLLSTC REQUEST=WRITE_LCONTROLS. If the request specifies a list key value greater than the maximum list key value, the IXLLSTE request is terminated.

  If the list structure was allocated with secondary keys, assign the value of SECONDARYKEY to the list entry. The position relative to secondary key ordering is based on the LISTNUM specified and the value of SKEYTARGETDIR.

If ASSIGN=NONE is explicitly specified or defaulted to, and the structure was allocated to use keyed entries, the entry key value is assigned as follows:

- If DIRECTION=HEADTOTAIL, the list entry is assigned an entry key value of all binary zeros.
- If DIRECTION=TAILTOHEAD, the list entry is assigned an entry ket value of all binary ones.

When creating a new list entry with a user-assigned ENTRYID, specify the user ID value with the ENTRYID keyword. If a list entry already exists with the specified ENTRYID, a new list entry is not created and the IXLLSTE request is terminated. ENTRYID can be specified if and only if the initial allocation of the structure specified IXLCONN ENTRYIDTYPE=USER and the structure is allocated in a coupling facility of CFLEVEL=8 or higher.

Requesting Comparison Criteria when Creating a List Entry

When creating a new list entry with ENTRYTYPE=ANY,REQUEST=WRITE, any and all requested comparison criteria must be met in order for the IXLLSTE request to be performed.

- List authority
- Lock comparison
- Key comparison

See “Selecting an Entry for Processing by IXLLSTE” on page 600 for a description of the criteria that can be used when creating a new list entry.

When creating a new list entry, you can specify a new list authority value (NEWAUTH) that can be used to update the current list authority value. See “Updating the List Authority Value” on page 513.

Reading a List Entry

Use IXLLSTE ENTRYTYPE=OLD,REQUEST=READ to read an existing list entry. The information returned can be entry data, adjunct data, and list control data (or any combination thereof), as determined by the output areas provided. Entry data is returned in the area specified by BUFFER or BUFLIST; adjunct data is returned in the area specified by ADJAREA. At the successful completion of the request, the area identified by ANSAREA contains the list control information as well as the number of entries or elements residing on the list and the total number of allocated entries in the structure.

See “Locating a List Entry” on page 598 for information about locating the list entry to be processed. If the entry is not found, the READ operation is not performed and no change is made to the list structure.

Requesting Comparison Criteria when Reading a List Entry

When reading a list entry with ENTRYTYPE=OLD,REQUEST=READ, any and all requested comparison criteria must be met in order for the IXLLSTE request to be performed. See “Selecting an Entry for

Processing by IXLLSTE" on page 600 for a description of the criteria that can be used when determining whether to read the list entry.

Writing a List Entry

Use IXLLSTE REQUEST=WRITE with ENTRYTYPE=OLD or ENTRYTYPE=ANY to update a list entry. The list entry contents of the area specified by BUFFER or BUFLIST and the adjunct data contained in the area specified by ADJAREA are written to the designated list entry. See "Passing Data for a WRITE Request" on page 532. Upon successful completion of the request, the answer area contains the list entry controls, the number of entries or elements residing on the list, and the total number of allocated entries in the structure.

See "Selecting an Entry for Processing by IXLLSTE" on page 600 for a description of the criteria that can be used when determining whether to write a list entry.

See "Understanding the Write Operation" on page 531 for basic information about writing a single list entry and "Specifying the Type of Write Operation" on page 531 and "Specifying the Size of the Data Entry to Hold the Data" on page 532 for information about how the list entry is to be written.

Moving a List Entry

A list entry can be moved from one list to another or to another location on the same list. See "List Cursor Placement on a MOVE Request" on page 551 for basic information about source and target lists and cursor placement when moving a list entry.

Requesting Comparison Criteria when Moving a List Entry

When moving a list entry, any and all requested comparison criteria must be met in order for the IXLLSTE request to be performed. See "Selecting an Entry for Processing by IXLLSTE" on page 600 for a description of the criteria that can be used when determining whether to move the list entry.

Specifying the List Entry to Be Moved

Use the LOCATOR keyword to identify the list entry to be moved. See "Locating a List Entry" on page 598. To request that the entry be moved, specify either:

- IXLLSTE ENTRYTYPE=OLD,REQUEST=MOVE,ACTION=NONE

  Use ACTION=WRITE to specify that in addition to moving the list entry, entry data and adjunct data are to be written to the list entry. Use ACTION=READ to specify that in addition to moving the list entry, entry data and adjunct data are to be read from the list entry.
- IXLLSTE ENTRYTYPE=ANY,REQUEST=MOVE

  If the designated list entry does not exist, a new list entry will be created. See "Creating a New List Entry with an IXLLSTE Move Request" on page 604.

Specifying the Target List and List Position

The location to which the entry is to be moved is identified by the MOVETOLIST and MOVETODIRECTION keywords. The MOVETOLIST keyword identifies to list number of the list to which the entry is to be moved. MOVETODIRECTION specifies the target direction (the head or the tail of the list identified by MOVETOLIST).

For structures allocated to use keyed entries, the MOVETOKEY allows you to specify how an entry key is to be assigned to the moved list entry, which then affects how the list entry is positioned on the list. Options include assigning a user-specified key to the list entry, assigning the current list key value of the target list to the list entry, or maintaining the list entry's current key.

- If the structure was not allocated to use keyed entries, the moved list entry is placed at the head or the tail of the list as designated by MOVETODIRECTION.

- If the structure was allocated to use keyed entries, the moved list entry is placed at the head or the tail of the sublist as designated by MOVETODIRECTION and the resultant entry key as designated by MOVETOKEY.
- If the structure was allocated to use secondary keys, the moved list entry is placed at the head or the tail of the sublist as designated by SKEYTARGETDIR and the secondary key of the specified entry.

For structures allocated to use keyed entries, the MOVETOKEY allows you to specify how an entry key is to be assigned to the moved list entry, which then affects how the list entry is positioned on the list. Options include assigning a user-specified key to the list entry, assigning the current list key value of the target list to the list entry, or maintaining the list entry's current key.

When the list entry is to be moved from its current position on a sublist to another position on the same list, use the KEYPOSITION keyword. (This keyword is meaningful only for list structures allocated in a coupling facility of CFLEVEL=9 or higher.)

- To move a list entry from its current position on the sublist to a position on the sublist as specified by MOVETODIRECTION and MOVETOKEY, specify KEYPOSITION=UPDATE.
- To keep a list entry in its current position on the sublist, assuming that the list number specified by MOVETOLIST matches the list number currently containing the list entry and the list entry key is not changed, specify KEYPOSITION=KEEP.

When the list entry is to be moved from its current position on the secondary sublist to another position on the same list, use the SKEYPOSITION keyword. (This keyword is meaningful only for list structures allocated in a coupling facility of CFLEVEL=9 or higher.)

- To move a list entry from its current position on the secondary sublist to a position on the secondary sublist as specified by SKEYTARGETDIR, specify SKEYPOSITION=UPDATE.
- To keep a list entry in its current position on the secondary sublist, assuming that the list number specified by MOVETOLIST matches the list number currently containing the list entry, specify SKEYPOSITION=KEEP.

Changing the List Entry Key

For a list structure allocated to use keyed entries, the IXLLSTE service can be used to assign a new entry key to a moved list entry. Use the MOVETOKEY keyword to:

- Keep the current entry key of the list entry (MOVETOKEY=UNCHANGED).
- Assign a new key to the list entry if it is moved (MOVETOKEY=TARGETKEY). The assigned entry key is used in conjunction with MOVETOLIST and MOVETODIRECTION to designate the target keyed position of the list entry.
- Assign the current list key value of the target list to the moved list entry (MOVETOKEY=LISTKEY). Note that if you chose to increment the list key value after assigning it to the entry key and the resultant value is greater than the maximum list key value, the IXLLSTE request will be terminated.

Creating a New List Entry with an IXLLSTE Move Request

IXLLSTE ENTRYTYPE=ANY,REQUEST=MOVE allows you to create a new list entry if one does not currently exist.

Specifying Where to Place the New List Entry

If the structure was not allocated to use **named entries or keyed entries**, the newly created entry will be placed on the list specified by MOVETOLIST at the head or tail as specified by MOVETODIRECTION.

If the structure was allocated to use **named entries**, the entry name specified for ENTRYNAME is assigned to the newly created list entry, provided a list entry does not already exist with the same entry name. The newly created entry will be placed on the list specified by MOVETOLIST at the head or tail as specified by MOVETODIRECTION.

Identifying the New List Entry

When the new list entry is created, specify whether the list entry is to be assigned an entry ID, entry key, or list key.

If the structure was allocated to use a user-provided **entry ID**, the newly created list entry is assigned the ENTRYID specified by ASSIGNENTRYID, provided a list entry does not already exist with the same ENTRYID.

If the structure was allocated to use **keyed entries**, an **entry key** (and therefore, the target list keyed position) is assigned to the newly created list entry as follows:

- If ENTRYKEY is not specified, and TARGETKEY=NO_TARGETKEY is specified (explicitly or by default) with ASSIGNLISTKEY=NO or with ASSIGNLISTKEY=MOVE, then:
  - If MOVETODIRECTION=HEADTOTAIL is specified (explicitly or by default), the newly created list entry is assigned an entry key value of all binary zeros.
  - If MOVETODIRECTION=TAILTOHEAD is specified, the newly created list entry is assigned an entry key value of all binary ones.
- If ENTRYKEY is specified, and TARGETKEY=NO_TARGETKEY is specified (explicitly or by default) with ASSIGNLISTKEY=NO or with ASSIGNLISTKEY=MOVE, the newly created list entry is assigned the value specified by ENTRYKEY.
- If TARGETKEY is specified with ASSIGNLISTKEY=NO or with ASSIGNLISTKEY=MOVE, the newly created list entry is assigned the value specified for TARGETKEY.
- If ASSIGNLISTKEY=CREATE or ASSIGNLISTKEY=ANY is specified, the newly created list entry is assigned the **list key** value of the target list.
- The newly created list entry is placed on the list specified by MOVETOLIST at the head or tail of the sublist composed of list entries whose entry keys are equal to the assigned entry key. The newly created list entry is placed at the head or tail of this sublist as specified by MOVETODIRECTION. If a sublist of entries with entry keys equal to the assigned entry key does not yet exist, the newly created list entry is placed on the list in key sequence.
- The newly created list entry is placed on the list specified by MOVETOLIST at the head or tail of the sublist composed of list entries whose secondary keys are equal to the secondary key of the moved entry. The newly created list entry is placed at the head or tail of this sublist as specified by SKEYTARGETDIR. If a sublist of entries with secondary keys equal to the assigned secondary key does not yet exist, the newly created list entry is placed on the list in secondary key sequence.

Deleting a List Entry

Use IXLLSTE ENTRYTYPE=OLD,REQUEST=DELETE to delete a list entry from the list on which it resides.

Use IXLLSTE ENTRYTYPE=OLD,REQUEST=READ,ENTRYDISP=DELETE to read the list entry and then delete it from the list on which it resides.

To identify the list entry to be deleted, use the LOCATOR keyword. See "Locating a List Entry" on page 598.

See "Selecting an Entry for Processing by IXLLSTE" on page 600 for a description of the criteria that can be used when determining whether to delete the list entry.

See "Deleting a Keyed List Entry in a CFLEVEL=3 or Higher Coupling Facility" on page 558 and "Receiving Data on a DELETE Request" on page 559 for additional information about deleting a list entry from a list.

IXLLSTM: List Structure Multiple Entry Services

Use the IXLLSTM service to operate on multiple list entries in a list structure. The following services are available:

- Read list entries from a list.
- Delete list entries from a list.

- Read list entries from multiple lists.
- Delete list entries from multiple lists.
- Move list entries identified by a list of entry identifier or entry names.
- Delete list entries identified by a list of entry identifiers or entry names.

The following table lists the IXLLSTM request types:

Table 44. Request Types for IXLLSTM

Request Type	Description	Where described
DELETE_ENTRYLIST	Delete multiple list entries that are designated in an entry list contained in the storage area specified by BUFFER or BUFLIST.	"Deleting a List of List Entries" on page 614
DELETE_LIST	Delete multiple entries from a list.	"Deleting List Entries from a List" on page 609
DELETE_MULT	Delete multiple list entries from a coupling facility list structure.	"Deleting List Entries from Multiple Lists" on page 612
MOVE_ENTRYLIST	Move multiple list entries that are designated in an entry list contained in the storage area specified by BUFFER or BUFLIST.	"Moving a List of List Entries" on page 613
READ_LIST	Read multiple list entries from a list.	"Reading List Entries from a List" on page 607
READ_MULT	Read multiple list entries from a list structure.	"Reading List Entries from Multiple Lists" on page 611

Selecting Entries for Processing by IXLLSTM

Prior to the processing of an IXLLSTM request, you can specify comparison criteria that will control whether the entry is processed. In order for the IXLLSTM request to be performed, any and all of the requested comparison criteria must be met.

Requesting a Lock Operation as Part of an IXLLSTM Request

To perform a serialized operation, one in which a lock operation is performed together with an IXLLSTM operation, specify the LOCKOPER parameter on the IXLLSTM request. If list services cannot perform both the lock operation and the IXLLSTM operation, the request fails.

You can specify the following LOCKOPER values on an IXLLSTM request:

- NOTHELD
- HELDBY

See "LOCK: Performing a Lock Operation" on page 568 for detailed information about the LOCKOPER parameter.

Requesting List Authority Checking

The AUTHCOMPARE keyword specifies whether list authority checking is to be performed to determine whether the entry should be processed. List authority comparison, if requested, precedes processing of the IXLLSTM request. If the comparison does not meet the condition specified by the AUTHCOMPTYPE keyword (EQUAL or LESSOREQUAL), the request fails.

List authority checking is meaningful only for list structures allocated in a coupling facility of CFLEVEL=1 or higher.

Requesting List Number Checking

The LISTCOMPARE keyword specifies whether list number comparison is to be performed to determine whether the entry should be processed. List number comparison, if requested, compares the list number on which the entry resides to a specified list number value.

Requesting Version Number Checking

The VERSCOMPARE keyword specifies whether version number comparison is to be performed to determine whether the list entry should be processed. Version number comparison, if requested, compares the version number of the designed list entry to a specified version number value. Version number comparison can request that the version number of the designated list entry be equal or less than or equal to a specified version number value.

Version number comparison is meaningful only for list structures allocated in a coupling facility of CFLEVEL=1 or higher.

Requesting Entry Key Comparison Checking

The KEYCOMPARE keyword specifies that key comparison should be performed to determine whether the list entry should be processed. Key comparison can request that the designated list entry be equal, less than or equal, or greater than or equal to a specified entry key value.

Entry key comparison is meaningful only for list structures allocated in a coupling facility of CFLEVEL=9 or higher. If the structure was not allocated with keyed list entries, the KEYCOMPARE keyword is ignored.

Requesting Secondary Key Comparison Checking

The SKEYCOMPARE keyword specifies that secondary key comparison should be performed to determine whether the list entry should be processed. Secondary key comparison can request that the designated list entry be equal, less than or equal, or greater than or equal to a specified secondary key value.

Secondary key comparison is meaningful only for list structures allocated in a coupling facility of CFLEVEL=9 or higher. If the structure was not allocated with secondary keys, the SKEYCOMPARE keyword is ignored.

Receiving Answer Area Information

When the IXLLSTM operation completes, information is returned in the list answer area, mapped by IXLYLAA.

Restarting IXLLSTM Requests

An IXLLSTM request may complete prematurely. See each of the IXLLSTM request types for the protocol to be followed if the IXLLSTM request does not fully complete.

Reading List Entries from a List

Use IXLLSTM REQUEST=READ_LIST to request that a list scan process be performed such that reading the entries that meet a specific set of criteria. The entry data, adjunct data, list entry controls, or any combination of these for the selected entries on the list might be read into the buffer storage area specified for the request (designated by BUFFER or BUFLIST).

When adjunct data is requested, the adjunct data for the first entry processed is returned in the storage area specified by ADJAREA. The adjunct data for all other entries is returned in the buffer storage area.

When list entry controls are requested, the entry controls for the first entry processed are returned in the answer area specified by ANSAREA. The entry controls for all other entries are returned in the buffer storage area.

The LOCATOR keyword designates the entry, and the DIRECTION keyword specifies the direction. The Processing begins with the entry and proceeds sequentially along the list in the direction until the head

or tail of the list is reached. If the entry key comparison or the secondary key comparison is requested, the process ends when the scan has progressed past all entries for which those key comparisons is successful.

Specifying AUTHCOMPARE=YES in conjunction with AUTHCOMP causes list authority comparison for the designated list to precede processing of any list entries. If the list authority verification fails, the list authority, the list control information and appropriate return and reason codes are provided in the ANSAREA.

Specifying LOCKINDEX in conjunction with LOCKOPER causes lock comparison to precede processing of any list entries. LOCKINDEX can optionally be specified to indicate the index of the serialized list lock to be compared within the lock table for the list structure. If the lock comparison fails, the lock table entry and appropriate return and reason codes are provided in the ANSAREA.

Note: The use of serialized list functions might be useful to serialize the list against other concurrent operations that could otherwise execute in parallel with the READ_LIST scan, because these concurrent operations can cause entries on the list to be skipped or scanned more than once during READ_LIST processing.

Specifying LISTCOMPARE=YES in conjunction with LISTNUM causes list number comparison for the designated starting list entry to precede processing of any list entries. If the designated list entry exists, but the list number verification fails, the list entry controls and appropriate return and reason codes are provided in the ANSAREA.

Specifying VERSCOMPARE=YES in conjunction with VERSCOMP might optionally be used as a filter to restrict processing to those entries with a version number as specified by VERSCOMPTYPE.

Specifying KEYCOMPARE=YES in conjunction with ENTRYKEY causes key comparison to be performed as a filter, as specified by KEYREQTYPE.

Specifying SKEYCOMPARE=YES in conjunction with SECONDARYKEY causes secondary key comparison to be performed as a filter, as specified by SKEYREQTYPE.

DIRECTION can optionally be specified to indicate direction of processing for traversing the list.

The absence of AUTHCOMPARE or LISTCOMPARE, or specifying AUTHCOMPARE=NO or LISTCOMPARE=NO indicates no list authority or list number comparisons are to be performed before processing any list entries. The absence of LOCKINDEX indicates that no lock comparison is to be performed before processing any list entries.

The absence of VERSCOMPARE, KEYCOMPARE, or SKEYCOMPARE or specifying VERSCOMPARE=NO, KEYCOMPARE=NO, or SKEYCOMPARE=NO indicates no version number, entry key value, or secondary key value comparisons are to be performed as a filter when selecting entries to be processed.

For any list entries to be processed, the list number comparison, the list authority comparison and the lock operation, must succeed if they are requested.

For a particular list entry to be read, the version number comparison, the entry key comparison, and the secondary key comparison, must succeed if they are requested. Otherwise, no processing is performed for the current entry and processing continues with the next entry to be considered.

When the request completes successfully, the number of entries for which entry data, adjunct data, or list entry controls or both was read is returned in the answer area specified by ANSAREA.

A READ_LIST request might complete prematurely due to exhaustion of the storage specified for the buffer storage area, or if coupling facility model dependent timeout criteria is exceeded. In this event appropriate return and reason codes are provided, and the number of entries for which data has been returned on the current request is provided in ANSAREA. The list entry controls for the next appropriate entry in the list sequence to be processed is returned in ANSAREA. These list entry controls can be used to designate the entry with which to resume processing on a subsequent, resuming READ_LIST request, so as to continue the overall scan process for the list.

Note: The disposition of this list entry might change as a result of another operation (for example, the entry may be deleted or moved to another position on the same list or a different list) after the completion of the first READ_LIST request and before the invocation of the resuming READ_LIST request. This might

cause the resuming READ_LIST request to fail, skip entries, or reprocess some entries that have already been processed. If a resuming READ_LIST request fails with an "entry not found" condition, the list might not yet have been completely scanned and the scan needs to be restarted from the beginning.

If other concurrent operations, such as other commands that can create entries, delete entries, or move entries to a different position on the same list or a different list, are permitted to execute while READ_LIST processing is ongoing, this can result in anomalous behavior for the overall READ_LIST scan process. Entries on the list can be skipped or reprocessed more than once, and this might result in entries that should have been read being "missed" by the scan, or might result in the same entry being read multiple times. For example:

- The list is being scanned in a left-to-right direction by a READ_LIST command, and a concurrently-executing command moves an entry on the list to the right. If the moved entry was being processed by the READ_LIST command at the time it was moved, the intervening entries between the moved entry's old position and its new position on the list can be skipped by the scan process.
- The list is being scanned in a left-to-right direction by a READ_LIST command, and a concurrently executing command moves an entry on the list to the left. If the moved entry was being processed by the READ_LIST command at the time it was moved, the intervening entries between the moved entry's old position and its new position on the list might be reprocessed by the scan process, and if the entries pass the requested filtering criteria, might be read in again.
- An entry that matches the READ_LIST request's filtering criteria is created on, or moved into, the list that is being scanned. Depending on whether the entry is placed onto the list "ahead of" or "behind" the ongoing scan process, the entry might or might not be read.
- Other examples are also possible.

Note: Such anomalies can occur both within the processing of a single READ_LIST request, and in the gap between the completion of one READ_LIST request and the initiation of a subsequent, resuming one.

In order to avoid such anomalies, consider making use of serialized list functions (LOCKINDEX and LOCKOPER) to lock out concurrent operations for the duration of the entire READ_LIST scan process, from before initiating the first request, through any premature completion and re-drive processing that may occur, until the scan process indicates that the list has been processed to completion.

Resumed requests are processed identically to non-resumed requests and must meet the same interface requirements as non-resumed requests. For example, the buffer storage area boundary and length requirements are unchanged. Resumed requests might in turn experience premature completion.

See “READ_LIST: Reading Multiple List Entries from a List” on page 541 for additional information about reading list entries, including how to handle an incompletely processed request.

Deleting List Entries from a List

Using IXLLSTM REQUEST=DELETE_LIST to request that a list scan process be performed such that entries meeting a specified set of criteria are removed from the list on which they reside and returned to the pool of free entries for reuse.

Processing begins with the entry located by the LOCATOR keyword and proceeds sequentially along the list in the direction specified by DIRECTION until the head or tail of the list is reached, or if entry key comparison or secondary key comparison is requested, until the scan has progressed past all entries for which those key comparisons could be successful.

Specifying AUTHCOMPARE=YES in conjunction with AUTHCOMP causes list authority comparison for the designated list to precede processing of any list entries. If the list authority verification fails, the list authority, the list control information and appropriate return and reason codes are provided.

Specifying LOCKINDEX in conjunction with LOCKOPER causes lock comparison to precede processing of any list entries. LOCKINDEX may optionally be specified to indicate the index of the serialized list lock to be compared within the lock table for the list structure. If the lock comparison fails, the lock table entry and appropriate return and reason codes are provided.

Note: The use of serialized list functions might be useful to serialize the list against other concurrent operations that could otherwise execute in parallel with the DELETE_LIST scan, because these concurrent

operations can cause entries on the list to be skipped or scanned more than once during DELETE_LIST processing.

Specifying LISTCOMPARE=YES in conjunction with LISTNUM causes list number comparison for the designated starting list entry to precede processing of any list entries. If the designated list entry exists, but the list number verification fails, the list entry controls and appropriate return and reason codes are provided.

Specifying VERSCOMPARE=YES in conjunction with VERSCOMP can optionally be used as a filter to restrict processing to those entries with a version number matching that specified by VERSCOMPTYPE.

Specifying KEYCOMPARE=YES in conjunction with ENTRYKEY causes entry key comparison to be performed as a filter, as specified by KEYREQTYPE.

Specifying SKEYCOMPARE=YES in conjunction with SECONDARYKEY causes secondary key comparison to be performed as a filter, as specified by SKEYREQTYPE.

DIRECTION can optionally be specified to indicate direction of processing for traversing the list.

The absence of AUTHCOMPARE or LISTCOMPARE or specifying AUTHCOMPARE=NO or LISTCOMPARE=NO indicates no list authority or list number comparisons are to be performed before processing any list entries. The absence of LOCKINDEX indicates that no lock comparison is to be performed before processing any list entries.

The absence of VERSCOMPARE, KEYCOMPARE, or SKEYCOMPARE, or specifying VERSCOMPARE=NO, KEYCOMPARE=NO or SKEYCOMPARE=NO indicates no version number, entry key value, or secondary key value comparisons are performed as a filter when selecting entries to be processed.

For any list entries to be processed, the list number comparison, the list authority comparison, and the lock operation, must succeed if they are requested.

For a particular list entry to be deleted, the version number comparison, the entry key comparison, and the secondary key comparison, must succeed if they are requested. Otherwise, no processing is performed for the current entry and processing continues with the next entry to be considered.

When the request completes successfully the number of entries deleted for this request is returned in the answer area specified by ANSAREA.

A DELETE_LIST request might complete prematurely if the coupling facility model dependent timeout criteria is exceeded. In this event appropriate return and reason codes, and the number of entries which have been deleted by the current request are provided. The list entry controls for the next appropriate entry in the list sequence to be processed is returned in ANSAREA. These list entry controls can be used to designate the entry with which to resume processing on a subsequent, resuming DELETE_LIST request, so as to continue the overall scan process for the list.

Note: The disposition of this list entry might change as a result of another operation (for example, the entry might be deleted or moved to another position on the same list or a different list) after the completion of the first DELETE_LIST request and before the invocation of the resuming DELETE_LIST request. This might cause the resuming DELETE_LIST request to fail, skip entries, or reprocess some entries that have already been processed. If a resuming DELETE_LIST request fails with an "entry not found" condition, the list might not yet have been completely scanned and the scan needs to be restarted from the beginning.

If other concurrent operations, such as other commands which can create entries, delete entries, or move entries to a different position on the same list or a different list, are permitted to execute while DELETE_LIST processing is ongoing, this can result in anomalous behavior for the overall DELETE_LIST scan process. Entries on the list might be skipped or reprocessed more than once, and this might result in entries that should have been deleted being "missed" by the scan. For example:

- The list is being scanned in a left-to-right direction by a DELETE_LIST command, and a concurrently-executing command moves an entry on the list to the right. If the moved entry was being processed by the DELETE_LIST command at the time it was moved, the intervening entries between the moved entry's old position and its new position on the list can be skipped by the scan process.

- The list is being scanned in a left-to-right direction by a DELETE_LIST command, and a concurrently executing command moves an entry on the list to the left. If the moved entry was being processed by the DELETE_LIST command at the time it was moved, the intervening entries between the moved entry's old position and its new position on the list can be reprocessed by the scan process.
- An entry that matches the DELETE_LIST request's filtering criteria is created on, or moved into, the list that is being scanned. Depending on whether the entry is placed onto the list "ahead of" or "behind" the ongoing scan process, the entry might or might not be deleted.
- Other examples are also possible.

Note: Such anomalies can occur both within the processing of a single DELETE_LIST request, and in the gap between the completion of one DELETE_LIST request and the initiation of a subsequent, resuming one.

To avoid such anomalies, consider using serialized list functions (LOCKINDEX and LOCKOPER) to lock out concurrent operations for the duration of the entire DELETE_LIST scan process, from before initiating the first request, through any premature completion and re-drive processing that may occur, until the scan process indicates that the list has been processed to completion.

Resumed requests are processed identically to non-resumed requests, and they must meet the same interface requirements as non-resumed requests. Resumed requests might in turn experience premature completion.

This is a new list services function introduced in Release 9 and cannot be invoked on systems running a level of OS/390 lower than Release 9. DELETE_LIST is only valid when the structure is allocated in a CFLEVEL=9 or higher CF.

Reading List Entries from Multiple Lists

Use IXLLSTM REQUEST=READ_MULT to read the entry data, adjunct data, and list entry controls for all allocated entries in the list structure that meet a specified set of criteria.

Request Comparison Checking on a READ_MULT Request

Comparison checking that can be done prior to processing any list entries can be one or more or the following:

- List authority comparison
- Lock operation

In order for any list entries to be processed, the list authority comparison and the lock operation, if requested, must succeed. If a requested comparison fails, the IXLLSTM request is terminated.

Additional comparison checking that can be done prior to processing each list entry can be one or more of the following:

- List number comparison
- Version number comparison
- Entry key comparison
- Secondary key comparison

In order for a particular list entry to be read, the list number comparison, version number comparison, entry key comparison, and secondary key comparison, if requested, must succeed. If any of the comparisons fails, no processing is performed for the current list entry and processing continues with the next entry.

Handling an Incompletely Processed READ_MULT Request

A READ_MULT request might complete prematurely if not enough buffer space is available or the coupling facility model dependent timeout criteria is exceeded. If this happens, the following is returned in the answer area:

- Appropriate return and reason codes
- Number of list entries for which data has been returned
- A restart token or extended restart token, which can be used to resume processing the request.

To continue processing, reissue the request specifying the restart token or extended restart token as input.

Continue reissuing the READ_MULT request until the return code indicates that all processing has completed.

See "READ_MULT: Reading Multiple List Entries from One or More Lists" on page 547 for additional information about reading list entries from multiple lists, including how to handle an incompletely processed request.

Deleting List Entries from Multiple Lists

Use IXLLSTM REQUEST=DELETE_MULT to delete all list entries that meet a specified set of criteria be deleted from whichever list on which they reside in the structure.

Request Comparison Checking on a DELETE_MULT Request

Comparison checking that can be done prior to processing any list entries can be one or more or the following:

- List authority comparison
- Lock operation

In order for any list entries to be processed, the list authority comparison and the lock operation, if requested, must succeed. If a requested comparison fails, the IXLLSTM request is terminated.

Additional comparison checking that can be done prior to processing each list entry can be one or more of the following:

- List number comparison
- Version number comparison
- Entry key comparison
- Secondary key comparison

In order for a particular list entry to be read, the list number comparison, version number comparison, entry key comparison, and secondary key comparison, if requested, must succeed. If any of the comparisons fails, no processing is performed for the current list entry and processing continues with the next entry.

Handling an Incompletely Processed DELETE_MULT Request

A DELETE_MULT request might complete prematurely if the coupling facility model dependent timeout criteria is exceeded. If this happens, the following is returned in the answer area:

- Appropriate return and reason codes
- Number of list entries for which data has been returned
- A restart token or extended restart token, which can be used to resume processing the request.

To continue processing, reissue the request specifying the restart token or extended restart token as input.

Continue reissuing the DELETE_MULT request until the return code indicates that all processing has completed.

See "DELETE_MULT: Deleting Multiple List Entries" on page 560 for additional information about deleting list entries, including how to handle an incompletely processed request.

Moving a List of List Entries

Use IXLLSTM REQUEST=MOVE_ENTRYLIST to move the list entries identified in a list of entry IDs or entry names from the current source location to a designated target location. The LISTTYPE parameter indicates whether you are providing a list of entry IDs or entry names. IXLLSTM REQUEST=MOVE_ENTRYLIST is valid only when the structure is allocated in a coupling facility of CFLEVEL=9 or higher.

Request Comparison Checking on a MOVE_ENTRYLIST Request

Comparison checking that can be done prior to moving a list entry can be one or more of the following:

- List authority
- Lock operation

In order for any list entries to be moved, the list authority comparison and the lock operation, if either or both are requested, must succeed. If during the processing of the first list entry, the list authority comparison or lock operation fails, the IXLLSTM request is terminated. If the authority comparison or lock operation fails during processing of a subsequent entry after the first, the request will time out and can be restarted. See "Handling an Incompletely Processed MOVE_ENTRYLIST" on page 614 for information about restarting a MOVE_ENTRYLIST request.

Additional comparison checking that can be done prior to processing each list entry can be one or more of the following:

- Version number comparison with optional version number replacement
- List number comparison
- Key comparison (both entry key and secondary key)

In order for a particular list entry to be moved, the version number comparison, list number comparison, and key comparison, if requested, must succeed. If any of the comparisons fails, and MISCOMPARE=HALT was specified, the IXLLSTM request is terminated. If MISCOMPARE=CONTINUE was specified or defaulted to, processing of the current list entry is terminated and processing continues with the next entry.

Passing the List of Entries to be Moved

An array of elements in the BUFFER or BUFLIST storage areas contains the information about the list entries to be moved. Each array element identifies the entry to be moved and may also designate version number comparison and key assignment information. The array elements are mapped by the IXLYMELI macro and are indexed with the first element starting at offset zero in the BUFFER or BUFLIST area. Identify the index of the first element to be processed with FIRSTELEM and the index of the last element with LASTELEM.

IXLYMELI, Move Entrylist Input, contains three mappings:

- Each element in the array is mapped by MELI1 when:
  - The structure does not support keyed entries and VERSCOMPARE=YES or NO is specified.
  - The structure does support keyed entries and entry keys and secondary keys are not to be updated (MOVETOKEY=UNCHANGED), (MOVETOSKEY=UNCHANGED) with VERSIONCOMPARE=NO, or VERSIONCOMPARE=YES is specified.
  - The structure does not support keyed entries and entry keys are to be updated with the list key value (MOVETOKEY=LISTKEY), MOVETOSKEY=UNCHANGED with VERSIONCOMPARE=NO, or VERSIONCOMPARE=YES is specified.
- Each element in the array is mapped by MELI2 when VERSCOMPARE=BYENTRY or entry key update (MOVETOKEY=TARGETKEY) is specified with MOVETOSKEY=UNCHANGED (secondary key not changed).
- Each element in the array is mapped by MELI3 when the secondary key is to be updated (MOVETOSKEY=TARGETKEY is specified).

Each IXLYMELI element also contains the target list to which the list entry is to be moved, the direction in which the entry is to be positioned on the list or sublist, and whether the list limit set for the target list should be enforced or ignored.

Updating Key Values When Moving List Entries

Both the list entry key and the secondary key may be updated when moving a list entry from the source to the target list. The MOVETOKEY keyword indicates whether the entry key is to remain unchanged, updated to the list key value, or assigned a new entry key value. The MOVETOSKEY keyword indicates whether the secondary key is to remain unchanged or assigned a new secondary key value. Note that the MOVE_ENTRYLIST request is the only request that can be used to change the secondary key value for an existing entry.

Handling an Incompletely Processed MOVE_ENTRYLIST

If any entry specified in the input array does not exist, then processing is halted. The count of the list entries moved and the current array element index of the non-existent list entry are returned in the answer area. To continue processing, reissue the request starting with the entry after the one that could not be found (increment the current array element index by 1).

A MOVE_ENTRYLIST request might also complete prematurely if the coupling facility model dependent timeout criteria is exceeded. If this happens, the following is returned in dthe answer area:

- Appropriate return and reason codes
- Number of list entries that have been moved up to this point
- Index of the first unprocessed list entry
- If MOVETOKEY=LISTKEY was specified on the request, the entry key value for each array entry is also returned in LISTKEYAREA.

To continue processing, reissue the request starting with the entry indexed by the current array element index returned in the answer area. Continue reissuing the MOVE_ENTRYLIST request until the return code indicates that all processing has completed.

Deleting a List of List Entries

Use IXLLSTM REQUEST=DELETE_ENTRYLIST to delete the list entries identified in a list of entry IDs or entry names. As with IXLLIST REQUEST=DELETE_ENTRYLIST, the LISTYPE parameter indicates whether you are providing a list of entry IDs or entry names. See “DELETE_ENTRYLIST: Deleting a List of Entries” on page 562.

Request Comparison Checking on a DELETE_ENTRYLIST Request

Comparison checking that can be done prior to deleting a list entry can be one or more of the following:

- List authority
- Lock operation

In order for any list entries to be deleted, the list authority comparison and the lock operation, if either or both are requested, must succeed. If the list authority comparison or lock operation fails, the IXLLSTM request is terminated. If the authority comparison or lock operation fails during processing of a subsequent entry after the first, the request fails and can be restarted. See “Handling an Incompletely Processed MOVE_ENTRYLIST” on page 614 for information about restarting a DELETE_ENTRYLIST request.

Additional comparison checking that can be done prior to processing each list entry can be one or more of the following:

- Version number comparison
- List number comparison
- Key comparison (both entry key and secondary key)

In order for a particular list entry to be deleted, the version number comparison, list number comparison, and key comparison, if requested, must succeed. If any of the comparisons fails, and MISCOMPARE=HALT was specified, the IXLLSTM request is terminated. If MISCOMPARE=CONTINUE was specified or defaulted to, processing of the current list entry is terminated and processing continues with the next entry.

Passing the List of Entries to be Deleted

An array of elements in the BUFFER or BUFLIST storage areas contains the information about the list entries to be deleted. Each array element identifies the entry to be moved and may also designate version number comparison information for a structure allocated in a coupling facility of CFLEVEL=9 or higher. The array elements are mapped by the IXLYDELI macro and are indexed with the first element starting at offset zero in the BUFFER or BUFLIST area. Identify the index of the first element to be processed with FIRSTELEM and the index of the last element with LASTELEM.

IXLYDELI, Delete Entrylist Input, contains three mappings:

- Each element in the array is mapped by DELI1 when LISTTYPE=NAMELIST and VERSCOMPARE=YES or NO is specified.
- Each element in the array is mapped by DELI2 when LISTTYPE=IDLIST and VERSCOMPARE=YES or NO is specified.
- Each element in the array is mapped by DELI3 when VERSCOMPARE=BYENTRY is specified for either LISTTYPE=NAMELIST or LISTTYPE=IDLIST.

The mappings for DELI1 and DELI2 may also be used with the appropriate IXLLIST REQUEST=DELETE_ENTRYLIST request type. The DELI3 mapping applies only to IXLLSTM REQUEST=DELETE_ENTRYLIST.

IXLLSTC: List Structure Control Services

Use the IXLLSTC service to operate on control information for a list structure. The following services are available:

- Read control information for a list.
- Update control information for a list.
- Read event queue control information for a list.
- Read event monitor controls (EMC) control information for a list.
- Dequeue event monitor controls from an event queue.
- Start or stop event queue monitoring.
- Start or stop list monitoring.
- Start or stop keyrange monitoring of a list.
- Start or stop monitoring of a sublist.
- Start or stop monitoring of a set of sublists.
- Lock operations.
- Read structure counts.

The following table lists the IXLLSTC request types:

Table 45. Request Types for IXLLSTC

Request Type	Description	Where described
DEQ_EVENTQ	Dequeue event monitor controls from an event queue.	"Dequeuing Event Monitor Controls from an Event Queue" on page 623

Table 45. Request Types for IXLLSTC (continued)		
Request Type	**Description**	**Where described**
LOCK	Operate on a lock entry in the lock table associated with the list structure.	"Performing Lock Operations" on page 623
MONITOR_EVENTQ	Start or stop list notification vector monitoring of an event queue.	"Monitoring an Event Queue" on page 622
MONITOR_KEYRANGE	Start or stop key-range monitoring of a list.	"Monitoring a List by Keyrange Values" on page 620
MONITOR_LIST	Start or stop monitoring a list for an empty or not-empty condition.	"Monitoring a List" on page 618
MONITOR_SUBLIST	Start or stop monitoring a sublist.	"Monitoring a Sublist" on page 620
MONITOR_SUBLISTS	Start monitoring a set of sublists.	"Monitoring a Set of Sublists" on page 620
READ_EMCONTROLS	Read the EMC control information for a user's registered interest in monitoring a particular sublist.	"Reading EMC Control Information" on page 622
READ_EQCONTROLS	Read event queue control information for the user's event queue.	"Reading Event Queue Control Information" on page 622
READ_LCONTROLS	Read the control information for a list.	"Reading Control Information for a List" on page 618
READ_STRCOUNTS	Read the structure counts for a list.	"Reading Structure Counts for a List Structure " on page 623
WRITE_LCONTROLS	Update one or more of the list controls for a list.	"Updating Control Information for a List" on page 618

Understanding Threshold Counts

Threshold counts specify the minimum and maximum values that are used to define the empty or not-empty state of a list. Threshold counts are used when monitoring lists and sublists.

Using Threshold Counts when Monitoring List State Transitions

Depending on the CFLEVEL of the coupling facility in which the list structure is allocated, list structure transitions from the empty state to the not-empty state will be recognized differently.

- For structures allocated in a coupling facility of CFLEVEL=8 or lower, a list changes from the empty to the not-empty state when the list has no entries and an entry is created on or moved to the list. A list changes from the not-empty state to the empty state when the list has one entry and the entry is deleted or moved to another list.
- For structures allocated in a coupling facility of CFLEVEL=9 or higher, the initial state of a list is empty. The subsequent empty or not-empty state is determined by the value of a set of threshold counts for the list. The threshold counts specify the number of list entries required to define the list as either empty or not-empty. The values of LISTEMPTY and LISTNOTEMPTY specity the empty and not-empty threshold counts for a list.

A list is in the empty state until the number of list entries on the list becomes greater than the list not-empty threshold.

Setting the Threshold Counts for List Monitoring

For keyed list structures allocated in a coupling facility of CFLEVEL=9 or higher, use LISTSTATE=DEFINE to set or update the threshold counts that define the empty or not-empty state of a list. LISTEMPTY and LISTNOTEMPTY specify the values to be assigned as the empty and not-empty values. The not-empty value must be greater than the empty value. When the structure is allocated, the threshold count values are initialized to zero for all lists. For structures allocated in a coupling facility at CFLEVEL=8 or lower, LISTSTATE=DEFINE is ignored and the list empty and list not-empty threshold counts are always zero.

Use IXLLSTC REQUEST=MONITOR_LIST to register interest in monitoring transitions between the empty and not-empty states.

Using Threshold Counts when Monitoring a Range of Key Values

Operating with a range of key values is valid only for list structures that support keyed list entries allocated in a coupling facility of CFLEVEL=9 or higher. A key-range is either in the empty state or the not-empty state.

- KREMPTY specifies the key-range empty threshold count to be associated with the list. A key-range is in the empty state if the number of list entries in the key-range is either less than or equal to the KREMPTY value or zero. Once the key-range state becomes empty, it remains empty until the number of list entries in the key-range becomes greater than the KRNOTEMPTY value.
- KRNOTEMPTY specifies the key-range not-empty threshold count to be associated with the list. The key-range not-empty value must be greater than or equal to the key-range empty count. A key-range is in the not-empty state if the number of list entries in the key-range is greater than the KRNOTEMPTY value. Once the key-range state is not-empty, it remains not-empty until the number of list entries in the key-range either becomes less than or equal to the KREMPTY threshold or becomes zero.

Setting the Threshold Counts for Keyrange Monitoring

For keyed list structures allocated in a coupling facility of CFLEVEL=9 or higher, use KEYRANGESTATE=DEFINE to set or update the threshold counts that define the empty or not-empty state of a keyrange. KREMPTY and KRNOTEMPTY specify the values to be assigned as the empty and not-empty values. The not-empty value must be greater than the empty value. Key-range start and stop values are initialized to binary zeros when the structure is allocated. The keyrange for the list can be set with the IXLLSTC KEYRANGE=SET; starting and ending values of the keyrange are specified with KEYRANGESTART and KEYRANGEEND.

For structures allocated in a coupling facility at CFLEVEL=8 or lower, KEYRANGESTATE=DEFINE is ignored and the list empty and list not-empty threshold counts are always zero.

Use IXLLSTC REQUEST=MONITOR_KEYRANGE to register interest in monitoring transitions between key-range states.

Handling an Incompletely Processed WRITE_LCONTROLS Request that Specifies KEYRANGESTATE=DEFINE

A WRITE_LCONTROLS request that specifies KEYRANGESTATE=DEFINE might complete prematurely if the coupling facility model dependent timeout criteria is exceeded. The caller is expected to reissue the request until it completes without timing out.

If this happens, the appropriate return and reason codes are retuned in the answer area. Continue reissuing the WRITE_LCONTROLS request until the return code indicates that all processing has completed.

Reading Control Information for a List

Use IXLLSTC REQUEST=READ_LCONTROLS to obtain the list control information for a specific list. See "READ_LCONTROLS: Reading List Controls" on page 565 for general information about reading list control information and the information returned when the request completes.

For structures allocated in a coupling facility of CFLEVEL=9 or higher, list monitoring by threshold counts and keyrange monitoring are supported. The list control information for a list for which either list monitoring by threshold counts or keyrange monitoring is in effect is returned both in the answer area, mapped by IXLYLAA, and in the buffer area, mapped by IXLYLMI, List Monitoring Information.

Updating Control Information for a List

Use IXLLSTC REQUEST=WRITE_LCONTROLS to alter the list control information associated with a list. In addition to the control information that can be altered using IXLLIST REQUEST=WRITE_LCONTROLS (see "WRITE_LCONTROLS: Writing List Controls" on page 567), the following information can be altered using IXLLSTC REQUEST=WRITE_LCONTROLS:

- KEYRANGESTART and KEYRANGEEND

  Specify the keyrange start and end values. (KEYRANGE=SET)
- KREMPTY and KRNOTEMPTY

  Specify the keyrange empty and keyrange not-empty threshold counts. (KEYRANGESTATE=DEFINE)
- LISTEMPTY and LISTNOTEMPTY

  Specify the list empty and list not-empty threshold counts. (LISTSTATE=DEFINE)
- KEYRNOTIFYDELAY

  Specifies whether the keyrange notification delay value for the structure is applied when notifying monitoring instances of empty to non-empty state transitions of a list's keyrange. For more information about the notification delays associated with keyrange monitoring, see "Understanding Keyrange Monitoring Notification Delay" on page 621.
- LISTNOTIFYDELAY

  Specifies whether the list notification delay value for the structure is applied when notifying monitoring instances of empty to non-empty state transitions for a list. For more information about the notification delays associated with list monitoring, see "Understanding List Monitoring Notification Delay" on page 619.

The WRITE_LCONTROLS request requires that list authority checking be done. If the specified authority fails to equal that for the designated list, the IXLLSTC operation is terminated. The structure is not changed and return and reason codes are returned in the answer area. If the authority checking is successful, you can specify a new value to be established as the list authority of the designated list; otherwise, the list authority for the list remains unchanged.

Monitoring a List

Use IXLLSTC REQUEST=MONITOR_LIST to monitor a list to determine whether its state is empty, not-empty, full, or not-full. For more information about setting empty and not-empty thresholds and list limit values defining the list empty, not-empty, full, and not-full states for a list, see "Updating Control Information for a List" on page 618 and the IXLLSTC WRITE_LCONTROLS service.

For list structures allocated in a coupling facility of CFLEVEL=22 or higher, XES list services provide the following list monitoring options. The support for CFLEVEL=22 must be installed on the z/OS system to use these options.

- A list monitoring function that allows a list structure connector to determine whether a list in the structure is full or not-full. If the list is full, the list count limit for entries or elements has been reached. If the list is not-full, the list count limit for entries or elements has not been reached. Use IXLLSTC REQUEST=MONITOR_LIST to monitor a list to determine whether the state is full or not-full.

The parameter, MONITORTYPE=NOTFULL, is used to specify full or not-full monitoring. The parameter, DRIVEEXIT=YES, causes XES to drive the connector's list transition exit for list full to not-full list state transitions.

As with list monitoring for empty and not-empty, the IXLVECTR service can be used to determine whether the list is considered full or not-full based on the list limits established with an IXLLSTC WRITE_LCONTROLS request.

An indicator in the IXLLSTC answer area is returned when the MONITOR_LIST request is successful, which indicates the state of the list when monitoring is established. For an IXLLSTC REQUEST=MONITOR_LIST, MONITORTYPE=NOTFULL, the value of LAAMNL_LISTNOTFULL indicates if the list is in a full or not-full state.

The macro IKCYQUAA defines the `QuRegRfNotFullMonitoring` bit in the `QuRegFeatures` string that is used to determine whether list not-full monitoring support and the IXLLSTC MONITORTYPE keyword are supported on the system. Use IXCQUERY REQINFO=FEATURES to retrieve the `QuRegFeatures` string.

- Aggressive list notification

  When establishing monitoring for a list, the requester can specify the frequency under which the coupling facility updates the list notification vector for empty and not-empty list state transitions.

  The list notification vector is updated when the list state transitions from the not-empty state to the empty state or from the empty state to the not-empty state. For list structures allocated in a coupling facility of CFLEVEL=22 or higher, more frequent updates to the list notification vector can be requested by specifying NOTIFICATION=EVERY on the IXLLSTC REQUEST=MONITOR_LIST request. NOTIFICATION=EVERY causes the list notification vector to be updated when the list state transitions from the not-empty state to the empty state and every time a list entry is added to the list and the list not-empty threshold count is exceeded.

  When NOTIFICATION=EVERY and DRIVEEXIT=YES is specified on a REQUEST=MONITOR_LIST ACTION=START request, XES processing is initiated to drive the connection's list transition exit when a list entry is added to the monitored list and the list not-empty threshold count for the list is exceeded. The combination of NOTIFICATION=EVERY and DRIVEEXIT=YES gives the connection's list transition exit the opportunity to receive control more frequently. This more frequent notification is referred to as an aggressive list notification.

  The macro IXCYQUAA defines the `QuRegRfAggresiveNotify` bit in the `QuRegFeatures` string that can be used to determine whether aggressive list notification (IXLLSTC NOTIFICATION=EVERY keyword for REQUEST=MONITOR_LIST) is supported on the system. Use IXCQUERY REQINFO=FEATURES to retrieve the `QuRegFeatures` string.

  Note: Aggressive list notification is not available for full/not-full monitoring.

Understanding List Monitoring Notification Delay

For list structures allocated in a coupling facility of CFLEVEL=22 or higher, exploiters of list monitoring can transparently reduce scheduling overhead (for example, false scheduling of list transition exits) with the list monitoring notification delay function.

By specifying a non-zero value for the LISTNOTIFYDELAY sub-parameter of the CFRM STRUCTURE definition statement in the Administrative Data Utility (IXCMIAPU), users can specify in their policy definitions the amount of time between notification of a system selected list monitoring exploiter instance and the notification of the other exploiter instances. If LISTNOTIFYDELAY is not specified, a default of zero is used and the list monitoring notification delay function is not enabled for the allocated list structure.

When the notification delay function is requested, a single system selected exploiter instance receives notifications of empty to not-empty list transitions before any other exploiter instances are notified. The other instances are not notified if the initial exploiter instance processes the list in a timely manner. The exploiter instance that receives the initial notification, is selected in a round-robin fashion.

To benefit from this enhancement, the sysplex environment needs to meet the following conditions:

- The monitored list structure is allocated in a coupling facility of CFLEVEL=22 or higher.
- The support for CFLEVEL=22 is installed on a z/OS system in the sysplex.

For full benefit, all z/OS systems in the sysplex need to support CFLEVEL=22.

For more information about list monitoring notification delay, see CFRM parameters for administrative data utility in *z/OS MVS Setting Up a Sysplex*.

Monitoring a Sublist

Use IXLLSTC REQUEST=MONITOR_SUBLIST to start or stop monitoring a sublist that is designated by a list number and either an entry key or a secondary key. Secondary keys can be specified only when the structure is allocated in a coupling facility of CFLEVEL=9 or higher and has been allocated to support secondary keys (IXLCONN KEYTYPE=SECONDARY).

In addition to the functions provided by IXLLIST REQUEST=MONITOR_SUBLIST (see "MONITOR_SUBLIST: Monitoring a Single Sublist" on page 577), IXLLSTC REQUEST=MONITOR_SUBLIST allows you to specify the condition for which an EMC is to be queued to the event queue. The NOTIFICATION keyword indicates either that an EMC is to be queued to the event queue when the monitored sublist transitions from the empty to the not-empty state or that an EMC is to be queued to the event queue whenever a list entry is queued to the monitored sublist.

Monitoring a Set of Sublists

Use IXLLSTC REQUEST=MONITOR_SUBLISTS to monitor a set of sublists, designed by list number and either entry key or secondary key. Secondary keys can be specified only when the structure is allocated in a coupling facility of CFLEVEL=9 or higher and has been allocated to support secondary keys (IXLCONN KEYTYPE=SECONDARY).

See "MONITOR_SUBLISTS: Monitoring Multiple Sublists" on page 578 for information about monitoring multiple sublists, including how to restart a request that ends prematurely. Additional information that is returned, mapped by IXLYMSRI, for structures allocated in a coupling facility of CFLEVEL=9 or higher, is:

- An indication of whether the sublist monitoring is for sublists identified by entry key or for sublists identified by secondary key.
- An indication as to whether an EMC should be queued to the event queue for every list entry added to the sublist or for only the first list entry added to the sublist.
- Secondary list entry key of the sublist for which sublist monitoring is requested, if applicable.

Monitoring a List by Keyrange Values

Use IXLLSTC REQUEST=MONITOR_KEYRANGE to start or stop keyrange monitoring of a particular list.

Keyrange monitoring allows you to determine when a range of keys within a list is empty or not-empty, depending on threshold counts that you have specified. The range of keys and the threshold counts that define the empty and not-empty state are specified with the IXLLSTC WRITE_LCONTROLS service. See "Using Threshold Counts when Monitoring a Range of Key Values" on page 617.

For list structures allocated with keyed list entries in a coupling facility of CFLEVEL=22 or higher, XES list services provide the following keyrange monitoring option:

- Aggressive keyrange notification

  When establishing keyrange monitoring for a range of keys within a list, the requester can specify the conditions and frequency under which the coupling facility updates the list notification vector for empty and not-empty state transitions of the keyrange.

  More frequent updates to the list notification vector can be requested by specifying NOTIFICATION=EVERY on the IXLLSTC REQUEST=MONITOR_KEYRANGE request. NOTIFICATION=EVERY causes the list notification vector to be updated when the keyrange state transitions from the not-empty state to empty state and every time a list entry is added to the keyrange on the list and the keyrange not-empty threshold count is exceeded.

When NOTIFICATION=EVERY and DRIVEEXIT=YES is specified on a REQUEST=MONITOR_KEYRANGE ACTION=START request, XES processing is initiated to drive the connection's list transition exit when a list entry is added to the monitored keyrange and the keyrange not-empty threshold count is exceeded. The combination of NOTIFICATION=EVERY and DRIVEEXIT=YES gives the connection's list transition exit the opportunity to receive control more frequently. This more frequent notification is referred to as, aggressive keyrange notification.

Macro IXCYQUAA defines the `QuRegRfAggressiveNotify` bit in the `QuRegFeatures` string that is used to determine if keyrange notification (IXLLSTC NOTIFICATION=EVERY keyword for REQUEST=MONITOR_KEYRANGE) is supported on the system. Use IXCQUERY REQINFO=FEATURES to get the `QuRegFeatures` string.

Using the List Notification Vector

When you connect to the list structure and indicate your interest in using list keyrange monitoring, the system allocates a list notification vector. When a list transition occurs for the monitored list, the system updates the associated entry in the list notification vector to reflect the empty or not-empty state of the keyrange according to the established threshold values. The IXLVECTR service provides the interface to the list notification vector. See "The List Notification Vector" on page 507 for additional information.

The IXLVECTR service can also be used to alter the size of the list notification vector, and thus change the number of lists that can be monitored. Only one keyrange can be monitored per list.

Under most circumstances, if a list notification vector index is in use for monitoring, a request to stop monitoring should be issued before using the same vector index to start another monitor. With keyrange monitoring, it is not necessary to stop monitoring with an old index before starting monitoring with a new index. With keyrange monitoring, a request to start monitoring causes the vector index in use to be replaced by the new vector index.

Starting and Stopping Keyrange Monitoring

To begin keyrange monitoring for the connection specified by CONTOKEN for the list specified by LISTNUM, specify ACTION=START. Only one keyrange can be monitored per list. You must also identify the list notification vector index that is to reflect the empty or not-empty state of the monitored keyrange. If your list transition exit is to receive control when the state of the keyrange changes from empty to not-empty, specify DRIVEEXIT=YES.

To stop keyrange monitoring for the connection specified by CONTOKEN for the list specified by LISTNUM, specify ACTION=STOP.

Handling an Incompletely Processed MONITOR_KEYRANGE Request

A MONITOR_KEYRANGE request might complete prematurely if the coupling facility model dependent timeout criteria is exceeded. The caller is expected to reissue the request until it completes without timing out.

If this happens, the appropriate return and reason codes are retuned in the answer area. Continue reissuing the MONITOR_KEYRANGE request until the return code indicates that all processing has completed.

Understanding Keyrange Monitoring Notification Delay

For keyed list structures allocated in a coupling facility of CFLEVEL=22 or higher, exploiters of keyrange monitoring can transparently reduce scheduling overhead (for example, false scheduling of list transition exists) with the keyrange monitoring notification delay function.

By specifying a non-zero value for the KEYRNOTIFYDELAY sub-parameter of the CFRM STRUCTURE parameter in the Administrative Data Utility (IXCMIAPU), users can specify in their policy definitions the amount of time between notification of a system selected keyrange monitoring exploiter instance and the notification of the other keyrange monitoring exploiter instances. If KEYRNOTIFYDELAY is not specified,

a default of zero is used and the keyrange monitoring notification delay function is not enabled for the allocated list structure.

When the keyrange monitoring delay function is requested, a single system selected exploiter instance receives notifications of empty to not-empty keyrange transitions before any other exploiter instances are notified. The other instances are not notified if the initial exploiter instance processes the keyrange in a timely manner. The exploiter instance that receives the initial notification, is selected in a round-robin fashion.

To benefit from this enhancement, the sysplex environment needs to meet the following conditions:

- The monitored list structure is allocated as a keyed list structure in a coupling facility of CFLEVEL=22 or higher.
- The support for CFLEVEL=22 is installed on a z/OS system in the sysplex.

For full benefit, all z/OS systems in the sysplex need to support CFLEVEL=22.

For more information about list monitoring notification delay, see CFRM parameters for administrative data utility in *z/OS MVS Setting Up a Sysplex*.

Monitoring an Event Queue

Use IXLLSTC REQUEST=MONITOR_EVENTQ to start or stop monitoring an event queue to determine whether the queue is in an empty or not-empty state. Specify whether the event queue is to be monitored for state transitions of sublists identified by list entry key or for state transitions of sublists identified by secondary key.

If the structure is allocated in a coupling facility of CFLEVEL=9 or higher and has been allocated with secondary keys (IXLCONN KEYTYPE=SECONDARY), there are two event queues that can be monitored. The IXLLSTC KEYTYPE parameter (ENTRY or SECONDARY) designates which event queue is to be monitored. KEYTYPE=ENTRY specifes that the event queue for state transitions of sublists identified by list entry key is to be monitored. KEYTYPE=SECONDARY specifies that the event queue for state transitions of sublists identified by secondary key is to be monitored.

See “MONITOR_EVENTQ: Monitoring an Event Queue” on page 574 for additional information about requesting event queue monitoring.

Reading EMC Control Information

Use IXLLSTC REQUEST=READ_EMCONTROLS to obtain control information for the user's registered interest in monitoring a particular sublist. The sublist is designated by list number and either entry key or secondary key. Secondary keys can be specified only when the structure is allocated in a coupling facility of CFLEVEL=9 or higher and has been allocated to support secondary keys (IXLCONN KEYTYPE=SECONDARY).

See “READ_EMCONTROLS: Reading Event Monitor Controls” on page 580. For list structures allocated in a coupling facility of CFLEVEL=9 or higher, additional information that is returned in the answer area mapped by IXLYLAA includes:

- An indication of whether an EMC will be queued to the associated event queue whenever a list entry is added to the sublist or whenever the first list entry is added to the sublist.
- An indication of whether an EMC is associated with a sublist for a secondary key or a sublist for a list entry key.
- The secondary key of the sublist with which an EMC is associated, if applicable.

Reading Event Queue Control Information

Use IXLLSTC REQUEST=READ_EQCONTROLS to obtain control information associated with the connector's event queue. The event queue contains event monitor controls (EMCs) associated with a monitored sublist. The sublist is designated by list number and either entry key or secondary key. Secondary keys can be specified only when the structure is allocated in a coupling facility of CFLEVEL=9

or higher and has been allocated to support secondary keys (IXLCONN KEYTYPE=SECONDARY). The IXLLSTC KEYTYPE parameter (ENTRY or SECONDARY) designates which type of event queue is to be processed. KEYTYPE=ENTRY specifes that information about the event queue for state transitions of sublists identified by list entry key is to be returned. KEYTYPE=SECONDARY specifies that information about the event queue for state transitions of sublists identified by secondary key is to be returned.

See "READ_EQCONTROLS: Reading Event Queue Controls" on page 581. For list structures allocated in a coupling facility of CFLEVEL=9 or higher, additional information that is returned in the answer area mapped by IXLYLAA includes:

- An indication of whether the queue of EMCs is associated with sublists for secondary keys or sublists for entry keys.

Dequeuing Event Monitor Controls from an Event Queue

Use IXLLSTC REQUEST=DEQ_EVENTQ to dequeue event monitor controls (EMCs) from an event queue. Specify whether the EMCs to be dequeued are for state transitions of sublists identified by list entry key or for state transitions of sublists identified by secondary key. Secondary keys can be specified only when the structure is allocated in a coupling facility of CFLEVEL=9 or higher and has been allocated to support secondary keys (IXLCONN KEYTYPE=SECONDARY).

If the structure is allocated in a coupling facility of CFLEVEL=9 or higher and has been allocated with secondary keys (IXLCONN KEYTYPE=SECONDARY), there are two event queues that can be processed. The IXLLSTC KEYTYPE parameter (ENTRY or SECONDARY) designates which event queue is to be processed. KEYTYPE=ENTRY specifies that EMCs for state transitions of sublists identified by list entry key are to be dequeued. KEYTYPE=SECONDARY specifies that EMCs for state transitions of sublists identified by secondary key are to be dequeued.

See "DEQ_EVENTQ: Retrieving Events from the Event Queue" on page 582 for additional information, including how to handle an incompletely processed request to dequeue EMCs. For list structures allocated in a coupling facility of CFLEVEL=9 or higher, additional information that is returned in the BUFFER or BUFLIST area mapped by IXLYEMC includes:

- An indication of whether an EMC will be queued to the associated event queue whenever a list entry is added to the sublist or when only the first list entry is added to the sublist.
- An indication of whether the monitored sublist is for a secondary key or for a list entry key.
- The secondary key of the sublist with which the EMC is associated, if applicable.

Performing Lock Operations

Use IXLLSTC REQUEST=LOCK to operate on a lock table entry associated with the list structure. Operations that can be specified are SET, RESET, TEST, and READNEXT. The function provided by ISLLSTC REQUEST=LOCK is comparable to that provided by IXLLIST REQUEST=LOCK. See "LOCK: Performing a Lock Operation" on page 568.

Reading Structure Counts for a List Structure

Use IXLLSTC REQUEST=READ_STRCOUNTS to obtain the list structure count information on the maximum and in-use counts for a specific list structure.

The information for the structure that is returned in the answer area mapped by IXLYLAA includes:

- The current in-use number of elements and maximum number of elements defined for the structure if data elements are defined for the structure.
- The current in-use number of entries and the maximum number of entries defined for the structure.
- The current in-use number of event monitor controls and the total number of event monitor controls defined for the structure if event monitor controls are supported by the structure.
- The number of lock entries defined for the structure if the structure is defined as a serialized list structure.

Chapter 10. Using Lock Services (IXLLOCK)

The XES lock services allow sysplex-wide serialization in a multi-system data sharing environment. The services provided through the IXLLOCK macro enable authorized applications to obtain shared or exclusive serialization on user-defined logical resources. Additionally, you can implement your own locking protocols through the inclusion of user data. The XES lock services offer the additional benefits of allowing you to assist in the management of contention in the data sharing environment and of providing failure recovery options by retaining data about serialized resources that will persist across system outages.

The IXLLOCK macro provides services that allow you to request:

- Shared or exclusive ownership of a resource (OBTAIN)
- A change to the attributes of a resource that you currently own or are attempting to own (ALTER)
- Release of the shared or exclusive ownership of a resource or cancel a previously submitted request that is pending (RELEASE).
- Processing of multiple resource requests with a single macro invocation (PROCESSMULT). The types of resource requests that are supported are a function of the version of the IXLLOCK macro.

When you request an XES lock service, you must be connected to a lock structure in a coupling facility. The lock structure is the repository for the lock table that is used to monitor the serialization of resources in the sysplex and for the data being recorded for recovery purposes.

Intrinsic to the XES lock services are the user exit routines that provide the negotiation and contention management protocols for the data sharing application. The contention exit and the notify exit collaborate to resolve contention for shared resources. Other exits used by the XES lock services are the complete exit, to report the completion of a previously submitted request for a resource and the event exit, to report the occurrence of an event in the sysplex, such as another user failing, which might affect your processing.

Specifying the IXLLOCK ADUPREQSEQNUM or REQVERSION keywords will result in the generation of at least a version 4 parameter list. A version 4 parameter list requires system-managed asynchronous duplexing support. If a version 4 parameter list is used on a system that does not support it, the request is rejected for the unsupported parameter list version (IxlRsnCodeBadVersion#). Macro IXCYQUAA defines the QuReqRfAsyncDuplex bit in the QuReqFeatures string that can be used to test for system-managed asynchronous duplexing support. Use IXCQUERY REQINFO=FEATURES to get the QuReqFeatures string. It can be assumed that systems at a z/OS level that later than V2R2 will support system-managed asynchronous duplexing.

Resource Concepts

This section discusses the entity for which you want to provide serialization (a resource) and how XES and the IXLLOCK services keep track of users' requests for resource serialization.

What Is a Resource?

A resource can be any logical entity depending on your application. For data base products, a resource could be anything from a record to a block of records, to an entire data set. You define the resources for which serialization is required. You assign a name to each resource so that you or any other user can identify the resource for processing.

A request to access a resource for either shared or exclusive ownership is called a resource request. Each resource request indicates who the requestor is, in what state (either shared or exclusive) the resource is requested, and user-defined data that the requestor can specify for use in contention management. The resource request also might specify another type of user-defined data that the system is to record for recovery purposes.

© Copyright IBM Corp. 1994, 2023

XES keeps track of requests for a specific resource in a **resource request queue**.

State of a Resource Request Queue

The **composite state** of a resource request queue is determined by evaluating all resource requests on the queue. A resource request queue can be in one of three composite states — free, shared, or exclusive.

- Free — There are currently no owners or waiters (requests to own) the specified resource.

  (You are a resource owner if you have been granted access to the resource. You are a waiter if your request has not yet been granted.)
- Shared — All owners and waiters for the resource are in the shared state.
- Exclusive — There is at least one owner or waiter for the specified resource in the exclusive state.

Since each individual resource request indicates the state in which the resource is requested, XES is able to maintain the composite state of the entire resource request queue.

When a new request for a resource is received, XES determines the compatibility of the request before adding it to the resource request queue. Figure 67 on page 626 illustrates the compatibility rules used by XES and the resultant state of the resource request queue. A “C” indicates a compatible state and an “X” indicates an incompatible state.

Composite State →	FREE	SHARED	EXCLUSIVE
NEW Request's Requested State ↓			
SHARED	C	C	X
EXCLUSIVE	C	X	X

Figure 67. XES Compatibility Rules

Figure 68 on page 627 depicts two resource request queues — for resource XYZ and resource JKL.

- Resource Request Queue for Resource XYZ

  The composite state (C/S) of the resource request queue for XYZ is shared and all entries on the queue are compatible; both User A and User B have requested the resource in a shared state. The entry for both User A and User B show that the resource is held and that they have each specified user data associated with the request.
- Resource Request Queue for Resource JKL

  The composite state of the resource request queue for JKL is exclusive; User C has been granted exclusive use of the resource, while User A has requested shared use. The request queue is also said to be incompatible because it contains entries requesting access to the resource in conflicting states. The entry for User C shows that the resource is held in an exclusive state and that user data is associated with the resource request. The entry for User A shows that User A's request is pending and that user data is associated with the request.

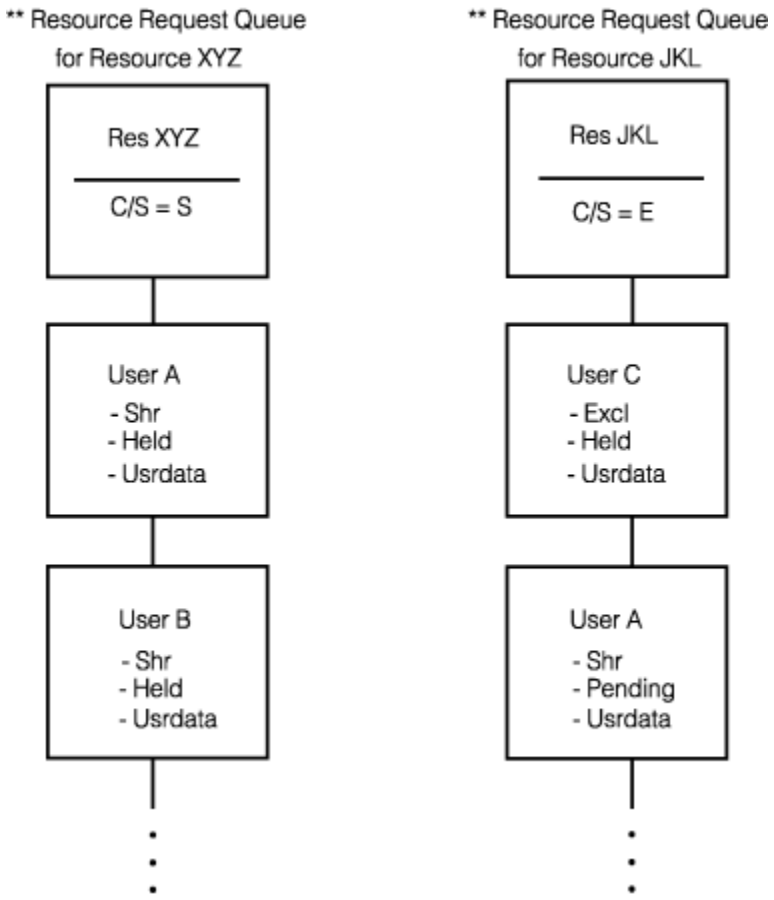


Figure 68. Resource Request Queue Compatibility

What can you do with the XES lock services?

The serialization requirements of your application determine how and when you will need to use the XES lock services. Resources that the application needs can be associated with lock entries in the lock structure. To obtain serialization on the resource, you first must be connected to the lock structure, which then allows you access to the XES lock services. The following topics briefly outline the XES lock services available to the connected lock structure user.

- **Obtaining a Lock**

  A user wanting to gain serialization on a resource uses the IXLLOCK service, specifying an OBTAIN request, to request shared or exclusive ownership of a resource. With this request, you can also specify additional user-defined data. This optional data can be used by the application to implement user-defined locking protocols and lock states, as well as for use in providing recovery capabilities.

- **Altering the Lock**

  Once serialization on a resource is held, or the request to obtain serialization is pending, you can request that the attributes of the request be changed. With the ALTER request, you can request changes to the state, user data, or recovery data that was initially specified with the OBTAIN request. The ALTER request also allows you to request that new recovery data be added to the structure for this resource, if none had been specified before.

- **Releasing the Lock**

  When serialization is no longer required, you can use the RELEASE request to relinquish the shared or exclusive use of the resource. The data that was associated with the request is released, unless the data was recovery data. For recovery data, you can specify whether to keep or delete the data. Keeping the recovery data allows resource ownership information to remain in the coupling facility structure, even though the resource is no longer owned.

The RELEASE request also allows you to cancel a prior request (either OBTAIN or ALTER) that has not yet been granted.

- **Processing Multiple Requests**

  A user wanting to process multiple IXLLOCK requests can use the PROCESSMULT option. Version 1 of the IXLLOCK macro supports the RELEASE request as a valid PROCESSMULT type. With IXLLOCK Version 1 and a coupling facility of CFLEVEL=2 or higher, you can perform a PROCESSMULT request that is the equivalent of up to 128 RELEASE requests. Using the PROCESSMULT option should reduce the number of coupling facility accesses as compared with issuing multiple separate RELEASE requests.

  For each resource to be released, you can specify that the associated record data is either to be kept or deleted. The record data cannot be modified.

- **Recording Recovery Data**

  XES allows you to plan for recovery if a connected user fails. At CONNECT time, you can specify that you want to maintain information about resources that you might own in the form of record data. The record data can persist across system failures. If you fail before you are able to release your ownership of any of these resources, peer connected users can access the information that you have maintained and use it to recover the resource(s). You can record 64 bytes of this information, called record data, when you issue an OBTAIN or ALTER request. The record data is for your use and the use of peer connections only and is not used by XES. Record data could include such information as:

  – The resource to which this entry applies
  – Something to identify the unit of work holding the lock to which this entry applies.

  XES maintains the record data as part of the lock structure and provides a unique identifier for you to retrieve the data when necessary. In the event of a connected user's failure, other connected users can use the IXLRT macro to retrieve the record data associated with resources that were held by the failed user.

- **Failure and Recovery Considerations**

  If the user fails while holding serialization on a resource, the application should have recovery actions in place to recover the data resource. XES provides the mechanisms by which an application can implement protocols to recover resources and maintain the integrity of shared data in the event of connector failures. When a connector (or the system on which the connector is running) fails, XES informs the surviving connectors of the failure through their event exits. XES then waits for all surviving connectors to provide a confirmation before proceeding to clean up for the failed user. The surviving connectors may choose to perform application-specific cleanup prior to providing this confirmation.

  Note that when an asynchronously duplexed structure fails over to the secondary structure instance, locks held by any connector that is disconnecting or terminating might become immediately available to active connectors when failover completes. Without asynchronous duplexing, such locks would not be available until all active connectors provided a confirmation for the disconnecting/terminating connector. This would only occur for locks that were not committed to the secondary structure instance and thus is not behavior that can be relied upon.

Managing Contention

Contention occurs when a resource request that is not compatible with the existing entries on the resource request queue is added to the queue.

Contention is handled by both XES and the exploiting user. XES recognizes the contention; the user resolves the contention through its contention and notify exits.

When XES recognizes contention, it selects one of the connected users to manage the resource and assigns management responsibilities to that user. The user selected is not necessarily a requestor of the resource. (Note that the application should make no assumptions regarding where contention management will occur.) XES passes the resource request and the associated resource request queue to the contention exit of the selected user. The contention exit's purpose is to resolve the contention based on the user's defined protocols. Subsequent requests for the “in contention” resource are presented to

the selected user's contention exit in time-of-arrival sequence. XES ensures that at most one new request at a time is presented to the contention exit.

Once a user is selected to manage the resource, that user remains the manager of the resource until the contention is resolved. If the selected user should disconnect or abnormally terminate while still the manager of the resource in contention, then XES assigns management responsibilities to another connected user.

Defining a Protocol to Handle Contention

You can define user protocols for your application by specifying user data on a resource request. User data is 64 bytes of data that can be specified on any IXLLOCK request. Within the contention and notify exits, contents of the 64-byte user data field for each request on the resource request queue can be examined or modified — whatever the application requires to maintain its own controls about the serialization of the resource. An example of your use of user data is if your locking protocol supports lock states other than shared and exclusive. You can put your lock state information in the user data.

How is Contention Resolved?

When XES recognizes that contention exists, the responsibility for managing — and ultimately resolving — the contention is assigned to the contention exit of a connected user. The contention exit has as input the contention exit parameter list, the CEPL, mapped by the macro IXLYCEPL. The CEPL consists of a header area (mapped by CEPL) followed by a series of entries (mapped by CEPLENT), each of which represents a connector with an interest in the resource (in other words, an entry on the current resource request queue). The header information includes general status data about the resource and about this instance of contention. Each CEPLENT entry represents the current ownership state and any pending request to update that state, which has already been presented to the contention exit. Additionally, each CEPLENT entry contains a set of flags that allow the contention exit to inform XES as to what action, if any, should be performed for this owner and/or pending request represented by this entry.

What Can You Do in a Contention Exit?

The contention exit may inform XES about what actions, if any, are to be performed for the owners and pending IXLLOCK requests represented on the resource request queue. Through modification of the appropriate CEPLENT entry, the contention exit may choose to:

- Grant a pending request, perhaps with changed ownership attributes.

  The contention exit allows the resource request, while possibly changing the ownership attributes requested.
- Deny a pending request.

  The contention exit does not allow the ownership state requested.
- Regrant an owned resource with changed ownership attributes.

  The contention exit changes the ownership data of a resource that is currently owned.
- Keep a pending request in a pending state.

  The contention exit neither grants nor denies the resource request. The request remains pending on the resource request queue until it is granted, denied, or superseded. (When a pending request from a user on the resource request queue is replaced by a more current request (that is, an ALTER or RELEASE request) from that user, the previous request is said to be superseded.)
- Notify a current resource owner that contention exists

  The contention exit may choose to inform one or more users that contention exists for a resource it owns by executing the notify exit of those users. The notify exit receives as input a notify exit parameter list (NEPL) representing the current resource request queue. Based on its evaluation of the resource request queue, the notify exit may choose to take actions to alleviate the contention. The IXLSYNCH service provides the mechanism by which the notify exit of a connected user may synchronously update or release its interest in a resource. After the specified notify exits have been executed, the

resultant resource request queue (containing any changes made by the notify exits) is presented to the contention exit. Through the use of the notify exit, an application can implement protocols that allow owners and requestors of a resource to negotiate for ownership.

Sample Locking Protocol — Definition

The following illustrates a protocol in which an application uses IXLLOCK with user data to achieve its multi-system data sharing.

Application "A" is a multi-system application whose data is maintained in data sets residing on shared DASD. The application is required to access the data on behalf of requests from end users of the application as well as on behalf of utility functions that are periodically scheduled to perform maintenance activities. User requests are for a single record in the data set; utility requests are for a block of records on which maintenance is to be performed. The application is required to maintain the integrity of the shared data while providing efficient access to both types of processes. To accomplish this, the application has designed a locking protocol based on the XES lock services. A detailed description of the protocol follows.

Purpose

To provide a protocol that allows user requests for a resource to take precedence over utility requests for the same resource.

Design

All resource requests are to indicate whether they are user-initiated or utility-initiated and are to specify the exact records of the data set to which they require access. User requests will be served on a first-in first-out (FIFO) basis and will take precedence over utility requests. Under certain circumstances, utility functions that are current owners of a resource might need to negotiate the resource ownership. The negotiation is to be accomplished in the notify exit and could result in the utility function maintaining a subset of its resource ownership in order to allow the user request to be granted.

Requirements

The application must conform to the following:

- Any request to access the data must result in the application issuing an IXLLOCK request specifying a resource name equal to the name of the data set containing the specified data.
- A request to read the data must result in an IXLLOCK request for shared access; a request to update the data must result in an IXLLOCK request for exclusive access.
- The user data specified with the IXLLOCK request must contain:
  - Values indicating the first and last sequential records to be accessed.
  - A process identifier field to indicate on whose behalf the serialization is being obtained. For example, if the request is to service a user, then the field will contain a value that indicates "user-initiated".

The following table shows the required information that the application must specify to accomplish various requests for data access.

Table 46. Required Information for Application A. Information to be specified on an IXLLOCK request as a result of various tasks.

Action	Required Access
User request to read record 2 of data set ABC.	Shared access of resource, data set ABC, with user data indicating a first record of 2, last record of 2, and process identifier indicating "User".
User request to update record 1 of data set DEF.	Exclusive access of resource, data set DEF, with user data indicating first record of 1, last record of 1, and process identifier indicating "User".

Table 46. Required Information for Application A. Information to be specified on an IXLLOCK request as a result of various tasks. *(continued)*	
Action	**Required Access**
Utility routine to perform maintenance activities on the first 100 records of data set GHI.	Exclusive access of resource, data set GHI, with user data indicating a first record of 1, last record of 100, and process identifier indicating "Utility".

Rules of the Sample Protocol

The protocol defines the following rules for the management of contention.

1. If a request is made that requires access to a record within a block that is already serialized, then it is treated as a conflict.
2. Conflicting user requests are processed in FIFO (first-in first-out) order.
3. User access is to be given priority over access by utility functions.
4. Negotiation is required when utility functions that hold serialization conflict with a new user request for the resource.

- **Rule 1 — Conflict**

  XES recognizes contention on a resource level, which is the data set level in this protocol. This implies that requests to access the same data set in incompatible states will result in XES assigning management responsibilities to the contention exit of one of the instances of the application. Contention exit processing is to examine the first and last record indicators in the user data to determine if the requests are to access different portions of the data set and thus are compatible. For example,

  – Instance 1 of Application A receives a user request to read record 2 of data set XYZ. Instance 1 issues an IXLLOCK request for shared access of data set XYZ, with user data indicating that the request is to access record 2.
  – XES grants the IXLLOCK request.
  – Instance 2 of Application A receives a user request to update record 8 of data set XYZ. Instance 2 issues an IXLLOCK request for exclusive access of data set XYZ, with user data indicating that the request is to access record 8.
  – XES recognizes that there is an incompatible request for the resource, data set XYZ, and chooses a contention exit to manage the contention.
  – The contention exit chosen to manage the resource contention examines the user data and determines that the requests are to access different records in the data set. The contention exit instructs XES to grant Instance 2's request.

- **Rule 2 — FIFO Order**

  If two or more user-initiated requests for the same resource cause contention, contention exit processing is to handle the requests in the order in which they are received. For example,

  – Instance 1 of Application A receives a user request to read record 2 of data set XYZ. Instance 1 issues an IXLLOCK request for shared access of data set XYZ, with user data indicating that the request is to access record 2.
  – XES grants the IXLLOCK request.
  – Instance 2 of Application A receives a user request to update record 2 of data set XYZ. Instance 2 issues an IXLLOCK request for exclusive access of data set XYZ, with user data indicating the request is to access record 2.
  – XES recognizes that there is an incompatible request for the resource, data set XYZ, and chooses a contention exit to manage the contention.

- The contention exit chosen to manage the resource contention examines the user data and determines that the requests are to access the same record in the data set and notes that both requests are user-initiated. The contention exit instructs XES to leave Instance 2's request pending.
- When Instance 1 (who owns the resource in a shared state) completes its processing, it issues an IXLLOCK request to release its interest in data set XYZ.
- The release request is added to the resource request queue and presented to the contention exit who continues to manage this occurrence of contention.
- The contention exit examines the request queue and sees that Instance 1 is releasing its interest in the resource and that the request by Instance 2 is still pending on the request queue. The contention exit instructs XES to grant Instance 2's request.

  Note that when XES receives control back from the contention exit and processes these requests, the resource will no longer be in contention as the request queue will contain one exclusive owner (Instance 2). The contention exit will, therefore, have completed its duties as contention manager. Should another instance of contention for this resource occur, this contention exit is not necessarily the one that XES will choose to manage the contention.

- **Rule 3 — User Precedence**

  When requests from a utility function conflict with user-initiated requests that currently hold serialization on the resource, the utility request must wait until the user releases its serialization.

- **Rule 4 — Negotiation**

  When requests from a user conflict with a utility function that currently holds serialization on the resource, the conflict is resolved by negotiation, as follows.

  - XES informs the current resource owner (the utility function) that there is contention for the resource through its notify exit.
  - The notify exit of the utility function can examine the resource request queue, as presented in the notify exit parameter list (NEPL), to determine if it is able to change its current ownership characteristics and thus allow access to be granted to the user.

Sample Locking Protocol — Implementation

The following illustrates Application A's implementation of the protocol using the user data.

- A utility program is scheduled to perform maintenance operations on the first 100 records of data set XYZ and submits a request to do so. Instance 1 of Application A receives the request and issues an IXLLOCK request for exclusive access of data set XYZ, with user data indicating that the request is to access records 1 through 100.
- XES grants the IXLLOCK request. The utility program begins its maintenance procedures starting with record 1 of data set XYZ.
- Instance 2 of Application A receives a user request to update record 6 of data set XYZ. Instance 2 issues an IXLLOCK request for exclusive access of data set XYZ, with user data indicating that the request is to access record 6.
- XES recognizes that there is an incompatible request for the resource, data set XYZ, and chooses a contention exit to manage the contention.
- The contention exit chosen to manage the resource contention examines the resource request queue, as presented in the contention exit parameter list (CEPL), and determines that the serialization held by Instance 1 on behalf of a utility program is preventing access by Instance 2 on behalf of a user. The contention exit instructs XES to schedule the notify exit of Instance 1 by setting the appropriate indicators in the CEPL entry that represents Instance 1's ownership of the resource.
- When control returns from the contention exit, XES examines the CEPL and determines that the contention exit has requested that the notify exit of Instance 1 be run. XES schedules the notify exit of Instance 1, passing the resource request queue in the form of a NEPL.
- The notify exit of Instance 1 receives control and examines the resource request queue in the NEPL. Meanwhile, the utility program continues its processing and has completed processing the first 20

records of data set XYZ. The notify exit determines that the utility program no longer needs access to those 20 records, and updates the user data in the NEPL to indicate that it needs serialization only to records 21 through 100. The change in ownership reflected in the updated NEPL is committed by invoking the IXLSYNCH service. Additionally, the notify exit might need to update the application's control structures to reflect the change in ownership status that was committed with the IXLSYNCH service. The notify exit then returns control to XES.

- XES presents the updated resource request queue (in the CEPL) to the contention exit.

  Note that XES will not add any new requests for the resource to the resource request queue until it has had a chance to examine any changes made as a result of the invocation of the notify exit.
- The contention exit examines the resource request queue and determines that the user request for record 6 is NOT in conflict with the utility program, whose user data now indicates that it is serializing records 21 through 100. The contention exit instructs XES to grant Instance 2's request.

Informing a User of Request Completion

A user requesting access to a resource must allow for the possibility that factors might exist that could prevent the request from being satisfied immediately. The reasons why request processing might experience delays range from user-controlled conditions, such as resource contention, to conditions that are not controllable by the connected user, such as internal XES serialization that could not be immediately obtained.

XES processes IXLLOCK requests for a resource either synchronously or asynchronously.

- A request is defined as synchronous when processing for the request is complete when control returns to the next sequential instruction following the request.
- Asynchronous means that processing for the request is not complete when control returns to the next sequential instruction following the request. XES provides a return and reason code (IXLRSNCODEASYNCH) to indicate that processing is not complete and that additional communication will be required. When processing for the request is complete, XES schedules the user's complete exit.

Using the IXLLOCK MODE Parameter

There might be times when the system is not able to process your IXLLOCK immediately. Some reasons for this delay might be:

- The requested resource is being globally managed at the time of the request.
- The system could not obtain its internal latches needed to process the request.
- The system detected contention for the requested resource when the coupling facility was accessed.

You can specify how you want the system to process your request if it cannot be serviced immediately by using the MODE parameter on IXLLOCK. **If the request can be processed immediately, then the MODE parameter is ignored.** The following are valid MODE specifications for an IXLLOCK request.

SYNCSUSPEND
Specifies that you do not want control returned until processing for the request is complete. If request processing is delayed because of the reasons listed previously, you will be suspended until the request completes. You will receive control at the next sequential instruction with the request complete and the final disposition determined.

SYNCEXIT
Specifies that you want control returned to you immediately if request processing is delayed. If there is to be a delay, you will receive return and reason codes to indicate that the request is being processed asynchronously. When processing for the request is complete, XES schedules your complete exit to return the results of your request. Note that the complete exit might be given control before control returns to the next sequential instruction after your IXLLOCK request.

NORESPONSE
Specifying this mode is valid only for a RELEASE request and indicates that you do not want notification of the request's completion. You will receive a return code indicating that the IXLLOCK

RELEASE request has been accepted; however, the complete exit will not be invoked to report request completion.

SYNCFAIL
Specifies that if the system cannot process your request without a delay, the request is to be cancelled. You will receive return and reason codes indicating that disposition of your request. This mode is valid only for OBTAIN and ALTER requests.

VALUE
Specifies that the contents of MODEVAL are to be used in determining how the request is to be processed if it cannot be serviced immediately. The constant values that are valid for MODEVAL are defined in IXLYCON. If you specify a value for MODEVAL other than one of the IXLYCON constants that is valid for a particular request type, the system fails the IXLLOCK request.

You should be aware that XES guarantees that you receive notification of requests completing in logical order. You receive notification that an OBTAIN request completed before notification that an ALTER or RELEASE request for the same resource completed. If, for some reason, the OBTAIN request failed, and you had already issued an ALTER and/or a RELEASE request for the same resource, XES invalidates the remaining ALTER and/or RELEASE requests and notifies you that the request was denied because you do not own the resource.

Asynchronous duplexing

When the structure has asynchronous duplexing established, XES reports the completion of the request when the update to the primary structure instance has been made. The update to the secondary structure instance occurs asynchronously. It is this asynchronous background processing to update the secondary structure instance that can allow processing to achieve performance characteristics that are more similar to simplex than synchronous duplexing. For more information, see “Working with structures in the Async Duplex Established phase” on page 285.

Lock Structure Concepts

This section discusses basic concepts relating to the lock structure and the functions it provides.

A lock structure can consist of two parts. The first part is a lock table, a series of lock entries that the system associates with resources. The lock table is always present in a lock structure. The second part is a set of record data entries. A record data entry contains information about a connected user's ownership interest in a particular “resource” and can be used for recovery if the connected user fails. Record data entries are present in the lock structure only if you specify at connect time that you want to record this type of recovery information.

Figure 69 on page 635 shows a lock structure with multiple locks, each represented by an entry in the lock table. It also shows the optional record data entries that an application can associate with a connected user. In the event of a connector's failure, another user could use the IXLRT service to read all the record data entries for resources owned by the failing connector.

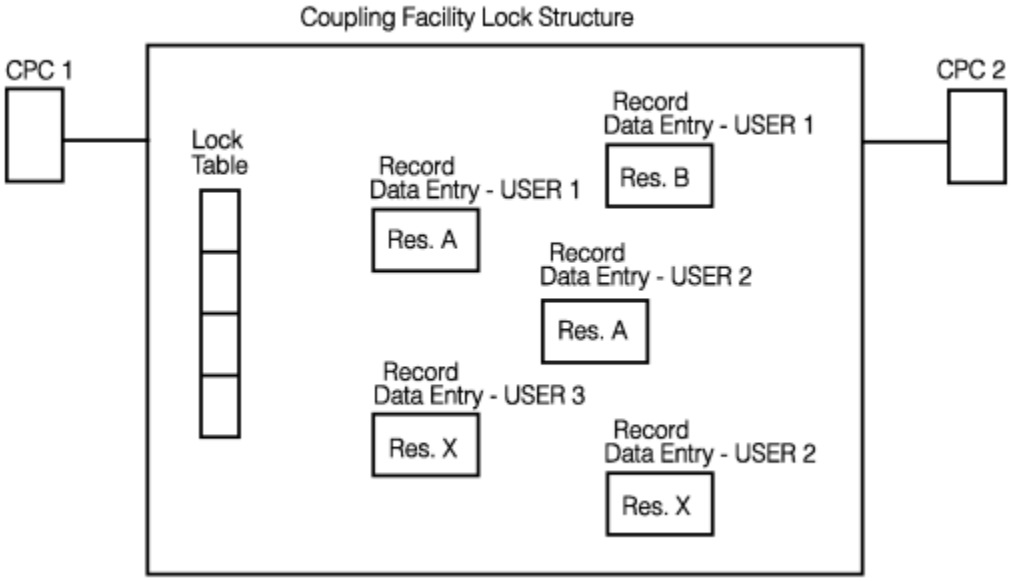


Figure 69. Lock Structure with Optional Record Data Entries

Each part of the lock structure can be addressed independently — a lock table entry through a resource name and a hash value, and a record data entry through an identifier provided when the entry is allocated as part of an IXLLOCK request.

The Lock Table

The purpose of the lock table is to detect contention efficiently, so that your contention exit will receive control only when necessary. The number of entries in the lock table is determined by the first connector to the lock structure.

Identifying a Lock Table Entry

Defining a resource for which you want serialization requires that you specify both a resource name and a hash value. XES uses the hash value to map to a specific entry in the lock table and then determines from the resource name whether or not contention exists.

Assigning a Resource Name

The resource name identifies the entity for which you want serialization. The length of the resource name can be fixed (64 bytes long) or variable (from 1 to 300 bytes long). You specify whether you are using fixed-length or variable-length resource names when you connect to the lock structure. This resource name length attribute cannot be changed either by subsequent connectors to the lock structure or when the structure is rebuilt.

Assigning a Hash Value

The hash value is the result of an application-specified algorithm to designate a specific lock table entry. The goal of this algorithm should be to distribute the resource name mappings evenly across as many hash values as possible.

Mapping a Resource Name to a Lock Table Entry

It is possible for more than one resource name to hash to the same lock table entry, depending on the hashing algorithm used. Figure 70 on page 637 depicts two resource request queues — one for resource name XYZ and one for resource name JKL. Each entry in the resource request queues show the requested resource name (RN) and the specified hash value (HV).

The hashing algorithm employed results in both resource names mapping to lock table entry 2 in the lock structure.

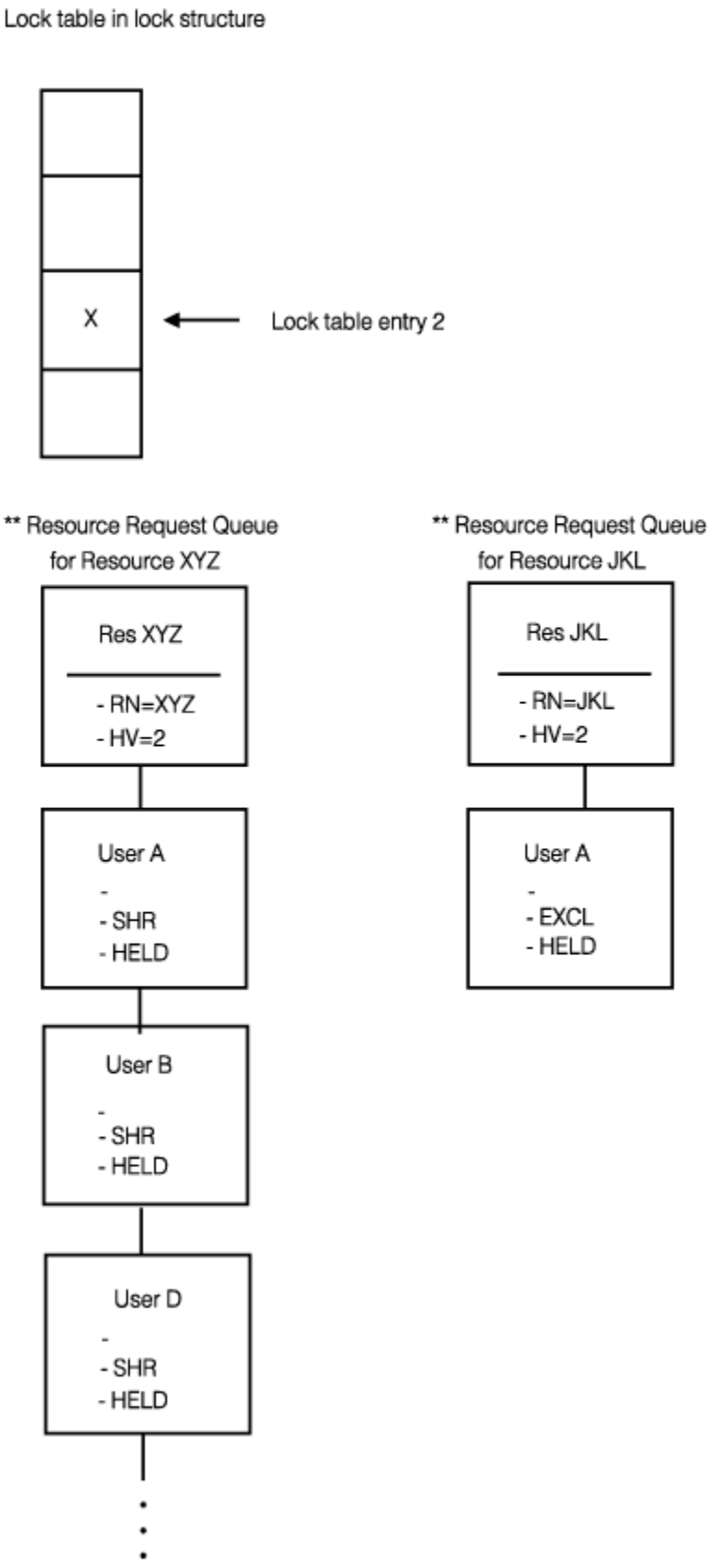


Figure 70. Lock Table — Using a Hash Value

Note that in this example, if the hashing algorithm results in a resource name mapping to lock table entry 6 in the lock structure, that value is outside the range of allocated entries. XES will “wrap-around” and again map the new resource name to lock table entry 2.

Composite State of a Lock Table Entry

The term “composite state” was previously introduced to depict the cumulative state, in terms of shared or exclusive, of a resource request queue. This same term can be used to describe the state of a coupling facility lock table entry. Whereas the composite state of a resource request queue reflects the state of all owners and requestors of a particular resource, the composite state of a lock table entry denotes the state of the owners and requestors of ALL resources mapping to that lock table entry. Similar to a resource request queue, a lock table entry can be in one of three composite states — free, shared, or exclusive.

- Free - There are currently no owners or waiters for any resource that maps to this lock table entry.
- Shared - All owners and waiters for resources that map to this lock table entry are in the shared state.
- Exclusive - There is at least one owner or waiter for a resource mapping to the lock table entry in the exclusive state.

The same rules used to determine compatibility of requests that are added to a resource request queue can also be used to determine the effect of a resource request on the state of the corresponding lock table entry. Figure 67 on page 626 illustrated this concept.

For example, Figure 70 on page 637 depicts a four-entry lock table with multiple owners of resources that hash to lock table entry 2. The composite state of this lock entry is determined to be “exclusive” because there is at least one exclusive owner of a resource mapping to this entry. Additionally, the lock entry is recognized as being incompatible because the resources (XYZ and JKL) that map to this entry are owned in conflicting states.

Understanding Contention

Once a lock table entry has became incompatible, XES will begin to incur significant overhead when processing requests for ANY resource that maps to that entry. This overhead will continue to be incurred until the composite state of the lock table entry returns to the shared or free state. A lock table entry can become incompatible for one of the following reasons:

- Resource Contention

  As previously described, resource contention occurs when the corresponding resource request queue becomes “in contention”, that is, an entry is added that is incompatible with the existing entries. For example, User 1 requests to own resource ABC with hash value 3 in the exclusive state but the resource is already owned by User 2. In this case, XES will assign management of the resource request queue to a connected user.

  Because all the entries on a particular resource request queue map to the same lock table entry, it follows that if a particular resource request queue is “in contention”, then the corresponding lock entry also is “in contention”. Thus, resource contention incurs the overhead associated with processing an incompatible lock table entry PLUS the cost of any actions performed by the application's contention exit.

- False Contention

  False contention is the term that describes other conditions that could cause a lock entry to reach the incompatible state. One such condition that could result in false contention is when two different resource names have the same hash value thus causing a collision at the lock table entry level. This is referred to as “hash class” contention.

  In the following example, User 1 owns a resource with a name of ABC and a hash value of 27 in the exclusive state and User 2 owns a resource with a name of DEF and a hash value of 27 in a shared state. Note that the requests represent different resources, each belonging to a separate compatible resource request queue. While there is no contention at the resource level (and thus no need to involve

the contention exit), the corresponding lock table entry is incompatible causing the application to incur the previously described processing overhead.

Hash Class Contention

```
  User 1 - Owner                  User 2 - Requestor

IXLLOCK REQUEST=OBTAIN          IXLLOCK REQUEST=OBTAIN

   RNAME=ABC                       RNAME=DEF
   HASHVAL=27                      HASHVAL=27
   STATE=EXCL                      STATE=SHR
```

Another condition which could result in overhead due to false contention is when two resources with distinct hash values collide at the lock table entry due to the wrap-around condition described previously.

In the following example, assume a lock table with 8 entries. User 1 owns a resource with a name of ABC and a hash value of 1. User 2 requests a resource represented by resource name DEF and a hash value of 9. Because the lock table contains only 8 lock table entries, hash value 9 will “wrap-around” and map to lock table entry 1. Once again, while the following example does not result in resource contention and the need to involve the contention exit, the application will incur the overhead required to process requests for resources mapping to a lock table entry that is in an incompatible state.

Wrap-Around Condition

```
  User 1 - Owner                  User 2 - Requestor

IXLLOCK REQUEST=OBTAIN          IXLLOCK REQUEST=OBTAIN

   RNAME=ABC                       RNAME=DEF
   HASHVAL=1                       HASHVAL=9
   STATE=EXCL                      STATE=SHR
```

In summary, the overhead incurred by XES to process requests that experience contention (whether it be false contention or resource contention) is significant. The performance of your application can be severely impacted as the number of requests that experience contention increases. For this reason, an application should design a protocol which attempts to reduce the likelihood of this occurrence.

Creating an Efficient Locking Protocol

While the requirements and characteristics of an application will ultimately influence the design of its locking protocol, it should remain a goal to produce a protocol which will result in a minimum amount of contention. The following guidelines may prove helpful in attaining this goal:

1. **Scope of Resource Definition**

   Because a resource represents an entity that is to be serialized, defining these entities to be small in scope reduces the chance of two users experiencing contention while trying to obtain the serialization needed to reference them. In the example locking protocol (“Sample Locking Protocol — Definition” on page 630), an application was required to access records in a data set. In the example, the resource was defined to be the name of the data set, with the user data containing the ranges of records within that data set that were to be accessed. When contention occurred at the resource (data set) level, the contention exit was used to determine if the update was for the same portion of the data set. While this illustrated the ability to create user locking protocols and negotiate resource ownership between connected users, it might not prove to be the most efficient method of satisfying the application's serialization requirements. An alternative approach may have been to define the resource at a more granular level, such as the record level. Using this technique would have eliminated the instances in which contention was realized at the data set level only to ultimately have the contention exit, through examination of the user data, determine that the requests were for different records within the data set.

2. **Range of Hash Values**

The hash value portion of the resource definition allows XES to map the resource to a lock table entry. The hash value is typically an output of an application defined hashing algorithm which accepts a resource name as input. If this is the case in your application, to minimize the chance of false contention, care should be taken to design an algorithm that produces hash values that are distributed evenly across a wide range of lock table entries.

3. **Accurate Planning Information**

   While an application may take great care to design a hashing algorithm that meets the criteria described above, the installation may reduce the effectiveness of this algorithm by not providing enough space within the coupling facility to allocate a lock structure with enough lock entries to accommodate the range of hash values. When providing information about structure size to your application's users, explain the performance ramifications of their specifying a smaller lock structure than recommended.

Analyzing Your Locking Protocol

The XES Accounting and Measurement service, IXLMG, provides information to assist an exploiter of XES locking services in analyzing the efficiency of its locking protocol. The service returns information about a connection's use of a particular structure in an area mapped by the IXLYAMDA macro. The following fields are of interest while doing this analysis:

- **IXLYAMDSTRL_NLE**

  This field contains the number of entries allocated in the lock table portion of the lock structure. The value of this field may help the application determine when it is experiencing false contention due to having insufficient lock table entries to accommodate the range of hash values.

- **IXLYAMDSTRL_REQCT**

  This field contains the total number of IXLLOCK, IXLRT, and IXLSYNCH requests issued for this lock structure.

- **IXLYAMDSTRL_NLTEC**

  This field contains the total number of lock table entries whose composite state is shared or exclusive; that is, a count of lock table entries that currently have resources mapping to them. This field can be used along with the IXLYAMDSTRL_REQCT field to determine how evenly the application's hashing algorithm is distributing the hash values across the allocated lock table entries. This count is only substantially accurate.

- **IXLYAMDSTRL_REQCTASYNC**

  This field is a subset of the total request field and contains the number of requests which XES experienced a delay in servicing. The delay can occur for various reasons such as contention or XES not being able to immediately obtain serialization on its own structures. This field can be used in conjunction with the IXLYAMDSTRL_REQCT field to determine the percentage of requests which XES was able to service immediately without delays.

- **IXLYAMDSTRL_CONTCT**

  This field is a subset of the IXLYAMDSTRL_REQCTASYNC field and contains the number of requests that were delayed due to contention. Note that this field contains the number of instances of both resource contention and false contention.

- **IXLYAMDSTRL_FCONTCT**

  This field is a subset of the IXLYAMDSTRL_CONTCT field and contains the number of requests that were delayed due to false contention. An application can determine the number of requests experiencing resource contention by subtracting this value from the IXLYAMDSTRL_CONTCT field.

- **IXLYAMDSTRL_MLSEC**

  This field contains the maximum number of record data elements allocated in the lock structure. This count is only substantially accurate.

- **IXLYAMDSTRL_CRITICALREQUESTCOUNT**

This field contains the total number of IXLLOCK requests specified with CRITICALREQUEST(IxllockCriticalRequestYes). This count is only substantially accurate.

This information may also be obtained by running the appropriate Resource Measurement Facility (RMF) Coupling Facility Activity reports. See *z/OS Resource Measurement Facility User's Guide* for additional information about these reports.

Record data entries

A record data entry contains information about a resource that is or has been held by a connected user. The information in the record data entry is relevant to the user holding the resource and is normally used by no other user sharing the resource unless the other user needs to use the data to recover for a failure of the resource owner.

Record data entries are 64 bytes long. The first connector to the lock structure specifies whether record data entries are to be used. A record data entry can be allocated within the lock structure when an obtain or an alter request is issued. When a record data entry is allocated in conjunction with an IXLLOCK request, the entry is created with a record data type (RDATATYPE) of zero. See "Using the Lock Cleanup and Recovery Service (IXLRT)" on page 672 for information about RDATATYPE.

Each record data entry records information about a connected user's interest in a specific resource. Because record data can persist across system or sysplex outages, if a recovery situation occurs, the users of the lock structure can make use of the record data entries to perform recovery for resources that were held by a user at the time of the failure.

In general, use of a record data entry must be serialized. This is especially important when working with structures in the async duplex established phase. Consider that a record data entry can be read before the update to the secondary structure instance occurs. If both duplexing failover and failure of the connector that made the update occur before the update to the secondary structure instance is made, the read record data entry might not end up in the resultant simplex structure. Also, in cases where a loss of connectivity to the primary structure instance occurs, the request will be redriven to the resultant simplex structure, possibly resulting in a second update to the record data entry. Without proper serialization, another connector could observe that the record data entry was updated multiple times.

Associating Record Data Entries with Connected Users

A record data entry is identified by an entry-identifier, ENTRYID, which is returned to you when you allocate the entry with an IXLLOCK request to OBTAIN or ALTER ownership of a resource.

In a failure situation, the users of the lock structure can use the IXLRT service to access the record data entries for the failed user and thus coordinate the recovery processing. See "Using the Lock Cleanup and Recovery Service (IXLRT)" on page 672.

Capacity Planning for Record Data Entries

On each IXLLOCK OBTAIN and ALTER request that specifies record data is to be written, XES returns information about the number of record data entries currently in use. Either the ENTRYCOUNT output field in the IXLLOCK parameter list or the CMPLRTENTRYCOUNT field in the IXLYCMPL parameter list, if processing was asynchronous, contains this value. Use the value returned in ENTRYCOUNT or CMPLRTENTRYCOUNT to monitor how much storage remains in the structure for record data entries.

The approximate maximum number of record data entries that the structure supports is returned in the answer area by IXLCONN (field CONALOCKMAXRECORDELEMENTS). You can also determine this value by using the XES Accounting and Measurement service, IXLMG, to request structure information. By comparing the value of ENTRYCOUNT with the value of the maximum supported number of record data entries, you can anticipate a "structure full" situation and take appropriate actions to avoid the occurrence of such a condition.

When a structure resides in a CFLEVEL=25 CF or higher, you can use the IXLCONN PCTENTRYRSV to specify the percentage of the total number of record data entries that are to remain free (reserved) after

completion of a request that attempts to create a record data entry. By setting aside a percentage of record data entries, you can better manage the availability of record data entries and can thereby create a record data entry during application and connector recovery scenarios when deemed necessary as long as there is a free record data entry to satisfy the request.

When a percent entry reserve has been established for the structure, a request will not be permitted to create a record data entry if request processing creates a record data entry and the resulting percentage of free entries at the completion of the request would be less than the established percent entry reserved threshold for the structure. A "structure full" condition will be reported, and the request will complete with an error return code of IxlRetCodeEnvError, reason code of IxlRsnCodeRtFull. When needed, an application may indicate that a lock request that attempts to create a record data entry may proceed with creating a record data entry even if the resulting percentage of free entries at the completion of the request would be less than the established percent entry reserved threshold. For more information see IXLLOCK Services and IXLRT – lock structure record data processing in *z/OS MVS Programming: Sysplex Services Reference*.

By comparing the ENTRYCOUNT with the maximum supported number of record data entries less the number of record data entries that represent the amount of record data entry reserve ((CONALOCKMAXRECORDELEMENT x CONALOCKPCTENTRYRSV) / 100), a "structure full" condition can be anticipated as a result of the established percent entry reserve value for the structure. For more information see "Determining Whether to Specify Record Data" on page 239.

Size Considerations for a Lock Structure

The initial size of the lock structure is specified by the installation in the CFRM policy, although that value might be overridden by the structure size specified on the IXLCONN macro. (The system uses the smaller of the sizes, if both are specified.) When providing guidance for determining a lock structure size, you must consider both parts of the structure — the lock table and the record data entries.

Lock Table Size

The size of the lock table is determined by the number of lock entries that are used to detect contention and the number of potential sharers of each lock table entry.

- Choosing the number of lock table entries

  In order to achieve optimum performance, the application should allocate a lock table with enough entries to accommodate the range of hash values. Allocating a lock table with entries greater than the maximum hash value results in unnecessary coupling facility storage being assigned to the structure, because XES will never need to reference a lock table entry beyond the range of hash values. (Note that the number of lock entries that you specify on the IXLCONN macro is rounded up to a power of 2 if the value is not already a power of 2.)

  Allocating a lock table with entries less than the maximum hash value could result in significant performance degradation because XES will be forced to assign multiple hash classes to a single lock entry. (XES will "wrap-around" when the hash value is outside the range of the last lock table entry.)

  You can have the system automatically generate the largest possible number of lock entries within the allocated lock structure when the following prerequisites are met:

  – The lock structure is allocated without record data.
  – The lock structure is allocated by an OS/390 Release 2 or higher system.
  – You specify LOCKENTRIES=0 on the IXLCONN invocation.

- Choosing the number of lock table users

  The number of lock table users determines the size of each lock table entry within the lock table. Each lock table entry consists of one byte and then one bit for each potential user of the lock table entry, as specified by the NUMUSERS or MAXCONN keyword on IXLCONN. Bit 0 is not used to represent a connected user. The size of the entry is rounded to a power of 2 number of bytes.

  For example, each lock table entry for 27 users, after rounding to a power of 2, would be 8 bytes.

27 Users

```
|________|0xxxxxxx|xxxxxxxx|xxxxxxxx|xxxx____|_...|...|..._|
   1 byte    <-------- 27 user bits ------->
```

Each lock table entry in the lock table is 8 bytes.

The following example shows a lock table entry for 16 users.

16 Users

```
|________|0xxxxxxx|xxxxxxxx|x_______|
  1 byte   <- 16 user bits ->
```

Each lock table entry in the lock table is 4 bytes.

Storage Required for Record Data Entries

If your protocol does not require the use of record data entries, then no additional storage in the coupling facility is required beyond that for the lock table. However, if you are using record data entries, keep in mind the following:

- Each record data entry is 64 bytes. (This is the user-available portion. The coupling facility control code requires additional coupling facility storage for control purposes.)
- The number of record data entries to estimate is a function of the number of users and their recording activity. The amount of recording activity is a product of the locking rate, the number of resources using record data, and the length of time resources and their associated record data entries are held.
- The coupling facility allocates storage for the lock table first and whatever storage remains in the structure can be used for record data entries.

See "Determining the Structure Size" on page 767 for detailed information about estimating structure size, including the additional coupling facility storage required by the coupling facility.

Effect of Structure Alter on a Lock Structure

The IXLALTER function provides for the expansion or contraction of the size of a structure and/or for the reapportionment of the entry-to-element ratio of the structure. For a lock structure, only a change to the size of the structure is valid. A request to change the entry-to-element ratio is meaningless because there are no data elements in a lock structure, only record data entries. The system rejects such a request to change the ratio with nonzero return and reason codes.

Depending on whether or not the lock structure was allocated with record data, changing the size of a lock structure either increases or decreases the record data entries only.

Recovery Considerations

As part of your application's design to use a coupling facility lock structure, you should plan for recovery if a connected user fails. Your recovery plan can allow for either the connected user to be restarted and continue processing or for peer connectors to assume the responsibilities of the failed connector or for a combination of both.

Designing for recovery

When a connector to a coupling facility lock structure fails, the following occurs:

- XES reports the failure event to all surviving connectors.
- The surviving connectors must respond to the event.
- When all responses have been received by XES, XES performs its cleanup.

As part of your recovery protocol, you might require that connectors do their application-specific cleanup before responding to the connection failure event.

When asynchronous duplexing failover to the secondary occurs at the same time as the failure of a connector, resources held by the failing connector might become available before surviving connectors respond to the event. This might happen when the lock update was not yet committed to the secondary structure instance.

Your recovery protocol has several dependencies:

- How connections are defined at connect time
- How recovery information is used for peer and restart recovery.

Defining the Connections

At connect time, with the IXLCONN macro, the persistence of a connection is defined with the CONDISP parameter. The disposition indicates how the connection is to be handled in the event the user terminates abnormally or disconnects from the structure with REASON=FAILURE. By specifying CONDISP=KEEP, you indicate that the connector is to enter a failed-persistent state when all surviving connections' event exit responses have been received. CONDISP=KEEP also ensures that the failed user's record data is to be kept. XES will not delete record data entries belonging to a connection that is entering the failed-persistent state. This allows data regarding owned resources to be available to the failing user upon restart, as well as during system outages in which peer recovery was not able to be performed. Note that XES releases the resources owned by a failing user regardless of the CONDISP specified at connect time.

Specifying Recovery Information

Also at connect time, you indicate whether or not you want to maintain record data entries for the connection. The record data entries can be used to hold recovery data, should the connector fail. Connectors to the structure can access the record data entries with the IXLRT programming interface. The failed connector, once restarted, can access its former record data entries with the REACQUIRE option of the IXLLOCK programming interface when it reobtains serialization on the specified resource.

When a structure resides in a CFLEVEL=25 CF or higher, you can use the IXLCONN PCTENTRYRSV to specify the percentage of the total number of record data entries that are to remain free (reserved) after completion of a request that attempts to create a record data entry. By setting aside a percentage of record data entries, you can better manage the availability of record data entries and can thereby create a record data entry during application and connector recovery scenarios when deemed necessary as long as there is a free record data entry to satisfy the request. For more information see “Determining Whether to Specify Record Data” on page 239.

XES cleanup processing

When all responses are received from surviving connectors, XES performs the following cleanup:

1. Remove entries associated with the failed user, whether the entry represents the user as an owner or a waiter for the resource, from any resource request queues.
2. If the user is not persistent, delete any record data entries associated with the failed user.
3. If the failed user was currently assigned contention management responsibilities, reassign those responsibilities to another connected user.

Resource Request Queues

XES removes the failed user's requests from any resource request queue on which the user is shown to be a resource owner or waiter (that is, has a request pending). Also, XES cancels any requests from the failed user that are waiting to be applied to a resource request queue. If a resource request queue from which the failed user's requests are removed is being managed by the contention exit of a surviving connector, then the updated resource request queue is presented to that user's contention exit. Reason flags in the CEPL will indicate that recovery has occurred. If the contention exit was waiting for a response from the failed connector at the time of the failure (such as a response from the notify exit), then XES will cancel the wait for the response.

Record Data Entries

Whether XES deletes the record data entries associated with the failed user depends on how the user's connection was defined at connect time. If the failed user is transitioning to a failed-persistent state, then the record data entries are kept and are available either for the restarted connector or for peer connectors as part of the recovery protocol. If the failed user was not defined to become failed-persistent, then XES deletes any associated record data entries for the failed user.

When a record data update is made in the async duplex established phase, updates that were made by the failed-persistent connector but were not committed to the secondary structure instance may be lost. For more information, see “Working with structures in the Async Duplex Established phase” on page 285.

Contention Management Responsibilities

The responsibility of managing resource request queues is shared among the instances of XES supporting the connectors to the structure. At any point in time a connector may be managing any number of resource request queues through its contention exit, and the instance of XES supporting the connector may internally be managing additional resource request queues that are not in contention. Whenever a connector disconnects or abnormally terminates, XES reassigns management responsibilities for any resource request queues that were being managed by that connector (or its associated instance of XES) among the remaining connectors and their supporting instances.

After removing the failed user's requests from any resource request queues, the system determines if the failed user's contention exit (or the instance of XES supporting the connector) had been managing any resource request queues at the time of the error. If so, and the queue still contains owners, XES reassigns management responsibilities to the contention exit of another connected user. When the newly selected user's contention exit is first invoked, the CEPL will indicate that recovery has occurred.

Note that because the management of resource request queues is reassigned in these instances without considering the current (or previous) state of the entries on the queue, a contention exit may be presented with a resource request queue containing entries that are compatible. This method of overindicating the state of a resource request queue in failure scenarios ensures that any communications that had been established with local connectors by the contention exit of the failing user will have a chance to be completed by the contention exit of the connector who become the new manager regardless of whether the queue is still in contention after the cleanup has occurred.

Note about Deadlock:

You should be aware of the potential for a deadlock environment when XES is waiting to reassign management responsibilities for a resource request queue. XES cannot assign another connected user to manage the resource request queue until all acknowledgments of the failed connection have been received from the surviving connectors through either their event exits or IXLEERSP. Therefore, any resource requests on the queue will remain outstanding until a new contention manager is assigned. A deadlock situation could occur if you delay confirming an XES event while awaiting the completion of an IXLLOCK resource request. The responsibility of detecting and resolving deadlock situations is that of the connected user and not of XES.

Sample Recovery Protocol

The following illustrates a recovery protocol in which the failed connector is to restart and complete any updates that were in progress at the time of its failure.

Application “B” is a multi-system application whose data is maintained on shared DASD. The application accesses the data as a result of user requests. Should an instance of Application B fail, the remaining instances are to prevent access to any shared data that may have been in the process of being updated by the failing instance at the time of the failure. Upon being restarted, the instance of Application B that failed will complete its update of the shared data.

Purpose: To provide a recovery protocol which maintains the integrity of the shared application data across system and sysplex outages.

Design: Any data that was in the process of being updated when an instance of the application fails is to be marked “reserved” until the failing instance is able to be restarted. Once restarted, the instance of the

application is to determine what updates were in progress at the time of the failure and continue with the update.

The surviving instances of the application must maintain structures at the application level to denote “reserved” resources because XES, after receiving cleanup confirmations, will release the resources owned by the failing connector. Once XES releases the failed connector's ownership of a resource, that resource is available to be obtained by other active connectors. This implies that any new requests to obtain the resource will be successful. Therefore, if the application wishes to prevent access to that resource until the failing instance is restarted, it must do so at the application level. Specifically, each instance of the application must verify that the resource is not reserved before initiating an IXLLOCK request.

Requirements: The application must conform to the following:

- Connections by the application must be persistent.

  The persistent connection ensures that any record data entries created by the application remain resident in the lock structure across failures.
- An IXLLOCK record data entry is to be recorded whenever the application requests an update to shared data. The record data entry is to contain the following information:
  - The name of the data set being updated.
  - The range of records requiring serialization.
- The application must maintain local structures to denote resources that are currently “reserved”.

  The local structures are required so that access to the reserved resources can be prevented until the failed instance is able to restart. The local structures must be updated with the list of reserved resources at the following times:
  - During startup of an instance of an application

    When an instance of an application establishes a connection to XES, the connector receives information about other connections (both active and failed-persistent) through its event exit. The instance of the application must use the IXLRT service to read the record data entries associated with any failed-persistent connector to determine the resources that might be reserved for reclamation when the failed user restarts.
  - Prior to providing cleanup confirmation for a peer user.

    When a failure of a peer user is reported through the event exit, the connector must use the IXLRT service to read the record data entries associated with the failing connector. Information in the record data entries is used to update the list of reserved resources in the user's local structure.
- Before initiating an IXLLOCK request, the application must verify that the resource being requested is not reserved.
- Resources owned by a failed persistent user are to be reacquired upon restart.

Sample Recovery Protocol - Implementation

The following illustrates Application B's implementation of the restart recovery protocol.

- Two instances of Application B are executing on different systems in a sysplex. Both are connected to coupling facility lock structure LOCKAA.

  ```
  INSTANCE 1 PROCESSING
  ```
- Instance 1 of Application B receives a client request to update record 2 of data set XYZ.
  - Instance 1 of Application B issues an IXLLOCK request for exclusive access of data set XYZ using lock structure LOCKAA.
  - The user data specified indicates that the IXLLOCK request is to access record 2.
  - The IXLLOCK request specifies that a record data entry is to be created.
- XES grants the IXLLOCK request for exclusive access of data set XYZ; the record data entry is created.

- Instance 1 of Application B begins its update of the serialized record, record 2.
- Before Instance 1 can complete its update, the system on which it is running fails.

```
XES PROCESSING
```

- XES informs the remaining instances (in this case, only Instance 2) about the failure of Instance 1 through the scheduling of their event exit. XES then waits for the remaining instances (in this case, Instance 2) to acknowledge the event before proceeding with the cleanup.

```
INSTANCE 2 EVENT EXIT PROCESSING
```

- Instance 2 invokes the IXLRT service to return the record data entries associated with the failing Instance 1. For each returned entry, Instance 2 adds an entry to its locally maintained reserved resource list.

  Note that if Instance 2 receives any new requests for a resource on the reserved resource list, the Instance 2 rejects the requests with an indication that the specified resource is temporarily not available.
- After building the reserved resource list, Instance 2 provides a cleanup confirmation to XES.

```
XES PROCESSING
```

- Having received the cleanup confirmation from Instance 2, XES performs the necessary cleanup processing. The state of Instance 1 is now:
  - Its interest in the resource Data Set XYZ has been released. (Instance 1's request has been removed from the resource request queue for Data Set XYZ.)
  - Its connection is in the failed-persistent state.
  - Its record data entries remain resident in the coupling facility lock structure.

```
INSTANCE 1 RESTART PROCESSING
```

- Instance 1 of Application B is restarted. Instance 1 reestablishes its connection to coupling facility structure LOCKAA.
- After reestablishing its connection, Instance 1 issues the IXLRT macro to determine what resources it held at the time the previous instance failed. Additionally, the data returned by IXLRT can be used to populate the local structures with resources that are reserved by other instances that may currently be failed.
- In order to reobtain ownership of the resources that are currently reserved on its behalf, Instance 1 issues IXLLOCK requests. Also, Instance 1 indicates to reassociate the current record data entry with the new instance of resource ownership by specifying the REACQUIRE keyword on the IXLLOCK invocation.
- When the restart processing is complete, Instance 1 notifies the other active connections to discard the appropriate resources from their reserved resource list.

Sample Recovery Protocol - An Alternative

An application might have requirements that could not tolerate “reserving” a resource for the length of an outage, as in the preceding example. In these types of environments, an application might choose to employ a “peer recovery” protocol. In a peer recovery protocol, surviving instances of an application would attempt to take over or complete unfinished work that had been started by the failing instance. XES locking services also assist in enabling an application to build such a protocol.

Requesting Lock Services

To request lock services, you issue the IXLLOCK macro from the same address space where the IXLCONN macro for the connection to the lock structure was issued. You identify the service you want (OBTAIN, ALTER, RELEASE) by specifying the name of the service on the REQUEST keyword. You also must specify

the CONTOKEN keyword to identify your connection and the RNAME and HASHVAL keywords to identify the resource.

z/OS MVS Programming: Sysplex Services Reference provides the required syntax for coding the IXLLOCK macro.

Connecting to a Lock Structure — A Review

To use a lock structure, you must first connect to the structure with the IXLCONN macro, TYPE=LOCK. On the IXLCONN macro, you specify the attributes you require the structure to have, such as the number of users to be supported and whether or not record data is to be used. Be aware, however, that despite your IXLCONN-specified attributes, XES might need to allocate a lock structure with different attributes. It is your responsibility to verify that the attributes as allocated will satisfy your application's requirements. Also, for a lock structure, a return and reason code of IXLRSNCODENOFAC (X'0C08') indicates that the allocation failed because there was no suitable coupling facility in which to allocate the structure based on the preference list in the CFRM policy. For example, a request to allocate a lock structure with 64 lock entries to be used by eight users may fail if a structure large enough to meet these requirements cannot be allocated in any coupling facility defined within the preference list.

One way of ensuring that you request a set of attributes that agrees with an installation's requirements is to interrogate the CFRM policy using the IXCQUERY macro. The CFRM policy provides the structure name and size, the number of connections supported by the policy, and other installation-specific information about the coupling facility. Use this information to specify lock structure attributes that match the installation's configuration.

Requesting Ownership of a Resource (REQUEST=OBTAIN)

Use the OBTAIN request to specify the state in which you want ownership of the resource, as well as to define certain resource attributes. You cannot issue multiple concurrent OBTAIN requests for the same resource. If you do so, XES rejects the request with a reason code indicating that the requested resource is either already owned or already pending ownership.

Note that if you wish to update the attributes for a resource that you already own or are attempting to own, you should use the ALTER option of IXLLOCK.

- **State**

  The OBTAIN request allows you to request shared or exclusive ownership of a resource. The system supports only the shared or exclusive lock states. However, you have the ability to define other lock states by using the user data. If you define other lock states, you must ensure that the additional lock states defined map to either a shared or an exclusive lock state.

  The resultant state (which may or may not have been modified by the contention exit) is returned in the STATEVAL keyword for synchronous requests and in the CMPLSTATE field for asynchronous requests.

  In addition to the requested ownership state, on an IXLLOCK OBTAIN request you can specify the following user-defined data:

  – Eight bytes of lock data
  – 64 bytes of user data
  – 64 bytes of record data.

- **Lock data**

  The eight bytes of lock data that you can specify on an OBTAIN request remains associated with the owned resource once the request is granted and is for use only by the requesting user. Lock data is presented to the complete and notify exits and typically would be used to contain an address or similar control information about the use of this resource. Thus, when updates are made to the resource and the complete exit is called or when the notify exit is invoked, the user can efficiently locate the control structures pertaining to the resource.

- **User data**

The 64 bytes of user data that you can specify on an OBTAIN request remains associated with the owned resource once the request is granted. Unlike the lock data, you can modify the user data on subsequent ALTER or RELEASE requests for the resource. The user data is presented to the complete exit, the notify exit, and the contention exit, and typically would be used to contain data necessary to implement your locking protocol.

The user data (which may or may not have been modified by the contention exit) is returned in the UDATAVAL keyword on synchronous requests and in the CMPLUDATA field for asynchronous requests.

If you do not specify user data, the area contains zeros.

- **Record data**

  The 64 bytes of record data that you can specify on an OBTAIN request represents the connected user's interest in a particular resource. The OBTAIN request can be to write the record data to an available record data entry in the lock structure or to reacquire a record data entry that already exists. The record data is presented in the complete exit and the contention exit.

  If a record data entry is created, a unique entry identifier and an indication of the number of record data entries currently allocated in the structure are returned to the user. These values are returned in the ENTRYID and ENTRYCOUNT keywords for synchronous requests and in the CMPLRTENTRYID and CMPLRTENTRYCOUNT fields for asynchronous requests. If a record data entry is not available, ownership of the resource is not granted, and the system provides an error return code.

  When a structure resides in a CFLEVEL=25 CF or higher, you can use the IXLCONN PCTENTRYRSV to specify the percentage of the total number of record data entries that are to remain free (reserved) after completion of a request that attempts to create a record data entry. By setting aside a percentage of record data entries, you can better manage the availability of record data entries. Using the ALLOWUSERSV keyword, you can access the reserved entries and create a record data entry during scenarios when the creation of a record data entry is deemed necessary, as long as there is a free record data entry in the structure. For more information, see IXLLOCK Services in "Determining Whether to Specify Record Data" on page 239.

  If a record data entry already exists for this resource request, you can use the REACQUIRE keyword on the OBTAIN request to reacquire both ownership of the resource and the associated record data entry. When specifying the REACQUIRE option, use the ENTRYID keyword to identify the record data entry and optionally, the CONID keyword to identify the connection from whom the record data entry is being reacquired. The REACQUIRE option is primarily intended for use in a recovery environment to facilitate recovery of resources. Consider the following examples:

  – Upon reconnecting, a previously failed-persistent user of locking services can re-obtain resources that were held by its previous instance and reacquire the existing record data entries to be associated with the new instance of ownership. It is possible to use the UPDATERDATA suboption to update the contents of the reacquired record data entries to reflect updated state information.
  – A connected user of locking services fails and the related surviving users wish to recover the resources held by the failing user. The survivors might wish to obtain the specified resources while reacquiring the associated record data entries from the failed connector. It is possible to use the CONID suboption to coordinate the surviving connectors' processing.

    Note that CONID is a one-byte connection identifier. CONID is returned in the connect answer area (IXLYCONA) upon the successful completion of an IXLCONN request. You can use CONID during recovery processing to identify a failed connection for which you are attempting to recover resources. Specifically, use CONID when reacquiring the record data entries for the failed user. When you reacquire a record data entry, XES associates the entry with your connection.

    If the record data entry specified by the ENTRYID does not already exist, or if the record data entry that you specify with ENTRYID is not associated with the connection specified by CONID, the ownership of the resource is not granted, and the system provides an error return code.

  The record data (which may or may not have been modified by the contention exit) is returned in the RDATAVAL keyword on synchronous requests and in the CMPLRDATA field for asynchronous requests.

Determining the Completion of an OBTAIN Request

Upon the successful completion of an OBTAIN request, the connected user is recognized as an owner of the specified resource. If ownership was granted through the contention exit then the attributes may have been modified. For example, the resource may have been granted with state, user data, or record data different from what was originally requested. The connected user can examine the appropriate output fields to determine if the attributes have in fact been modified.

If the OBTAIN request was not successful, then the requestor is not the owner of the resource. Any subsequent requests (or those that may have been issued while waiting for the results of the OBTAIN request) will fail with return and reason codes indicating that the requested resource is not owned by this connector. The reasons for which an OBTAIN request might fail include user-controlled conditions, such as the request being denied by the contention exit, and environmental conditions, such as loss of connectivity or structure failure.

Return and Reason Codes

When you invoke IXLLOCK, the macro returns the status of the request through return and reason codes. The return and reason code constants are defined in the IXLYCON macro.

Some examples of the type of status information returned from an OBTAIN request are:

- The request is being processed asynchronously. Results will be presented to the complete exit.
- The request is granted. You should check the appropriate output fields to determine if any attributes (such as state, user data, record data) were changed.
- The request is superseded and ownership is not granted.
- The request is denied by the contention exit that was managing contention for the resource.
- The request has failed because of an environmental condition, such as structure failure or loss of connectivity to the coupling facility.

Changing Ownership Attributes (REQUEST=ALTER)

Use the ALTER request to change the attributes of a resource that is already held or to replace a OBTAIN or ALTER request for the resource that is pending on the contention exit resource request queue with a more current request.

You can only alter a resource of which you are the owner or for which you have a pending request. Otherwise, the system rejects the ALTER request with return and reason codes.

Resource attributes that can be changed are the state, user data, and record data.

- **State**

  You are required to provide a requested state when issuing an IXLLOCK REQUEST=ALTER. If you do not wish to modify your current ownership state, you should provide the current value as input.

  The resultant state (which may or may not have been modified by the contention exit) is returned in the STATEVAL keyword on synchronous requests and in the CMPLSTATE field for asynchronous requests.

- **User data**

  On an ALTER request you also can specify the 64 bytes of user data. If you do not wish to modify your current user data, you should respecify its current value.

  The user data (which may or may not have been modified by the contention exit) is returned in the UDATAVAL keyword on synchronous requests and in the CMPLUDATA field for asynchronous requests.

- **Record data**

  On an ALTER request you can specify whether or not to update the 64 bytes of record data that is currently associated with the resource, to delete the record data entry that is currently associated with the resource, or to create a new record data entry to be associated with the resource (if one did not previously exist). The record data is indicated by the RDATAVAL keyword.

– If a record data entry was not associated with this resource previously, the system attempts to write to an available record data entry. If a record data entry is not available, the request is rejected and a return code is returned. If a record data entry is available and can be allocated, the record data entry is written to the allocated entry. The identifier of the record data entry and the number of record data entries currently in use are returned in the ENTRYID and ENTRYCOUNT areas in the IXLLOCK parameter list for synchronous requests and in the CMPLRTENTRYID and CMPLRTENTRYCOUNT fields for asynchronous requests.

  When a structure resides in a CFLEVEL=25 CF or higher, you can use the IXLCONN PCTENTRYRSV to specify the percentage of the total number of record data entries that are to remain free (reserved) after completion of a request that attempts to create a record data entry. By setting aside a percentage of record data entries, you can better manage the availability of record data entries. Using the ALLOWUSERSV keyword, you can access the reserved entries and create a record data entry during scenarios when the creation of a record data entry is deemed necessary, as long as there is a free record data entry in the structure. For more information, see IXLLOCK Services in "Determining Whether to Specify Record Data" on page 239.

– If a record data entry already is associated with the resource and the ALTER request is to change this record data entry, the system replaces the contents of the prior record data entry with the record data entry specified in the current ALTER request. The entry identifier remains the same.
– If a record data entry already is associated with the resource and the ALTER request is to delete this record data entry, the system deletes the record data entry. If no record data entry is associated with the resource, the RDATA=DELETE keyword is ignored.

The record data entry (which may or may not have been modified by the contention exit) is returned in the RDATAVAL keyword on synchronous requests and in the CMPLRDATA field on asynchronous requests.

Determining the Completion of an ALTER Request

Upon successful completion of an ALTER request, the user's ownership attributes will have been updated. If the request was granted through the contention exit then the attributes may have been modified. For example, the request may have been granted with the state, user data, or record data different from what was originally requested. The user can examine the appropriate output fields to determine if the attributes have been modified.

If the ALTER request was not successful, the attributes specified are not modified. The reasons for which an ALTER request might fail include user-controlled conditions, such as the request being denied by the contention exit, and environmental conditions, such as loss of connectivity or structure failure.

Return and Reason Codes

When processing is complete for an ALTER request, the system provides a return and possibly a reason code to indicate the status of the request.

Some examples of the type of status information returned from an ALTER request are:

- The request is being processed asynchronously. Results will be presented to the complete exit.
- The request is granted. Check the appropriate output fields to determine if any attributes were changed.
- The request has failed because you requested that a record data entry was to be written, but there were no record data entries available.
- The request is superseded (or cancelled due to a more current request by this user being received by the contention exit that is managing the resource).
- The request is denied by the contention exit that was managing contention for the resource.
- The request has failed because of an environmental condition, such as structure failure or loss of connectivity to the coupling facility.

Releasing ownership of a resource (REQUEST=RELEASE)

Use the RELEASE request to relinquish ownership of a resource or to replace a request to OBTAIN or ALTER the resource that is pending on the contention exit resource request queue with a more current request to release it (in effect, canceling the pending request).

You can only release a resource of which you are the owner or for which you have a pending request. Otherwise, the system rejects the RELEASE request with return and reason codes.

On an IXLLOCK RELEASE request you can modify the user data and the record data and also specify the disposition of the record data that is associated with the resource request.

- **User data**

  The user data (which may or may not have been modified by the contention exit) is returned in the UDATAVAL area in the IXLLOCK parameter list for requests that complete synchronously and the CMPLUDATA field for asynchronous requests.

- **Record data**

  On a RELEASE request you can specify what to do with the record data entry that is associated with the resource. The RDATA keyword indicates whether to delete, to keep, or to keep and update this record data entry. Keeping the record data in the coupling facility lock structure allows ownership information to remain and continue to be available to connected users after the ownership of the resource has been released. However, it is the responsibility of the connected users to either reacquire or free these record data entries.

  When the record data entry is deleted, it is deleted after ownership of the resource is released.

Determining the Completion of a RELEASE Request

In general, a request to RELEASE a resource cannot fail nor be denied by the contention exit. When RELEASE processing is complete, the resource will have been released.

For synchronous processing, the system provides a return code and possibly a reason code when a RELEASE request is complete.

For asynchronous processing, the system provides a return code and a reason code to indicate that processing is not complete. Completed asynchronous processing is reported in the complete exit unless you specified the MODE=NORESPONSE option. If you specify MODE=NORESPONSE and the request is processed asynchronously, your complete exit will not receive control when the request processing completes.

Return and Reason Codes

When processing is complete for a RELEASE request, the system provides a return and possibly a reason code to indicate the status of the request.

Some examples of the type of status information returned from a RELEASE request are:

- The request is being processed asynchronously. Results will be presented to the complete exit unless MODE=NORESPONSE was specified on the request.
- The request to release the resource is successful, possibly with one of the following exceptions:
  - You requested that the record data entry was to be kept (RDATA=KEEP), but there was no record data entry associated with the resource.
  - You requested that the record data entry was to be kept and its contents updated (UPDATERDATA=YES), but the system was unable to update the contents. The record data entry is kept.
  - You requested that the record data entry was to be deleted (RDATA=DELETE), but the system was unable to delete the entry.

Processing multiple resource requests (REQUEST=PROCESSMULT)

Use the PROCESSMULT request to have the system process multiple requests for resources with a single invocation of IXLLOCK. IXLLOCK Version 1 supports the PROCESSMULT option with the 'RELEASE' type. As with a single 'RELEASE' request, you can specify either to keep or to delete the record data associated with the resource request. However, note that there is no support for updating the record data when keeping it with the PROCESSMULT RELEASE option. REQUEST=PROCESSMULT also does not support resource names with a length greater than 64 characters.

The PROCESSMULT request type is valid only for a structure allocated in a coupling facility with CFLEVEL=2 or higher.

For each resource request that you wish to process, you build a lock request block (LRB) to represent that request. An LRB is mapped by the macro IXLYLRB. You can specify up to 128 resource requests on a PROCESSMULT invocation. You build the LRBs representing these resource requests in the virtual storage area specified by REQBUFFER. The REQBUFFER area can hold from 1 to 128 individual lock request blocks. For a description of IXLYLRB, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

IXLLOCK Version 4 supports the IXLLOCK PROCESSMULT REQVERSION keyword. With a REQVERSION value of at least 1, the user can be provided with the ADUPREQSEQNUM for system-managed asynchronous duplexing support.

Table 47 on page 653 shows the information that each lock request block contains.

Table 47. IXLLOCK Lock Request Block Information

Field Name	Description
LRB_XTYPE	Type of request Value must be LRB_XTYPE_RELEASEVERS0
LRB_XRNAME	Resource name
LRB_XHASHVAL	Hash value
LRB_XUDATAVAL	User data value
LRB_XMODE	Mode in which the request is to be processed if it cannot be serviced immediately: **0 (LRB_XMODE_SYNCEXIT)** Process the request asynchronously and give control to the user's complete exit when the request is complete. **1 (LRB_XMODE_NORESPONSE)** Do not inform the caller when the request is complete.
LRB_XRDATA	Record data options **X'20' (LRB_XRDATA_DELETE)** Delete the record data entry associated with the resource that is being RELEASEd. **X'04' (LRB_XRDATA_KEEP)** Keep the record data entry associated with the resource that is being RELEASEd.
LRB_XRETCODE	Return code from this request for this LRB.
LRB_XRSNCODE	Reason code from this request for this LRB.
LRB1_xADupReqSeqNum	Only when IXLLOCK REQVERSION value is greater than 0. Asynchronous duplexing request sequence number. A value of zero implies no asynchronous duplexing request sequence number was generated for the request. See the ADupReqSeqNum keyword on the IXLLOCK macro for more information.

Processing a Lock Request Block

Each lock request block is processed in the order in which it appears in the REQBUFFER area. If the system encounters an error while processing a resource request associated with a lock request block, processing for that request is halted with the appropriate return and reason codes and the system continues to the next LRB. If the system encounters an error attempting to access the next LRB, the entire PROCESSMULT request is halted.

If you specified the REQPROC keyword, the system returns in that area the number of lock request blocks that were processed before the PROCESSMULT request was halted when the request completes synchronously.

Determining the completion of a PROCESSMULT request

When control returns to the caller from a PROCESSMULT request, the request is complete and the result of its completion is indicated by the return and reason codes in RETCODE and RSNCODE. The PROCESSMULT request might have completed fully or partially, depending on whether all the LRBs in the buffer area were processed. The number of LRBs processed is returned in REQPROC when the PROCESSMULT request completes synchronously.

Examining PROCESSMULT Return and Reason Codes

Some examples of the types of status information returned from a PROCESSMULT request are:

- The request failed because the number of lock request blocks specified by REQNUM was not in the range of 1 to 128.
- The request was halted because a lock request block contained a request-type value that is not supported. The number of lock request blocks processed prior to the error is returned in the REQPROC field when the PROCESSMULT request completes synchronously.
- The request was halted because a lock request block contained a mode value that is not supported. The number of lock request blocks processed prior to the error is returned in the REQPROC field when the PROCESSMULT request completes synchronously.
- The request was halted because the system encountered an error while attempting to access storage in the area specified by REQBUFFER. The number of lock request blocks processed prior to the error is returned in the REQPROC field when the PROCESSMULT request completes synchronously.

Examining Lock Request Block Return and Reason Codes

The return and reason codes associated with the processing of each LRB are returned in LRB_XRETCODE and LRB_XRSNCODE. It is the caller's responsibility to examine each of these individually for each LRB processed to determine the outcome of the RELEASE request. The return code might indicate that the resource request is complete, or it might indicate that the request is being processed asynchronously (in which case the system will report the completion as specified with the LRB_XMODE value).

Some examples of the types of status information returned for a lock request block are:

- The request is being processed asynchronously. Results will be presented to your complete exit if you specified the LRB_XMODE value of LRB_XMODE_SYNCEXIT on the request.
- The request to release the resource is successful, possibly with one of the following exceptions:
  - You requested that the record data entry was to be kept, but there was no record data entry associated with the resource.
  - You requested that the record data entry was to be deleted, but the system was unable to delete the entry.

Using Exits for Coupling Facility Lock Services

The IXLLOCK services require the specification of three user exit routines — a complete, a contention, and a notify exit routine. These routines support the application's management of resource contention. The

exit routine names are specified at connect time. (You also must specify an event exit at connect time for any type of structure connection.)

General Requirements

The following requirements are common to the complete, contention, and notify exits:

Environment

The requirements at the time of exit invocation are:

Authorization:	Supervisor state, key 0
Dispatchable unit mode:	SRB
Cross memory mode:	PASN=HASN=SASN; PASN same as PASN at time of connect
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held

Input Specifications

On invocation of the IXLLOCK exits, (complete, contention, and notify), the general purpose registers (GPRs) contain:

Register
Contents

0
Does not contain any information for use by the exit.

1
Address of a fullword containing the address of the exit parameter list.

2-12
Does not contain any information for use by the exit.

13
Address of a 72-byte work area for use by the exit routine. The exit routine does not have to and restore registers in this work area. The exit routine can use this work area in any way it chooses.

14
Return address of XES.

15
Entry point address

When the exit receives control, the access registers (ARs) contain no information for use by the exit.

Return Specifications

At the completion of processing, the user exit must return to XES at the return address indicated by GPR 14 on entry. The user must restore all GPRs (and ARs, if necessary) prior to returning.

Serialization

Each of the IXLLOCK exits (complete, contention, and notify) is invoked and serialized on a resource basis by structure connection. Because of this serialization, it is not possible for more than one type exit to be running in parallel for the same resource related to the same connection. For example, if multiple complete exits are running in parallel on behalf of a specific connection to a lock structure, the exits are running on behalf of different resources within that structure.

Within a complete exit, MVS does not support the issuance of a request for the same resource that caused the complete exit to be called originally.

For example, assume that you request a resource in the shared state and the request is asynchronous. When processing for this request is complete, the system presents the event completion notification to your complete exit. In your complete exit, you specify another IXLLOCK request for the same resource and also specify that control is not to be returned until processing is complete. In this instance, the second request is lost because the complete exit has not completed the original request processing.

Ordering of IXLLOCK Exit Routines

When multiple exit routines are running on behalf of a resource request, they generally are scheduled in the order in which they were called. At times, XES may not schedule an exit until another currently executing exit completes.

XES guarantees the sequence of the exits will follow a predefined order.

Contention/Notify Exit Sequencing

In the contention exit, the connected user managing contention may indicate that the notify exit of a set of resource owners is to be scheduled. After all specified notify exits complete, the contention exit receives control again. Note that the results of notify exit processing are always presented to the contention exit except when the contention exit requests (through a return code) that it does not want control returned.

Complete/Notify Exit Sequencing

For an asynchronous request, XES guarantees that the complete exit processing to report resource ownership or a regrant will be completed before the notify exit is given control for the resource ownership. For example, if a request is processed asynchronously, and the contention exit issues both a grant and a request to run the granted user's notify exit, the notify exit is not given control until the complete exit processing for the grant has returned to XES.

Note that when the request is processed synchronously (and therefore the complete exit does not receive control), there is no guarantee as to which will occur first — control returns to the next sequential instruction or control passes to the notify exit. It is your responsibility to provide this type of serialization if required.

Exit Recovery

An error in an XES exit that prevents the exit from completing actions required by XES or the application can impact not only the connected instance that suffers the error but all instances that are dependent on the completed actions. Two examples of this type of error in an XES exit are:

- An event exit fails without providing a required event exit response. The sysplex-wide process that is dependent on that response hangs.
- A contention/notify pair of exits fails without completely updating the CEPL or fields appropriate to processing the request. The results are unpredictable and could possibly lead to integrity exposures as well as sysplex-wide hang waiting for lock resources.

Because of the potential sysplex-wide impact of these errors, XES ends any connector's task that returns to XES abnormally with a non-retryable X'026' abend. The reason code from the abend indicates which of your exits suffered the error, and diagnostic information is available to help diagnose the error.

For all XES exits, connectors should establish the appropriate recovery to handle errors.

Programming Considerations

In certain instances, XES must quiesce the activity of user exits in order to perform cleanup processing. The following illustrates scenarios where this processing occurs:

- Connection Termination

When a user disconnects or abnormally terminates, XES will force to completion any user exits executing on behalf of that user by issuing a PURGEDQ against the appropriate units of work. Note that if a connector terminates while a rebuild is in progress, any exits pertaining to both the original and the new structures will be forced to completion. In addition to forcing the currently executing user exits to completion, XES will also prevent any new invocations of these exits by cancelling any events that are pending presentation.

- Rebuild Stop

  When a connector provides an event exit response for the Rebuild Stop event, XES will force to completion any exits that are executing on behalf of that user's connection to the new structure by issuing a PURGEDQ against the appropriate units of work. Similar to connector termination processing, the user exits pertaining to the new structure will not be presented with any additional events. Note that any user exits executing on behalf of the original structure are unaffected by rebuild stop processing.

- Completion of a Rebuild

  When a connector provides an event exit response for the Rebuild Cleanup event, XES will force to completion any user exits that are executing on behalf of that user's connection to BOTH the original and the new structures by issuing a PURGEDQ against the appropriate units of work. No new events will be presented to the user exits on behalf of the original structure (as it is being discarded). Normal user exit processing will resume for the rebuilt structure upon completion of the rebuild process.

A user exit must be sensitive to conditions that can occur as a result of actions taken by XES and must be able to handle these as appropriate. For example, if a user exit has suspended itself, when the PURGEDQ is issued the system abends the user exit's unit of work with a retryable X'47B' abend and gives control to the user exit's recovery routine. (Note that although the recovery routine can retry, the user exit can not re-suspend itself because the system will fail any request to suspend a unit of work that has been the target of a PURGEDQ.) If the recovery routine percolates back to the system, its associated connection is terminated.

Avoiding Deadlocks

XES serialized connection recovery processing, such as that for cleaning up for a disconnected or failed peer connection, and rebuild processing serializes against mainline IXLLOCK and IXLSYNCH requests by obtaining internal latches. To avoid potential deadlocks, exploiters of IXLLOCK and IXLSYNCH requests should consider having a separate unit of work available or specify a MODE/MODEVAL with the request that allows control to be returned whenever XES encounters a delay. This would allow responses to required events to be provided in a timely manner.

In situations such as when a connector to a lock structure fails and a surviving connector's event exit receives the DISCFAILCONN event, but cannot provide the response for it until a previously issued IXLLOCK or IXLSYNCH request completes, a deadlock condition might be experienced when the request gets serialized behind an internal latch held by XES serialized connection recovery. The deadlock might result bacause the exploiter cannot respond to the required DISCFAILCONN event until the IXLLOCK or IXLSYNCH request completes, and XES cannot complete IXLLOCK or IXLSYNCH processing until the response to the DISCFAILCONN is received (allowing XES serialized connection recovery serialization to be released).

Performance Implications

Any action that delays the completion of a complete, contention, or notify exit will have an adverse effect on the performance of a connected user. For that reason, do not suspend processing in an exit without understanding the performance implications.

An IXLLOCK exit is considered complete when it returns to XES based on the address in GPR 14.

Coding a Complete Exit

The complete exit is used to inform you that your asynchronously-processed request is complete, or that your ownership status for a resource has been updated (regranted) by the contention exit of the connected user who has been chosen to manage the resource contention. You provide the address of your

complete exit using the COMPLETEEXIT parameter when you issue the IXLCONN macro to connect to the lock structure.

Information passed to the complete exit

When the complete exit receives control, it receives information about the event whose completion is being reported in the complete exit. The complete exit parameter list (CMPL), mapped by the IXLYCMPL macro, contains the following information:

CMPLCONTOKEN
The IXLLOCK invoker's connect token.

CMPLCONNAME
The IXLLOCK invoker's connect name.

CMPLREBUILD
Rebuild status of the target lock structure.

CMPLRNAME@
Resource name address.

CMPLRNAMELEN
Resource name length.

CMPLHASHVAL
Hash value

CMPLEVENT
Type of event whose completion is being reported. (See the constants for use with IXLLOCK events in IXLYCON.)

CMPLRETCODE
Return code from IXLLOCK request. Return code values are defined in the IXLYCON macro.

CMPLRSNCODE
Reason code from IXLLOCK request. Reason code values are defined in the IXLYCON macro.

CMPLLOCKDATA
Lock data that is associated with the owned resource, assigned at the time the IXLLOCK request to OBTAIN the resource was granted.

CMPLSTATE
Ownership state of the requested resource if the return code indicates a successful update; otherwise, the requested state, which may have been updated and therefore be different from the original request. (See IXLYCON for ownership state constants.)

CMPLUDATA
User data associated with the resource request if the return code indicates a successful update; otherwise, the requested user data including any updates made by the contention exit.

CMPLRDATA
Record data associated with the resource request, if the return code indicates a successful update; otherwise, the requested record data including any updates made by the contention exit.

CMPLRTENTRYID
Record data entry identifier if the return code indicates that the record data entry was successfully created or updated.

CMPLRTENTRYCOUNT
Number of record data entries that are currently in use for this lock structure if the return code indicates a successful update.

CMPLRDATAINFO
Flags for further information about record data options and validity of record data fields.

CMPLLOCKSECTIONVER
Version number of the lock section data.

CMPLLOCKSECTIONLEN
Length of the lock section data. Valid only when CmplLockSectionVer is equal to or greater than CmplLockVer1.

CMPL1ADUPREQSEQNUM
Asynchronous duplexing request sequence number assigned to the request if one was generated. Otherwise this field contains zero. See the ADupReqSeqNum keyword on the IXLLOCK macro for more information. Valid only when CmplLockSectionVer is equal to or greater than CmplLockVer1.

See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for the IXLYCMPL macro.

Return and Reason Codes

There are no return and reason codes from the complete exit. However, the CMPL contains the return code and reason code (if applicable) from the IXLLOCK request.

Programming Considerations

The IXLYCMPL macro provides the format of the complete exit parameter list. Include that macro as well as IXLYCON in your program.

When the complete exit returns to XES, you can no longer access the data in the CMPL.

You should be aware that in order to preserve the logical ordering of events, XES will not inform the user of other events related to this resource until it has received control back from the complete exit. Any processing that is to be performed by XES to inform the user of additional status of the subject resource (such as executing the notify exit, informing the user of asynchronous request completion through a subsequent invocation of the complete exit, or resuming a suspended work unit representing a synchronous request) will not be attempted until the complete exit returns control to XES. These interdependences with regard to the presentation of resource information to the user introduce the possibility of a user's protocol creating a deadlock scenario. For example, issuing a synchronous IXLLOCK request to alter a resource from within a complete exit that is executing on behalf of the same resource is one such case in which a deadlock could occur. (The complete exit will be suspended awaiting completion of the alter request, but XES will not be able to perform this processing until the exit returns control from the current invocation.

XES does not support the detection or resolution of deadlock scenarios. The prevention of such occurrences is the responsibility of the connected users of XES services.

Coding a Contention Exit

The purpose of the contention exit is to resolve contention based on your user-defined protocols. The contention exit receives as input a parameter list containing general information about the resource and the instance of the contention, as well as the resource request queue representing the current set of owners and waiters for the resource. The contention exit can use this information to take actions to resolve the contention. Such actions are:

- Grant pending requests with or without changes to the requested state, user data, or record data
- Deny pending requests
- Regrant an owned resource with a different state or user data
- Keep a pending request in a pending state
- Direct XES to run the notify exit of selected resource owners.

Assigning Resource Contention Management

XES chooses the contention exit of a connected user to manage resource contention in one of the following cases:

- When a resource request queue becomes incompatible

When a new request for a resource is received that is incompatible with the existing entries on the resource request queue, the queue is said to be “in contention”. In this case, XES selects one of the connected users to manage the contention and presents the current resource request queue (containing the new, pending request) to the contention exit of the selected user.

- When a previously selected resource contention manager fails.

  If a connected user disconnects or abnormally terminates while it is managing one or more resource request queues through its contention exit, XES, as part of its cleanup processing, reassigns management responsibilities to one of the surviving connectors.

The following diagram depicts XES processing when it receives a new request for a resource.

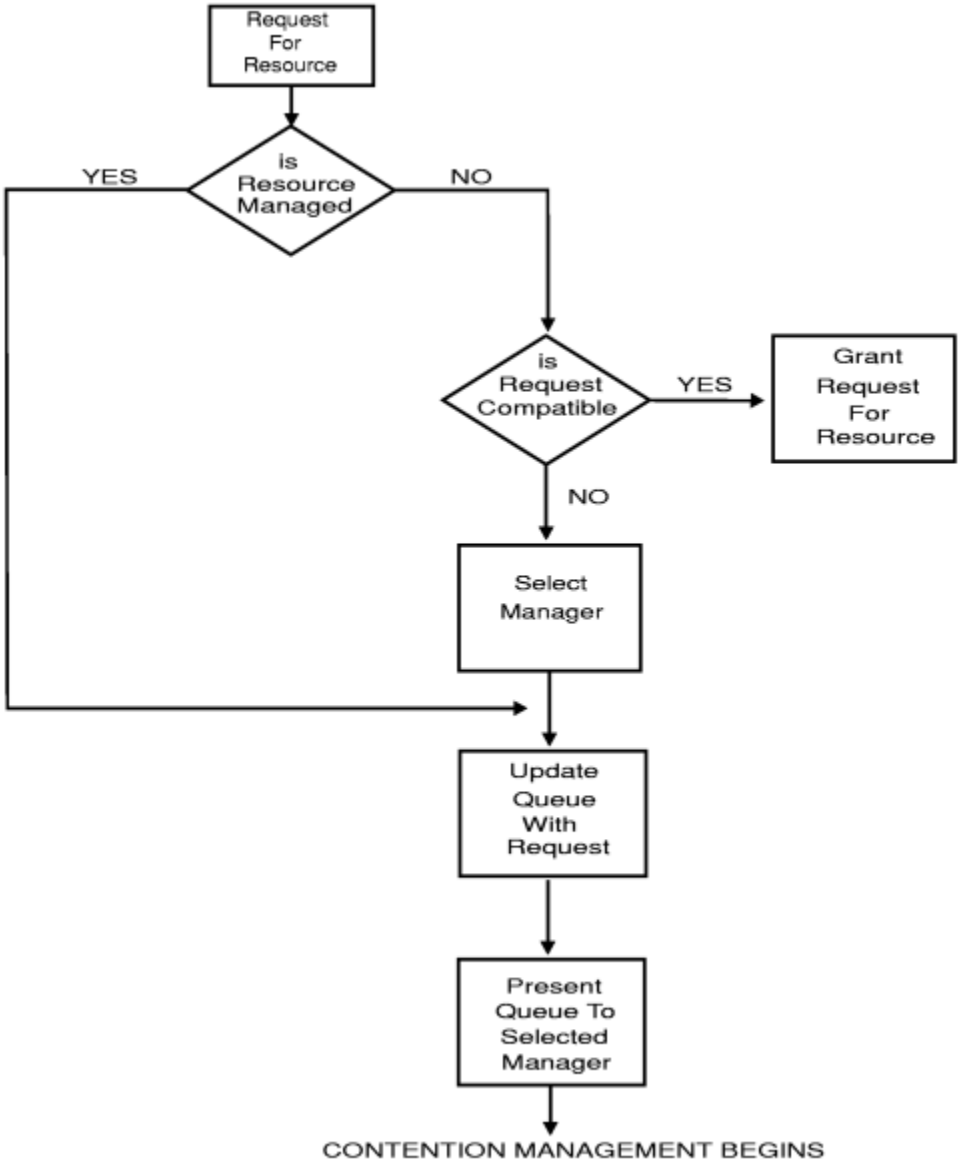


Figure 71. Receiving a Resource Request

The selected connector will continue to manage the resource through its contention exit until XES is able to determine that all entries on the resource request queue are once again compatible. When this determination is made, contention management ends and XES grants any pending requests remaining on

the resource request queue. The selected connector is no longer responsible for managing the resource. Should contention for the resource reappear, a different user might be selected to manage the resource.

Passing information to the contention exit

The contention exit receives control because a new request has been added to the resource request queue, because of previous actions of the contention exit, or in reaction to certain recovery processing (for example, when contention management responsibility is reassigned after the prior manager failed.) XES communicates with the contention exit through the contention exit parameter list — the CEPL, mapped by the macro IXLYCEPL. The CEPL that is passed to the contention exit contains an image of the resource request queue for the resource being managed, as well as summary information about the contention. Each current resource owner and waiter is reflected in the CEPL.

When the system first chooses a connected user to manage a resource that is in contention, a 32-byte work area, CEPLWORK, in the CEPL is set to zero. This work area is for the use of the managing connected user with any updates persisting across contention exit invocations until this instance of contention management has ended.

Each CEPL entry on the resource request queue also has an associated 32-byte work area, CEPLEWORK, that is shared with the notify exit of the connector represented by that entry. That is, if the notify exit of a connector is executed, the contents of the CEPLEWORK field from the requesting contention exit will be presented to the notify exit in the NEPL work area (NEPLWORK). Similarly, any updates made to the area by the notify exit will be added to the queue and presented to the contention exit. This provides the contention exit the ability to communicate with the notify exits of all the resource owners reflected on the request queue. When the resource request is first presented to the contention exit, the system sets this work area to zero. The contention exit can modify data in this work area and the data will be presented to subsequent invocations of the contention exit while contention for the resource continues to exist. (As with CEPLWORK, the CEPLEWORK work area persists across contention exit invocations.) This work area also appears in the notify exit parameter list (NEPL), if the notify exit is called. Any updates made to the work area during either notify exit processing or contention exit processing will be reflected in subsequent invocations of each exit while contention for the resource exists.

The CEPL contains a header section containing general information about the resource and the resource request and a section containing an entry for each request, either held or pending, for the resource. The CEPL information includes:

- Header information
  - Information about the resource in contention (**CEPLRNAME@**, **CEPLRNAMELEN**, **CEPLHASHVAL**)
  - Flags to indicate why the exit received control and whether the structure is being rebuilt (**CEPLREASONFLAGS**, **CEPLREBUILD**, **CEPLREBUILDORIG**)
  - A workarea for use by the contention exit (**CEPLWORK**)
  - A return code field by which the contention exit can communicate with XES (**CEPLRETCODE**)
  - Summary information about all entries on the request queue:
    - Total number of resource owners and waiters (**CEPLENT#**)
    - Number of new requests currently on the queue (**CEPLNEW#**)
    - Address of any new entry on the queue (**CEPLNEW@**)
    - Address of the first entry on the queue (**CEPLENT@**).
- Entry information. There is a CEPLENT entry for each resource request, both owned and pending. The CEPLENT data includes:
  - Information about the owner or waiter for the resource (**CEPLECONVERSION**, **CEPLECONID**, **CEPLECONNAME**)
  - Ownership status flag indicating whether resource is owned or pending (**CEPLESTATUSFLAGS**)
  - Action flag, to be set by the contention exit, to indicate to XES what action should be taken for the request when the contention exit completes (**CEPLEACTIONFLAGS**)
  - Owned state and user data for resources that are owned (**CEPLEHELD**)

- Requested state, user data, and record data for resources that are pending update (**CEPLEREQ**)
- Address of the next entry on the queue (**CEPLENEXT**)
- A workarea for use by both the contention and the notify exits (**CEPLEWORK**).

See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for the IXLYCEPL macro.

Contention Exit Processing

The contention exit can specify that the following actions be taken to resolve contention by setting indicators for one or more resource requests in the CEPL:

- Grant requests that are pending with or without changes to the requested state, user data, and/or record data.
- Deny requests that are pending.
- Regrant an owned resource with changed state, user data, record data.
- Keep a request pending.
- Direct XES to run the notify exit of selected users who own the resource.

If the contention exit returns to XES with a mixture of grants, regrants, or notify exit processing to be done, the system processes grants, regrants, and denys first, followed by notify exit processing.

The following describes what actions the contention exit can take.

- **Grant a request that is pending**

  The contention exit can grant any pending resource request by setting the grant indicator (CEPLEGRANT) in the corresponding CEPL entry. While granting the request, the contention exit might also change the ownership attributes originally requested by modifying the grant/regrant area in the CEPL entry.

  XES processes the grant request and any record data entry updates that might have been specified on the original IXLLOCK OBTAIN request. If XES is unable to grant the request (for example, it attempted to grant a request that required a record data entry to be created and there were no entries available), XES proceeds as follows:

  – XES returns failing return and reason codes to the issuer of the IXLLOCK request.
  – XES presents no new requests for the resource to the contention exit until:
    - All the requested actions (grants, regrants, denys) from the previous invocation of the exit have been processed.
    - The updated resource request queue is presented to the contention exit with the reason that an attempt to grant a request (as instructed by the previous invocation of the contention exit) has failed. The connector(s) whose request(s) have failed will be represented on the resource request queue with the ownership attributes that applied prior to the new request. This implies the following:
      - If the request that failed was an attempt to gain ownership of a resource that the requestor did not currently own (for example, an IXLLOCK OBTAIN request), then the updated resource request queue will not contain an entry for this connector. Any subsequent requests by this connector to alter or release the resource (including those that were outstanding at the time of failure) will be failed with return and reason codes indicating that the specified resource is not owned or pending ownership. These subsequent requests are not presented to the contention exit.
      - If the request that failed was an attempt by a requestor to update the ownership attributes of a resource that it currently owns (for example, an IXLLOCK ALTER request against an owned resource), then the connector will be represented on the resource request queue with the ownership attributes that applied prior to the failed attempt to update.
- **Deny a request that is pending**

The contention exit can deny a pending resource request by setting the deny indicator (CEPLEDENY) in the corresponding CEPL entry.

Note that the contention exit can not deny a request from a connector to release its ownership of a resource (IXLLOCK RELEASE). XES will ignore any attempt by a connector to deny this type of request.

- Denying a request for a resource that is not currently owned (IXLLOCK OBTAIN) results in the request being removed from the resource request queue. If the requestor had issued any subsequent requests to alter or release the resource whose ownership has been denied, XES fails the subsequent requests with return and reason codes indicating that the specified resource is not owned. These subsequent requests are not presented to the contention exit.
- Denying a request to update a currently owned resource (IXLLOCK ALTER) results in the corresponding update being cancelled. The entry remains on the resource request queue in its previous ownership state.

While a request being denied results in the requestor's ownership status not being updated as requested, any modifications made to the requested user data (CEPLEGUDATA) by the contention exit will be presented to the requestor as part of request completion. For instance, the contention exit may deny a request and also update the user data field to include a value indicating why the request was denied. The requestor, upon being informed of the request being denied, could potentially take some action based on the value in the changed user data.

- **Regrant an owned resource**

  The contention exit can regrant a resource by setting the regrant indicator (CEPLEREGRANT) in the appropriate CEPL. The state and/or user data can be changed on a regrant; record data cannot be updated on a regrant. XES allows only owned resources to be regranted. The connector whose ownership has been updated is informed through its complete exit.

  - Regranting a resource that is currently owned results in the state (CEPLEGSTATE) and/or user data (CEPLEGUDATA) being updated.
  - Regranting a resource that is owned and pending update results in the “owned” state and/or user data being updated, but the pending request remains unaffected. For example, the contention exit may encounter an entry which represents a connector as an owner of a resource with a pending request to update its ownership status (IXLLOCK ALTER). If the exit specifies to regrant the owner's interest in the resource, the owned status will be updated and the pending IXLLOCK ALTER request will remain pending.

  Special Note about the Regrant Function: Be aware that using the regrant function to downgrade a connector's ownership of a resource to be less restrictive introduces the possibility of an integrity exposure. This exposure could occur as the result of an asynchronous process (namely, the contention exit) modifying the serialization that was originally acquired by a connector to make an update without first informing that connection that its ownership status is being changed. Specifically, in the period between the time that the contention exit indicates to regrant the connector's ownership status and the time that the affected connector is able to be informed of the change (and subsequently take actions based on this), an update of a shared resource without the proper serialization could occur. If an application's locking protocol requires the contention exit to modify the attributes of owned resources in this manner, it should consider using the notify function.

- **Keep a request pending**

  The contention exit can choose neither to grant nor to deny a pending resource request. However, be aware of the following when leaving a request pending on the queue:

  - The request can be superseded (replaced on the resource request queue) by a more recent request from the connector.
  - The request will be granted when the resource request queue ceases to be in contention.
  - The request might also be granted by actions taken on a subsequent invocation of the contention exit.

- **Direct XES to run notify exits**

  The contention exit can direct XES to run the notify exit of selected resource owners by setting the invoke notify exit indicator (CEPLENOTIFY) in the appropriate CEPL entry. Keep in mind that XES

processes requests to grant, regrant, and deny requests before it handles requests to run the notify exits of selected users. Therefore, the resource request queue presented to the notify exits (in the NEPL) will contain any updates resulting from these prior actions. If a failure occurs while performing one of these prior actions (for instance, a grant fails), XES returns to the contention exit to report the failure and cancels the requests to run any notify exits.

When directed to run the notify exit of selected users, XES proceeds as follows:

– The resource request queue (in the form of the notify exit parameter list, NEPL) is presented to the notify exits of owners specified by the managing user's contention exit.
– No new requests for the resource are processed until:
  - All the indicated notify exits have completed processing,
  - XES updates the resource request queue to reflect the changes (if any) made in the notify exits and presents them to the contention exit, and
  - The contention exit indicates that no more notify exits are to be given control.

    Note that if the contention exit indicates that notify exits are again to be given control, then processing resumes with notify exit scheduling for the indicated notify exits.

XES/Contention Exit Communication

The contention exit communicates with XES through a return code. The exit can specify that:

- It has completed normally and contention management should continue.
- It should not be called at the completion of notify exit processing if contention has ceased.
- It should be called again immediately after XES has updated the resource request queue.
- It should not be called again until structure rebuild processing has completed.

See Table 48 on page 668 for a description of these return codes.

Continue Contention Management

Subsequent to its first invocation of the contention exit, XES may invoke the exit at each of the following times while the connector remains the contention manager:

- **A new request is received for the resource while contention exists**

  Once the entries on a resource request queue have been determined to be incompatible and as long as the incompatible condition exists, any new request for the resource causes the contention exit to be given control. Note that XES determines the state of the resource request queue after all actions specified during the previous invocation of the contention exit have been performed.
- **The completion of notify exits**

  When a resource request queue is presented to the contention exit, the connected user that is managing the contention can indicate that the notify exit of resource owners be scheduled for processing. After all specified notify exits have completed, the contention exit is again given control. The resource request queue reflects the changes, if any, that connected users made to their ownership of the resource in the notify exit.

  Note that the contention exit may supply a return code specifying that if the results of notify exit processing cause contention to cease, XES is not to redrive the contention exit.
- **Failure of a previous grant request**

  If XES is unable to grant a pending request as specified by the contention exit, XES redrives the contention exit with an indication in the CEPL of the failure. Among the reasons for the failure to grant the request are an associated record data entry operation which could not be completed or a loss of connectivity to the lock structure.

  The resource request queue presented to the contention exit reflects all updates made in the previous invocation of the exit, but may or may not contain an entry for the user whose request has failed.

- If the failed request was for a resource that was not previously owned, the updated resource request queue will not contain an entry for the failed request.
- If the failed request was to update (ALTER) an already owned resource, the updated resource request queue will contain an entry for the resource that reflects the ownership state of the resource before the failed attempt to update it.

Note: Any requests to execute notify exits that were made during the invocation of the contention exit which specified to grant a request are cancelled.

- **During recovery processing for a failed connection**

  XES removes entries representing a failed or disconnected user from all resource request queues as part of its cleanup processing. If cleanup occurs for a request queue that is being managed by a surviving connected user then the updated request queue is presented to the contention exit of that user with an indication that recovery has occurred.

  Note that XES will not interrupt the normal processing flow to inform the contention exit that a failure has occurred. For example, if a contention exit is waiting for responses from notify exits to complete at the time that cleanup is being performed on the resource request queue then the contention exit will be informed of the failure at the time that it is invoked with the results of the notify exits. In this case, the contention exit parameter list will indicate that the queue reflects results of notify exit processing, as well as recovery actions.

  In summary, the contention exit may be informed of a change in the resource request queue due to failures during any invocation. This includes when the exit is called for normal processing such as presentation of new requests and results of notify exit processing, as well as through a separate invocation when the failure represents the only change in the queue. If an application's protocol calls for its contention exit to be sensitive to failures of related users, the exit should check the failure indicator (CEPLRECOVERY) during each invocation of the exit.

  Note that the CEPLREASONFLAGS indicate why the contention exit has been given control. The CEPLNOTIFYRESPONSE, CEPLGRANTFAILED, and CEPLRESTARTAFTERDEFER flags are mutually exclusive. However, it is possible for the CEPLRECOVERY flag to be set to ON in conjunction with one of the other CEPLREASONFLAGS.

Stop Contention Management

The contention exit can specify that management of the resource is to stop once contention no longer exists. If any notify exits are scheduled, and they complete with no actions to be taken because contention has ceased to exist, do not call the contention exit. When contention again occurs, another connected user might be chosen to manage contention.

Call Contention Exit Again

The contention exit may wish to examine the resource request queue again immediately after XES has updated it according the actions specified in the exit. Normally, XES would not present the resource request queue again until a new request was received. The contention exit can use the IXLRCCONTEXITCALLAGAIN return code to request that it be called again immediately.

When the contention exit is called again, all of the CEPLREASONFLAGs are OFF and CEPLNEW# is zero. The resource request queue will reflect any changes that were specified in the previous invocation of the exit. For example, if you had set the CEPLENOTIFY bit in the previous contention exit invocation, its action would have been processed and you would be notified of the result in this invocation of the exit. If the CEPLRECOVERY bit had been set to ON in the initial invocation of the contention exit, it will be OFF when the contention exit is called again, unless a new recovery event has taken place.

Defer Contention Management during Rebuild

A contention exit managing a resource request queue can request that contention processing be quiesced until the structure has either completed rebuilding or the rebuild process has been stopped. If the structure is in the rebuild process (either the CEPLREBUILD or the CEPLREBUILDORIG bit will be set ON), the contention exit can set the IXLRCCONTEXITREBUILDDEFER return code to specify that XES should

not invoke the contention exit again for this resource until rebuild processing has completed. Requests for the resource continue to be queued and will be handled when the rebuild process ends.

If rebuild is not in progress (neither CEPLREBUILD nor CEPLREBUILDORIG is ON) and the contention exit returns to XES with the IXLRCCONTEXITREBUILDDEFER return code, XES issues an abend dump and terminates the connection.

Whether the contention exit actually is restarted depends on whether the contention exit is processing on behalf of the original structure or the new structure, and on the outcome of the structure rebuild.

- If the contention exit is processing on behalf of the new structure (CEPLREBUILD=ON), the contention exit will be restarted when rebuild processing is complete. That is, after the connector has responded to the rebuild cleanup event with IXLEERSP EVENT=REBLDCLEANUP.
- If the contention exit is processing on behalf of the original structure (CEPLREBUILDORIG=ON), the contention exit will be restarted only if the rebuild processing is stopped. That is, after the connector has issued IXLEERSP EVENT=REBLDSTOP.

At the completion of rebuild processing that was deferred with the IXLRCCONTEXITREBUILDDEFER return code, the contention exit will be restarted with the CEPLRESTARTAFTERDEFER bit set ON to indicate that this is the initial invocation of the contention exit for this resource after its deferral for rebuild processing.

When the contention exit is restarted, the contents of the CEPL will be identical to its previous contents when the request to defer processing was specified. During the previous invocation of the contention exit, if updates to the CEPL work areas were made, those updates are preserved.

A connector who was reflected on the resource request queue of the previous invocation of the contention exit might no longer be represented on that queue. This situation could occur because the connector failed or disconnected. The CEPLRECOVERY flag is set ON to indicate that the failed or disconnected user has been removed, if cleanup has completed.

Summary of XES Contention Exit Processing

In order to make full use of the capabilities provided through the contention exit, it is important that you understand the processing performed by XES to carry out these actions. The following schematic depicts this processing:

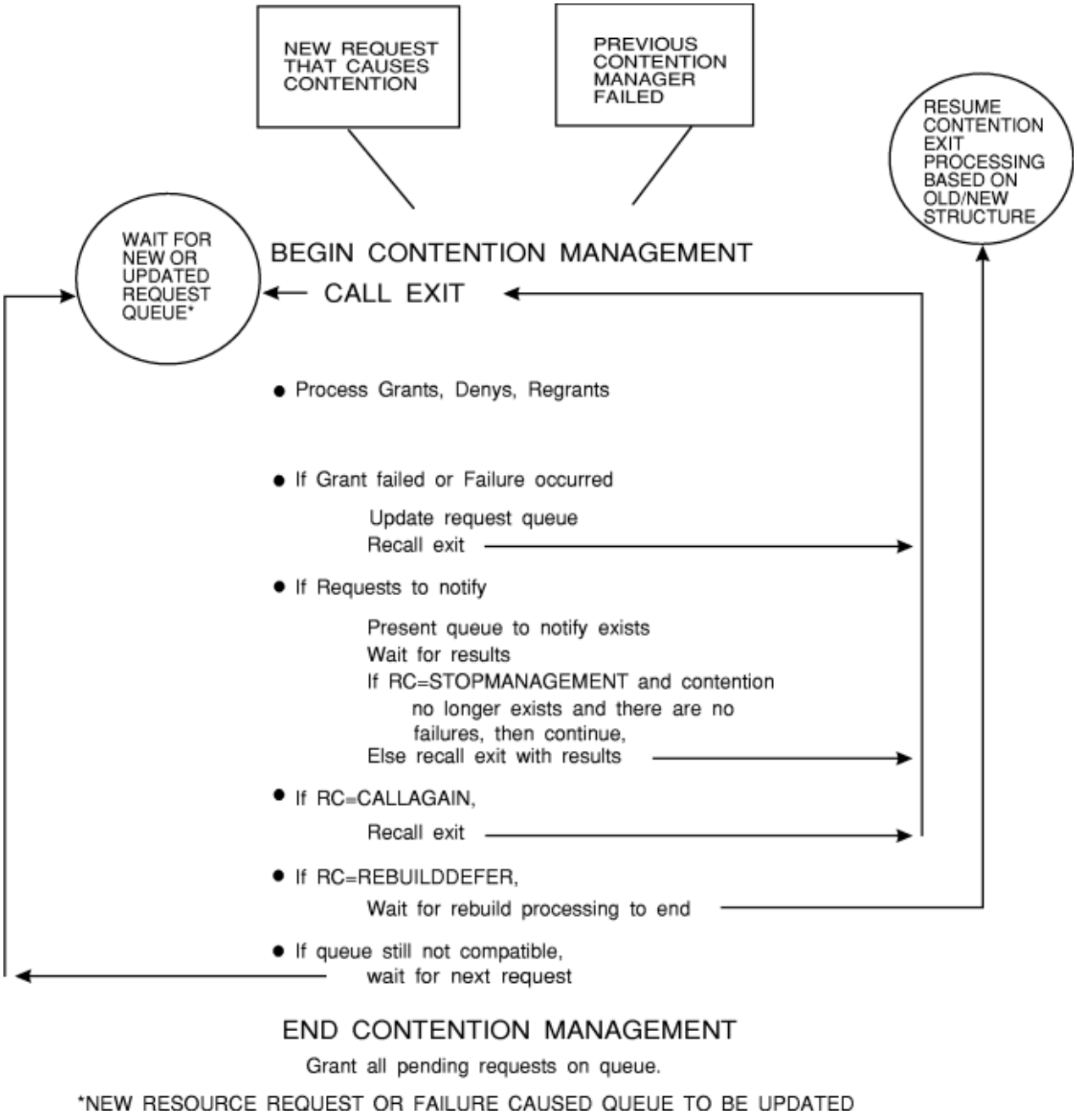


Figure 72. Contention Exit Processing

Return and Reason Codes

When the contention exit returns control to the system, *ceplretcode* contains a return code.

The following table identifies return codes from a contention exit, tells what each means and the actions that XES should take.

Table 48. Return Codes for the Contention Exit	
Hexadecimal Return Code	**Meaning and Action**
0	**Equate Symbol**: IXLRCCONTEXITCONTINUEMANAGEMENT **Meaning**: Contention exit complete. **Action**: Continue normal management of the resource. If notify exits are to be scheduled, call the contention exit after all notify exits complete.
4	**Equate Symbol**: IXLRCCONTEXITSTOPMANAGEMENT **Meaning**: Contention exit complete. **Action**: Terminate management (assuming that no contention exists). If any notify exits are to be scheduled, *do not* call the contention exit after all notify exits complete if actions taken by the notify exit cause the resource to no longer be in contention.
8	**Equate Symbol**: IXLRCCONTEXITCALLAGAIN **Meaning**: Invoke the contention exit again with the resource request queue updated to reflect the actions specified. **Action**: Invoke the exit again without waiting for the arrival of a new request.
C	**Equate Symbol**: IXLRCCONTEXITREBUILDDEFER **Meaning**: Do not invoke the contention exit again for this resource until structure rebuild processing has completed. **Action**: Restart the contention exit either after this connector has responded to the REBLDCLEANUP event (for a new structure) or the REBLDSTOP event (for the old structure).

Programming Considerations

The IXLYCEPL macro provides the format of the contention exit parameter list. Include that macro as well as IXLYCON in your program.

When the contention exit returns to XES, you can no longer access the data in the CEPL.

The contention exit cannot consider that a request that it has granted is processed to completion until the system informs you of the result of your request. In general, the contention exit should neither make any assumptions about the success of its grants of resource requests nor keep a local image of the resource request queue.

You should be aware that if the contention exit indicates that the notify exits of resource owners are to be executed, the contention exit will not be invoked again until XES has been able to execute the specified notify exits and apply the results to the resource request queue. This dependency introduces the possibility of a user's protocol creating a deadlock condition. For example, issuing a synchronous IXLLOCK request to release a resource from within a notify exit that is executing on behalf of the same resource is one such case in which a deadlock could occur. (The contention exit will not be able to receive control again to process the request until the current set of notify exits (which are suspended awaiting completion of the request) are able to complete.)

XES does not support the detection or resolution of deadlock scenarios. The prevention of such occurrences is the responsibility of the connected users of XES services.

Coding a Notify Exit

The notify exit is the method by which resource owners are made aware that contention exists for the resource that they own. The notify exit is given control at the request of the contention exit. The connected user that is managing the resource in contention modifies the contention exit parameter list (CEPL) in the contention exit to indicate the connectors whose notify exits are to be called. Only resource owners are eligible to be notified.

The notify exit allows the resource owner to modify one or more of the following attributes of the owned resource for which the exit has been called:

- The ownership state of the resource

  The modification can be to change the state from shared to exclusive, from exclusive to shared, or relinquish ownership.
- The user data associated with the resource
- The record data associated with the resource.

XES/Notify Exit Communication

Similar to the contention exit, the notify exit receives the current resource request queue as input. The queue is presented in the form of a notify exit parameter list (NEPL), mapped by the macro IXLYNEPL. The NEPL for locking requests has a header that contains information pertaining to the connector and the lock structure along with a lock section that includes the identification of the resource that is in contention. These sections are followed by a series of entries (mapped by NEPLENT) that reflect the interest of other connectors (both owners and waiters) in the specified resource. The NEPL information includes:

- Header section information
  - Connect data, such as the connect token and connect name of the resource owner (**NEPLCONTOKEN**, **NEPLCONNAME**)
  - Structure information, such as rebuild status (**NEPLREBUILD**)
- Lock section
  - Lock data, if any, that was specified when resource ownership was obtained (**NEPLLOCKDATA**)
  - Resource identifiers, such as resource name, length, and hash value (**NEPLRNAME@**, **NEPLRNAMELEN**, **NEPLHASHVAL**)
  - 32-byte work area, passed from the contention exit (**NEPLWORK**)

    This work area which is shared between the contention exit and the notify exit. Specifically, when the contention exit requests that the notify exit be executed, the contents of the work area within the corresponding CEPL entry (CEPLEWORK) is passed to the notify exit. The notify exit receives this information by way of the NEPLWORK field. Any changes made to this area by the notify exit are subsequently presented to the contention exit.
  - Ownership data, such as state and user data for this resource owner (**NEPLSTATE**, **NEPLUDATA**)
  - An input/output area, which can be used in conjunction with the IXLSYNCH service to update the ownership data (**NEPLOUT**).
  - A version number of the lock section (**NEPLLOCKVERSION**)
  - An area used in conjunction with the IXLSYNCH service to provide an asynchronous duplexing request sequence number (**NEPL1ADUPREQSEQNUM)**. Valid only when the lock section version number is at least **NeplLockVer1**.
- Entry information
  - Requester information, such as version, connection identifier, and connect name (**NEPLECONVERSION**, **NEPLECONID**, **NEPLECONNAME**)
  - Ownership data, such as state and user data, for both the held state and the requested state (**NEPLEHELD**, **NEPLEREQ**).

See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for the IXLYNEPL macro.

Notify Exit Routine Processing

Within the notify exit, you can modify the NEPL to indicate the action to be taken. To commit the changes to the NEPL, you must call the IXLSYNCH service from the notify exit. The IXLSYNCH service provides a synchronous update of the state and/or user data associated with a resource. If you modify the NEPL and do not call the IXLSYNCH service, any changes made in the NEPL are ignored. When all notify exits are complete, the contention exit is given control with a CEPL reflecting all changes made in the notify exits.

Changes that the notify exit might make in the NEPL are:

- **Update state and/or user data**

  The notify exit can update the state data through the NEPLOSTATE field and then issuing the IXLSYNCH macro. Similarly, the user data can be updated through the NEPLOUDATA field. Updates to these areas will be reflected in the CEPL fields CEPLEHSTATE and CEPLEHUDATA passed to the contention exit.

- **Release ownership of the resource**

  The notify exit can release ownership of the resource by setting the NEPLOSTATE field to the value (in IXLYCON) meaning “free” (IXLSTATEFREE) and then issuing the IXLSYNCH macro. When the contention exit receives the updated information in the CEPL, both CEPLEHSTATE and CEPLERSTATE will be set to “free”. This avoids the confusion that might arise with a request to obtain a resource, where CEPLEHSTATE is set to “free” and CEPLERSTATE is shared or exclusive.

  When a user relinquishes interest in a resource within the notify exit, any outstanding requests from the user to alter or release the resource will fail with an indication that the specified resource is not found.

Upon completion of each invoked notify exit, XES gathers the ownership information that corresponds to the resource that caused the notify exit to be given control and applies the changes to the resource request queue. When the results of all invoked notify exits are available, XES invokes the managing contention exit and passes to it the modified resource request queue.

Note that a connector can be notified multiple times by the contention exit that is managing the resource contention. This will occur whenever the contention exit requests that notify exits again be given control after the results of the previous set of notify exits are presented to the contention exit. This ability to communicate allows multiple users to negotiate for lock ownership.

Return and Reason Codes

None.

Programming Considerations

The IXLYNEPL mapping macro provides the format of the notify exit parameter list. Include that macro as well as IXLYCON in your program.

When the notify exit returns to XES, you can no longer access the data in the NEPL.

The IXLSYNCH service is the only IXL-type service that should be issued in the notify exit on behalf of the resource that caused the Notify exit to be called. Other IXL-requests that are issued on behalf of the same resource might not complete.

Using the Synchronous Update Service (IXLSYNCH)

The IXLSYNCH service allows you to modify information about your ownership of a resource while running in your notify exit. The following attributes can be modified:

- The ownership state of the resource

  The modification can be to change the state from shared to exclusive, from exclusive to shared, or relinquish ownership.

- The user data associated with the resource
- The record data associated with the resource.

Notify Exit/IXLSYNCH Communication

The notify exit passes the address of the NEPL to IXLSYNCH. The address passed must be of the actual NEPL originally passed to the notify exit and not a copy of it.

If you do not provide a valid NEPL address, there is a possibility that XES will terminate abnormally while processing the request. You should establish recovery procedures for this circumstance.

Addressing the Notify Exit Parameter List

The system schedules a notify exit so that an owner of a resource for which there is contention can modify information about that ownership. The data to be modified is contained in the notify exit parameter list (NEPL), the address of which is passed to the connected user's notify exit. Changes to the information in the NEPL are recognized by the system only if the IXLSYNCH service is used to record those changes.

Processing the Modifications

During notify exit processing, you may either release its ownership of a resource or change its corresponding state, user data, or record data. Once the changes have been made in the NEPL, you issue IXLSYNCH to notify the system.

The effects of issuing IXLSYNCH for a modification to record data are as follows:

- **Update the record data associated with a resource**

  When IXLSYNCH is issued and the NEPLORTWRITE flag is set to ON (write record data), XES performs the following update:

  – If a record data element is currently associated with the resource, then its contents will be updated.
  – If there is not a record data element currently associated with the resource, then XES attempts to write to an available record data entry. If an available record data entry cannot be found, XES rejects the request and provides error return and reason codes to the requestor.

  When a structure resides in a CFLEVEL=25 CF or higher, you can use the IXLCONN PCTENTRYRSV to specify the percentage of the total number of record data entries that are to remain free (reserved) after completion of a request that attempts to create a record data entry. By setting aside a percentage of record data entries, you can better manage the availability of record data entries. Using the NEPLORTWRITE and NEPLORTALLOWUSERSV flags (set ON), you can access the reserved entries and create a record data entry during scenarios when the creation of a record data entry is deemed necessary, as long as there is a free record data entry in the structure.

- **Delete the record data associated with a resource**

  When IXLSYNCH is issued and the NEPLORTDELETE flag is set to ON (delete record data), XES attempts to delete the record data element currently associated with the resource. If a record data element does not exist for the resource, XES ignores the request.

Informing a User of Request Completion

A user requesting access to a resource must allow for the possibility that factors might exist that could prevent the request from being satisfied immediately. The reasons why request processing might experience delays include conditions that are not controllable by the connected user, such as internal XES serialization that could not be immediately obtained.

XES processes IXLSYNCH requests synchronously. A request is defined as synchronous when processing for the request is complete when control returns to the next sequential instruction following the request.

Using the IXLSYNCH MODEVAL Parameter

There might be times when the system is not able to process your IXLSYNCH immediately, such as when the system could not obtain its internal latches needed to process the request.

You can specify how you want the system to process your request if it cannot be serviced immediately by using the MODEVAL parameter on IXLSYNCH. **If the request can be processed immediately, the MODEVAL parameter is ignored.** The following are valid MODEVAL specifications for an IXLSYNCH request:

IXLMODESYNCSUSPEND
Specifies that you do not want control returned until processing for the request is complete. If request processing is delayed because of needed internal latches, you will be suspended until the request

completes. You will receive control at the next sequential instruction with the request complete and the final disposition determined.

IXLMODESYNCFAIL
Specifies that if the system cannot process your request without a delay, the request is to be cancelled. You will receive return and reason codes indicating that disposition of your request.

The constant values that are valid for MODEVAL are defined in IXLYCON. If you specify a value for MODEVAL other than one of the IXLYCON constants that is valid for a particular request type, the system fails the IXLSYNCH request.

Using the Lock Cleanup and Recovery Service (IXLRT)

As part of the IXLLOCK interface, connected users to a lock structure can specify 64 bytes of record data to be written in the lock structure. This record data can contain information about the associated resource and its ownership state so that the resource can be recovered in the event of the owning user's failure. The record data entry can also have an associated record data type which can be used to identify the type of data being recorded.

The IXLRT service is the recovery interface to obtain or clean up this recording information in a lock structure. The recording information is associated with a connected locking user. Use the IXLRT macro to request the following:

- Create a record data entry and write data to the entry. Optionally, you can assign a record data type to the newly created record data entry.
- Read the entire set of record data entries in the lock structure or those associated with a particular record data type.
- Read the entire set of record data entries associated with a particular connected user or those associated with both a particular connected user and a particular record data type.
- Read a particular record data entry by entry identifier.
- Delete all record data entries identified by a list of entry identifiers.
- Delete a particular record data entry by entry identifier.
- Delete all record data entries associated with a particular connected user or those associated with both a particular connected user and a particular record data type.
- Update a record data entry by entry identifier and optionally, assign a record data type to the updated record data entry.

Identifying the User

XES recognizes a valid connection to a lock structure through the connect token (CONTOKEN), which the system returns after the successful completion of the IXLCONN service. To use the IXLRT service, you must specify your valid connect token on every IXLRT request.

If you do not provide a valid CONTOKEN, there is a possibility that XES will terminate abnormally while processing the request. You should establish recovery procedures for this circumstance.

Providing areas for returned data

You must provide areas into which the system places the information requested with IXLRT.

- The answer area (ANSAREA) contains information about the data returned. The contents of the answer area are mapped by the IXLYRTAA macro. For asynchronous system-managed duplexing, the size of the answer area should be large enough to hold level 1 data (mapped by RTAA1) to allow an asynchronous duplexing request sequence number to be returned in Rtaa1ADupReqSeqNum when applicable.
- The data area (DATAREA) contains the record data entry information requested. The contents of the data area are mapped by the IXLYMRTD macro.

Specifying the Level of Information

The IXLYMRTD macro supports several levels of information that IXLRT returns. Certain IXLRT requests may provide data that was not returned when the IXLRT service was first made available. For these request types, you can specify the level of information you want with the MRTDLEVEL parameter on IXLRT. The MRTDLEVEL parameter is available with version 4 of the IXLRT macro. The system returns base MRTD information when you specify or default to MRTDLEVEL=0 on your request; the system returns level-1 MRTD information when you specify or default to MRTDLEVEL=1 on your request. You should be aware of the type of output that you are requesting and be able to process it correctly. IBM recommends that you use the level-1 level of IXLYMRTD in case additional new data is returned by the IXLRT service. Note that the level-1 IXLYMRTD records are larger than the level-0 IXLYMRTD records.

See the IXLYMRTD macro in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for a description of the information returned.

Identifying the Record Data

When a record data entry is created, the system identifies the entry with an entry identifier (ENTRYID). Each record data entry can be referenced by its unique 12-character entry identifier.

If the entry identifier is known, the IXLRT service can be used to either read or delete a record data entry.

Assigning a Record Data Type to the Record Data

When a record data entry is created, if the system supports the specification of the type of record data contained in that field, RDATATYPE identifies the value of the record data type. If omitted, the default value for RDATATYPE is zero.

What You Can Request with IXLRT

The following options are available with the IXLRT service:

- **Creating a Record Data Entry**

  You can use the IXLRT service to create a record data entry in a lock structure by specifying the CREATENTRY keyword. You specify the record data to be written with the RDATAVAL keyword. XES attempts to write the 64 bytes of record data to an available record data entry in the lock structure. If there are no available record data entries, an error return code is provided.

  When a structure resides in a CFLEVEL=25 CF or higher, you can use the IXLCONN PCTENTRYRSV to specify the percentage of the total number of record data entries that are to remain free (reserved) after completion of a request that attempts to create a record data entry. Using the ALLOWUSERSV keyword, you can access the reserved entries and create a record data entry during scenarios when the creation of a record data entry is deemed necessary, as long as there is a free record data entry in the structure. For more information, see IXLRT – lock structure record data processing in *z/OS MVS Programming: Sysplex Services Reference*.

  When creating a record data entry, you optionally can request that the record data entry be associated with another connected user. Use the TCONNAME keyword to identify the target connection name of the user with which to associate the record data entry. If you do not specify a target connection name, XES associates the record data entry with the user identified by the connect token provided to IXLRT.

  Optionally, when creating a record data entry, you can assign a record data type to the entry by specifying the RDATATYPE keyword. If omitted, the default value for RDATATYPE is zero.

  When the request to create a record data entry completes successfully, XES returns the following:

  – The unique entry identifier (in the area specified by ENTRYID)
  – The number of entries associated with the target connection (in the area specified by ANSAREA)
  – The total number of allocated entries in the record list (in the area specified by ANSAREA).

Note that using the CREATENTRY option is not the normal way in which to allocate a record data entry. The record data entry that is created is not associated with an IXLLOCK OBTAIN or ALTER request, nor with the resource ownership resulting from such a request.

- **Reading All Record Data Entries in a Lock Structure**

  You can use the IXLRT service to read all record data entries in a lock structure by specifying the READALL keyword. You must provide a 4096 byte area, identified by the DATAREA keyword, to contain the output.

  Optionally, you can limit the READALL request to read all record data entries of a particular record data type by specifying the RDATATYPE keyword to identify the type of record data to be read.

  When the READALL request completes successfully, XES returns the following:

  – Depending on the level of IXLYMRTD requested, one of the following is returned in the area specified by DATAREA:

    - If MRDTLEVEL=0 is specified or defaulted to, an array of records consisting of the 64 bytes of record data, the corresponding entry identifier, and the connection identifier of the associated connected user. The format of this area is mapped by the MRTD macro.
    - If MRDTLEVEL=1 is specified, the same information as above plus the type of record data. The format of this area is mapped by the MRTD1 macro.

  – The number of record data entries read (in the area specified by ANSAREA).

  If the request to read all entries completes prematurely, XES provides return and reason codes to the requestor. XES also returns the following:

  – The number of reliable record data entries (in the area specified by ANSAREA)

  – A restart token (in the location specified by the RESTOKEN or EXTRESTOKEN keyword), which you can use as input on subsequent requests in order to continue processing with the next record data entry. You should issue the request repeatedly until you receive a return code indicating that there are no more entries to read.

- **Reading All Entries in a Lock Structure Associated with a User**

  You can use the IXLRT service to read all record data entries in a lock structure associated with a particular connected user by specifying the READBYCONN keyword. You must provide a 4096 byte area, identified by the DATAREA keyword, to contain the output.

  You can request that the record data entries associated with another connected user be returned to you. Use the TCONNAME keyword to identify the target connection name of the user whose record data entries are to be returned. If you do not specify a target connection name, XES returns the record data entries associated with the user identified by the connect token provided to IXLRT. You can specify on a READBYCONN request whether the system is to process the request using an optimized access method by specifying the FASTPATH keyword. If using the optimized access method, you must have serialization that ensures that the record data entries that belong to the target connection remain unchanged throughout the IXLRT process. For example, you can use FASTPATH=YES when reading record data belonging to a failing or failed connector because those entries will remain unchanged.

  Optionally, you can limit the READBYCONN request to read those record data entries associated both with a particular connected user and with a particular record data type by specifying the RDATATYPE keyword to identify the record data entries to be read.

  When the READBYCONN request completes successfully, XES returns the following:

  – Depending on the level of IXLYMRTD requested, one of the following is returned in the area specified by DATAREA:

    - If MRDTLEVEL=0 is specified or defaulted to, an array of records consisting of the 64 bytes of record data, the corresponding entry identifier, and the connection identifier of the associated connected user. The format of this area is mapped by the MRTD macro.
    - If MRDTLEVEL=1 is specified, the same information as above plus the type of record data. The format of this area is mapped by the MRTD1 macro.

– The number of record data entries read (in the area specified by ANSAREA).

If the request to read all entries completes prematurely, XES provides return and reason codes to the requestor. XES also returns the following:

– The number of reliable record data entries (in the area specified by ANSAREA)

– A restart token (in the location specified by the RESTOKEN or EXTRESTOKEN keyword), which you can use as input on subsequent requests in order to continue processing with the next record data entry. You should issue the request repeatedly until you receive a return code indicating that there are no more entries to read.

- **Reading an Existing Record Data Entry**

  You can use the IXLRT service to read an existing record data entry by specifying the READENTRY keyword. Use the ENTRYID keyword to specify the entry identifier of the record data entry to be read.

  When reading an existing record data entry, you optionally can specify a verification connection name. XES will verify that the record data entry is associated with the verification connection name before the request is attempted. Use the VERCONNAME keyword to identify the verification connection name of the user with which to associate the record data entry. If the record data entry is not associated with the verification connection name, XES cancels the request and provides an error return code to the requestor. If you do not specify a verification connection name, XES attempts the read without verification.

  With the version 4 level of IXLRT, you can also request that the system return the record data type associated with the record data element.

  When the request to read a record data entry completes successfully, XES returns the following:

  – A 64 byte output area containing the contents of the record data entry requested (in the area specified by RDATAVAL)

  – If applicable, an 8-byte output area containing the record data type value associated with the record data element (in the area specified by OUTRDATATYPE)

  – The total number of entries associated with the user whose entry was read (in the area specified by ANSAREA)

  – The total number of allocated entries in the record list (in the area specified by ANSAREA).

  If the record data entry specified by ENTRYID is not allocated, XES provides an error return code to the requestor.

- **Deleting All Entries by a List of Entry Identifiers**

  You can use the IXLRT service to delete a set of record data entries by specifying the DELETENTRYLST keyword. Use the ENTRYIDLIST keyword to specify the 4096 byte area containing a list of entry identifiers corresponding to the record data entries to be deleted. Specify the list as 12-byte elements starting at offset zero in the area.

  You also must use the FIRSTELEM and LASTELEM keywords to specify the range of entry identifiers within the ENTRYIDLIST. FIRSTELEM specifies the first element in the list of entry identifiers to be processed; LASTELEM specifies the index of the last element in the list to be processed. The value of either index must be in the range of 1 to 341, inclusive. The value of LASTELEM must be greater than or equal to the value of FIRSTELEM.

  If an element specified in the entry identifier list does not exist, XES halts processing and returns the index of the non-existent element to the connected user in the RTAA. To continue processing the list, the connected user can update the FIRSTELEM keyword with the incremented index of the starting point in the entry identifier list and reissue the DELETENTRYLIST request.

  If the request to delete a set of entries completes prematurely, XES provides return and reason codes to the requestor. XES also returns the followng:

  – The number of entries deleted thus far (in the area specified by ANSAREA)

– The value of the first unprocessed ENTRYIDLIST index (in the area specified by ANSAREA). You can use this value for FIRSTELEM when reissuing the request to delete the remaining unprocessed elements in the entryid list.

When the request to delete a set of record data entries completes successfully, XES returns the following:

– The number of entries deleted by this request (in the area specified by ANSAREA).

- **Deleting an Existing Record Data Entry**

  You can use the IXLRT service to delete an existing record data entry by specifying the DELETENTRY keyword. Use the ENTRYID keyword to specify the record data entry to be deleted.

  When deleting an existing record data entry, you optionally can specify a verification connection name. XES will verify that the record data entry is associated with the verification connection name before the request is attempted. Use the VERCONNAME keyword to identify the verification connection name of the user with which to associate the record data entry. If the record data entry is not associated with the verification connection name, XES cancels the request and provides an error return code to the requestor. If you do not specify a verification connection name, XES attempts the deletion without verification.

  When the request to delete a record data entry completes successfully, XES returns the following:

  – The total number of remaining entries associated with the user whose entry was deleted (in the area specified by ANSAREA)

  – The total number of remaining allocated entries in the record list (in the area specified by ANSAREA)

  If the record data entry specified by ENTRYID is not allocated, XES provides an error return code to the requestor.

- **Deleting All Entries in a Lock Structure Associated with a User**

  You can use the IXLRT service to delete all record data entries in a lock structure that are associated with a particular user by specifying the DELETEBYCONN keyword

  You can request that the record data entries associated with another connected user be deleted. Use the TCONNAME keyword to identify the target connection name of the user whose record data entries are to be deleted. If you do not specify a target connection name, XES deletes the record data entries associated with the user identified by the connect token provided to IXLRT. Optionally, you can limit the DELETEBYCONN request to delete those record data entries associated both with a particular connected user and with a particular record data type by specifying the RDATATYPE keyword to identify the record data entries to be deleted.

  When the DELETEBYCONN request completes successfully, XES returns the following:

  – The number of record data entries deleted (in the area specified by ANSAREA)

  If the request to delete all entries completes prematurely, XES provides return and reason codes to the requestor. XES also returns the following:

  – The number of record data entries deleted (in the area specified by ANSAREA).

  – A restart token (in the location specified by the RESTOKEN or EXTRESTOKEN keyword), which you can use as input on subsequent requests in order to continue processing with the next record data entry. You should issue the request repeatedly until you receive a return code indicating that there are no more entries to delete.

- **Updating an Existing Record Data Entry**

  You can use the IXLRT service to update an existing record data entry by specifying the UPDATENTRY keyword. Use the ENTRYID keyword to specify the record data entry to be updated. Specify the record data to be written with the RDATAVAL keyword, which identifies the 64 byte area containing the data to be written. Use the RDATATYPE keyword to specify the record data type that is to be updated. If omitted, the default value for RDATATYPE is zero.

When updating a record data entry, you optionally can specify a verification connection name. XES will verify that the record data entry is associated with the verification connection name before the request is attempted. Use the VERCONNAME keyword to identify the verification connection name of the user with which to associate the record data entry. If the record data entry is not associated with the verification connection name, XES cancels the request and provides an error return code to the requestor. If you do not specify a verification connection name, XES attempts the update without verification.

When the request to create a record data entry completes successfully, XES returns the following:

– The total number of entries associated with the user whose entry was updated (in the area specified by ANSAREA)
– The total number of allocated entries in the record list (in the area specified by ANSAREA)

If the record data entry specified by ENTRYID is not allocated, XES provides an error return code to the requestor.

Handling an incompletely processed IXLRT request

An IXLRT READALL or READBYCONN request can complete prematurely (that is, without fully completing the requested service) if the data area (DATAREA) is filled before all data is returned. If that occurs, IXLRT:

- Sets the IXLRT return code to IXLRETCODEWARNING and the reason code to IXLRSNCODETIMEOUT to indicate that processing did not complete.
- Returns in the RTAAREADCNT field of the answer area, the count of data entries read on this IXLRT invocation.
- Returns in either the RESTOKEN, EXTRESTOKEN, or FASTRESTOKEN field that you specified, a restart token to be provided when you reissue the request to continue the IXLRT request.

An IXLRT READALL, READBYCONN, DELETENTRYLST, or DELETEBYCONN request can also complete prematurely if the request exceeds timeout criteria (timeout criteria for a coupling facility is model-dependent) before completing all is processing. If that occurs, IXLRT:

- Sets the IXLRT return code to IXLRETCODEWARNING and the reason code to IXLRSNCODETIMEOUT to indicate that processing did not complete.
- Returns in the RTAAREADCNT field of the answer area, the count of data entries read or in the RTAADELCNT field, the count of data entries deleted on this IXLRT invocation.
- For DELETENTRYLST, returns in the RTAAFAILINDEX field of the answer area, the index of the first unprocessed entry in the list of entry identifiers specified for ENTRYIDLIST. This value should be specified for FIRSTELEM when reissuing the IXLRT request.
- For requests other than DELETENTRYLST, returns in either the RESTOKEN, EXTRESTOKEN, or FASTRESTOKEN field that you specified, a restart token to be provided when you reissue the request to continue the IXLRT request.

An IXLRT DELETENTRYLST request can also complete prematurely when either the first element index specified (FIRSTELEM) or the last element index specified (LASTELEM) are not valid, or any element that is specified in the entry ID list (ENTRYIDLIST) specifies an element that doesn't exist. If that occurs, IXLRT:

- Sets the IXLRT return code to IXLRETCODEPARMERROR and the reason code to IXLRSNCODEBADIDINDEX to indicate that processing did not complete.
- Returns in the RTAADELCNT field of the answer area, the count of data entries deleted on the IXLRT invocation.
- Returns in the RTAAFAILINDEX field of the answer area, the index of the failing entry in the list of entry identifiers specified for ENTRYIDLIST. An index value past this one should be specified for FIRSTELEM when reissuing the IXLRT request.

An IXLRT READBYCONN request that specifies FASTPATH=YES can also complete prematurely if the record table entry represented by FASTRESTOKEN has been deleted or reacquired. If that occurs, IXLRT:

- Sets the IXLRT return code to IXLRETCODEPARMERROR and the reason code to IXLRSNCODEENTRIESCHANGED to indicate that processing did not complete. In this case, FASTRESTOKEN must be reset to zero to start the READBYCONN request from the beginning.

To complete the IXLRT processing, for requests other than DELETENTRYLST, continue to reissue the IXLRT request, specifying the restart token (RESTOKEN, EXTRESTOKEN, or FASTRESTOKEN) returned on this request. Do not change the contents of the restart token as returned by the system unless you are resetting it back to zero to restart an IXLRT READBYCONN request specifying FASTPATH=YES.

For DELENTENTRYLST requests, adjust the index of the first ENTRYIDLIST list element to be processed (FIRSTELEM) and index of the last ENTRYIDLIST list element to be processed (LASTELEM) as appropriate. Be sure to process the information that is returned from this request before reissuing the request. The data that is returned from this request will be overwritten if you specify the same data area. Continue to reissue the request until the return code indicates that all processing has completed.

For asynchronous system-managed duplexing, the highest asynchronous duplexing request sequence number (if any) for completed request processing is returned in Rtaa1ADupReqSeqNum.

Chapter 11. Supplementary List, Lock, and Cache Services

Using the IXLFCOMP Macro

Note

The following information assumes that you are familiar with either the IXLLIST macro or the IXLCACHE macro and that you are using either a list or cache structure. For more information about the IXLLIST macro, see Chapter 8, “Using List Services (IXLLIST),” on page 481. For more information about the IXLCACHE macro, see Chapter 7, “Using Cache Services (IXLCACHE),” on page 357.

If you are an IXLLIST or IXLCACHE macro user who specified MODE=ASYNCTOKEN or specified MODE=SYNCTOKEN but had the request processed asynchronously, you can use the IXLFCOMP macro to do either of the following:

- **Test whether your list or cache request has completed (OPTYPE=TEST)**. Choose this option if your task cannot be suspended or your program can perform other work while the list or cache request is being processed. IXLFCOMP's return code indicates whether the request has completed.

  If the request has already completed, control returns to you so you can check the results of the request in the output areas you specified on the list or cache request.
- **Have your task suspended until your list or cache request completes (OPTYPE=COMPLETE).** Choose this option if your task can be suspended and you have no other work to perform while the list or cache request is being processed.

  Once the request completes or if it has already completed when you issued IXLFCOMP, control returns to you so you can check the results of the request in the output areas you specified on the list or cache request.

When you issue the IXLFCOMP macro, you identify the target request using the request token returned from the IXLLIST or IXLCACHE invocation.

Before accessing the answer area information returned from an IXLLIST or IXLCACHE request, be sure to read:

- “Determining if the Answer Area is Valid” on page 535, which describes the circumstances under which the answer area information returned by IXLLIST is not valid.
- “Determining Valid Information in the Answer Area” on page 397, which describes the circumstances under which the answer area information returned by IXLCACHE is not valid.

Issuing IXLFCOMP During Recovery Processing

If you issue the IXLFCOMP macro with OPTYPE=COMPLETE for an IXLLIST or IXLCACHE request that has already been purged (IXLPURGE macro), your task will not be suspended because the IXLLIST or IXLCACHE request has already terminated and the processing results are already available to you.

If your application's resource manager issues the IXLFCOMP macro to determine the results of an IXLLIST or IXLCACHE request issued by a connected user whose task has been terminated, you must ensure that IXLFCOMP processing has completed before you return control to RTM. Once control returns to RTM, the system performs its own clean-up and deletes any information relating to the terminated user's IXLLIST or IXLCACHE request.

© Copyright IBM Corp. 1994, 2023

Purging a Coupling Facility Operation

The IXLPURGE macro allows you to complete IXLCACHE, IXLLIST, and IXLRT operations in progress to a coupling facility and to purge operations that have not yet processed. The operations to be purged must have been invoked on the same system on which IXLPURGE is issued. Use IXLPURGE to complete or purge outstanding XES operations:

- For a specific task.
- For a specific address space.
- For a specific connector.
- For one or more requests by request ID (REQID) associated with a specific connector.

When you invoke IXLPURGE, XES completes all pertinent operations or purges all those operations waiting to be completed. When IXLPURGE completes, all XES-established storage binds are broken. Request notification completion for purged requests is scheduled asynchronously, if the requestor is able to receive the completion information. If the requestor is no longer defined, then it cannot receive the completion information.

Handling Operations in Progress

Each operation in progress will be forced to completion and the completion information will be returned to the requestor using the notification mechanism specified by the requestor.

Handling Operations Yet to be Processed

Each operation that is waiting to be processed will be purged and a return code indicating that the operation has not been completed will be returned to the requestor using the notification mechanism specified by the requestor.

Timing Considerations

IXLPURGE detects only those outstanding IXLCACHE, IXLLIST, and IXLRT operations at the time of the IXLPURGE invocation. IXLPURGE may not detect and does not inhibit XES operations started subsequent to the IXLPURGE invocation.

Using the IXLVECTR Macro

The IXLVECTR macro allows you to perform the following functions on a list notification vector or local cache vector associated with a coupling facility structure:

- Test a list notification or a local cache vector entry
- Load and test a range of list notification or local cache vector entries
- Modify the size of a list notification vector or local cache vector.

Note: The following information assumes that you are familiar with either the IXLLIST macro or the IXLCACHE macro and that you are using either a list or cache structure. For more information about the IXLLIST macro, see Chapter 8, “Using List Services (IXLLIST),” on page 481. For more information about the IXLCACHE macro, see Chapter 7, “Using Cache Services (IXLCACHE),” on page 357.

List Notification Vector

When you issue the IXLVECTR macro, you identify your list notification vector using the vector token returned in the connect answer area (mapped by the CONAVECTORTOKEN field of the IXLYCONA macro) when you issued the IXLCONN macro to connect to the list structure. You receive a vector token from IXLCONN only if you coded the VECTORLEN parameter.

If the structure was rebuilt, be sure to use the vector token returned from the IXLCONN REBUILD request instead of the original vector token.

Changing the Number of Entries in a List Notification Vector

The MODIFYVECTORSIZE request allows you to change the number of entries in your list notification vector so you can monitor a different number of objects in the list structure.

Reducing the size of your list notification vector when it is larger than necessary frees storage for the list notification vectors of other users on your system.

Use the VECTORLEN parameter to indicate the new number of entries you would like your list notification vector to contain.

The number of vector entries must be a multiple of 32. If the value you specify is not a multiple of 32, the system rounds the value up to a multiple of 32.

The number of entries the system actually assigns to your list notification vector is returned to you as output through the ACTUALVECLEN parameter.

- **Decreasing the Number of Entries**: If you request a decrease in the number of entries in your list notification vector, your request will always be satisfied.

  When a list notification vector's size is decreased, the number of entries is reduced by removing entries starting with the highest number. The remaining entries are unchanged and retain their original values (empty or non-empty).

  Before eliminating any entries, you must ensure that the entries that will be deleted are not being used to monitor lists or an event queue.

  If multiple users could be accessing vector entries concurrently, you should obtain exclusive serialized access to the vector before decreasing its size. Otherwise, users that issue the TESTLISTSTATE or LTVECENTRIES request must be prepared to handle a return code of IXLRETCODEINDXINVALID, indicating that the specified vector index is no longer valid.
- **Increasing the Number of Entries**: If you request an increase in the number of entries in your list notification vector and the system is unable to obtain sufficient storage to satisfy your request, the new number of entries might be unchanged or smaller than you requested. In this case, the number of entries returned in ACTUALVECLEN will be smaller than the requested number and you will receive return code IXLRETCODELESSTHAN to inform you of the result.

  When a list notification vector's size is increased, the number of entries is increased by adding additional entries after the current highest-numbered entry. Existing entries are unchanged and retain their original values (empty or non-empty). New entries are initialized to the non-empty state.

Testing Whether a List or Event Queue Is Empty

The TESTLISTSTATE request allows you to test the entry representing a particular list or event queue to determine whether that list or event queue is empty. List notification vector updates are performed asynchronously by the system, so a vector entry might not show a particular list or event queue state change at the time you check it. However, the system always performs the change in the vector (and the notification, if applicable). The system also ensures that, if the list or event queue transitions to non-empty and then back to empty and so on multiple times, the final state reflected in the vector will match the final state of the list or event queue. Individual transitions, however, might not be applied to the vector if subsequent changes supersede them. For example, if the initial state of the vector entry indicates that the list or event queue is empty and then the list or event queue transitions to non-empty and then becomes empty again in a short period of time, it is not guaranteed that the interim non-empty state will be reflected in the vector or that notification will occur through your list transition exit. However the final state (empty, in this case) is guaranteed to be correct.

To use the TESTLISTSTATE request, you must have registered your interest in monitoring the particular list and/or event queue. IXLVECTR assumes you have previously issued the IXLLIST MONITOR_LIST request or the IXLLIST MONITOR_EVENTQ request and have associated that list or event queue with the specified vector index. The system does not check whether you have done this.

Testing Whether a List is Full or Not-Full

When the CFLEVEL of the coupling facility in which the list structure is allocated is CFLEVEL=22 or higher, the TESTLISTSTATE request tests the entry that is representing a particular list to determine whether that list is one of the following:

- Empty or not-empty
- Full (list count limit for entries or elements is reached) or not-full (list count limit for entries or elements is not reached)

An entry in a list notification vector can be used to determine either the empty/not-empty state or full/not-full state of a list. A connector to a list structure can monitor a specific list for either the empty/not-empty state or full/not-full state of a list, but not both. A connector can dynamically change the type of list monitoring that is performed for a list (without stopping monitoring) by reregistering (issue another start monitoring request) with the opposite list-notification monitor type (MONITORTYPE=[NOTEMPTY or NOTFULL]).

List notification vector updates for list full/not-full state changes are performed in a similar manner as the updates that are made for empty/not-empty state changes. For more information about how the system updates a list notification vector to reflect the state of a list, see "Testing Whether a List or Event Queue Is Empty" on page 681.

To use the TESTLISTSTATE request to test for the full/not-full state of a list, you must have registered your interest in monitoring the particular list using IXLLSTC MONITOR_LIST and specified the MONITORTYPE=NOTFULL keyword. IXLVECTR assumes that you have previously issued the IXLLSTC MONITOR_LIST request and have associated that list with the specified vector index. The system does not check whether you have done this.

Testing a Range of List Notification Vector Entries

The LTVECENTRIES request allows you to test up to 32 consecutive vector entries to determine whether their associated event queue or lists are empty. When the list notification vector is for a list structure that is allocated in a coupling facility at CFLEVEL=22 or higher, vector entries for monitored lists can indicate whether a monitored list is full or not-full. The output from this request is a bit string with one bit per vector entry, starting with the vector entry you specify as the starting vector entry number and continuing until 32 bits are loaded. Vector entries range from 0 to n-1, where n is the number of entries in the vector.

The bits in the bit string are interpreted as follows:

0
: The vector entry corresponding to this bit position indicates that the monitored list or event queue is not empty. If the list is being monitored for full/not-full transitions, then the vector entry indicates that the list is not-full.

1
: The vector entry corresponding to this bit position indicates that the monitored list or event queue is empty. If the list is being monitored for full/not-full transitions, then the vector entry indicates that the list is full.

Use IXLLSTC REQUEST=MONITOR_LIST to specify whether to monitor a list for empty/not-empty states or full/not-full states (keyword MONITORTYPE).

Local Cache Vector

When you issue the IXLVECTR macro, you identify your local cache vector using the vector token returned in the connect answer area (mapped by the CONAVECTORTOKEN field of the IXLYCONA macro) when you issued the IXLCONN macro to connect to the cache structure. A local cache vector is required with the use of a cache structure.

If the structure was rebuilt, be sure to use the vector token returned from the IXLCONN REBUILD request instead of the original vector token.

Changing the Number of Entries in a Local Cache Vector

The MODIFYVECTORSIZE request allows you to change the number of entries in your local cache vector so you can maintain concurrent registered interest in a different number of data items.

Reducing the size of your local cache vector when it is larger than necessary frees storage for the local cache vectors of other users on your system.

The number of vector entries must be a multiple of 32. If the value you specify is not a multiple of 32, the system rounds the value up to a multiple of 32.

Use the VECTORLEN parameter to indicate the new number of entries you would like your local cache vector to contain. The number of entries the system actually assigns to your local cache vector is returned to you as output through the ACTUALVECLEN parameter.

- **Decreasing the Number of Entries**: If you request a decrease in the number of entries in your local cache vector, your request will always be satisfied.

  When a local cache vector's size is decreased, the number of entries is reduced by removing entries starting with the highest number. The remaining entries are unchanged and retain their original values (valid or not valid).

  Before eliminating any entries, you must ensure that the entries that will be deleted are not associated with any data items.

  If multiple users could be accessing vector entries concurrently, you should obtain exclusive serialized access to the vector before decreasing its size. Otherwise, users that issue the TESTLOCALCACHE or LTVECENTRIES request must be prepared to handle a return code of IXLRETCODEINDXINVALID, indicating that the specified vector index is no longer valid.
- **Increasing the Number of Entries**: If you request an increase in the number of entries in your local cache vector and the system is unable to obtain sufficient storage to satisfy your request, the new number of entries could be unchanged or smaller than what you requested. In this case, the value returned in ACTUALVECLEN will be smaller than the requested number of entries and you will receive return code IXLRETCODELESSTHAN to inform you of the result.

  When a local cache vector's size is increased, the number of entries is increased by adding additional entries after the current highest-numbered entry. Existing entries are unchanged and retain their original values (valid or invalid). New entries are initialized to the invalid state.

Checking the Validity of Data Items in a Local Cache Buffer

The TESTLOCALCACHE and LTVECENTRIES requests allows you to determine whether data items in your local cache buffer are valid. A data item is valid when the user has registered interest in the data item, and no other user of the structure has caused that item to be invalidated.

The TESTLOCALCACHE request allows you to check a single local cache vector entry to determine the validity of a single data item. The LTVECENTRIES request allows you to check 32 consecutive local cache vector entries to determine the validity of their associated data items.

To use the TESTLOCALCACHE or LTVECENTRIES request, you must establish a serialization protocol to be followed by all programs with which you are sharing access to the data items. Without adhering to such a protocol, you cannot prevent a data item you are accessing from being rendered invalid by another user at any time. “Using the TESTLOCALCACHE and LTVECENTRIES Requests with a Serialization Protocol” on page 685 provides information about possible serialization protocols.

To use the TESTLOCALCACHE request with the VECTORINDEX parameter or to use the LTVECENTRIES request, you must have previously associated each specified vector entry with a data item of interest (using the IXLCACHE macro.) IXLVECTR assumes you have associated any specified vector index with a data item in the cache structure. It does not do any checking to enforce this.

Vector entries range from 0 to n-1, where n is the number of entries in the vector.

There are several options for checking the validity of data items. These are described below. "Using the TESTLOCALCACHE and LTVECENTRIES Requests with a Serialization Protocol" on page 685 shows how these options are used together.

The TESTLOCALCACHE request has the following variations:

- TESTLOCALCACHE with VALIDATE=YES and VECTORINDEX omitted, which requests that the system validate connectivity to the coupling facility
- TESTLOCALCACHE with VALIDATE=YES and VECTORINDEX specified, which requests that the system validate connectivity to the coupling facility and check the validity of a particular data item
- TESTLOCALCACHE with VALIDATE=NO and VECTORINDEX specified, which requests that the system check the validity of a particular data item without validating connectivity to the coupling facility.

The LTVECENTRIES request allows you to test a range of 32 consecutive local cache vector entries to determine the validity of their associated data items. The output from this request is a bit string with one bit per vector entry, starting with the vector entry you specify as the starting vector entry number and continuing until 32 bits are loaded.

The bits in the bit string are interpreted as follows:

0
: The vector entry corresponding to this bit position indicates that the local cache buffer is not valid

1
: The vector entry corresponding to this bit position indicates that the local cache buffer is valid.

Note: The LTVECENTRIES request does not validate connectivity to the coupling facility.

- **Validating Connectivity to the Coupling Facility**:

  Use the TESTLOCALCACHE request with VALIDATE=YES and the VECTORINDEX parameter omitted to determine whether connectivity between your system and the coupling facility was temporarily interrupted (which might have caused the loss of one or more cross-invalidate signals.) Specifying VALIDATE=YES allows you to determine if an interruption has prevented any previous cross-invalidate requests from being performed against your local cache vector.

  If connectivity has been maintained, your local cache vector will reflect all previous cross-invalidate requests because they are performed synchronously. If connectivity has been interrupted, all entries in the local cache vector will be invalidated to ensure no data items are incorrectly marked as valid.

- **Validating Connectivity to the Coupling Facility and Checking the Validity of a Data Item**:

  Use the TESTLOCALCACHE request with VALIDATE=YES and the VECTORINDEX parameter specified to do both of the following:

  – Validate connectivity between your system and the coupling facility
  – Determine the validity of a data item in your local cache buffer.

  You do not get a specific indication of whether connectivity to your local cache vector has been maintained. However, the system invalidates all entries in a local cache vector when it detects that connectivity between the system and the coupling facility has been temporarily interrupted. If the data item is shown as valid you can also be assured that the local cache vector has maintained connectivity with the coupling facility.

- **Checking the Validity of Data Items without Validating Connectivity to the Coupling Facility**:

  You should only check the validity of data items without validating connectivity to the coupling facility **after** you have already issued TESTLOCALCACHE with VALIDATE=YES under the same serialization as this request.

  Use the TESTLOCALCACHE request with VALIDATE=NO and the VECTORINDEX parameter specified to determine the validity of a single data item in your local cache buffer. Use the LTVECENTRIES request to test 32 consecutive local cache vector entries to determine the validity of their associated data items.

These options do not involve checking whether there has been an interruption in connectivity between the system and the coupling facility.

Using the TESTLOCALCACHE and LTVECENTRIES Requests with a Serialization Protocol

To guarantee that the data item in your local cache is valid and will remain so while you reference or update it, you must ensure the following:

1. All previous cross-invalidate requests have been received and recorded in your local cache vector (verified by issuing the IXLVECTR macro with TESTLOCALCACHE and VALIDATE=YES)
2. No new cross-invalidate requests can be issued for the data item while you are using it (because you have serialized access to the data items.)

If these two conditions are met, you can check the vector index associated with the data item of interest and be sure that it is correct and accurate.

Figure 73 on page 686 shows a sample serialization protocol for a single data item. The flowchart assumes that the data item exists in the user's local cache.

Serialization for Multiple Data Items: Figure 74 on page 687 shows a sample serialization protocol for multiple data items comprising a single resource, for example, a set of data blocks functioning as a unit and represented by several entries in the local cache vector. The flowchart assumes that the data items are in the user's local cache. In this case, one lock provides serialization for multiple data items, each represented by a separate vector index.

The flowchart uses the TESTLOCALCACHE request to test each vector index individually. If the vector indexes are consecutive, you can issue the LTVECENTRIES request once instead of issuing the TESTLOCALCACHE request multiple times. Figure 75 on page 688 illustrates this process.

Because IXLVECTR's performance is significantly slower with VALIDATE=YES, you should validate connectivity between the coupling facility and your local cache vector either when you check the validity of the first data item (TESTLOCALCACHE with VALIDATE=YES and VECTORINDEX specified) or before checking the validity of any of the data items (TESTLOCALCACHE with VALIDATE=YES and VECTORINDEX omitted). After this, you can perform the validity checks on the other data items either using TESTLOCALCACHE with VALIDATE=NO or using LTVECENTRIES This method lets you avoid having to issue IXLVECTR with TESTLOCALCACHE and VALIDATE=YES multiple times.

To use this approach, you must obtain the lock for the group of data items before you issue TESTLOCALCACHE with VALIDATE=YES and continue to hold the lock while you issue either TESTLOCALCACHE with VALIDATE=NO or LTVECENTRIES. Since other users cannot cross-invalidate a data item in the group while you hold the lock, you need only check for connectivity interruptions (VALIDATE=YES) on the first invocation of IXLVECTR for that group of data items.

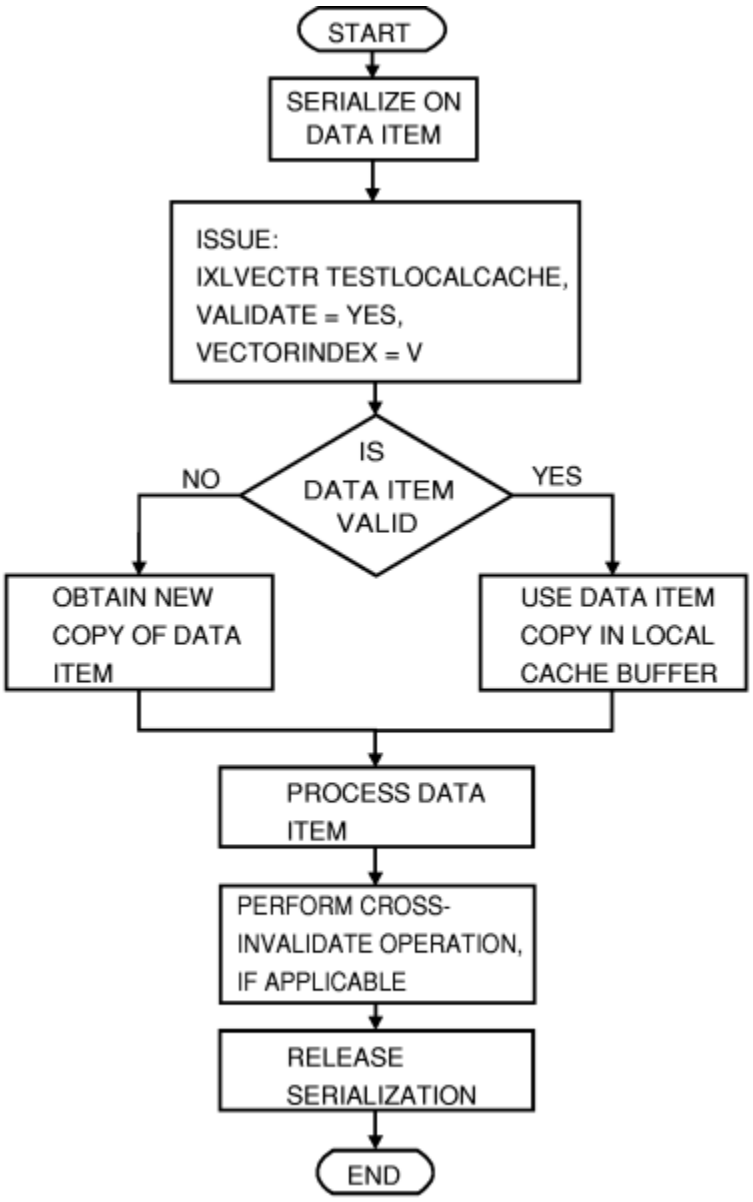


Figure 73. Sample Serialization Protocol for Single Data Item

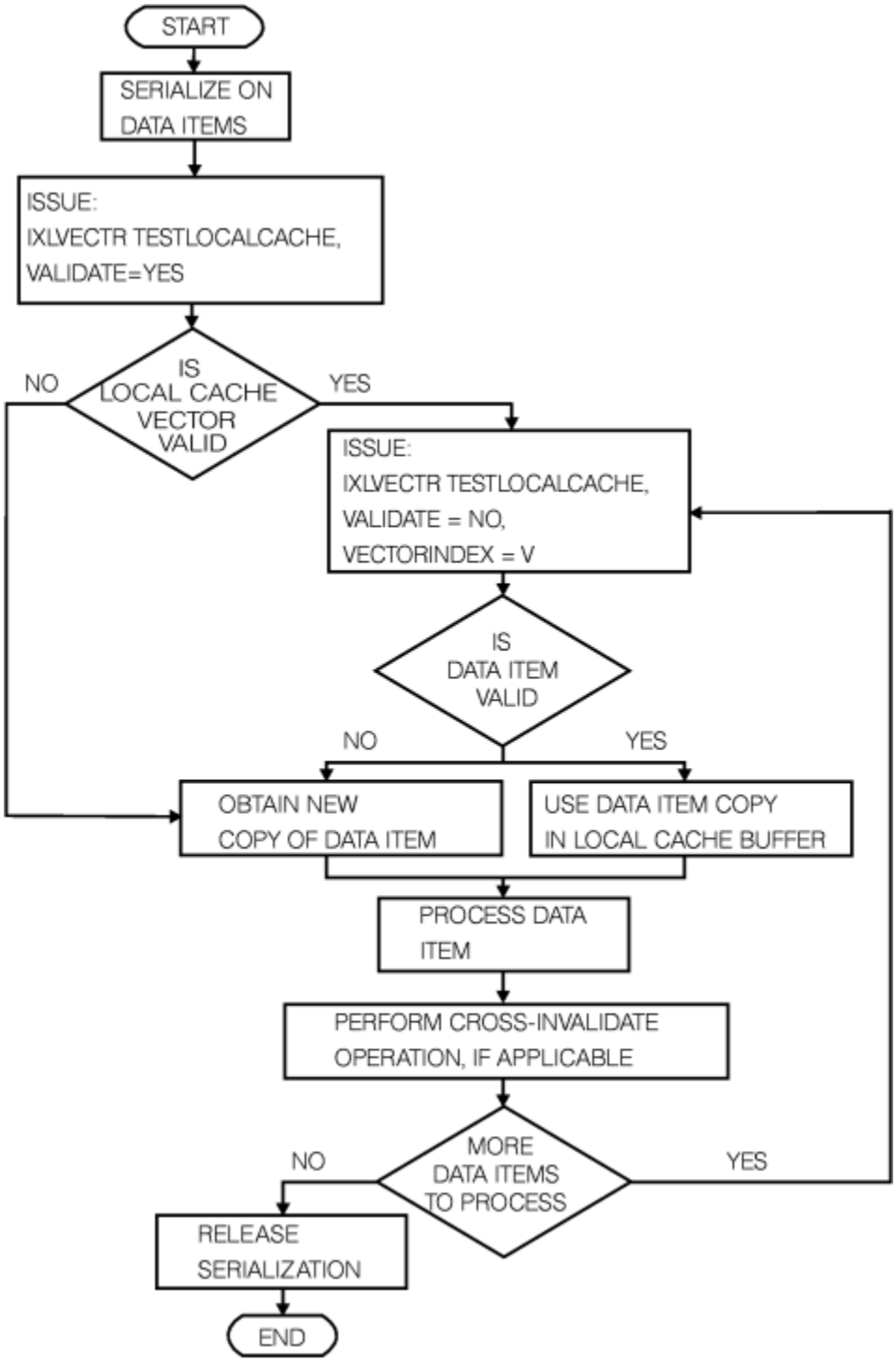


Figure 74. Sample Serialization Protocol for Multiple Data Items

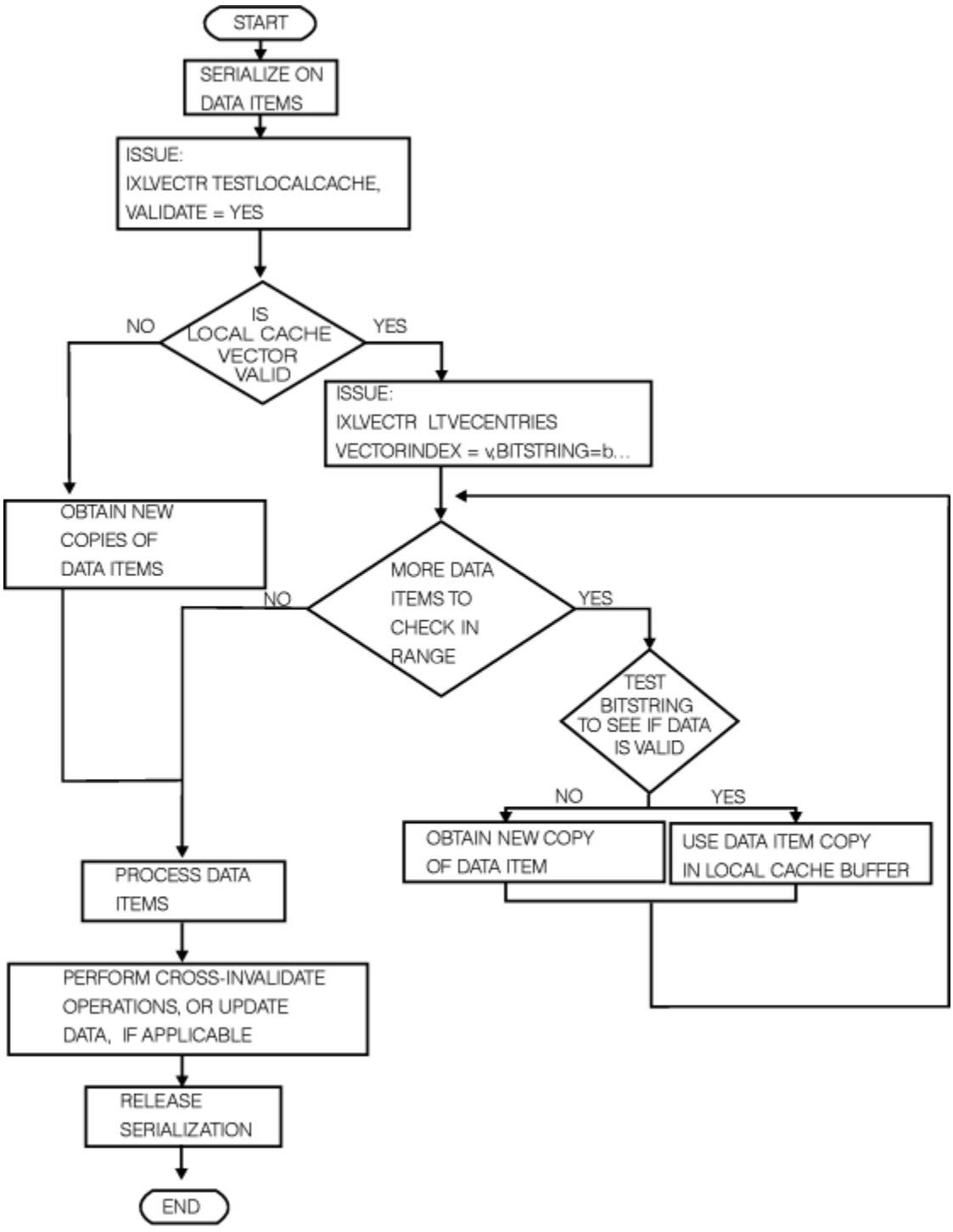


Figure 75. Sample Serialization Protocol for a Range of Data Items

Using the IXLADUPX macro

The IXLADUPX service allows a connected user to ensure that a request that was previously committed in the primary instance of an asynchronously duplexed structure is also committed in the secondary instance of the structure. The request is identified by the asynchronous duplexing request sequence number (ADUPREQSEQNUM). The IXLADUPX service supports the following operation types:

- OPTYPE=TEST. This option allows you to test whether the request has been committed in the secondary structure. Choose this option if the invoking unit of work cannot be suspended while the request is being

processed. The return and reason codes indicate whether the request identified by ADUPREQSEQNUM has been committed in the secondary structure.

- OPTYPE=SUSPEND. This option suspends the invoking unit of work until the request identified by ADUPREQSEQNUM is committed in the secondary structure. Choose this option if the invoking unit of work can be suspended while the request is being processed.

Issuing the IXLADUPX requires system-managed asynchronous duplexing support. Use of IXLADUPX on a system that does not support it will have unpredictable results. Macro IXCYQUAA defines the QuReqRfAsyncDuplex bit in the QuReqFeatures string that can be used to test for system-managed asynchronous duplexing support. Use IXCQUERY REQINFO=FEATURES to get the QuReqFeatures string. It can be assumed that systems at a z/OS level higher than V2R2 support system-managed asynchronous duplexing.

For more information, see "Working with structures in the Async Duplex Established phase" on page 285.

Using the IXLAXISN macro

The IXLAXISN service provides the capability for a connected cache structure user to determine when asynchronous cross-invalidations associated with cache requests issued by the connector have completed. Asynchronous cross-invalidations for a cache request are identified by the asynchronous cross-invalidation sequence number (CAAASYNCXISEQNUM) returned in the IXLCACHE Answer Area (IXLYCAA) on IXLCACHE requests that generated asynchronous cross-invalidations. The IXLAXISN service supports the following operation types:

OPTYPE=TEST
This option allows you to test whether the cross-invalidations associated with the asynchronous cross-invalidation sequence number (ASYNCXISEQNUM) have completed. The return and reason codes indicate whether the cross-invalidation processing has completed in the coupling facility.

This option may be used to poll for cross-invalidation completion status by invokes that cannot be suspended.

OPTYPE=SUSPEND
This option suspends the invoking unit of work until the cross-invalidations associated with ASYNCXISEQNUM have been completed by the coupling facility. Choose this option if the invoking unit of work can be suspended while cross-invalidations are being processed by the coupling facility.

Issuing the IXLAXISN service requires XES asynchronous cross-invalidation support. Use of IXLAXISN on a system that does not have XES asynchronous cross-invalidation support installed will have unpredictable results. Macro IXCYQUAA defines the QuReqRfAsyncXI bit in the QuReqFeatures string that can be used to test for asynchronous cross-invalidation support. Use IXCQUERY REQINFO=FEATURES to get the QuReqFeatures string. Systems at a z/OS level of V2R2 and above with APAR OA54688 installed or which are at z/OS V2R4 and above support XES asynchronous cross-invalidation.

For more information, see sections "Connecting to a Cache Structure" on page 230 and "Deregistering Interest in a Data Item and Invalidating Local Copies" on page 375.

Chapter 12. Using Note Pad Services (IXCNOTE)

Programs issue the assembler macro IXCNOTE to use the XCF Note Pad Services to:

- Create and delete an XCF note pad
- Dynamically modify the number of notes that an XCF note pad needs to hold
- Create and delete connections to a note pad
- Create, update, read, and delete notes in a note pad on behalf of a connection

To the programmer, an XCF note pad can be viewed as shared storage that is directly accessible to note pad connectors distributed throughout the sysplex. A note pad is a named collection of notes. Each note contains application provided content. After a note pad is created, programs establish connections to the note pad in order to process notes. Note pad connectors create notes in the note pad. The creator of a note supplies its content and gives it a name. Once a note is created, any note pad connector (with the appropriate authority) can read, replace, or delete its content. The note itself can also be deleted. Connectors specify the note name to identify the note to be created, updated, or deleted. Each connection is represented by a connection token. The token is valid for use on the system where the connection was created.

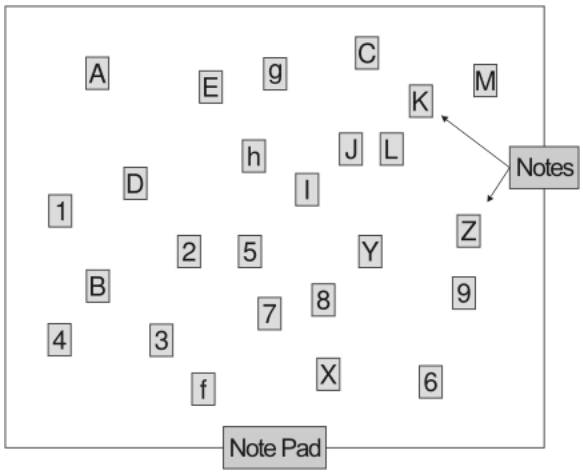


Figure 76. Conceptual model of a note pad

© Copyright IBM Corp. 1994, 2023

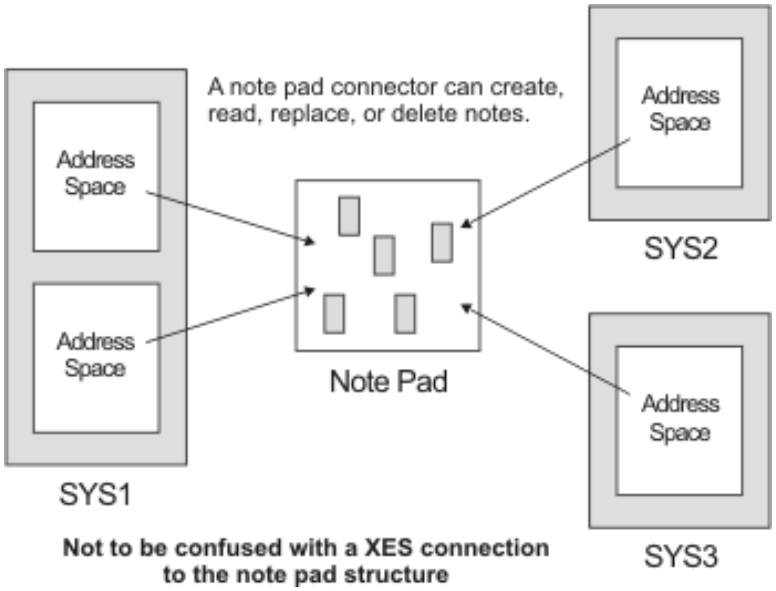


Figure 77. Connections to a note pad

A note pad might be used, for example, as a data repository to assist with recovery actions after an application failure. An application might write notes to a note pad to record information about the state of its processing. As progress is made, notes are updated or deleted. If the application fails, it could be restarted. Upon restart, it reads the notes from the note pad to determine its state at the time of failure, takes appropriate recovery actions, and resumes normal processing. If the system failed, the application could be restarted on some other system in the sysplex since note pads are generally accessible from every system in the sysplex. Alternatively, surviving peer instances could read its notes and perform appropriate recovery actions, or perhaps even take over its work. Writing notes to a note pad will generally be much faster than other data replication techniques such as writing to DASD or sending signals to other systems. Use of a note pad might therefore reduce the performance impact of maintaining this state information for recovery purposes.

An XCF note pad is actually implemented as a set of list entries on a list in a coupling facility (CF) list structure. Thus the XCF Note Pad Services provide a simple way for programs to create data that can be accessed from any system in the sysplex. However, the note pad interface does not expose all the rich functionality made available to programs that directly exploit the list structure interfaces (see Chapter 9, “Using List Services (IXLLSTE, IXLLSTM, IXLLSTC),” on page 677). For example, note pad connectors are not notified when a note is created, updated, or deleted. Nor are they notified when a connection to the note pad is created or deleted. Thus the note pad abstraction tends to be most suitable for programs that need a simple shared data repository. In cases where the note pad programming model is suitable, the note pad interface is generally easier to use than the list structure interface. For example, no exit routines are needed. Furthermore, the note pad interface supports unauthorized callers.

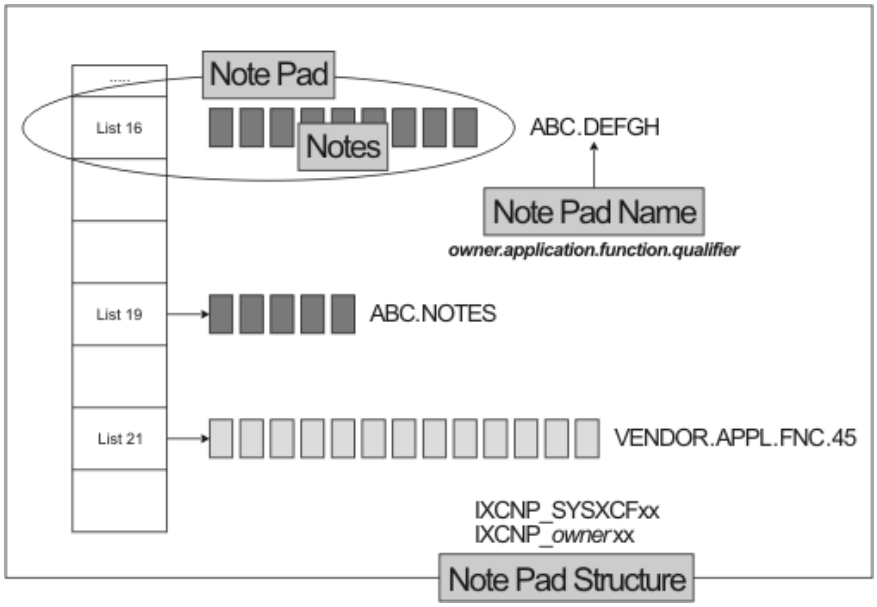


Figure 78. Physical embodiment of a note pad

When system administrators or system programmers install software that requires an XCF note pad, they create a Coupling Facility Resource Management (CFRM) policy that specifies the name, size, and attributes of each structure that is to be used for note pads. The CFRM policy also allows the installation to limit the amount of storage each structure can occupy and through a *preference list* of one or more coupling facilities, control where each structure is allocated. The policy also indicates whether the structure is eligible for duplexing.

When a program uses the XCF Note Pad Services to create a note pad, it gives the note pad a name, indicates the number of notes it needs to hold, and a duplexing preference. XCF uses these specifications in conjunction with the structure definitions in the CFRM policy to choose a structure to host the note pad. A given structure can host one or more note pads. The duplexing preference provides guidance as to whether the note pad should be hosted by a structure that is subject to being duplexed via System Managed CF Structure Duplexing. If the CFRM policy indicates that a structure is pending delete, that structure is not a candidate for hosting a new note pad.

Products and subsystems that exploit XCF note pads must document their requirements as part of their installation information. The system programmer needs to know the name of the note pad, the number of notes needed, and the duplexing preference in order to define suitable host structures in the CFRM policy. The create of the note pad will fail if none of the candidate host structures have space for the requested number of notes. XCF tries to honor the duplexing preference specified by the creator of the note pad, but will not fail the create request if it is unable to do so.

The security administrator needs to know the name of the note pad so that appropriate security profiles can be defined to control use of the note pad. Through these profiles, the security administrator can control the ability of a program to (1) create and delete a note pad, (2) create, update, or delete notes in a note pad, and (3) read notes in a note pad. The implementation of the exploiting application will likely influence the nature of the security profiles that need to be defined. In order for programs that run problem state with a PKM allowing key 8 to 15 to use a note pad, a security profile must be defined for the note pad.

XCF maintains a catalog of all of the note pads that have been defined in the sysplex. Among other things, the catalog indicates which coupling facility structure hosts the note pad. With the exception of cases where a system is unable to access the note pad catalog, the existence and use of the catalog is largely transparent to the applications that exploit note pads. However, if a system loses access to the note pad catalog, it also loses access to the note pads.

Note pad concepts and terminology

To use an XCF note pad, you must understand the following concepts and related terminology:

Note
Application provided data identified by a note name.

Note pad
A collection of notes identified by a note pad name.

Connection
An entity that manipulate notes in a note pad, identified by a connection token.

To use an XCF note pad, you must understand the various functions provided by the XCF Note Pad Services:

Note Pad Services
These services enable your application to create or delete a note pad, and get information about a note pad. The creator of the note pad determines the attributes of the note pad and the number of notes it can hold. The Note Pad Services can be used to request a dynamic increase or decrease in the number of notes in a note pad in response to increased or decreased demand on note pad capacity to avoid disruptions in application availability during times of high demand. Some specifications require users of the note pad (connections) to adhere to certain conventions or protocols. The query service can be used to determine whether a note pad exists. If it does exist, the query service can be used to get information about the note pad. The note pad will generally exist until it is explicitly deleted or fails. A note pad does not survive a sysplex outage. When deleting a note pad, you can optionally specify various conditions that must be satisfied in order for the delete request to go forward. For example, you might want the delete request to be rejected if the note pad still has connections.

Connection Services
The connection services enable your application to create or delete a note pad connection, and pause or resume some particular connection thread. A connection must be created in order for a program to manipulate notes in the note pad. When creating a connection, the program can specify a termination scope to bind the connection to a particular task or address space. When the designated task or space terminates, XCF automatically deletes the connection. Alternatively, a program can explicitly delete the connection when it is no longer needed. XCF implicitly deletes a connection if the system that created the connection terminates. XCF also deletes a connection if the relevant note pad fails or is otherwise deleted.

There might be times when the note pad becomes inaccessible. For example, connectivity to the coupling facility that contains the note pad could be lost, or the CF structure that hosts the note pad could be in rebuild. Such a note pad is said to be in a quiesced state. When quiesced, requests to manipulate notes in the note pad are rejected. To determine when the note pad is once again accessible, the connection can use the connection service to pause a particular work unit. This service suspends the calling work unit until access to the note pad is restored, or until various other conditions are satisfied (such as the expiration of a timeout value). The connection service can also be used to resume the paused work unit at the discretion of the application.

Note Services
The note services enable your application to manipulate one or more notes in a note pad. To use these services, your application must first create a connection to the note pad. Once connected, the note services can be used to create, update, read, or delete one particular note (a single note request). In this case, the note to be processed is identified by its user provided name. The note services can also be used to read or delete a collection of notes (a multi-note request). In this case, the notes to be processed are identified by *selection criteria*. The selection criteria define the attributes of the notes to be processed. XCF finds the notes in the note pad that satisfy the selection criteria and applies the requested operation to those notes.

To use an XCF note pad, you must understand how to code the IXCNOTE macro to exploit the functions provided by the XCF Note Pad Services and the contexts in which these functions can be used by your program. You must understand how to format the input data areas required for the requested function,

how to interpret the request results, and how to use the information stored in various output data areas. These data areas include:

Answer Area
When your program issues the IXCNOTE macro to call the XCF Note Pad Services, it can provide an answer area. The storage for the answer area is obtained by your program. XCF stores data relevant to the result of the request in the answer area. Although an answer area is optional for most requests, you can always provide one if you like. In many cases, XCF will store diagnostic data and other potentially useful details in the answer area if one is provided. If your program needs to use these details or if the diagnostic data is needed to diagnose a problem, an answer area must be provided.

Buffer Area
When your program issues the IXCNOTE macro to process a single note request or a multi-note request, it can provide a buffer area. The storage for the buffer area is obtained by your program. The buffer area is used for note content. When creating or replacing a note, your program stores the desired note content in the buffer area and then issues the IXCNOTE request. XCF fetches the data from the buffer area and stores a copy of it in the designated note out in the note pad. When reading or deleting a note, XCF fetches the note content from the designated note in the note pad and stores a copy of it in the buffer area. When reading a collection of notes, the content of each note in the collection is stored in the buffer area (for as many notes as will fit).

Selection Criteria
When processing a multi-note request, your program can optionally provide an input data area containing selection criteria. The storage for the selection criteria data area is obtained by your program. A multi-note request processes a collection of notes. Your program formats the data area to define the criteria that XCF will used to determine which notes are to be selected for the collection.

Mappings for the data areas related to use of the IXCNOTE macro are declared in the IXCYNOTE macro.

Use of the term *connection*

The XCF Note Pad Services use terms and concepts similar to those used by the Sysplex Services for Data Sharing (XES). Although the terms and concepts are similar, they are not the same. This similarity could be a source of confusion to those who are familiar with the terms and concepts as used by XES. Most of the confusion will arise over the interpretation and use of the term *connection*. Both services use this term to express a similar idea. With XES, a connection must be established in order to access a coupling facility structure. With the XCF Note Pad Services, a connection must be established in order to access a note pad. The fact that XCF Note Pad Services establish a XES connection to support the note pad connection tends to further compound the confusion.

The key distinctions between the two notions of connection are listed below:

- The XES Connection Service (IXLCONN) creates a connection to a coupling facility structure. If the structure does not exist, it will be created as a side effect of creating the first connection.

  With the XCF Note Pad Services (IXCNOTE), the create of a note pad and the create of a connection to a note pad are distinct operations. A note pad must be created before any connections to the note pad can be created. If a note pad does not exist, any attempt to create a connection to the note pad will fail. The create of a note pad does not cause a connection to the note pad to be created.
- If a system loses connectivity to a coupling facility that contains a structure, XES invalidates the connection to the structure (in cases where no alternate instance of the structure is accessible). Programs must then invoke the XES Disconnect Service (IXLDISC) to delete the connection. When connectivity to the coupling facility containing the structure is restored, the XES Connect Service (IXLCONN) must be invoked to create a new connection to the structure.

  If a system loses connectivity to a coupling facility that contains a note pad, the note pad connection remains intact. The connector will not be able to access the note pad until connectivity to the relevant coupling facility is restored. But when connectivity is restored, the connector can resume its processing with the same connection token. There is no need to delete and then create a new connection when a system loses connectivity to a note pad.

In the context of a note pad, the term *connection* refers to a note pad connection. When referring to a connection to a coupling facility structure, the terms *XES connection* or *IXLCONN connection* are used.

Designing your application to use an XCF Note Pad

The process of designing your application to exploit a note pad involves the following tasks. If you plan to use more than one note pad, you need to perform the tasks listed below for each note pad.

- Study the attribute options for the note pad and the functions provided by the IXCNOTE programming interface. Determine:
  - How your application will exploit the note pad and its functions
  - How you will organize your data in the note pad
  - The note pad attributes you require
  - The name of the note pad
  - The number of notes you require
  - The names of your notes and their content
- Address issues such as serialization that relate to sharing the note pad among multiple users.
- Understand timing issues relating to processing of multiple, concurrent requests.
- Determine how your application will respond when it is unable to create a note because the note pad is either full or constrained. A note pad is full if the maximum number of notes allowed for the note pad exist. A note pad is constrained if a note cannot be created even though the note pad is not yet full.
- Determine how your application will respond when the note pad becomes temporarily unavailable due to conditions such as structure rebuild and loss of connectivity to the coupling facility that contains the note pad. During such periods, your program will not be able to manipulate notes in the note pad. Typically an application will suspend note processing and issue an IXCNOTE pause connection request to determine when normal processing can resume. You might need to document operational or recovery procedures for the installation if your application does not tolerate the temporary loss of access to the note pad.
- Determine how your application will respond when the note pad becomes permanently unavailable due to conditions such as failure of the coupling facility that contains the note pad, or an explicit action taken by the installation to forcibly delete the note pad. For example, you might choose to create a new instance of the note pad, or you might choose to terminate the application. You might choose to perform various appropriate recovery actions. Note that the loss of the note pad can sometimes be prevented if the host structure is duplexed. Consider directing the XCF Note Pad Services to put your note pad in a duplexed structure when creating the note pad.

  You might need to document the consequences of losing the note pad and describe procedures for the installation to use to recover your application.
- Determine the number and nature of your connections
- Determine how your application will respond when a note pad connection fails.

  You might need to document the consequences of losing the connection. You might need to describe the procedures that should be used by the installation to recover your application if such a loss should occur.
- Determine when your note pad connection is to be deleted. The connection should be explicitly deleted by your program as part of its normal shut down procedure. The termination scope of your connection should be defined so that XCF will delete the connection if your program terminates abnormally. Note that the connection will also be deleted by XCF when the note pad is deleted.

  If the connection persists after your program terminates, the installation might need to delete the connection manually. For example, suppose the connection was the only one permitted to have update access to the note pad. If your program was restarted and this old connection still existed, your program would not be able to connect to the note pad with update access. Alternatively, existence of the connection might prevent a new connection from being created due to the limit on the maximum number of note pad connections per address space. Perhaps the installation simply wants to tidy up.

The fact that the connection still exists suggests that the entities identified by the termination scope of the connection did not terminate. Perhaps there are shut down procedures for your application that will cause these entities to terminate. If not, the installation will either need to induce an appropriate termination event or delete the note pad. Provide documentation so that the installation can determine the most appropriate technique for getting the connection deleted. Describe the appropriate application recovery procedures or shut down procedures, if any. You might identify the specific task or address space to be terminated. You might suggest that the note pad be deleted. Document the risks and potential consequences of the suggested actions with respect to your application, particularly if abnormal termination is to be induced or if the note pad is to be deleted. Depending on the business impact, the installation might choose to schedule an IPL of the system to recover from the problem.

- Determine when the note pad is to be deleted. For some applications, the very purpose of the note pad is to retain state information that allows the application to be restarted. In that case, the note pad needs to persist after the application terminates. In other cases, the note pad might be needed only while the application is running. The needs of the application dictate when the note pad is to be deleted. In general, the application should take responsibility for deleting the note pad. If the note pad is not deleted when the application terminates, a mechanism should be provided to enable the installation to delete the note pad when it is no longer needed.

  If your program fails to delete the note pad or does not provide the installation with the means to do so, document the circumstances under which the installation should manually delete the note pad on your behalf. The installation uses the XCF delete utility (IXCDELNP) to delete a note pad. The delete utility deletes a note pad even if it contains notes. By default, the delete utility will not delete the note pad if it has connections. However, the installation can optionally specify that the note pad be deleted even if it has connections. See *z/OS MVS Setting Up a Sysplex* for more information about the XCF deletion utility for note pads. Provide documentation to help the installation understand the risks and consequences of deleting your note pad.
- Document the following information in the installation instructions for your application so that the system programmer can properly configure the sysplex for your note pad. In particular, this information is needed to create profiles for the System Authorization Facility (SAF) and policies for the Coupling Facility Resource Manager (CFRM).
  - Name of the note pad
  - Number of notes required
  - Duplexing preference

  The duplexing preference indicates whether the note pad is to be preferentially hosted in a duplexed CF structure. A duplexed structure offers the potential for improved availability since the data is replicated in two different coupling facilities. A note pad hosted by a duplexed structure will in general survive even if one of the coupling facilities was to suffer an outage.

To use a note pad, your application must first create the note pad (if it does not already exist) and create a connection to the note pad (if a suitable connection does not already exist). Once connected, your application can manipulate notes in the note pad. To access the note pad, the system on which a note pad connector resides must have a direct attachment to the coupling facility that contains the note pad.

The next sections provide detailed information about notes, note pads, note pad connections, as well as the related functions offered by the XCF Note Pad Services. The order in which the topics are presented is not the order in which they would be used by your program. However, this order of presentation is suitable for pedagogical purposes. The following topics are presented:

- What is a note
- What is a note pad
- What is a connection
- Using the IXCNOTE macro
- Note pad requests
- Connection requests
- Single note requests

- Multi-note requests

What is a note

A note in an XCF note pad has the following:

- An 8 byte name
- An 8 byte instance number
- A 16 byte tag value
- Note content, which is either null (no data) or 1024 bytes of data
- A 12 byte connection identifier
- A persistence attribute

When creating a note, the application provides the note name, note content, and persistence attribute. XCF sets the instance number and connection identifier. The creator of the note pad determines who has responsibility for setting the tag value, either the application or XCF.

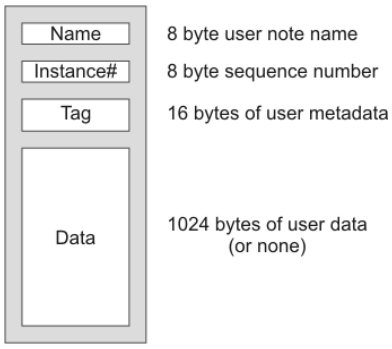


Figure 79. A note in a note pad

Note name

Every note in a note pad has an 8 byte *name* by which it can be identified. The name is assigned by the application when the note is created. Subsequent requests to read, update, or delete the note specify this name to indicate which note in the note pad is to be processed. All the notes that exist in a given note pad at a given point in time will have unique names. Once a note is deleted, its name can be reused for a newly created note.

The designer of the note pad application must determine how to name the notes. There are no restrictions on the content of the note name. Thus the note pad designer is at liberty to use whatever naming scheme best suits the application. Some applications restrict names to alphanumeric characters to make the names human readable. Other applications model the note pad as an array of notes and use the index of the array entry as the name of the note. Some applications would like to use very long names to identify their data, but must devise a translation scheme to convert those long names into an 8 byte note name.

Note instance number

Every note in the note pad has an 8 byte *instance number* that is managed by XCF. The intended use of the instance number is to provide a simple compare and swap like serialization mechanism that enables note pad connectors to make updates to a given note with integrity. XCF assigns a unique instance number to a note when it is created and every time it is updated. When an application attempts to process a note, it can optionally specify the instance number for the instance of the note that is to be processed. If the specified instance number equals the current instance number assigned to the note, XCF processes the request. If the instance numbers are not equal, XCF rejects the request due to an instance number mismatch. When a request is rejected due to an instance number mismatch, XCF returns the current instance number of the note. However, to preserve the integrity of the note, the application will likely need to issue a new read request to fetch the latest copy of the note content from the note pad.

For example, an application might issue a read note request to get the current content of a note and its instance number. Call this *copy1a*. The application might then modify the in-storage copy of the note (call this *copy1b*) and issue a replace request to update the note out in the note pad. However, the content of the note might already have been changed out in the note pad by some other thread. The copy of the note out in the note pad would have a new instance number. Call this *copy2*. Without an instance number comparison, the new replace request issued for *copy1b* would unconditionally overwrite *copy2* of the note, which likely causes a loss of data (the changes written by the other thread). With instance number comparison, the new replace request issued for *copy1b* would be rejected due to an instance number mismatch. The application could then read *copy2* of the note into local storage and reapply its updates in an appropriate manner.

The creator of the note pad determines whether instance number comparisons are required when updating or deleting a note. If so required, a request to update or delete a note is rejected if the requester fails to ask for an instance number comparison. Thus the creator of the note pad can have XCF enforce a design requirement that all note updates be made with instance number comparisons. However, the application must still take responsibility for appropriately refreshing the local in-store copy of the note content before reissuing the request after an instance number mismatch. Without instance number enforcement by XCF, use of instance number comparisons would be achieved as needed through convention within the application. Since not all applications require the use of instance number comparisons to ensure the integrity of the note when making updates, such conventions and enforcement by XCF might not be needed.

Note tags

Every note in the note pad has a 16 byte *tag* value. The creator of the note pad determines who is responsible for setting the tag values, either XCF or the application. If XCF is responsible for setting the tag values, the tags will be ever increasing values set whenever a note is created or updated. If the application is responsible for setting the tag values, the tags are set whenever a note is created, updated, or deleted.

The creator of the note pad can optionally request that XCF track the maximum tag value (*maxtag*). The maximum tag value can be retrieved by issuing a query note pad request. Maximum tag values can be tracked in one of two ways, either *current* or *lifetime*. If tracking current tag values, *maxtag* is the maximum tag value of all the notes that exist in the note pad at the time of the query. If tracking lifetime tag values, *maxtag* is the maximum tag value of all notes that ever existed in the note pad. Thus the lifetime *maxtag* takes into account the tag values of all the notes that currently exist in the note pad at the time of the query, as well as the tag values of all the note instances that were deleted prior to the query.

If the creator of the note pad directs XCF to track the maximum tag value, the tag values set by the application must be nondecreasing for a given note so long as the note exists in the note pad. That is, the tag value set by the application when updating or deleting an existing note must be greater than or equal to the current tag value of the note. If XCF is not required to track the maximum tag value for the note pad, the application can set an arbitrary tag value when updating or deleting an existing note. Regardless of whether XCF is to track the maximum tag value, the application can set an arbitrary tag value when creating a new note.

If the application is responsible for setting tag values, a new tag value can be assigned to a note when the note is deleted. In such cases, XCF will first logically assign the new tag value to the note, and then delete the note. Since the note is deleted, the newly assigned tag value disappears with it. Thus, assigning a new tag value when deleting a note is only meaningful when XCF is tracking maximum tag values. In that case, the new tag value must be greater than or equal to the current tag value of the note. The fact that the delete request is rejected if the proposed tag value is less than the current tag value might be useful to you. For example, it might keep your program from accidentally deleting a newer instance of the note. Furthermore, for a note pad with lifetime tag tracking, the newly assigned tag value could be a new maximum tag value for the note pad. If so, that new tag value would be retained as the new *maxtag* value until such time as a higher tag value was set for some note.

If the application is responsible for setting tag values, the tag provides a way to associate 16 bytes of metadata with the note. For example, the tag could contain control information to describe the type of data contained in the note. One way to use this metadata would be for your program to issue a read notes

request to get the metadata for each note in the note pad. The control information in the tags might then be used to determine which notes are of interest for further processing. For some applications, the tag value could contain all the data required for a (null) note.

If XCF is not tracking the maximum tag value, the tag values have no restrictions and arbitrary values can be set by the application. But if XCF is tracking the maximum tag value, the tag value to be set for an existing note must be greater than or equal to the current tag value of the note. So if you want to use the tag for descriptive control information and want to change that control information when updating an existing note, you need to devise a scheme to ensure that the new tag value adheres to the requirement that it be nondecreasing. For example, you could put a sequence number in the high order bytes of the tag value and increment this number each time the tag is updated. The control data in the low order bytes of the tag value could then be set to arbitrary values without fear that the updated control information in the tag violates the tag sequencing requirements.

If XCF is responsible for setting the tags, the tag value is a 16 byte sequence number that is global to the note pad. The sequence number is incremented every time a note is created or updated in the note pad. Thus the XCF tag values for any given note will be ever increasing, but not necessarily sequential. The XCF tag value might be used, for example, to help manage a check point for a sequence of updates. The XCF tag value could also be used to generate a sysplex wide sequence number. For example, your application could create a note pad consisting of a single note with no content. Any connection to the note pad could issue a request to replace the note. The tag values returned by these replace requests would form an ever increasing sequence.

Note content

A note in a note pad either does or does not have *content*. A note without content is said to be a *null note*. A note with content contains a copy of data supplied by the application. The format and content of these data are determined by the requirements and design of the application. The *note size* is the number of bytes of data that the note contains. Two fixed note sizes are supported, 0 and 1024. A note of size 0 is a null note. A note with content has 1024 bytes of data.

Null notes might be useful in a variety of ways. For example, note content might not be needed at all if some combination of note name, tag value, and perhaps instance number can be used to meet the needs of the application. Alternatively, an application might use a null note to represent some special state such as *logically deleted*. A null note consumes less space in the note pad than does a note with content. Most applications are likely to need notes with content.

To create a note with content, your program issues a create note request and provides a buffer containing the desired note content. If the note already exists, your program can replace the current note content by providing a buffer containing new content when issuing a replace request (or a write request). If the note was a null note, the replace request (or write request) would cause the note to have content.

To create a null note, your program issues a create request but does not provide any content. If the note already exists, your program can convert the note into a null note by deleting the note content (not the note itself). The note content is deleted by issuing a replace request (or write request), and providing a buffer of length zero.

The IXCNOTE macro provides two different ways of specifying *no content*. You can either specify the keyword NOBUFFER, or you can specify the keywords BUFFER and BUFLEN with the buffer length set to zero. When creating a note, either specification causes a null note to be created. However, these specifications have different behaviors when replacing an existing note. Issuing a replace note request with NOBUFFER will update the tag value and instance number, but not the note content. Issuing a replace note request with BUFFER and BUFLEN=0 will update the tag value and instance number, and will also delete the note content (which makes it a null note). A write request behaves like a create request if the designated note does not exist, and behaves like a replace request if the note does exist.

Note connection identifier

Every note in a note pad is associated with some particular connection. The association is first established when the note is created. The association is updated whenever the note is replaced (or written). The

connection that issues the relevant create, replace, or write request is the one that becomes associated with the note. Reading or deleting a note does not change the association.

The associated connection is identified by the unique 12 byte connection identifier assigned to the connection by XCF when the connection is created. Note that the connection identifier is not the same as the connection token. The connection identifier can be obtained from the answer area (if any) returned by the IXCNOTE request that created the connection. The connection identifier of the connection associated with a note can be obtained from the answer area (if any) returned by an IXCNOTE request that processes the note.

As described below, you can provide selection criteria to identify the notes to be processed when calling the XCF Note Pad Services to read or delete a collection of notes. One of the possible selection criteria is connection identifier. Thus your program could issue a request to read (or delete) all of the notes in the note pad associated with a given connection identifier.

Associating a note with a connection can also be useful in the context of connection termination. When a connection is deleted, XCF can find and optionally delete the notes associated with that connection. Depending on the needs of the application, you might want XCF to delete some, none, or all the notes associated with a deleted connection. The persistence attribute of a note determines whether the note is to be deleted by XCF.

Note persistence

Whenever a note is created or replaced, the connection that issues the request can indicate whether the note is to be automatically deleted by XCF when the connection is deleted. That is, the connection to which the note is associated can indicate whether the note is to survive after the connection is deleted. When a connection is deleted, whether due to an explicit request or implicitly as the result of a failure, XCF finds the notes associated with the connection and deletes the ones that were designated as nonpersistent. The notes designated as persistent are left intact. Thus a given note could persist long after the associated connection was deleted.

The persistence attribute and associated connection are intended to help applications accomplish cleanup when a note pad connection is deleted. Since XCF does not notify surviving note pad connections when a peer connection is deleted, your program might not have any impetus to perform cleanup on behalf of the deleted connection. You can use the persistence attribute to designate the notes that need to be cleaned up when a connection is deleted. XCF knows when a connection is deleted. On your behalf, XCF can find and delete the nonpersistent notes associated with the deleted connection.

The design of the application determines whether it is appropriate to have XCF delete some, none, or all the notes associated with a given connection when that connection is deleted. Even if such cleanup is appropriate, it might not be sufficient if the connections maintain local data about their peers. Having XCF delete the notes in the note pad would not accomplish the cleanup of the local data that might be needed when a peer connection is deleted. As applicable, the application needs to implement its own protocol for accomplishing such cleanup.

For example, your program might create a pair of special notes precisely for the purpose of determining whether a connection has terminated. When a connection is created, your program would first create a nonpersistent note to represent the connection, and then create a persistent note to represent the connection. At times of its choosing, the application would read this collection of connection notes. If both notes exist, the connection exists. If only the persistent note exists, the connection was deleted. If only the nonpersistent note exists, the connection is still in the midst of creating these special notes. After the connection terminates, the data in the surviving persistent note could then be used to accomplish whatever application related cleanup was required on behalf of that connection. For example, the note might contain the connection identifier of the subject connection. This connection identifier could then be used to read all the persistent notes associated with the connection. After all the necessary cleanup is completed for those notes, the persistent note used to represent the connection could then be deleted.

What is an XCF Note Pad

An XCF note pad has the following:

- A 32 byte name composed of four 8 byte sections
- A 32 byte description
- A 64 byte static information area
- A note limit
- A duplex preference
- Protocols
- A timestamp of when it was created

When creating a note pad, the application specifies the note pad name, the note pad description, the static note pad information, the desired number of notes, whether duplexing of the host note pad structure is desired, and various protocols that are to be applied to use of the note pad. The protocol choices include indications of whether:

- The number of connections with update access is to be limited to one
- Instance number comparisons are required when connectors update and delete notes
- The connectors are responsible for setting the note tag values
- The maximum note tag value is to be tracked by XCF
- XCF should automatically initiate note pad delete processing when a loss of connectivity to the coupling facility containing the note pad is experienced by all systems with connectors to the note pad (LOSSCONNDELETE protocol)

XCF provides a timestamp to indicate when the note pad was created. This timestamp can be used to identify a particular note pad instance.

XCF considers the note pad name, note limit, and duplex preference when choosing a structure to host the note pad. In general, the note pad name determines the set of structures to be considered, the duplex preference determines the order in which those structures are examined, and the first accessible structure that has space for the requested number of notes is chosen to host the note pad. However, note that the structure selection algorithms might vary according to the installed level of the XCF Note Pad Services. Furthermore, there are cases where XCF will move the note pad to a different structure.

Once created, an XCF note pad persists until it is explicitly deleted or fails. A note pad is explicitly deleted when:

- A program issues the IXCNOTE macro to delete the note pad
- The installation runs the XCF delete utility (IXCDELNP) to delete the note pad

A note pad fails and is implicitly deleted when:

- A logically created note pad cannot be physically instantiated
- The coupling facility containing the note pad fails (and there is no duplexed copy)
- The coupling facility structure containing the note pad fails (and there is no duplexed copy)
- The coupling facility structure containing the note pad is forced (deleted)
- A sysplex outage occurs
- The coupling facility containing the note pad is assigned to a different sysplex
- The XCF note pad catalog fails
- A loss of connectivity to the coupling facility containing the note pad is experienced by all systems with connectors to the note pad and the LOSSCONNDELETE protocol is in effect

Since the note pad persists until it is deleted, the application must in general take responsibility for deleting the note pad when it is no longer needed. If the program does not explicitly delete the note pad, the application should document the circumstances under which the note pad is to be manually deleted by the installation with the XCF delete utility (IXCDELNP).

Note that deleting all the notes in the note pad does not cause the note pad to be deleted. The note pad simply continues to exist in an empty state with no notes. Nor does deleting all the connections to a note pad cause the note pad to be deleted. The note pad simply continues to exist with no connections.

The installation can use the z/OS operator command DISPLAY XCF,NOTEPAD to get information about the note pads that are defined to the sysplex. See *z/OS MVS System Commands* for more information.

XCF issues message IXC472I when a note pad is created and issues message IXC471I if it is unable to create a note pad. Message IXC473I is issued when a note pad is deleted. See *z/OS MVS System Messages, Vol 10 (IXC-IZP)* for more information.

Note pad name

Every note pad has a unique 32 byte name determined by its creator. The note pad name is divided into four 8 byte sections. Each 8 byte section must be left justified, padded on the right with EBCDIC blanks as needed. Each section can contain any upper case alphabetic (A-Z), numeric (0-9), national (@, #, $), or underscore (_) character. The first two sections (owner and application) must not be all blanks. The remaining sections can be all blank. The IXCYNOTE macro defines a mapping for the note pad name (ixcynote_tNotePadName).

The following notation is used for note pad names:

owner.application.function.qualifier

Owner, application, function, and qualifier each represent one 8 byte section of the note pad name. In this notation, the sections within the note pad name are demarcated by a period. XCF also uses this dot qualified format when the note pad name is used in operator commands and messages (blanks are also suppressed). However, when your program composes a note pad name, there is no separator for the sections.

To avoid names used by IBM, do not begin note pad names (*owner*) with the letters A through I or the character string SYS. Names beginning with the string SYSXCF are reserved for use by XCF.

The note pad name must be carefully chosen. To avoid conflicting usage, the note pad name must be unique across all applications that make use of note pads. You might need to account for the possibility that some installations might be running multiple copies of your application within the same sysplex. Since the note pad name is an input to the XCF algorithm that determines which coupling facility structure is to host the note pad, you need to understand the potential consequences of various naming schemes on structure selection for note pad placement. The note pad name is also used by security administrators to establish security profiles that control access to note pad functions and resources. So you need to understand how XCF uses the note pad name when making calls to the System Authorization Facility (SAF) to determine whether a program is permitted to access note pad resources.

Choosing a note pad name

The *owner* and *application* sections of the note pad name are required. The *function* and *qualifier* sections are optional.

The intended purpose of the *owner* section is to provide uniqueness so that the note pad names used by different software vendors will not conflict with each other. For IBM software, the *owner* name typically begins with the component prefix or perhaps SYS*xxx* where *xxx* is the component name (hence the restriction that note pad names beginning with the letters A to I and SYS be reserved for use by IBM). For the OEM software community, various naming conventions are used to avoid conflicts. As described in Standard Packaging Rules for z/OS-based Products (publibz.boulder.ibm.com/epubs/pdf/gimpkg80.pdf), vendors can send a request to element@us.ibm.com to register a prefix (component code) with IBM. The registration ensures that your component code is not used by other products that are also registered. Implementing your application to allow the installation to optionally set the *owner* section of your note pad name provides a mechanism to overcome any potential name conflicts that might arise from products that have not registered with IBM.

The intended purpose of the *application* section is to provide uniqueness so that the note pad names used by different applications from a given software vendor will not conflict with each other. For example, a given software vendor might have two different products, each of which needs to have its own note pad. The vendor could assign each product a unique name for the *application* section of the note pad name.

The intended purpose of the *function* section is to enable a given vendor application to use multiple note pads. For a given application, two or more note pads might be used in support of the various functions or services provided by the application.

The intended purpose of the *qualification* section is to enable a given application function to make use of more than one note pad. Alternatively, this section might be used to distinguish among multiple instances of an application that might be running in the same sysplex. For example, there might be a production version of the application and a test version of the application. Or it might be used to distinguish among different releases of an application that might be running in the same sysplex.

The *owner* section of the note pad name is used by XCF when choosing a coupling facility structure to host the note pad. The *owner* and *application* sections of the note pad are used when calling SAF to verify that the program is authorized to perform a given request. Given these uses, there might be cases where the application provider might choose to incorporate instance or release level information into either the *owner* section of the note pad name, or *application* section, or both, instead of using the *qualification* section for this purpose.

IBM suggests that you implement your application so that the installation has the option of setting your note pad names. A reasonable default name should be provided to simplify note pad configuration for installations that do not need to perform local customization. If you implement your application so that the installation has the option of setting the note pad names, you will maximize the ability of the system programmers to configure your note pads and their note pad structures in a way that best suits the needs and goals of the installation. For example, the option to set the *owner* section would allow the installation to direct the placement of note pads to specific coupling facility structures. The option to set the *qualification* could allow the installation to distinguish production note pads from test note pads.

The installation and configuration documentation for your application should indicate the default names of your note pads and describe how these names can be customized by the installation.

Note pad names and structure selection

The structure names for coupling facility structures to be used for XCF note pads can be of the following forms:

IXCNP_SYSXCF*xx*, and
IXCNP_***owner****xx*

where *xx* is the EBCDIC representation of a hexadecimal number in the range X'00' to X'FF' and ***owner*** is derived from the note pad name in the obvious manner.

When called to create a note pad, the XCF Note Pad Services extract the *owner* section of the note pad name. The Coupling Facility Resource Management (CFRM) policy is queried to determine whether the installation has defined any structures with names of the form IXCNP_***owner****xx*. If so, the note pad will be allocated in one of those structures. The term *owner specific structures* is used to refer to the set of structures with names of this form. If none of the owner specific structures are suitable, the create note pad request is rejected. When creating a new note pad, the set of owner specific structures does not include any structure that is pending delete.

If owner specific structures are not defined for the note pad in the CFRM policy, the note pad is allocated in one of the structures with names of the form IXCNP_SYSXCF*xx*. The term *community structures* is used to refer to the set of structures with names of this form. If no community structures are defined in the CFRM policy, or if none of them are suitable for the note pad, the create note pad request is rejected. When creating a new note pad, the set of community structures does not include any structure that is pending delete.

Thus when choosing a structure to host the note pad, the XCF Note Pad Services exclusively consider owner specific structures if any are defined, and exclusively considers community structures if not. It does not, for example, consider owner specific structures and then move on to community structures if none of the owner specific structures are suitable.

An installation can define owner specific structures, community structures, or some combination of the two. Owner specific structures might be defined for some, none, or all the note pads. Generally owner specific structures are defined only if the installation wants to isolate a particular set of note pads in a

particular set of structures. Thus the *owner* section of the note pad name has a direct bearing on the degree to which the installation can manage note pad placement.

When defining the *owner* section of the your note pad name, consider the following consequences with respect to placement of note pads within owner specific structures. If the installation defines owner specific structures, all of the note pads with the same *owner* will be allocated in one of those structures. In the most extreme case, this set of owner specific structures could consist of exactly one structure. Depending on the choice of *owner*, this group of note pads could be large or small.

- If you specify just your component code as the *owner* for all your note pads, the note pads for all of your applications across all your product suites across all versions and release levels will be placed together in the same set of structures. This naming technique allows a potentially large number of note pads to be placed within a small set of structures and tends to optimize utilization of space within the coupling facility. However, there might be concerns about such issues as placing note pads for the production workload in the same structures as note pads for applications that are under test.
- You might append additional characters to your component code to provide more granularity. For example, you might append characters to create groups of note pads based on product suite, or application, or release level, or perhaps some combination thereof. This technique allows the installation to have finer control of the placement of note pads since the number of note pads in the group is presumably smaller. In the extreme case, the *owner* might be unique for each note pad in which case the owner specific structure would have but one note pad. However, a single note pad per structure does not provide for a particularly efficient use of coupling facility storage.
- You might provide a way for the installation to optionally define part of the *owner* in either of the naming techniques above. Doing so would give the installation the ability to group your note pads in meaningful ways. For example, the installation might then be able to isolate note pads for test versions of the application from the production note pads.

Enabling the installation to optionally specify the entirety of the *owner* section for your note pad names maximizes the ability of the installation to direct placement of the note pads to specific sets of structures as appropriate for the needs of the business. Some installations might even desire, for example, to bundle the note pads of several vendors under the same *owner* so that a disparate collection of note pads could be placed together in a specific structure. However, doing so could cause conflicts since the *owner* was intended to provide the uniqueness needed to guarantee that the note pads of different vendors did not collide. So if you allow the installation to define the entirety of the *owner* section of your note pad name, you likely ought to allow the installation to define the entire note pad name.

Note pad names and SAF authorization

Programs require appropriate SAF (System Authorization Facility) authorization to the FACILITY class resource IXCNOTE.*owner*.*application* when creating, deleting, or querying a note pad, and when creating a connection to a note pad. The *owner* and *application* are derived from the note pad name.

- To create or delete a note pad, the program must have CONTROL access.
- To query a note pad, the program must have READ access.
- To create a connection to a note pad that can be used to create, read, write replace, or delete notes, the program must have UPDATE access.
- To create a connection to a note pad that can only read notes, the program must have READ access.
- In cases where XCF does not otherwise recognize the program as being a valid user of a connection, the program must have READ access to read notes in the note pad, and must have UPDATE access to create, write, replace, or delete notes in the note pad.

If your program runs unauthorized, the installation must define a SAF profile that grants your program the access it needs to be able to issue its IXCNOTE requests for your note pad. If SAF is not installed, or no SAF profile is defined for your note pad, your program will not be able to use the note pad. Otherwise, XCF honors whatever decision is returned by SAF. If SAF determines that your program has the required access, XCF permits the IXCNOTE request to be processed. If SAF determines that your program does not have access, XCF rejects the IXCNOTE request.

If your program runs authorized, XCF calls SAF to determine whether your program has been granted the access needed to issue its IXCNOTE requests. If SAF is not installed, or the installation has not defined a SAF profile for your note pad, XCF permits the request to go forward (because it is running authorized). However, if your program is permitted to create a note pad connection in this manner, your program must be running authorized when it issues an IXCNOTE request for that connection. If SAF is installed and the installation has defined a SAF profile for your note pad, XCF honors whatever decision is returned by SAF. If SAF determines that your program has the required access, XCF permits the IXCNOTE request to be processed. If SAF determines that your program does not have access, XCF rejects the IXCNOTE request.

The installation and configuration documentation for your application should indicate what the installation needs to provide in the way of SAF profiles to enable your application and its component programs to have appropriate access rights.

Note pad description

The creator of the note pad provides a 32 byte description that is intended to help installations and service personnel understand the function, purpose, or role of the note pad. The description will appear in various XCF messages and diagnostic data reports.

Note pad information

The creator of the note pad can provide 64 bytes of data that is to be associated with the note pad. A copy of the data is visible to other processes in the sysplex via queries that return information about the note pad. The content and interpretation of this data is determined by the creator of the note pad. The intended purpose is to provide an easy way for the creator of the note pad to make application specific control data available to the programs that use the note pad. For example, the data could be used to document application protocols that users of the note pad are to follow. Because the data is fixed when the note pad is first created, it is not suitable for state information that might need to be updated over the life of the note pad.

Note limit

The creator of the note pad indicates the number of notes needed for the note pad. In general, this number will be the maximum number of notes that need to reside in the note pad at the same time.

Note Pad Services can be used to request a dynamic modification to increase or decrease the number of notes that can reside in a note pad after a note pad is created. For example, a request to increase the number of notes allowed for the note pad may be issued in response to a note pad full condition detected on a note request.

The note limit is one of the factors considered by XCF when choosing a CF structure to host the note pad. Structures that do not appear to have enough available space to accommodate the requested number of notes will be excluded from the candidate list. The create request is rejected if there is no candidate structure with enough space for the requested number of notes.

XCF rejects a note request if it would cause the number of notes in the note pad to exceed the specified note limit. In such cases, the note pad is said to be full. There could also be situations where XCF is forced to reject a request that creates a new note even though the note pad is not full. In these cases, the note pad is said to be constrained. See “Constrained conditions” on page 720 for more information.

Duplexing preference

The creator of the note pad can indicate a preference as to whether the note pad should be hosted in a coupling facility structure that can be duplexed through System Managed CF Structure Duplexing. There is no guarantee that the specified preference can be satisfied. Even if the preference is satisfied initially, there is no guarantee that the preference will remain in effect for the life of the note pad since the installation can dynamically change the Coupling Facility Resource Management (CFRM) policy to enable and disable duplexing.

See Chapter 6, “Connection Services,” on page 203 for more information about System Managed CF Structure Duplexing. See *z/OS MVS Setting Up a Sysplex* for more information about what the installation must do to configure structure duplexing. However, as a note pad exploiter, you need only concern yourself with stating a preference for duplexing. The XCF Note Pad Services provide the code needed to support exploitation of structure duplexing as described in Chapter 6, “Connection Services,” on page 203.

A duplexed structure will generally provide greater availability since the second copy makes it more resilient to failure than a simplex structure which only has one copy. However, a simplex structure will generally provide faster note request response times than a duplexed structure since the note operations do not incur the overhead of replicating changes in two structures. The requirements of the exploiting application determine which option is to be preferred. In general, factors such as the impact of losing the note pad and the time needed for recovery actions to restore the application to normal service must be considered. However, it is ultimately up to the installation to determine whether the robust recovery capability of a duplexed structure is worth the various costs.

The XCF Note Pad Services do not provide information as to whether the note pad resides in a duplexed structure, nor does it report changes to the duplexing state of the note pad structure. Programs that run authorized can use the XCF Query Service (IXCQUERY) to determine whether the coupling facility structure is duplexed. The name of the structure that currently hosts the note pad is returned in the answer area stored by the XCF Note Pad Services in response to a query note pad request.

Note pad protocols

The creator of the note pad determines the various protocols that are to be applied to the note pad. Choices need to be made with respect to the following protocol options.

Multi-write access

The creator of the note pad specifies the MULTIWRITE keyword to indicate whether the number of connections having update access to the note pad is to be restricted. If MULTIWRITE=NO is specified, only one connection in the entire sysplex is allowed to have update access to the note pad. If such a connection exists, a request to create a second connection with update access is rejected. If the existing connection with update access is deleted, whether explicitly by the application or implicitly by XCF when the connector terminates, a new connection with update access can be created. If MULTIWRITE=YES is specified, more than one connection can be created with update access to the note pad.

For example, an application might use MULTIWRITE=NO to simplify the serialization of changes to notes in the notepad. If there is but one connection capable of creating, replacing, and deleting notes, the application might not need code to account for the conflicts that could arise when multiple competing connections manipulate the same note. However, note that MULTIWRITE=NO does not prevent multiple work units from using the same connection token simultaneously. If single threading is needed for serialization, you might need to devise a mechanism to ensure that the application has but one work unit issuing note requests.

Instance number comparison

When a note pad connector calls the XCF Note Pad Services to replace, write, or delete an existing note, an instance number comparison can optionally be performed to ensure that the correct instance of the note is being manipulated. The creator of the note pad can specify the INSTCOMP keyword to indicate whether connections are required to perform instance number comparisons when updating and deleting notes. This protocol applies only to requests that replace, write, or delete a single note. When writing a note, the protocol applies only in cases where the note already exists (in which case the write request is processed as a replace request). Since multi-note requests do not support instance number comparisons, they are not subject to this protocol.

If INSTCOMP=DISCRETIONARY is specified by the creator of the note pad, instance number comparisons are optional. When a connection issues a request to replace, write, or delete a note, XCF will process the request regardless of whether an instance number comparison was specified. An instance number comparison will be performed if specified, otherwise not.

If INSTCOMP=REQUIRED is specified by the creator of the note pad, instance number comparisons are required. When a connection issues a request to replace, write, or delete a note, XCF rejects the request if the connector fails to provide a nonzero instance number for comparison. When replacing or deleting a note, the request is immediately rejected if an instance number of zero is specified. When writing a note, the request proceeds. If the note exists, the write is processed as a replace and the request is rejected if the instance number is zero.

Typically, the note pad creator might insist that instance number comparisons be performed as a safeguard to help ensure that the application implementation adheres to certain conventions. Requiring instance number comparisons might help ensure that the application always detects cases where the note was changed out in the note pad. Ostensibly the intent is to prevent the accidental loss of note data. However, simple detection of a stale copy of the note is not sufficient. The application will likely need to read the note and reapply the desired changes to the latest copy of the note before it reissues the rejected replace request.

Note tag assignment

The TAGGING keyword indicates whether XCF or the application is responsible for setting note tag values. Every note in the note pad has a 16 byte tag value. The creator of the note pad determines whether the application is responsible for setting the note tag values, or whether XCF is responsible for setting them. When a single note request is issued, the TAGGING keyword must be specified to confirm who has responsibility for setting the tag values. The note request is rejected if the TAGGING specification does not match the TAGGING specification made by the creator of the note pad. This check ensures that the creator of the note pad and the users of the note pad exploit the same tagging protocol.

If XCF is responsible for setting the tag values, the tags will be ever increasing values set whenever a note is created or updated. If the application is responsible for setting the tag values, the tags are set whenever a note is created, updated, or deleted.

Tracking of maximum tag values

The creator of the note pad specifies the TRACKTAG keyword to indicate whether XCF needs to track the maximum note tag value. If tag values are to be tracked, TRACKTAG also indicates the set of notes to be considered when determining the maximum tag value. The maximum tag value can be obtained by issuing a query note pad request with an answer area. As determined by the TRACKTAG specification, XCF will either not report the maximum tag value at all, report the maximum tag value of all the notes that exist in the note pad at the time of the query, or report the maximum tag value assigned to any note that ever existed in the note pad. The answer area stored by the query note pad request has a flag to indicate whether the content of the field containing the maximum tag value is valid for use.

In cases where the user is responsible for assigning tag values (TAGGING=USER), the TRACKTAG specification can lead to additional considerations for your program. Some TRACKTAG specifications impose requirements that must be met when a note request sets the tag value. Some TRACKTAG specifications have the potential to create additional overhead for note processing.

The various possible choices for maximum note tag value tracking are:

- Do not track maximum tag value
- Track current notes
- Lifetime tracking

Do not track maximum tag value (TRACKTAG=NO)

XCF need not track the maximum tag value. Query note pad requests will not report a maximum tag value. The answer area will always indicate that the maximum tag value is not available.

For user assigned tags (TAGGING=USER), any value can be assigned to the tag when writing, replacing, or deleting a note. In particular, the new tag value for the note can be less than, equal to, or greater than the current tag value of the note.

Requests to create, replace, write, or delete a note will not incur any additional overhead related to tracking the maximum note tag value.

Track current notes (TRACKTAG=CURRENT)

XCF is to track the maximum tag value. Query note pad requests will report the maximum tag value of all the notes that exist in the note pad at the time of the query. If the note pad does not have any notes at the time of the query, the answer area will indicate that the number of notes is zero and the maximum tag value is not available. Note however, that the tag value might not be available because the note pad was not accessible at the time of the query. In this case, the answer area will indicate that neither the number of notes nor the maximum tag value are available.

For user assigned tags, the tag value assigned to an existing note must be greater than or equal to the current tag value of the note. A request to write, replace, or delete an existing note is rejected if the new tag value is less than the current tag value of the note.

Requests to create, replace, write, or delete a note will not incur any additional overhead related to tracking the maximum note tag value.

Lifetime tracking (TRACKTAG=LIFETIME)

XCF is to track the maximum tag value. Query note pad requests will report the maximum tag value of all the notes that ever existed in the note pad at the time of the query. To accomplish this, XCF maintains a record of the maximum tag value ever assigned to a note. This record is initialized to zero when the note pad is created. When a note is deleted from the note pad, XCF updates this record as needed. The maximum tag value reported by a query note pad request will either be the maximum tag value of all the notes that exist in the note pad at the time of the query, or this retained maximum tag value, whichever is greater. If the note pad does not have any notes at the time of the query, the retained maximum tag value is reported. If the note pad never had any notes, the maximum tag value is reported as zero since the retained tag value is initialized to zero. Note that the maximum tag value might not be available because the note pad was not accessible at the time of the query.

For user assigned tags, the tag value assigned to an existing note must be greater than or equal to the current tag value of the note. A request to write, replace, or delete an existing note is rejected if the new tag value is less than the current tag value of the note.

When XCF is responsible for setting tag values, there is no additional overhead associated with tracking the maximum tag value for the life of the note pad.

However, tracking the maximum tag value for the life of the note pad is not without cost when your application is responsible for setting tag values. When deleting a note, XCF might have to defer deletion of the note until the maximum tag value can be recorded. In such cases, the note is said to be *pending delete*. The delete is said to be a *deferred delete*. XCF marks the note as *logically deleted* and launches asynchronous processing to physically delete the note after the maximum tag value is appropriately recorded. The existence of a note that is pending delete might impact your program in the following ways:

- A note that is pending delete is still included in the count of the total number of notes in the note pad until such time as the note is physically deleted. So a pending delete could be observed as an unexpectedly high note count being returned by a query note pad request.
- A note that is pending delete might collide with a subsequent note request, causing that new request to incur additional overhead to resolve the pending delete.

  For example, consider a create note request. Since a note that is pending delete still exists in the note pad, the coupling facility would reject the create note operation. XCF recognizes that the existing note is pending delete. Logically, the create request should have worked, and it would have worked had the asynchronous XCF processing finished deleting the note in time. Rather than waiting for the asynchronous processing to complete, XCF finishes the deferred delete while running under the create note request thread. After doing so, it resends the create operation to the coupling facility. Functionally, the create note request works as expected. However, it incurred additional overhead because it was issued before the asynchronous XCF processing launched by the prior delete note request was able to physically delete the note. Not to worry, the asynchronous XCF processing will not delete the newly created instance of the note if it should happen to run (finally!) after the new note was created in the note pad.

From a functional perspective, the implementation of your program is not impacted by notes that are pending delete. XCF automatically detects and resolves any conflicts that might arise due to the

physical existence of the logically deleted note. From a performance perspective, your program is not impacted if the asynchronous XCF processing finishes deleting the note before a subsequent request tries to process the same note. In general, this will likely be the timing. In cases where it does not work out that way, the response time for the requests that collide with a logically deleted note might be elongated.

LossConnDelete

The creator of the note pad specifies the LOSSCONNDELETE keyword to indicate whether XCF should automatically initiate note pad delete processing when a loss of connectivity to the coupling facility containing the note pad is experienced by all systems with connectors to the note pad. If LOSSCONNDELETE=NO is specified or defaulted to, then XCF will not automatically initiate note pad delete processing. If LOSSCONNDELETE=YES is specified, then XCF will automatically initiate note pad delete processing when a loss of connectivity to the coupling facility containing the note pad is experienced by all systems with connectors to the note pad.

An application might use LOSSCONNDELETE=YES, if the application would like to perform recovery processing when a loss of connectivity to the coupling facility containing the note pad is experienced by all systems with connectors to the note pad. Without this protocol it would be difficult for the application to determine the scope of the loss of connectivity among the systems with connectors to the note pad and thus be difficult to start any recovery procedures. Delete processing provides a trigger point to fence access to the note pad and allow applications to perform their recovery procedures. An example recovery procedure could be failing over to a backup note pad contained in another coupling facility.

Timestamp when created

XCF provides a 16 byte timestamp to indicate when the note pad was created. This timestamp can be used to identify a particular instance of a note pad.

For programs that might have competing threads creating and deleting a given note pad, the timestamp can be specified on the delete note pad request to ensure that the intended instance of the note pad is deleted.

What is a connection

An XCF note pad Connection has the following:

- A 32 byte connection token
- A 12 byte connection ID
- A 32 byte description
- A 64 byte static information area
- An access scope
- A connection scope
- A usage classification
- A termination scope

To access the notes in a note pad, your program must first create a connection to the note pad. When creating a connection, the application indicates the name of the note pad to which the connection is to be established. It also provides a description of the connection, static information about the connection, and an indication of whether the connection is to be used strictly for reading notes or whether it is to be used to update notes in the note pad.

XCF sets the connection token and connection ID upon successful creation of the connection.

The connection scope is derived by XCF from the security environment that exists when the connection is created. The connection will either have address space scope or task scope.

The usage classification indicates how the application intends to use the connection. The primary purpose of the usage classification is to provide a way for authorized programs to create connections that can be used by a user that is not the connector.

The application specifies the termination scope to bind the connection to a particular task or address space. If the indicated task or space terminates, the connection is deleted by XCF.

In general, an application must create one connection for each address space that needs to access the note pad. XCF limits the number of connections that can be created for a given address space. The limit depends on the installed level of the XCF Note Pad Services. See "Note pad related limits" on page 748 for more information.

Note: A connection created for either server usage or client usage needs but one connection from the server address space, even though the note pad can be accessed from work units that originate in an arbitrary address space.

Once created, a note pad connection persists until it is explicitly deleted or fails. A connection is explicitly deleted when a program issues the IXCNOTE macro to delete it. A connection fails and is implicitly deleted when any of the following activities occur:

- The note pad is deleted
- The note pad fails
- The connector address space terminates
- The connector task terminates (if the connector has task scope)
- The task or address space designated by the termination scope terminates
- The connector system terminates

XCF does not provide a programmatic way to get information about peer note pad connections. If an application requires awareness of its connections, it must devise its own means of discovering this information. You might consider using notes in your note pad to do this. For example, create a nonpersistent note to represent your connection. You might establish a convention of using a tag value of zero for these notes alone. A multi-note read request with tag selection criteria for tag value zero would then return a note for each connection. Since a nonpersistent note is automatically deleted by XCF when the connection terminates, the returned notes would only include the current connections.

Connection token

When a connection to a note pad is successfully created, XCF constructs a 32 byte connection token to represent the connection. When the create request returns from the XCF Note Pad Services, the IXCNOTE macro expansion stores this connection token in the storage area designated by the CONNECTION keyword. A copy of this token must be passed on subsequent IXCNOTE requests that manipulate the connection or access notes in the note pad. Note that the requester must carefully preserve the connection token because the create request is the only opportunity to acquire the token. XCF does not make the token available via any other service or IXCNOTE request.

Connection identifier

When a connection to a note pad is successfully created, XCF constructs a unique 12 byte connection identifier to represent the connection. The connection identifier is not the same as the connection token. The connection identifier can be obtained from the answer area (if any) returned by the IXCNOTE request that created the connection. When a note is created or replaced in the note pad, the connection that issued the request becomes associated with the note. The connection identifier is used to identify the associated connection. The connection identifier of the connection associated with a note can be obtained from the answer area (if any) returned by an IXCNOTE request that processes the note.

When reading or deleting a collection of notes, you can provide selection criteria to identify the notes to be processed. One of the possible selection criteria is connection identifier. Thus your program could issue a request to read (or delete) all of the notes in the note pad associated with a given connection identifier.

Connection description

The creator of the connection provides a 32 byte description that is intended to help installations and service personnel understand the function, purpose, or role of the connection. The description will appear in various XCF messages and diagnostic data reports.

Connection information

The creator of the connection can provide 64 bytes of data that is to be associated with the connection. The content and interpretation of this data is determined by the creator of the connection. The intended purpose is to provide an easy way for the creator of the connection to make application specific control data available to the programs that use the note pad. For example, the data could be used to document application protocols supported by the connector. Because the data is fixed when the connection is first created, it is not suitable for state information that might need to be updated over the life of the connection.

Access scope

When a connection is created, the requester indicates how the notes in the note pad are to be accessed by the connection. The connection can either have read access or it can have update access. A connection with *update access* can create, write, replace, read, or delete notes in the note pad. A connection with *read access* can read notes in the note pad, but is not permitted to create, write, replace or delete notes.

Programs require appropriate SAF authorization to create a connection to a note pad. The access scope of the connection determines the type of authorization required. A connection to be created with update access to the note pad must be authorized for UPDATE access. A connection to be created with read access to the note pad must be authorized for READ access. See “System Authorization Facility (SAF) requirements” on page 718.

The creator of the note pad can request that there be at most one connection to the note pad with update access. If the note pad already has one such connection, a request to create a new connection with update access is rejected. See “Multi-write access” on page 707.

Connection scope

A connection will either have task scope or address space scope. If the task that creates the connection has its own security environment (TCBSENV is nonzero), the connection has *task scope*. For a connection with task scope, the task that creates the connection is said to be the connector task. If the connection is created by an SRB, or by a task that does not have its own security environment (TCBSENV is zero), the connection has *address space scope*. In either case, the home address space of the work unit that creates the connection is said to be the *connector address space*.

The *connector* is deemed to be any work unit that has the same security environment as the work unit that created the connection. In all cases, the home address space of the work unit must be the connector address space. In addition:

- For a connection with address space scope, the work unit must either be an SRB or it must be a task that does not have its own security environment.
- For a connection with task scope, the work unit must be the connector task.

In particular, a connection with address space scope can have multiple work units qualify as the connector. For a connection with task scope, only the connector task qualifies as the connector.

The phrase "requester is the connector" is often used to denote a work unit that is recognized as being the connector. A connection token is always valid for use if the requester is the connector.

The connection scope, in conjunction with the usage classification, determine when a work unit can validly use a connection token. See “Use of a connection token” on page 719.

Usage classification

The usage classification determines the conditions under which a work unit is deemed to be a valid user of the connection for note processing. See “Use of a connection token” on page 719 for more information. The creator of the connection specifies the USAGE keyword to indicate the manner in which the connection is to be used by the application. Three styles of usage are supported: connector, server, and client.

Connector usage

The connection is intended for use by work units that are deemed to be the connector. See “Connection scope” on page 712 for more information.

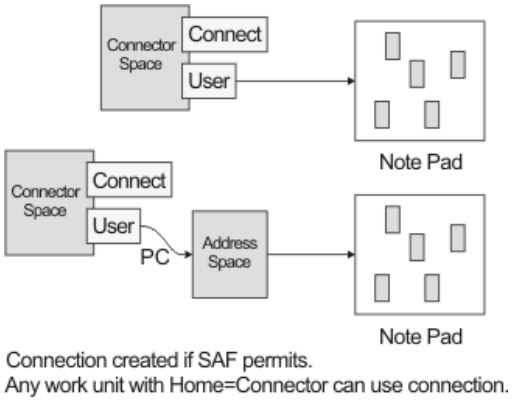


Figure 80. Note pad connection with USAGE=CONNECTION

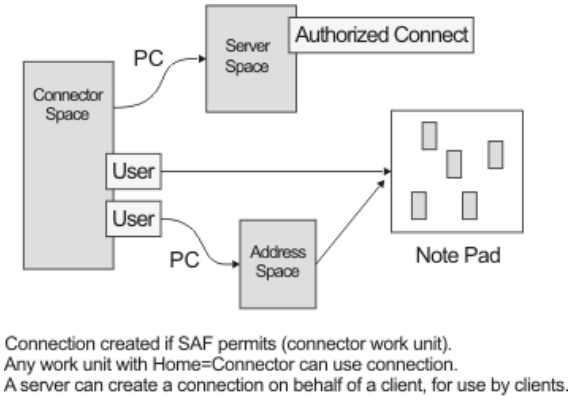


Figure 81. Creating a note pad connection on behalf of a client

When a work unit that is not running authorized creates a connection, the home address space and primary address space must be the same. A work unit running authorized can create a connection for connector usage while running in cross memory mode under the client work unit. For example, a server might receive control from a client via a space switch PC. The server creates the connection on behalf of the client. It is intended that the connection be used by the client, perhaps directly or perhaps via server routines that run out of the client space. The connection is associated with the home address space and the SAF checks are performed using the security environment of the client work unit. A server might create the connection in this manner (as opposed to creating the connection while running in the client space) in order to establish a termination scope that binds the connection to the server address space. Thus if the server address space or some designated task in the server address space terminates, the connection would be terminated as well.

Server usage

The connection is created by an address space that provides a service that can be called by other programs. When the service is called, it needs to access notes in the note pad while running under the

caller's work unit. Since the home address space of the work unit that calls the service will not normally be the server address space, the work unit is not deemed to be the connector and so would not qualify as a valid user of the connection. However, with a connection created for server usage, the work unit is permitted to access notes in the note pad when it is running authorized and the primary address space is the connector address space. In practice, the service might receive control via a space switch PC.

Only an authorized program can create a connection for server usage. Furthermore, the work unit that uses the connection in this manner must be running authorized. When the connection is created, the home address space and primary address space of the requesting work unit must be the same space (the server address space). When the connection is used, the primary address space of the work unit must be the server address space (the connector address space).

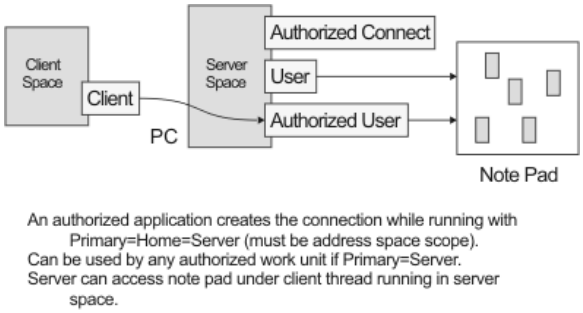


Figure 82. Note pad connection with USAGE=SERVER

In this style of use, the note pad is created by the server for its own purposes. It is intended for use by the server when running authorized in the server address space under a client work unit. The server should not normally give the caller direct access to the notes in the note pad. If it does so, the server is responsible for ensuring that the caller has the appropriate SAF authorization for such access. In cases where SAF authorization would be needed or the service routine needs to run unauthorized, it might be more appropriate for the server to create a connection for connector usage on behalf of the client (as described under “Connector usage” on page 713).

As an alternative, you could create a connection with connector usage and still provide a service as described above. For example, the service might receive control via a space switch PC. In cases where the calling work unit was not deemed to be the connector, the service routine would not be permitted to access notes in the note pad. To access the note pad, your program could queue work to a task in the server address space that is deemed to be the connector. That task would be permitted to use the connection to access the note pad. The requesting work unit might be suspended until the task completed its work. A connection created for server usage allows for synchronous access to the server note pad under the calling work unit, avoiding the complexity and overhead of an asynchronous protocol.

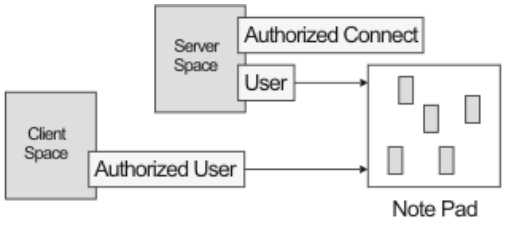
Note that the USAGE=SERVER connection can also be used by the server itself in any context where USAGE=CONNECTOR type accesses would be permitted. That is, had the connection been created with USAGE=CONNECTOR, work units originating from the server address space would have been permitted to use the connection. Any work unit that would have qualified for such usage qualifies as a valid user of the USAGE=SERVER connection. The work unit does not need to run authorized.

Client usage

The connection is created by an address space that provides a service that can be called by other programs. When the service is called, it needs to access notes in the note pad while running under the caller's work unit. Since the home address space of the work unit that calls the service will not normally be the server address space, the work unit is not deemed to be the connector and so would not qualify as a valid user of the connection. However, with a connection created for client usage, the work unit is permitted to access notes in the note pad when the work unit is running authorized. In practice, the service might receive control via a non-space switch PC.

Only an authorized program can create a connection for client usage. Furthermore, the work unit that uses the connection in this manner must be running authorized. When the connection is created, the home

address space and primary address space of the requesting work unit must be the same space (the server address space). When the connection is used, there are no requirements with respect to primary and home address spaces.



An authorized application creates the connection.
Any authorized application can use the connection.
Allows server to access note pad while running in the client space.

Figure 83. Note pad connection with USAGE=CLIENT

In this style of use, the note pad is created by the server for its own purposes. It is intended for use by the server when running authorized in an arbitrary address space under a client work unit. The server should not normally give the caller direct access to the notes in the note pad. If it does so, the server is responsible for ensuring that the caller has the appropriate SAF authorization for such access. In cases where SAF authorization would be needed or the service routine needs to run unauthorized, it might be more appropriate for the server to create a connection for connector usage on behalf of the client (as described under "Connector usage" on page 713). Note that the connection would be deleted, and therefore no longer usable by clients, when the server address space terminates.

As was the case for a connection for server usage, a connection for connector usage could similarly be used to provide the service. The service routine would suspend the caller and process the notes asynchronously under a work unit running in the server address space that is deemed to be the connector. A connection for client usage permits synchronous access to the note pad and avoids the complexity and overhead of an asynchronous protocol.

Note that the USAGE=CLIENT connection can also be used by the server itself in any context where USAGE=CONNECTOR type accesses would be permitted. That is, had the connection been created with USAGE=CONNECTOR, work units originating from the server address space would have been permitted to use the connection. Any work unit that would have qualified for such usage qualifies as a valid user of the USAGE=CLIENT connection. The work unit need not be running authorized.

Termination scope

The termination scope identifies a task to which the connection is to be associated for the purposes of termination processing. If the indicated task terminates, XCF deletes the connection. A connection is also deleted when the connector address space terminates, or when the connector system terminates. For a connection with task scope (TCBSENV is nonzero), the connection is also deleted when the connector task terminates.

A connection cannot be associated with a task or address space that has terminated, or that is in the midst of being terminated.

The connection can be bound to the task that issued the create connection request. Alternatively, the connection can be bound to any task in the Task Control Block (TCB) chain that runs from the requester task up to the job step program task. In other words, the connection can be bound to the connector task, or to the parent task of the connector task, or to the parent task of that task, and so on up to and including the job step program task. If the requester indicates that the connection is to be bound to an address space, the connection will in fact be bound to the task that owns the cross-memory resources for the address space (the TCB for this task is anchored in the ASCBXTCB field of the address space control block which is mapped by IHAASCB).

If a program is running authorized when it issues the request to create the connection, the connection can be bound to an arbitrary task in the home address space. If the authorized program is running

cross-memory mode with primary not equal to home when it creates the connection (which implies USAGE=CONNECTOR), the connection can be bound to an arbitrary task in the primary address space.

Using the IXCNOTE macro

The XCF Note Pad Services are available on systems running z/OS V2R1 and up, or on systems running z/OS V1R13 with APAR OA38450 installed. XCF Note Pad Resiliency functions which include support for REQUEST=NOTEPAD REQTYPE=MODIFY and LOSSCONNDELETE for REQUEST=NOTEPAD REQTYPE=CREATE are available on systems running with APAR OA60571 installed. Your program might need to determine whether the XCF Note Pad Services and XCF Note Pad Resiliency functions are available for use. To do so, issue the IXCQUERY macro specifying REQINFO=FEATURES to obtain data that indicates whether the XCF Note Pad Services and XCF Note Pad Resiliency function are available. See "Using the IXCQUERY Macro" on page 98 for more information.

Use the IXCNOTE macro to call the XCF Note Pad Services. Mappings of relevant data areas are defined in the IXCYNOTE mapping macro. The following requests are supported:

- Note Pad Requests
  - Create a note pad (see "Create note pad" on page 726)
  - Query a note pad (see "Query note pad" on page 727)
  - Modify a note pad (see "Modify note pad" on page 728)
  - Delete a note pad (see "Delete note pad" on page 729)
- Connection Requests
  - Create a connection (see "Create connection" on page 732)
  - Pause a connection (see "Pause connection" on page 733)
  - Resume a connection (see "Resume paused connection" on page 734)
  - Delete a connection (see "Delete connection" on page 733)
- Single Note Requests
  - Create a note in a note pad (see "Create note" on page 738 and "Write note" on page 739)
  - Replace a note in a note pad (see "Replace note" on page 738 and "Write note" on page 739)
  - Read a note from a note pad (see "Read note" on page 739)
  - Delete a note in a note pad (see "Delete note" on page 740)
- Multi-Note Requests
  - Read a collection of notes from a note pad (see "Read notes" on page 744)
  - Delete a collection of notes in a note pad (see "Delete notes" on page 747)

The IXCNOTE macro expands to fill in a parameter list and call the service routine for the XCF Note Pad Services. The service routine processes the request while running in the XCF address space and returns data to your program in one or more of the following ways:

- Setting a return and reason code to indicate whether the request completed successfully, or if not, why the request was rejected.

  Depending on the request type and the needs of your application, you might need special consideration for certain return and reason codes.

  - See "Quiescing conditions" on page 720 for cases where the note pad is temporarily unavailable.
  - See "Constrained conditions" on page 720 for cases where a note cannot be created because the coupling facility structure is out of space.
  - See "Timeout conditions" on page 721 for cases where a note pad request or a connection request times out.
  - See "Status unknown conditions" on page 723 for cases where XCF is unable to determine what happened.

  - See "XCF component failures" on page 725 for cases where XCF fails.
- Storing request results in an answer area.

  The storage for the answer area is provided by your program. For additional information, see "Answer area" on page 717.

  For most requests, the answer area is optional and the needs of your application will determine whether one is used. However, even if your program does not need the answer area for normal processing, be aware that XCF might store diagnostic data in the answer area for certain failures. Consider providing an answer area to capture this data for problem analysis. For example, if XCF were to reject a multi-note request because your selection criteria data area is not formatted correctly, the data stored in the answer area will indicate the specific problem.
- Storing note content in a buffer area.

  Depending on the request, data will either be fetched from the buffer, or stored into the buffer. The buffer contains the content of one or more notes. When reading multiple notes, data records in the answer area indicate the location in the buffer area where the content of the relevant note can be found. The size of the buffer must meet certain conditions, which vary according to the type of request. The specific conditions are described with each request type. In general, the buffer size must be some multiple of the maximum note size supported by the note pad.
- Storing information in storage areas identified by certain IXCNOTE keywords.

  Depending on the request specifications, the IXCNOTE macro expansion might copy data from these storage areas into the IXCNOTE parameter list. While processing the request, the XCF Note Pad Services might store information into the IXCNOTE parameter list. Upon return from the service routine, the IXCNOTE macro expansion copies data from the parameter list into the designated storage areas. This behavior applies to storage areas identified by the following IXCNOTE keywords:

  - CONNECTION
  - INSTANCE#
  - RESUMETOKEN
  - TAG

  Special care might be needed when coding these keywords as the IXCNOTE macro expansion unconditionally copies data from the parameter list into the storage areas designated by these keywords. Depending on the request result, the data stored might not be meaningful. In particular, you might need to maintain a separate copy of the data to preserve the original value in cases where the request fails. For example, this saved copy of the data might be used to reinitialize the storage area designated by the relevant keyword before the request is reissued.

Answer area

The storage for the answer area is provided by your program. For most requests, the answer area is optional. If provided, the answer area always contains a header record. Sometimes the header record is the only information stored in the answer area. If additional information is stored, the answer area header indicates where an array of data locator records can be found within the answer area. Depending on the request and the size of the answer area, zero or more data locator records are provided. Each entry in the data locator array identifies the location of an array of data records. These data records contain the functional output of the request. Both the data locators and the data records themselves are stored in the answer area. If the request is rejected, diagnostic data might be stored in the answer area header to provide detailed information about the failure. An answer area is optional for most requests, but desirable given the potential need for this detailed diagnostic data.

If the answer area provided by your program is not large enough to hold all of the relevant output data, the result will depend on the size of the answer area and the type of request.

- If an answer area is not provided for a request that requires one, the request is rejected.
- If the answer area is not large enough to hold the 64 byte header record, the request is rejected. No information is stored in the answer area. As appropriate, reissue the request with an answer area at

least as long as the header record. Depending on the request, additional storage might be needed. Some requests require an answer area that is longer than the header record.

- If the answer area is smaller than the minimum size required for the request, the request is rejected. In this case, the minimum amount of answer area storage needed to process the request is stored in the answer area header. As appropriate, reissue the request with an answer area at least as long as the indicated minimum amount. Depending on the request, more storage might be desirable. For some requests, an answer area larger than the required minimum allows more information to be returned on each call, thus reducing the total number of calls that must be made to get all the data.
- If the answer area is smaller than the size needed to hold all of the available information, the request could complete with a return and reason code indicating that some of the data was stored in the answer area and that more information is available. In this case, the amount of answer area storage needed to get all of the available information is stored in the answer area header. As appropriate, reissue the request with an answer area at least as long as the indicated amount. For some requests, the amount of information can vary with the dynamics of the system. In such cases, an answer area larger than the indicated amount might be desirable so as to allow room for potential growth in the amount of information to be returned.
- If the answer area is smaller than the size needed to hold all of the available information, the request could complete with a return and reason code indicating that the request should be reissued to get the remaining data. In these cases, the request is intentionally designed for iterative execution. Each time the request is issued, XCF stores as much information as will fit in the provided answer area.

System Authorization Facility (SAF) requirements

For the IXCNOTE requests listed below, your program needs appropriate SAF authorization to the FACILITY class resource IXCNOTE.*owner*.*application*, where *owner* and *application* are derived from the note pad name. See "Note pad name" on page 703 for more information about note pad names. The installation instructions for your application should document the names of your note pads so the security administrator can define these profiles.

If SAF is installed and a SAF profile is defined for the relevant class of note pads, the decision returned by SAF is always honored by XCF. If your program does not have SAF authorization, the IXCNOTE request is rejected.

If SAF is not installed, or if a SAF profile is not defined for the relevant class of note pads, XCF determines whether your program is running authorized. If your program is not running authorized, the IXCNOTE request is rejected. If your program is running authorized, the request is allowed to proceed.

The following IXCNOTE requests require SAF authorization:

- Create note pad requires CONTROL access
- Query note pad requires READ access
- Modify note pad requires CONTROL access
- Delete note pad requires CONTROL access
- Create connection requires:
  - READ access if the connection access scope is read
  - UPDATE access if the connection access scope is update

  See "Access scope" on page 712 for information about the connection access scope. As noted above, an authorized program can create a connection even if SAF is not installed or there is no SAF profile defined for the note pad. However, note that such a connection can only be used by programs that are running authorized.

In general, XCF does not perform SAF checks for requests that process notes in the note pad (single note and multi-note requests). Similarly, XCF does not generally perform SAF checks for delete connection requests. However, XCF might perform a SAF check when the security environment of the requesting work unit appears to differ from the security environment of the work unit that created the connection. For example, if a connection has address space scope, a SAF check might be performed if the work unit

that issues the request has its own security environment. For a connection with task scope, a SAF check might be performed if the work unit that issues the request is not the connector task. If a SAF check is performed, the program must have access that is appropriate for the type of request being issued:

- Read note requires READ access
- Create, write, replace, or delete note requires UPDATE access
- Delete connection requires:
  - READ access if the connection was created with access scope of read
  - UPDATE access if the connection was created with access scope of update

Use of a connection token

A connection token is a required input for most connection requests and all requests that manipulate notes in a note pad. When your program issues one of these requests, it must be running in a context where use of the connection token is deemed valid by XCF. Valid contexts are a function of the execution environment, request type, and in some cases, connection attributes (usage classification, access scope, and connection scope). See “What is a connection” on page 710 for more information about note pad connections and their attributes. In particular, the phrase "requester is the connector" is defined in “Connection scope” on page 712.

A connection token can only be used on the system that created the connection. XCF rejects an IXCNOTE request that uses a connection token from some other system.

When issuing a *pause connection* request or a *resume connection* request, the connection token is valid for use by any work unit that can satisfy at least one of the following conditions:

- The requester is the connector
- The home address space is the connector address space
- The primary address space is the connector address space
- The program is running authorized

When issuing a *delete connection* request, the connection token is valid for use by any work unit that can satisfy at least one of the following conditions:

- The requester is the connector
- The home address space is the connector address space and the requester has SAF authorization appropriate for the type of access specified when the connection was created
- The program is running authorized

When issuing a *single note* request or a *multi-note* request, the connection token is valid for use by any work unit that can satisfy at least one of the following conditions:

- The requester is the connector
- The home address space is the connector address space and requester has SAF authorization appropriate for the type of request:
  - A request to read a note must have READ authority.
  - A request to create, write, replace, or delete a note must have UPDATE authority.
- The work unit is running supervisor state or PKM allowing key 0 to 7, and either:
  - The connection was created with USAGE=CLIENT, or
  - The connection was created with USAGE=SERVER and the primary address space is the connector address space, or
  - The work unit is running as an address space resource manager.

Quiescing conditions

In cases where the coupling facility structure hosting the note pad becomes temporarily inaccessible, a single note request and a multi-note request will complete with a return and reason code indicating that the note pad is quiesced. A note pad can be quiesced, for example, if the local system loses connectivity to the coupling facility that contains the note pad or if the structure hosting the note pad is being rebuilt. The duration of the quiesce is indeterminate. A structure rebuild might complete in a few seconds, or it might not complete for several minutes. A loss of coupling facility connectivity might be resolved in a few seconds, or it might require manual intervention that could take minutes or hours to resolve.

Details about the quiescing condition might be stored in the answer area (if any). If present, the details are mapped by ixcynote_tDetailsQuiesced. Various flags indicate the current set of conditions for which the note pad is quiesced. One or more conditions could apply. Depending on the needs of the application, your program might take different actions based on the presence or absence of specific conditions.

In general, you should anticipate that a note pad will be quiesced at some point in its life since structure rebuild is a standard tool used by installations to manage coupling facility structures. Thus most applications should make reasonable attempts to tolerate a quiesced note pad, at least for a short while. If your application is not able to tolerate a quiescing condition, the documentation for your program should provide guidance to the installation. The installation needs to understand what actions should be taken to accommodate your application prior to rebuilding the hosting note pad structure for a planned maintenance action. The installation also needs to understand what actions should be taken to recover your application when the hosting note pad structure is rebuilt as the result of an unexpected failure, as opposed to a planned action.

You must decide how your application will respond when a note pad is quiesced. The needs of the application will determine what is most appropriate. Different actions might be taken for different notes. In some cases, it might be appropriate to apply combinations of actions. In general, one or more of the following actions might be appropriate.

- **Ignore** the failure. This action might be appropriate if the request is expendable, or if it will be reissued at a later time through normal operation. This action might also be appropriate if the failure is to be surfaced to a higher level in the calling sequence of your program.
- **Reissue** the request. Since the note pad might remain quiesced for several seconds or more, your program should allow time for the quiescing condition to clear before it reissues the request. Consider limiting the number of such attempts as a safeguard so that your program will not be continually be reissuing requests if the note pad should happen to remain quiesced for an unduly long period.
- **Wait** for the note pad to become accessible. The note pad connection service can be used to pause some particular work unit until the note pad becomes accessible, or until a specified timeout expires. Waiting for the note pad to become unquiesced can be advantageous to both the system and your application as compared to repeatedly reissuing the request. Reissuing the request while the note pad remains quiesced needlessly consumes system resources. Waiting any amount of time to reissue the request beyond when the note pad becomes accessible needlessly delays processing.
- **Delete** the connection. This action might be appropriate if the note pad should persist even though the application is to be terminated or the application is to continue without use of the note pad.
- **Delete** the note pad. This action might be appropriate if the application is to be terminated or if the application will continue operation without use of the note pad.

Constrained conditions

A note pad is said to be *full* if the number of notes in the note pad equals the note limit specified by the creator of the note pad. When a note pad is full, no new notes can be created. A note pad is said to be *constrained* if XCF is unable to create a new note even though the note pad is not full. For example, a note pad can become constrained if the coupling facility structure that hosts the note pad runs out of space. XCF generally attempts to manage the note pad structures so that note pads will not encounter constraints. However, the dynamics of the sysplex and various choices made by the installation can thwart these attempts. Thus you need to determine how your program will respond if an attempt to create a note is rejected for either of these conditions.

Full considerations

If a note cannot be created because the note pad is full, consider deleting one or more notes from the note pad to free up some note space and reissue the request. You might also consider reissuing the request, perhaps after a short delay, if you believe the dynamics of your note pad are such that notes are likely to be deleted through normal activity.

Alternatively, you might try to increase the capacity of your note pad by issuing IXCNOTE REQUEST=NOTEPAD REQTYPE=MODIFY to increase the number of notes that the note pad is allowed to hold (See "Modify note pad" on page 728). If dynamically increasing the note pad capacity is unsuccessful due to note pad host structure constraints (e.g., the MODIFY request fails with a reason code ixcnoteRsnNo#NotesStrResources), manual intervention might be required to resolve the constraint condition. For example, the installation might need to update the CFRM policy to change the size of the host note pad structure or use the IXLALTER service to expand the size of the structure. Although disruptive, you might choose to delete the note pad and create the note pad again with a higher note limit in order to increase the capacity of an existing note pad. XCF will attempt to locate the newly created note pad in a host structure with sufficient space to satisfy the higher note limit. Deleting the note pad could be disruptive to your application. Adding another note pad likely increases the complexity of your program. Note that there is no guarantee that a new note pad can be created successfully.

Constraint considerations

If a note cannot be created because the note pad is constrained, consider deleting one or more notes to free up some note space and reissue the request. Deleting notes will probably resolve the constraint, but not necessarily. When a note pad is constrained, note pads can interfere with each other in the sense that they compete to claim whatever scarce structure storage is available for notes. The storage consumed by the note that you delete could be claimed for a note in a different note pad before you are able to reissue your request.

Furthermore, a note pad can transition between the constrained and not constrained states as the result of various external actions. For example, structure alter, structure rebuild, structure create, structure delete, or even the creation and deletion of notes and note pads could affect the constrained condition. So deleting a note from a constrained note pad might not free up note space if the system takes the unused storage away from the note pad structure through alter processing.

Reissuing the request is certainly reasonable in the constrained case even if the dynamics are such that notes are not likely to be deleted from your note pad through normal activity. When a note pad is constrained, your reissued request might work as the result of notes getting deleted from other note pads in the structure.

As a last resort, you might delete the note pad and create it anew. Alternatively, you might create another note pad for your program to use. Note that there is no guarantee that a new note pad can be created successfully. However, a newly created note pad will not be constrained. If XCF cannot find a structure with enough space to accommodate all the notes requested by all the note pads that reside in the structure, the create request is rejected. Deleting your note pad could be disruptive to your application. Adding another note pad likely increases the complexity of your program.

Pending deletes

If the creator of the note pad requires XCF to perform lifetime tracking of the maximum note tag value, the note pad might contain notes that are pending delete. See "Note pad protocols" on page 707 for more information. When the note pad appears to be full or constrained, XCF initiates processing to physically delete the notes in a pending delete state and resends your note operation to the coupling facility (possibly more than once). Thus the existence of notes in a pending delete state will not generally be a contributing factor when XCF rejects a request because the note pad is full or constrained.

Timeout conditions

Note pad requests and connection requests have the potential to be long running. Depending on the request and the state of the system, it might take several seconds, perhaps minutes for a request to

complete. These requests are processed asynchronously under a task in the XCF address space. The work unit that issues the request is suspended while XCF processes the request.

As it processes the request, XCF might need to establish a XES connection (using the IXLCONN macro) from the local system to one or more coupling facility structures. I/O accesses to the Coupling Facility Resource Manager (CFRM) Couple Data Set (CDS) might be needed. The performance of these I/O accesses to the CDS can be impacted by a number of factors, including size of the CDS and contention. In some cases, portions of the request could be processed by other systems in the sysplex, who in turn might need to establish their own XES connections to the relevant structures. All of this processing is transparent to your program, except possibly for the elapsed time.

When issuing the IXCNOTE macro to process a note pad request or a connection request, your program can specify the TIMEOUT keyword to limit the amount of time that your work unit remains suspended waiting for the request to complete. Given that these requests can be long running, you should in general provide generous timeout values. The default timeout values used by XCF can vary with the installed level of the XCF Note Pad Services. Typically the default XCF timeout values are on the order of several minutes.

If the timeout value is too low, your work unit might be resumed before the results of the request are known. If XCF determines that the request was either not processed, or that it can be safely reissued, the return and reason code will indicate that the request timed out. Otherwise the return and reason code will likely indicate that the result of the request is unknown (see "Status unknown conditions" on page 723). For the timeout case, consider reissuing the request, possibly with a longer time out value.

Note the following for note pad requests:

- Create, modify and delete note pad are not considered by XCF to be requests that can be safely reissued. If these requests were to be reissued after a timeout, a reissued create request might be rejected because the note pad was in fact created by the original request. Similarly, a reissued delete note pad request might be rejected because the note pad was in fact deleted by the original request. A reissued modify request might be rejected because a previous modify request may still be in progress. Under the assumption that your program would treat these rejections as unexpected errors, XCF returns a status unknown condition instead of a timeout condition.
- Query note pad is considered by XCF to be a request that can be safely reissued.

Note the following for connection requests:

- Create connection requests can be safely reissued if they time out. Either the connection was not created at all, or XCF backed it out. The connection does not exist.
- In general, pause connection requests are issued to either wait for a quiescing condition to be resolved, or to wait for the pending delete of a connection to finish. Pause connection requests are by their very nature long running since they are intended to wait for these conditions to clear, potentially for as long as the specified timeout value.
- Resume connection requests are not considered by XCF to be requests that can be safely reissued. There are cases where the resume request times out even though the paused connection is in fact successfully resumed. A reissued resume request could then complete with a return and reason code indicating that the resume is pending. Under the assumption that some program might not expect this result, XCF returns a status unknown condition instead of a timeout condition.
- Delete connection requests are not considered by XCF to be requests that can be safely reissued. If this request were to be reissued after a timeout, the reissued delete connection request might be rejected because the connection was in fact deleted by the original request. Under the assumption that your program would treat this rejection as unexpected error, XCF returns a status unknown condition instead of a timeout condition.

  Note that the time needed to delete a connection could be impacted by the total number of notes in the note pad, the number of notes associated with the connection, and the number of nonpersistent notes that need to be deleted on behalf of the connection.

Status unknown conditions

Your IXCNOTE request could complete with a return and reason code indicating status unknown. A status unknown condition occurs when XCF is unable to determine the result of the request. In general, status unknown conditions can occur as the result of a timeout condition or a loss of connectivity to a coupling facility (CF).

For example, consider the case where the local system loses connectivity to the CF while in the midst of processing a create note request. Connectivity might be lost before the request could be sent to the CF. Connectivity might be lost after the request was sent to the CF but before the results could be received by the local system. If the request made it to the CF, it might have been processed successfully or it might have been rejected by the CF. Any of the following results are possible:

1. The note does not exist because the create note operation was never processed by the CF
2. The create note operation successfully created the note
3. The create note operation was rejected by the CF because the note already existed.

XCF reports status unknown because the loss of connectivity to the CF makes it impossible to determine which of the results actually occurred. Suppose your program immediately reissues the create note request in response to the status unknown condition. Any of the following are possible:

- Connectivity to the CF is not yet restored. The request is rejected with a return and reason code indicating that the note pad is quiesced.
- The request succeeds, ostensibly because case (1) appears to have been the actual result of the request that completed with status unknown. However, the successful create is also consistent with results (2) and (3) if some other thread deletes the note before you reissue the create request. These distinctions might be important to your program. In the case of result (2), your note would have been created twice.
- The request is rejected because the named note already exists. This rejection is consistent with either result (2) or (3). These distinctions might be important to your program. In the case of result (2), you would proceed as if the original request had been successful since the note was created as intended. In the case of result (3), you would proceed as if the original request had encountered an already existing instance of the note. You might need additional recovery actions to determine which course of action is appropriate.

You must decide how your application will respond to a status unknown condition. A variety of strategies might be appropriate depending on the request and the needs of the application. In general, you might:

- **Ignore** the failure. This action might be appropriate if the request is expendable, or if it will be reissued at a later time through normal operation. This action might also be appropriate if the failure is to be surfaced to a higher level in the calling sequence of your program.
- **Reissue** the request. This action might be appropriate if the actual (unknown) result of the original request is immaterial to how your program processes the result of the reissued request. For example, a request that uses an instance number comparison when replacing a note would likely have correct behavior regardless of what happened with the original request. Alternatively, this action might be appropriate if your program can use the results of the reissued request, possibly in conjunction with additional investigation, to determine the correct behavior. In the create note example above, if the reissued create request is rejected because the note already exists, a read note request might be issued so that the content of the note could be examined to determine whether the existing note was in fact the one created by the original request that failed with status unknown.
- **Investigate** the failure and proceed appropriately. This action might be appropriate if you can take recovery actions to determine the actual result of the request that failed with a status unknown condition. Based on that determination, your program could then proceed in an appropriate manner. For example, you might issue a read note request after a failed create note request to see if the note exists, and reissue the create request if not.

In the case where the status unknown condition arises as the result of losing connectivity to the CF that contains the note pad, it is quite likely that any requests issued as part of resolving the status unknown

condition will themselves be rejected for a quiescing condition. If so, your recovery actions will need to be deferred until connectivity is restored.

Considerations for note pad requests

If you were creating a note pad, the note pad might or might not have been created. If you were deleting a note pad, the note pad might or might not have been deleted. If you were modifying the number of notes that a note pad is allowed to hold, the update might or might not have been made effective. Your program might need to take a recovery action to determine how to proceed. For example, you might issue a query note pad request to see if the note pad exists (for a create or delete) or whether a modify request is pending and what the current number of notes allowed value is for the note pad and then proceed appropriately. If you simply reissue the request, your program might need to deal with additional failure conditions. For example, a second create note pad request might be rejected because the note pad already exists; a second delete note pad request might be rejected because the note pad no longer exists; a second modify may be rejected because a previous modify is in progress.

If you reissue a delete note pad request, consider specifying ETODCREATED to ensure that the correct instance of the note pad is deleted. If the original request actually deleted the note pad, some other thread might create a new instance of the note pad before the delete request can be reissued. Without the ETODCREATED specification, the reissued delete note pad request would unintentionally delete this new instance of the note pad.

A query note pad request can always be reissued.

Considerations for connection requests

A create connection request should never see a status unknown condition since XCF cancels the connection to be sure it does not exist in cases where the connection cannot be reliably established.

A delete connection request can be reissued because a connection token uniquely identifies a connection. There is no danger of accidentally deleting the wrong connection when the request is reissued. Your program would need to tolerate a return and reason code indicating that the connection did not exist.

In the case of a pause request, the fact that your program is running implies that you are not paused. If conditions warrant, reissue the pause.

In the case of a resume request, your program might be able to determine whether the work unit that is the intended target of the resume has in fact resumed. If not, you can likely reissue the resume. If you choose to do so, realize that the resume has a “pre-resume” flavor. If the second resume turns out to be unneeded because the paused work unit had already been resumed, the second resume will be retained by XCF. The next work unit to issues a pause connection request will be immediately resumed. The code that receives control after its pause connection service returns might therefore need to account for the fact that it was resumed prematurely.

Considerations for single note requests

When creating, writing, replacing, or deleting a note, your program might need to take a recovery action to determine how to proceed. For example, you might issue a read note request to see if the note exists. In the case of a create, write, or replace request, you might also check to see if the note contains the expected content. As needed, reissue the request. If instead you simply reissue the request, your program might need to deal with additional failure conditions. For example, a second create note request might be rejected because the note already exists, having been successfully created by the original request; a second delete note request might be rejected because the note no longer exists, having been successfully deleted by the original request.

A read note request can simply be reissued as needed.

Considerations for multi-note requests

A multi-note read request can be reissued. There are no notes to process as partial results are not returned when the read fails. The resume token is not advanced, so the same collection of notes remain available for selection when the read request is reissued.

A multi-note delete request might have deleted none, some, or all of the selected notes. If MAXTAG was specified, the designated tag value might not have been stored as the new maximum tag value for the note pad (applies to note pads that require lifetime maximum tag tracking of user assigned tag values). To ensure that all of the intended notes are deleted, reissue the delete notes request.

XCF component failures

In extremely rare cases, your request might complete with a return and reason code indicating that XCF itself failed. XCF will have already arranged for the gathering of appropriate diagnostic data. In general, it is reasonable to proceed as if the request completed with status unknown. If the problem occurs repeatedly or your program is otherwise unable to make progress, it might be appropriate for your program to gather diagnostic data to document the impact of the failure from the perspective of your application. Additional recovery actions such as deleting the connection or deleting the note pad might be warranted.

In the particular case of a create connection request, XCF will have arranged (as needed) for the connection to be cancelled. The cancellation of the failed create request is processed asynchronously and might not be accomplished immediately. Your program can simply reissue the create connection request, though the number of attempts should be limited to avoid the possibility of repeated failures. In general, the new request is likely to create the new connection successfully. However there might be some exceptions if XCF is still in the midst of cancelling the failed create request.

- Suppose the creator of the note pad specified MULTIWRITE=NO so that there could be at most one connection with update access to the note pad. Suppose that you are trying to create a connection with update access. It could be the case that the failed create request will appear to be the one and only one connection permitted to have update access to the note pad. If so, your reissued create connection request might be rejected until XCF is able to finish cancelling the failed request.
- The number of connections to any given note pad from any given address space is limited. It could be the case that the failed create request will appear to consume one of those slots. If so and all the slots are used, your reissued create connection request might be rejected until XCF is able to finish cancelling the failed request.

If the failure occurs when deleting a note pad, it might be necessary for the installation to use the XCF Delete Note Pad Utility (IXCDELNP) to manually delete the note pad.

Note pad requests

Issue the IXCNOTE macro with REQUEST=NOTEPAD to manipulate the note pad as a whole. Such a request is often simply called a *note pad request*. Use the NOTEPAD keyword to specify the name of the note pad to be processed. The note pad can be created, queried, modified, or deleted. The note pad must be created before connections can be created, and connections must be created before your application can manipulate notes in the note pad.

Use the REQTYPE keyword to indicate the type of operation to be performed for the note pad. Refer to the following material for specific information on each request type:

- For REQTYPE=CREATE, which is used to create a new note pad, see “Create note pad” on page 726.
- For REQTYPE=QUERY, which is used to get information about a note pad, see “Query note pad” on page 727.
- For REQTYPE=MODIFY, which is used to get information about a note pad, see “Modify note pad” on page 728.
- For REQTYPE=DELETE, which is used to delete an existing note pad, see “Delete note pad” on page 729.

When your program issues a note pad request, the requesting work unit is suspended. In general, the note pad request is processed asynchronously under a task in the XCF address space. The optional keyword TIMEOUT can be used to control how long the work unit remains suspended waiting for results. In general, the default values used by XCF can be used. See "Timeout conditions" on page 721 for additional information.

Your program needs appropriate SAF authorization to be able to create, query, modify, or delete a note pad. See "System Authorization Facility (SAF) requirements" on page 718 for more information.

In general, the XCF Note Pad Services need to have access to the XCF catalog of note pads and the coupling facility that hosts the note pad in order to process a note pad request. Depending on the request and the level of the XCF Note Pad Services installed on the various systems in the sysplex, the XCF Note Pad Services might be able to process a request even if it does not have direct access to these entities.

Create note pad

Issue a create note pad request to create a note pad. An answer area is optional. If an answer area is provided, it must be large enough to allow a note pad data record to be stored.

The create request is rejected if the note pad already exists. Otherwise, the XCF Note Pad Services read the CFRM Policy to determine the set of coupling facility structures that are candidates for hosting the note pad. The duplexing attributes and current duplex state of the candidate structures are extracted from the policy. These attributes are then used to sort the candidate structures so that the structures will be examined in an order that honors the structure duplexing preference specified by the create note pad request. The structures are then examined in turn until one is found with enough space for a note pad with the requested number of notes. The duplexing preference of the creator might not be satisfied if none of the preferred structures have space for the note pad, or if none of the structures have the desired duplexing attributes.

To successfully create the note pad, an entry must be added to the XCF note pad catalog, the note pad itself must be initialized in the host structure, and the catalog must be updated to indicate that the note pad is ready for use. In general, the XCF Note Pad Services successfully perform these operations and the note pad is functional upon return from the create request.

However, the host structure or the catalog might become inaccessible after the note pad is entered in the catalog. In such cases, the note pad is defined but not yet functional because the create is incomplete. The terms *logically created*, *being created*, and *create pending* are used to refer to a note pad in this state. Regardless, the create note pad request returns with a return code indicating success and your program can proceed to create connections to the note pad. XCF automatically finishes the create of the note pad when it becomes possible to do so. In some cases, XCF might later discover that the create of the note pad cannot be completed. If so, XCF fails the note pad.

If the create request completes successfully, a note pad data record is stored in the answer area if one is provided. Most of the information in the note pad data record reflects the parameters and options specified on the create request. However, the note pad data record also includes the timestamp of when the note pad was created. Your program might later need this timestamp when it deletes the note pad. The data record also includes a flag to indicate whether the note pad is still in the midst of being created or whether the create is complete.

If an answer area is not provided with the create note pad request, a query note pad request can be issued with an answer area at a later time to obtain the information. However, in cases where you need the timestamp of when the note pad was created, the data reported by this future query will be for the note pad instance that exists at that point in time. It might not be the same instance of the note pad that you created.

As a result of the create note pad request, XCF might issues messages. By default, these messages are written to the hard copy log.

- Message IXC472I is issued when a note pad is created. If the create is pending, there could be two instances of the message. The first instance indicates that the note pad is in the midst of being created. The second instance is issued when creation of the note pad is completed. Whichever system in the

sysplex happens to finish the pending create issues the message to indicate that the note pad is created and ready for use.

- If the note pad cannot be created in response to a valid create note pad request, message IXC471I is issued to explain why. In general, this message is issued when the create note pad request is rejected due to an environmental problem. It is not typically issued when the request is rejected due to programming errors.

Query note pad

Issue a query note pad request to get information about a note pad. An answer area is optional. If an answer area is provided, it must be large enough to allow a note pad data record to be stored. If a suitable answer area is provided, a note pad data record and zero or more system connection data records are stored in the answer area.

The note pad data record provides information about the note pad definition and the current state of the note pad. For example, the data record contains information to describe the attributes and protocols specified by the creator of the note pad. It also contains a count of the current number of notes in the note pad, a flag indicating whether the note pad is constrained, and the name of the coupling facility structure that hosts the note pad at the time of the query. If the note pad is not accessible, some of the state information might not be reported. In such cases, validity flags in the data record indicate whether the data is present.

System connection data records provide information about the systems that appear to have connections to the note pad. For example, the data record contains the name of the system and flags to indicate whether the system has any connections to the note pad. If the note pad was created with MULTIWRITE=NO, the data record has flags to indicate which systems have connections with read access and which system has the one connection with update access. The system that gathers the data always inserts a system connection data record for itself, regardless of whether that system actually has any connections to the note pad. The data record has a flag to indicate which system gathered the data.

If an answer area is not provided, the query is in effect testing for the existence of the note pad. If the note pad is defined, an otherwise valid query request completes with return code zero. If the note pad is not defined, the query completes with a return and reason code indicating that the note pad does not exist.

If an answer is provided, it must be large enough to allow a note pad data record to be stored. Thus a minimal answer area must have enough space for the answer area header, one data locator, and one note pad data record. If not, the request is rejected. If the answer area is not big enough for the header, the request is simply rejected. Otherwise, the number of bytes of storage needed to receive all of the available information is stored in the answer area header. This size accounts for the note pad data record as well as the number of system connection data records available at the time of the query.

If the answer area is large enough for the note pad data record, but not large enough to hold at least one system connection data record, an otherwise valid query request stores the note pad data record in the answer area. The number of bytes of storage needed to receive all of the available information is also stored in the answer area header. If system connection data records were available but not stored in the answer area, the query request completes with a return and reason code indicating that more data is available.

If the answer area is large enough for the note pad data record and at least one system connection data record, an otherwise valid query request stores in the answer area, the note pad data record and as many system connection data records as will fit. Such an answer area will have enough space for the answer area header, two data locators, one note pad data record, and at least one system connection data record. The number of bytes of storage needed to receive all of the available information is also stored in the answer area header. If system connection data records were available but not stored in the answer area, the query request completes with a return and reason code indicating that more data is available. If all of the available system connection data records were stored, the query request completes with return code zero.

Note: All offsets are relative to the start of the answer area header.

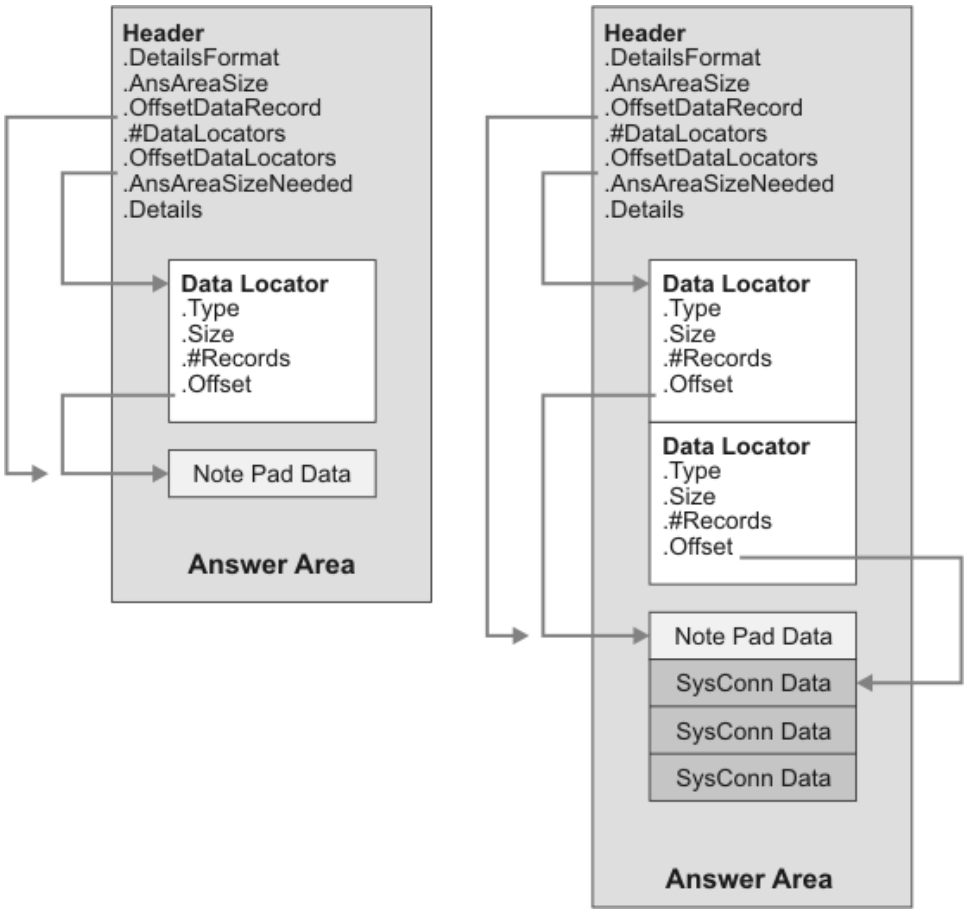


Figure 84. Answer areas for query note pad

Modify note pad

Issue a modify note pad request to modify the number of notes (#NOTES attribute) that a note pad is allowed to hold after creating a note pad. You may, for example, want to request an increase in the number of notes that a note pad is allowed to hold in response to a note pad full condition or prior to an expected need for increased note pad capacity during times of high demand. Upon successful completion of the request, the requested number of notes allowed becomes effective immediately. An answer area is optional. If an answer is provided, it must be large enough to hold the answer area header.

To successfully implement and complete a dynamically effective increase or decrease to the number of notes that a note pad is allowed to hold, XCF Note Pad Services must reflect the requested change in the XCF note pad catalog and the host structure where the note pad resides. In general, several tasks must be accomplished to propagate the change in notes allowed for a note pad. For example, XCF might need to fence the note pad in the host structure to temporarily prevent note processing while decreasing the number of notes that a note pad is allowed to hold, and update the XCF note pad catalog and note pad multiple times during the course of processing an increase or decrease modify request. Usually, XCF Note Pad Services successfully perform these operations as needed and the update to the #NOTES attribute is effective by the time the service routine returns control to your program.

However, XCF might not be able to perform some or all of these tasks if the note pad host structure or the note pad catalog structure is not accessible. If XCF Note Pad Services successfully initiated the modify request, XCF will automatically finish the necessary updates and complete the modify request when the note pad host structure and note pad catalog structure become accessible. In such cases, the modify request might complete with a return and reason code indicating:

- **MODIFY request pending completion**. XCF Note Pad Services successfully initiated the MODIFY request, but could not complete all the necessary tasks associated with completing the modify request.

The request will complete when the necessary updates are made in the note pad host structure and note pad catalog structure by a system that has access to the structures. No other modify request can be initiated until the current pending modify request completes successfully. An action to consider taking when receiving a return and reason code indicating that the request is pending completion is using a connection request to pause and resume the connection when XCF completes the MODIFY request. XCF Note Pad Services can also be used to query the note pad to determine whether a MODIFY request is pending against the note pad.

- **Status unknown**. The status of the modify request is unknown. The modify request might have completed updating the number of notes the note pad is allowed to hold, might not have ever been started, or might be pending. Consider using a query note pad request to determine the state of the note pad. For example, query the note pad to determine whether a modify request is in progress and what the current number of notes is that the note pad is allowed to hold. See "Status unknown conditions" on page 723.

If the modify request completes successfully and an answer area is provided, information returned in the answer area header details field includes the maximum number of notes that the note pad is currently allowed to hold as a result of the modify request, the number of notes that currently exist in the note pad and note pad host structure information that may be useful to the

application.

Delete note pad

Issue a delete note pad request to delete a note pad. An answer area is optional. If an answer is provided, it must be large enough to hold the answer area header. The delete request is rejected if the note pad does not exist.

You can optionally specify the timestamp when the note pad was created to ensure that the correct instance of the note pad is deleted. The note pad can be deleted conditionally or unconditionally. With a conditional delete, you can prevent the note pad from being deleted if it contains notes, or has connections, or both.

If the note pad contains notes when the note pad is deleted, the notes are deleted along with the note pad. If connections to the note pad exist when the note pad is deleted, the connections are also deleted. Depending on the timing as to when the deletion is recognized, connections deleted in this manner might have their IXCNOTE requests rejected for a variety of reasons. For example, a note request could be rejected because the specified note does not exist, or because the connection does not exist, or because the note pad does not exist.

In general, several tasks must be accomplished to delete the note pad. For example, XCF might need to fence the note pad in the host structure to prevent any further note processing, fence the note pad in the XCF catalog to prevent new connections, delete any connections that still exist in the sysplex, release the resources consumed by the note pad in the host structure, and remove the relevant entry from the XCF note pad catalog. Usually the XCF Note Pad Services successfully perform these operations as needed and the note pad is physically deleted by the time the service routine returns control to your program. However, XCF might not be able to perform some or all of these tasks if the note pad structure or the XCF catalog of note pads is not accessible.

In general, the delete request is deemed successful when XCF is able to fence the note pad to prevent note requests from being processed and new connections from being created. The note pad might not be physically deleted, but it is *logically deleted*. The physical existence of a logically deleted note pad is transparent to your program. A query note pad request would indicate that the note pad does not exist and a new note pad could be created with the same name. As needed, XCF will automatically finish the physical deletion of the note pad when the host structure and catalog become accessible. Although the physical existence of a logically deleted note pad will not directly impact your program, there could in some cases be an indirect impact. For example, the create of a new note pad might be rejected if the XCF catalog appears to be full because the logically deleted note pad still consumes an entry.

If the host structure or the XCF catalog is not accessible, XCF might not be able to fence the note pad. In such cases, the delete request might complete with a return and reason code indicating:

- **No system resources**. In this case, no progress was made. The note pad still exists. It has not been marked for deletion. In some cases, reissuing the request with a longer timeout value might allow the delete to make progress. Otherwise, reissue the delete request at a later time. Alternatively, your program might alert the installation regarding the need to intervene manually through use of the XCF delete utility (IXCDELNP) after the accessibility issues are resolved.
- **Delete pending**. The delete of the note pad is pending. The note pad was fenced in the host structure, but not in the catalog. The note pad exists both physically and logically, but it is not usable for note requests. XCF automatically finishes the delete of the note pad when circumstances allow. Your program could be impacted by a note pad in this state.

  In the meantime, you might be able to create a new connection, but it will not be able to process any notes. You will not be able to create a new instance of the note pad until XCF logically deletes the note pad. Note requests will be rejected, generally with a return and reason code indicating that the note pad is quiesced. If an answer area was provided for the note request, the quiesce details will indicate that the note pad is being deleted. If your program needs to wait for the note pad to be deleted, consider issuing an IXCNOTE pause connection request to wait for the note pad to be deleted.
- **Status unknown**. The status of the delete is unknown. The note pad might have been deleted, the delete might be pending, or it might be that no progress was made. Consider using a query note pad request to determine the state of the note pad. If it still exists, reissue the delete request. See "Status unknown conditions" on page 723 for more information.

  As a result of the delete note pad request, XCF might issues messages. By default, these messages are written to the hard copy log. Message IXC473I is issued when a note pad is deleted. If the note pad is logically deleted, two messages might be issued. The first message will indicate that the note pad is in the midst of being deleted. Whichever system in the sysplex finishes the pending delete will issue the message to indicate that the delete of the note pad has completed.

Designating a unique note pad instance

When deleting a note pad, consider using the ETODCREATED keyword to ensure that the intended instance of the note pad is deleted. Specify the 16 byte timestamp of when the note pad was created. When a note pad is created, XCF stores this timestamp in the answer area that was provided for the request (if any). The timestamp is also stored in the answer area in response to a query note pad request as well as a create connection request. For programs that might have competing threads concurrently creating and deleting a given note pad, the timestamp can be specified on the delete request to ensure that the intended instance of the note pad is deleted.

If ETODCREATED is not specified, or if the specified timestamp is zero, the currently defined instance of the note pad is deleted.

Conditional delete

When deleting a note pad, specify CONDITIONS=YES to impose conditions that must be satisfied in order for the note pad to be deleted. The MUSTBE keyword indicates the conditions that must be satisfied. You can require that the note pad not contain any notes, or that it not have any connections, or both. The needs of your application determine whether you need to impose conditions on the delete of the note pad. For example, deleting a note pad that contains notes might induce loss of data. Deleting a note pad that has connections might imply that the application has not been shut down properly.

MUSTBE=EMPTY

The note pad is to be deleted only if it is empty. If the note pad contains any notes, the delete request is rejected. If the note pad has connections, the connections are to be deleted.

A connection might not discover that it was deleted until it issues an IXCNOTE request. Depending on the request and the timing, the request might be rejected because the connection does not exist or because the note pad does not exist. If you delete a note pad out from under active connections, you need to understand the potential consequences and mitigate them as appropriate.

MUSTBE=UNUSED
The note pad is to be deleted only if it does not have any connections. If the note pad has connections, the delete request is rejected. If the note pad has notes, the notes are to be deleted. You need to understand the potential consequences to your application of losing notes. Your program might need to take recovery actions to restore the data. You might need to document recovery procedures for the installation.

MUSTBE=(EMPTY,UNUSED)
The note pad is to be deleted only if it has neither notes nor connections. If the note pad contains notes, the delete request is rejected. If the note pad has connections, the delete request is rejected.

Unconditional delete

Specify CONDITIONS=NO to unconditionally delete the note pad. With an unconditional delete, the note pad is deleted without regard to whether it contains notes or has connections. If the note pad is not empty, the notes are deleted. If the note pad has connections, the connections are deleted.

You need to understand the potential consequences to your application of losing notes. Your program might need to take recovery actions to restore the data. You might need to document recovery procedures for the installation.

A connection might not discover that it was deleted until it issues an IXCNOTE request. Depending on the request and the timing, the request might be rejected because a note does not exist, or the connection does not exist, or the note pad does not exist. If you delete a note pad out from under active connections, you need to understand the potential consequences and mitigate them as appropriate.

Connection requests

Issue the IXCNOTE macro with REQUEST=CONNECTION to manipulate a note pad connection. Such a request is often simply called a *connection request*. When creating a connection, use the NOTEPAD keyword to specify the name of the note pad to which the connection is to be established. Otherwise, the CONNECTION keyword is specified to identify the subject connection. The connection can be created, paused, resumed, or deleted. The note pad must be created before connections can be created, and a connection must be created before your application can manipulate notes in the note pad.

Use the REQTYPE keyword to indicate the type of operation to be performed for the connection. Refer to the following material for specific information on each request type:

- For REQTYPE=CREATE, which is used to create a connection to a note pad, see “Create connection” on page 732
- For REQTYPE=PAUSE, which is used to suspend a work unit until an event occurs, see “Pause connection” on page 733
- For REQTYPE=RESUME, which is used to release a work unit that is suspended, waiting for a pause connection request to complete, see “Resume paused connection” on page 734
- For REQTYPE=DELETE, which is used to delete an existing connection, see “Delete connection” on page 733

When your program issues a connection request, the requesting work unit is suspended. In general, the connection request is processed asynchronously under a task in the XCF address space. The optional keyword TIMEOUT can be used to control how long the work unit remains suspended waiting for results. In general, the default values used by XCF can be used. See “Timeout conditions” on page 721 for additional information.

Your program needs appropriate SAF authorization to create a connection to a note pad. In some cases, your program might need SAF authorization to delete a connection. See “System Authorization Facility (SAF) requirements” on page 718 for more information.

To pause, resume, or delete a connection, the requesting work unit must be recognized as a valid user of the connection. If the requester is the connector, the use is always valid. See “Use of a connection token” on page 719 for information about other valid uses.

In general, the XCF Note Pad Services need to have access to the XCF catalog of note pads and the coupling facility that hosts the note pad in order to process a connection. Depending on the request and the level of the XCF Note Pad Services installed on the various systems in the sysplex, the XCF Note Pad Services might be able to process a request even if it does not have direct access to these entities.

Create connection

Issue a create connection request to create a connection to a note pad. An answer area is optional. If an answer area is provided, it must be large enough for a connection data record. Thus the answer area must have space for the answer area header, one data locator, and one connection data record.

Programs require appropriate SAF authorization to create a connection to a note pad. The access scope of the connection determines the type of authorization required. Specify the ACCESS keyword to indicate the type of access required for the connection. See "System Authorization Facility (SAF) requirements" on page 718.

In general, home and primary of the calling work unit must be the same address space. In the specific case of creating a connection for connector use (USAGE=CONNECTOR), an authorized program is permitted to run with primary not equal to home.

When creating a connection, you must provide the name of the note pad to which the connection is to be established. The create request is rejected if the note pad does not exist.

The usage classification determines the conditions under which a work unit is deemed to be a valid user of the connection for note processing. The creator of the connection specifies the USAGE keyword to indicate the manner in which the connection is to be used by the application. Three styles of usage are supported: connector, server, and client. See "Usage classification" on page 713 for more information.

If the create request completes successfully, a connection data record is stored in the answer area if one is provided. Most of the information in the connection data record reflects the parameters and options specified on the create request. The connection related data that is set by XCF is also stored. In particular, the connection data record includes the connection identifier which might be useful when specifying selection criteria for a multi-note request. The connection data record also includes the timestamp of when the note pad was created. Your program might later need this timestamp when it deletes the note pad.

Using the CONNECTION keyword

For a create request, the CONNECTION keyword identifies a storage area into which XCF is to store the connection token used to represent the connection. The connection token is a required when issuing note requests. Do not lose the connection token. The connection token is only returned by a create connection request. If you lose your token or the token becomes corrupted, you will not be able to process any notes. Furthermore, you will not be able to issue a delete connection request. The connection would persist until the connection is implicitly deleted by XCF upon termination of the connector.

Upon return from the XCF Note Pad Services, the IXCNOTE macro unconditionally stores data in the storage area designated by the CONNECTION keyword. In particular, data is stored even if the create request is rejected. Thus the data stored might not be valid for use as a connection token. The connection token is valid for use if the create connection request is successful. It is also valid for use if the request completes with a return and reason code indicating that the create is pending or that the connection is quiesced.

The connection might not be immediately usable. For example, the create request might complete with a return and reason code indicating that the connection is quiesced. In such a case, the note pad might not be accessible. If you attempt to use the connection to manipulate notes, those requests will likely be rejected due to the quiescing condition. If an answer area was provided for the create request, it might contain detailed information about the quiescing conditions. In some cases, the note pad could still be in the midst of being created, or it could be in the midst of being deleted. More typically, the host structure is in the midst of CF structure rebuild processing. Alternatively, the local system might not have access to the note pad structure or the XCF catalog. Consider issuing a pause connection request to wait for the quiescing conditions to clear.

Delete connection

Issue a delete connection request to delete a connection to a note pad. An answer area is optional. If an answer area is provided, it must be large enough for the answer area header. The answer area for a delete connection request is primarily used for diagnostic data related to cases where the request is rejected.

In general, several tasks must be accomplished to delete the connection. For example, XCF might need to fence the note pad in the host structure to prevent any further note processing by the connector, delete nonpersistent notes associated with the connector, and update the note pad catalog. Usually the XCF Note Pad Services successfully perform these operations as needed. The connection is deleted and the service routine returns success.

However, XCF might not be able to perform some or all of these tasks if the host structure or the XCF catalog is not accessible. In such cases, the delete request completes with a return and reason code indicating that the delete of the connection is pending. XCF will automatically finish deleting the connection when it becomes possible to do so. Some applications can interpret this as being equivalent to a successful delete, and proceed. Some applications might not be able to proceed until the connection is known to have been fully deleted. For example, if this connector was the only connection permitted to have update access to the note pad (because the creator of the note pad specified MULTIWRITE=NO), a subsequent attempt to create a new connection with update access might be rejected until the pending delete is resolved. Alternatively, the application might have dependencies such that it cannot safely proceed until all its nonpersistent notes are known to have been deleted from the note pad.

If an application cannot safely proceed until the pending delete is resolved, consider issuing a pause connection request to wait for the connection to finish being deleted. The paused connection will be resumed when the delete is completed.

Pause connection

Issue a pause connection request to suspend some one work unit until an event occurs. An answer area is optional. If an answer area is provided, it must be large enough for the answer area header.

In general, a pause connection request is issued when the application needs to wait for a particular event. For example, note requests are rejected when a quiescing condition is encountered (such as a rebuild of the structure hosting the note pad). In such cases, the application might prefer to stop issuing note requests until the note pad is once again accessible. If so, it would issue a pause connection request to wait for that event. Alternatively, if a connection is pending delete, the application might need to wait for the delete to finish before it proceeds. It could then issue a pause connection request to wait for that event.

For a given connection, at most one work unit can be paused. If the XCF Note Pad Services already have a work unit paused for the connection, a new pause connection request is rejected.

The pause connection request completes successfully when there no quiescing conditions. If the paused connection is resumed before the quiescing conditions clear, the return and reason code indicating that it was resumed. In these cases, the details stored in the answer area provide information about the current quiescing conditions. They also indicate why the pause request was resumed. These details are mapped by ixcynote_tDetailsResumed.

When a work unit is suspended by a pause connection request that was accepted by the XCF Note Pad Services, the service routine might return control to the program for any one of the following events:

- The quiescing conditions are eliminated
- The quiescing conditions are changed
- The connection is deleted
- The note pad is deleted
- The timeout value expires
- An IXCNOTE resume connection request is issued

In particular, the fact that the pause connection request receives control back from the XCF Note Pad Services does not necessarily imply that the note pad is no longer quiesced. In general, it might be

appropriate to examine the information returned in the answer area to determine whether the note pad is still quiesced. If so, it might be appropriate to reissue the pause connection request.

Resume paused connection

Issue a resume connection request to resume a paused connection. An answer area is optional. If an answer area is provided, it must be large enough for the answer area header. The answer area for a resume connection request is primarily used for diagnostic data related to cases where the resume request is rejected.

Use the resume connection service to force XCF to return control to the work unit that issued a pause connection request prior to when XCF would normally return control. Since the pause connection request has a timeout value, XCF will always eventually return control. XCF will also return control if the connection is deleted. Thus the resume service tends to be used when the application needs to have the work unit that issued the pause request "wake up" to perform some other function. Some applications issue the resume as part of normal shut down procedures so that the work unit can be terminated before the connection is deleted.

If a work unit is currently paused for the connection, the resume connection request causes the suspended work unit to be resumed. The XCF Note Pad Services will then return control to the work unit that issued the IXCNOTE pause connection request. If an answer area was provided for the pause connection request, the detailed information stored therein will indicate that the connection was released as the result of a resume connection request.

If a work unit is not currently paused for the connection, XCF retains the fact that a resume connection request was issued. If a pause connection request is subsequently issued, the connection will be released immediately. Thus a given pair of pause and resume requests need not be issued in a particular order. Note that only one resume request is retained.

Single note requests

You can work with one specific note in the note pad.

Overview

Issue the IXCNOTE macro with REQUEST=NOTE to process one particular note in the note pad. Sometimes a single note request is simply called a *note request*. The note of interest is identified by its name. A note can be created in the note pad by issuing a create note request or a write note request. The content of a note can be updated by issuing a replace note request or a write note request. Both replace and write can be used to either delete the content of a note (causing it to become a null note) or to create content where the note previously had none. A note is deleted from the note pad by issuing a delete note request. A read request can be used to test for the existence of a note, or to read the metadata of a note, or to read the content of a note, or to read both the metadata and content of a note.

Use the REQTYPE keyword to indicate the type of operation to be performed for the note. Refer to the following material for specific information on each request type:

- For REQTYPE=CREATE, which is used to create a new note, see “Create note” on page 738
- For REQTYPE=WRITE, which is used to create or replace a note depending on whether it already exists or not, see “Write note” on page 739
- For REQTYPE=REPLACE, which is used to replace an existing note, see “Replace note” on page 738
- For REQTYPE=READ, which is used to read an existing note, see “Read note” on page 739
- For REQTYPE=DELETE, which is used to delete an existing note, see “Delete note” on page 740

In general, a single note request sends one operation to the coupling facility that contains the note pad. XCF asks XES to perform the operation as a synchronous coupling facility request. If the note pad is not accessible from the local system, the request is rejected and the note pad is said to be quiesced. See “Quiescing conditions” on page 720.

Using the CONNECTION keyword

To issue a single note request, your program must have a valid connection token for the note pad of interest. See “Create connection” on page 732 for more information.

When issuing a note request, your program must be running in a context where use of the connection token is valid. See “Use of a connection token” on page 719 for more information.

The connection token implicitly identifies the note pad to be processed. It also identifies the connection that will be associated with the note as the result of processing a create, write, or replace note.

Answer area

When invoking the IXCNOTE macro to issue a note request, your program can optionally provide an answer area. The answer area is the output area where XCF stores the metadata associated with the note. The metadata includes the note name, instance number, tag value, connection identifier, persistence attribute, and note size.

The metadata stored in the answer area always reflects the current state of the note. If the request is successful, the metadata contains data relevant to the state of the note upon completion of the operation in the note pad. If the request is rejected and the note exists, the metadata contains data relevant to the state of the note as it existed in the note pad just prior to the operation. For example, consider a replace note request that is to update a note named N. Call the current instance of the note N1. Call the new instance of the note N2. If the replace request is successful, the metadata provides information about the updated copy of the note (N2). If the request is rejected due to an instance number mismatch (for example), the metadata provides information about the current copy of the note (N1).

If an answer area is provided, it must be large enough to hold a note data record. Thus the answer area must have room for the answer area header, one data locator, and one note data record.

Note content with the NOBUFFER keyword

Specify the NOBUFFER keyword if you do not want to use a buffer area. When reading or deleting a note, the content of the note will not be read into local storage. When creating a note, the new note will be a null note. When replacing an existing note, the current content of the note is not changed. A write request creates a null note if the note does not exist. If the note does exist, the content of the note is not changed by a write request. In particular NOBUFFER preserves the content of an existing note. If you want the replace (or write) request to delete the content of an existing note, specify the BUFFER keyword with a BUFLEN value of zero.

Note content with the BUFFER and BUFLEN keywords

Use the BUFFER and BUFLEN keywords to provide a buffer area for the note request. In general, the buffer area provided for a single note request must be appropriate for the requested operation. The program must always indicate the length in bytes of the portion of the buffer area that XCF is to use. When reading a note from the note pad, the buffer length must be at least as big as the maximum size note supported by the note pad. If a buffer is provided when deleting a note, the request is processed as a "read and delete" and the buffer length must be at least as big as the maximum size note supported by the note pad. However, when the read operation stores the note content in the buffer area, no storage in the buffer beyond the actual size of the note will be updated. When creating, writing, or replacing a note in the note pad, the buffer length must equal a note size supported by the note pad. See “Note pad related limits” on page 748 for more specific details on supported note sizes and the relationship to the buffer size required for a single note request.

The BUFLEN can be zero when creating, writing, or replacing a note. When creating a note, the new note will be a null note. When replacing an existing note, the note becomes a null note. A write request creates a null note if the note does note exist. If the note does exist, it becomes a null note. In particular, BUFLEN=0 deletes the content of an existing note. If you want the replace (or write) request to preserve the content of an existing note, use NOBUFFER.

Note tags

When a single note request is issued, the TAGGING keyword must be specified to confirm who has responsibility for setting the note tag value. The creator of the note pad determines whether the application or XCF is responsible for setting the tag values. If the TAGGING keyword does not match the protocol specified by the creator of the note pad, the note request is rejected.

In general, the TAG keyword identifies a storage area containing a tag value for the note. Depending on the request and its result, XCF might fetch a tag value from this storage area, or store a tag value in this storage area, or both. Upon return from the service routine, the IXCNOTE macro expansion unconditionally stores data in the designated storage area. Depending on the request result, this data might not be a tag value that is valid for use. Some programs might need to preserve and restore the original content of this storage area to ensure correct operation in cases where the data stored is not valid for use as a note tag.

Independently of the TAG keyword specification, note that the metadata stored in the answer area (if any) includes the tag value for the note.

XCF Note Tagging

If XCF is responsible for setting the tag values, the TAG keyword is optional. If specified, the storage area identified by the TAG keyword is always an output area. XCF stores the tag value of the indicated note in this storage area. For a read request or a delete request, the current tag value of the subject note is stored. For a create, write, or replace request, the new tag value set as the result of performing the operation is stored. If the request is rejected due to an instance number mismatch, the current tag value of the note is stored. The tag value is valid for use if the request is either successful or rejected due to an instance number mismatch. Otherwise the stored tag value is not valid for use (normally the stored value will be zero).

User Note Tagging

If the application is responsible for setting the tag values, the TAG keyword can indicate a storage area for the note tag value. Alternatively, TAG=KEEP can be specified.

If TAG=KEEP is specified, or taken as the default, a tag value is neither fetched from nor stored into local storage. Furthermore, the current tag value of the note is not changed as the result of processing the request. If the request causes a new note to be created, the tag value of the new note is set to zero.

If the TAG keyword identifies a storage area, XCF will either fetch a tag value from the indicated storage area, or store the tag value of the note in the indicated storage area, or both.

- For a read note request, the IXCNOTE macro expansion stores the tag value of the note in the designated storage area upon return from the XCF Note Pad Services. The tag value is valid for use if the read request is either successful or rejected due to an instance number mismatch. Otherwise the stored tag value is not valid for use (normally the stored value will be zero).
- For a create note request, the storage identified by the TAG keyword contains the tag value to be set for the newly created note. Upon return from the XCF Note Pad Services, the IXCNOTE macro expansion stores into the designated storage area. In cases where the note exists, regardless of whether it was newly created or already existed, the value stored is the current tag value of the note. Thus the stored tag value is valid for use if the create request is either successful or rejected because the note already exists. Otherwise the stored tag value is not valid for use (normally the stored value will be the same as the original value).
- For a write, replace, or delete note request, the storage identified by the TAG keyword contains the new tag value to be set for the subject note. If the note exists and the request is otherwise successful, the note tag is set to the indicated value. If the note does not exist, a write request causes the note to be created with the indicated tag value. For replace or delete, the request is rejected if the note does not exist.

  If the creator of the note pad indicated that XCF needs to track the maximum tag value assigned to the notes in the note pad, the new tag value must be greater than or equal to the current tag value of the note. If the note exists and this condition is not satisfied, the request is rejected because the specified tag value is too low.

Upon return from the XCF Note Pad Services, the IXCNOTE macro expansion stores into the designated storage area. If the request is successful, the stored tag value is valid for use and equals the designated new tag value that was assigned to the note (that is, it equals the original input tag value). If the request is rejected due to an instance number mismatch or because the tag value is too low, the stored tag value is valid for use and equals the current tag value of the note. Otherwise the stored tag value is not valid for use (normally the stored value will be the same as the original value).

Note instance numbers

The INSTANCE# keyword determines whether certain requests will be processed with an instance number comparison. Instance number comparisons can be used to ensure that the expected instance of the note is being processed. If the creator of the note pad specified INSTCOMP=REQUIRED on the create note pad request, instance number comparisons are mandatory when writing, replacing, or deleting an existing note. Instance number comparisons are optional for read note requests.

If INSTANCE#=IGNORE is specified, or taken as the default, no instance number comparison is performed when processing the request.

Otherwise the INSTANCE# keyword identifies a storage area containing the instance number for the note. Depending on the request and its result, XCF might fetch an instance number from this storage area, or store an instance number in this storage area, or both. If the value fetched from the instance number storage area is zero, no instance number comparison is performed for the request.

- For a create note request, the IXCNOTE macro expansion stores the instance number of the note in the designated storage area upon return from the XCF Note Pad Services. The instance number is valid for use if the create request is either successful or rejected because the note already exists. Otherwise the stored instance number value is not valid for use (normally the stored value will be zero).
- For a write note request, the IXCNOTE macro expansion extracts the instance number from the designated storage area and calls the XCF Note Pad Services. If the designated note does not exist, the note is created and the instance number is set to a value determined by XCF. The specified input instance number is irrelevant and totally ignored in this case. If the designated note does exist, the note is replaced if the specified instance number is either zero or equal to the current instance number of the note (and the request is otherwise valid). Otherwise the request is rejected due to an instance number mismatch.

  Upon return from the XCF Note Pad Services, the IXCNOTE macro stores an instance number in the designated storage area. The instance number is valid for use if the write request was successful, or if the request was rejected due to either an instance number mismatch or a low tag value. Otherwise the instance number is not valid for use (normally the stored value will be the same as the original input value).
- For a read, replace, or delete request, the IXCNOTE macro expansion extracts the instance number from the designated storage area and calls the XCF Note Pad Services. If the designated note does not exist, the request is rejected. If the designated note does exist, the request is processed if the specified instance number is either zero or equal to the current instance number of the note (and the request is otherwise valid). Otherwise the request is rejected due to an instance number mismatch.

  Upon return from the XCF Note Pad Services, the IXCNOTE macro stores an instance number in the designated storage area. The instance number is valid for use if the request was successful, or if the request was rejected due to either an instance number mismatch or a low tag value. Otherwise the instance number is not valid for use (normally the stored value will be the same as the original input value).

Independently of the INSTANCE# specification, note that the metadata stored in the answer area (if any) includes the instance number for the note.

Note persistence

Use the KEEPNOTE keyword to indicate whether the note is to persist after the connection for the request is deleted. The KEEPNOTE keyword is valid for use when creating, writing, or replacing a note. Upon

successful completion of the request, the note becomes associated with the connection that issued the request. When the associated connection is deleted, XCF inspects each note that is associated with the connection. If KEEPNOTE=NO was specified when the note was created or updated, the note is nonpersistent and will be deleted by XCF. If KEEPNOTE=YES was specified, the note is persistent and will not be deleted. Thus a persistent note persists in the note pad after a connection is deleted. A nonpersistent note does not.



CAUTION: The KEEPNOTE keyword is optional with a default of KEEPNOTE=NO. If you want the note to persist you must specify KEEPNOTE=YES. In particular, you must specify KEEPNOTE=YES each and every time the note is created, written, or replaced.

Create note

Issue a create note request to create a new note with the indicated content. The create request is rejected if the named note already exists.

If the create note request is successful, the tag value, instance number, connection identifier, persistence attribute, and note size are also set for the note. The count of notes in the note pad is incremented by one.

The note content is determined by the IXCNOTE keywords NOBUFFER, BUFFER and BUFLEN. If either NOBUFFER or a BUFLEN value of zero is specified, a null note is created. If the BUFLEN value is nonzero, the indicated number of bytes of data are fetched from the BUFFER area and stored in the note. If nonzero, the BUFLEN value must equal a supported note size.

The create request is rejected if the note pad is either full or constrained. A note pad is full if the number of notes in the note pad equals the number of notes requested by the creator of the note pad. A note pad is constrained if XCF is unable to create the note even though the note pad is not full. See “Constrained conditions” on page 720 for more information. You need to determine how your program will respond to these conditions. It might be possible to delete some notes to make space available for new notes. It might be reasonable to reissue the request, perhaps after a short delay to allow time for the condition to clear.

Replace note

Issue a replace request to replace an existing note with the indicated content. The replace request is rejected if the named note does not exist.

If the replace note request is successful, the tag value, instance number, connection identifier, persistence attribute, and note size are also set. The count of notes in the note pad is not changed.

The new note content is determined by the IXCNOTE keywords NOBUFFER, BUFFER and BUFLEN:

- If NOBUFFER is specified, the current content of the note is not changed.
- If the BUFLEN value is zero, the current note content (if any) is deleted. The note becomes a null note.
- If the BUFLEN value is nonzero, it must equal a supported note size. The indicated number of bytes of data are fetched from the BUFFER area. The current content of the note (if any) is deleted and the data fetched from the BUFFER is stored as the new content of the note. The new content is a complete replacement of the old content. In particular, the size of the note changes if the length of the old content does not equal the length of the new content. For example, if the note was null it will now have content and a nonzero size.

If the replace request is rejected due to an instance number mismatch, the IXCNOTE macro expansion stores the current value of the instance number for the note in the storage area identified by the INSTANCE# keyword, and stores the current tag value of the note in the storage area identified by the TAG keyword. However, the current content of the note is not returned. Depending on the needs of your application, your program might issue a read request to get the current note content, reapply the desired updates, and then reissue the replace request. If instead your program were to immediately reissue the replace request without refreshing the local copy of the note content, it might well subvert the data integrity protection that the instance number comparison was intended to provide. Reissuing the replace request without first refreshing the note content might be reasonable for some applications if, for example, the tag value contained control data that could be used to determine whether the refresh was

needed. If a read operation is issued to refresh the local copy of the note, the instance number returned by the read request should be used when reissuing the replace request, as opposed to the instance number stored by the IXCNOTE macro expansion as the result of the instance number mismatch (because the note might have been updated yet again before the read is processed).

If the replace request is rejected because the proposed new tag value is less than the current tag value of the note, the IXCNOTE macro expansion stores the current tag value of the note in the storage area designated by the TAG keyword, and stores the current instance number for the note in the storage area identified by the INSTANCE# keyword.

In cases where the request has both an instance number mismatch and a low tag value, XCF reports the failure as an instance number mismatch.

Write note

Issue a write note request to either create or replace a note, as applicable. If the named note does not exist, proceed as for a create note request. If the note does exist, proceed as for a replace request.

The write request is useful in cases where the existence of the note is immaterial to the application. For some applications, the existence (or not) of a note conveys information that helps ensure that the application is operating as expected. Such an application might use a create request in contexts where the note is not expected to exist. If the note does exist, the application relies on the request being rejected to detect this unexpected state. Similarly, the application might use a replace request in contexts where the note is supposed to exist. If the note does not exist, the application relies on the request being rejected to detect this unexpected state. If the application does not have such requirements, or perhaps does not have them for certain notes, the write request is a simple way to accomplish a "create if not exist else replace" behavior. Using combinations of create and replace requests to achieve the same behavior is more complicated in the general case.

Read note

Issue a read note request to read an existing note from the note pad. The request is rejected if the named note does not exist.

If a suitable answer area is provided, a copy of the note metadata is stored in the answer area. The note metadata includes the note tag, instance number, connection identifier, persistence attribute, and note size.

If a suitable buffer area is provided, a copy of the note content is placed in the buffer. The buffer area must be large enough to hold the content of the largest note supported by the note pad. This buffer length requirement must be met even if the note to be read is known to be smaller than the maximum note size.

If neither an answer area nor a buffer area are provided, the read request is in effect testing the existence of the note. The IXCNOTE return and reason code will indicate *success* if the note exists and *not exist* if the named note does not exist.

If an answer area is provided but a buffer area is not provided, the read request is being issued strictly for the purpose of reading the note metadata.

If a buffer area is provided but an answer area is not provided, the read request is being issued strictly for the purpose of reading the note content. However, without the metadata in the answer area, you need an independent mechanism by which to determine the size of the note that was stored in the buffer. For example, it could be that each and every note in your note pad has the same known fixed size, or that the particular subject note has a known size. In cases where variable size notes are used, you might be able to pre-initialize the buffer with data so as to determine whether and how much of the buffer was overlaid with note content.

If the read request is rejected due to an instance number mismatch, the IXCNOTE macro expansion stores the current value of the instance number for the note in the storage area identified by the INSTANCE# keyword, and stores the current tag value of the note in the storage area identified by the TAG keyword. However, the current content of the note is not returned.

Delete note

Issue a delete note request to delete an existing note from the note pad. The request is rejected if the named note does not exist.

If the delete note request is successful, the note is deleted and the count of notes in the note pad is decremented by one. However, if the creator of the note pad requested lifetime tracking of the maximum tag values (TRACKTAG=LIFETIME), the note might be logically deleted. The delete note request still completes with a return code indicating success, but the physical deletion of the note and decrement of the note count could be deferred. See Lifetime tracking (TRACKTAG=LIFETIME) for more information about how the physical existence of a logically deleted note might impact your program.

The delete request is processed as a "read and delete" if either an answer area or a buffer area is provided.

If a suitable answer area is provided, a copy of the note metadata is stored in the answer area. The note metadata includes the note tag, instance number, connection identifier, persistence attribute, and note size.

If a suitable buffer area is provided, a copy of the note content is placed in the buffer. The buffer area must be large enough to hold the content of the largest note supported by the note pad. This buffer length requirement must be met even if the note to be deleted is known to be smaller than the maximum note size.

If the delete request is rejected due to an instance number mismatch, the IXCNOTE macro expansion stores the current value of the instance number for the note in the storage area identified by the INSTANCE# keyword, and stores the current tag value of the note in the storage area identified by the TAG keyword. However, the current content of the note is not returned. Depending on the needs of your application, your program might need to take an action to determine whether this new instance of the note needs to be deleted. For example you might issue a read request to get the current note content, inspect the note to see if it needs to be deleted, and then reissue the delete request as appropriate. If instead your program were to immediately reissue the delete request without refreshing the local copy of the note content, it might well subvert the data integrity protection that the instance number comparison was intended to provide. Reissuing the delete request without first inspecting the note content might be reasonable for some applications if, for example, the tag value contained control data that could be used to determine whether the note was still eligible for deletion. If a read operation is issued to refresh the local copy of the note, the instance number returned by the read request should be used when reissuing the delete request, as opposed to the instance number stored by the IXCNOTE macro expansion as the result of the instance number mismatch (because the note might have been updated yet again before the read is processed).

If the delete request is rejected because the proposed new tag value is less than the current tag value of the note, the IXCNOTE macro expansion stores the current tag value of the note in the storage area designated by the TAG keyword, and stores the current instance number for the note in the storage area identified by the INSTANCE# keyword.

In cases where the request has both an instance number mismatch and a low tag value, XCF reports the failure as an instance number mismatch.

Multi-note requests

You can work with multiple notes in the note pad.

Overview

Issue the IXCNOTE macro with REQUEST=NOTES to process a collection of notes in the note pad. Sometimes a multi-note note request is simply called a *notes request*. The notes of interest are identified by selection criteria. You can either read or delete the notes in the specified collection.

Use the REQTYPE keyword to indicate the type of operation to be performed for the notes. Refer to the following material for specific information on each request type:

- For REQTYPE=READ, which is used to read a collection of notes, see "Read notes" on page 744
- For REQTYPE=DELETE, which is used to delete a collection of notes, see "Delete notes" on page 747

In general, a multi-note request sends one or more operations to the coupling facility that hosts the note pad. These operations can read or delete notes. They are processed while running under the calling work unit. In general, XCF asks XES to perform these operations as synchronous coupling facility requests. If the note pad is not accessible from the local system, the note pad is said to be quiesced. See "Quiescing conditions" on page 720.

Relative to a single note request, multi-note requests are potentially long running. The duration of the request can be impacted by such factors as the number of notes in the note pad and the selection criteria. For a read notes request, you might have to issue the request multiple times to read all of the selected notes if the provided answer area (and optionally buffer area) are not large enough for them all.

Using the CONNECTION keyword

To issue a multi-note request, your program must have a valid connection token for the note pad of interest. See "Create connection" on page 732 for more information.

When issuing a multi-note request, your program must be running in a context where use of the connection token is valid. See "Use of a connection token" on page 719 for more information.

The connection token implicitly identifies the note pad to be processed.

Answer area

For a read notes request, an answer area must be provided. The answer area must be large enough for at least one note data record. Larger answer areas allow for more note data records to be stored. A note data record contains the metadata for one note. In general, you provide an answer area with enough space for the number of notes you want to process on each iteration of the read notes request. Thus the answer area must have space for the answer area header, one data locator, and as many note data records as you like. See "Read notes" on page 744 for more information.

For a delete notes request, the answer area is optional. If provided, the answer area must be large enough for the answer area header. Although optional, an answer area is desirable given the potential need for the detailed diagnostic data. For example, if the specified selection criteria do not have valid content, diagnostic information stored in the answer identifies the specific problem.

Selection criteria

When you issue the IXCNOTE macro to call the XCF Note Pad Services to process a collection of notes, you either specify CHOOSE=ALL to indicate that all the notes in the note pad are to be selected, or you specify CHOOSE=BYCRITERIA to provide a storage area (CRITERIA) containing selection criteria to describe which notes are to be selected. When deleting a collection of notes, the MAXTAG keyword specification is also considered when determining which notes are to be selected. See "Multi-note selection criteria" on page 741 for more information on use of CRITERIA. In general, the term *selection criteria* is used regardless of the CHOOSE specification.

Multi-note selection criteria

When using selection criteria (CHOOSE=BYCRITERIA), your program must provide a storage area initialized with data describing which notes are to be selected (CRITERIA). The mappings for the selection criteria are defined in the IXCYNOTE macro. The selection criteria, which is mapped by ixcynote_tSelectionCriteria, contains a type field, a count field, and an array of records.

- The type field indicates the test to be used to determine whether a note is to be processed.
- The count field indicates the number of selection criteria records in the array. The count must be greater than or equal to one. The maximum count value depends on the level of the XCF Note Pad Services that is running on the system that processes the request. See "Note pad related limits" on page 748.

- Within each array entry, the 32 byte selection criteria record contains parameters appropriate for the indicated type of test. Thus the type field also determines how each of the selection criteria records is mapped. When the array contains more than one record, a given note is processed if it can pass the indicated test using the test parameters from any of the selection criteria records in the array. In other words, if each selection criteria record defines a set of notes, the union of those sets would be the set of notes selected for processing.

You can specify any one of the following types of selection criteria.

- Use *Tag Range criteria* to select any note whose tag value satisfies the inequality:

  mintag ≤ tag value ≤ *maxtag*.

  The selection criteria record contains two 16 byte tag values, *mintag* and *maxtag*. *mintag* must be less than or equal to *maxtag*. You can use tag range selection criteria with any note pad, including note pads where user assigned tags can be arbitrary values. Tag range criteria are mapped by ixcynote_tSelectByTagRange.
- Use *Tag Mask criteria* to select any note whose tag value, when masked with a given bit mask, equals a given filter tag value. The selection criteria record contains two 16 byte values, *tagmask* and *tagfilter*. *tagmask* determines which bits in the note tag are to be compared to bits in the *tagfilter*. The note is selected if for every bit that is ON (B'1') in *tagmask*, both of the corresponding bits in the note tag and the *tagfilter* have the same value. If a mask bit is OFF (B'0'), the corresponding bits in the note tag and filter will not be compared. A *tagmask* where all the bits are OFF implies that all notes will be selected. A *tagmask* where all the bits are ON implies that any note whose tag value equals *tagfilter* will be selected. Tag mask criteria are mapped by ixcynote_tSelectByTagMask.
- Use *Connection Identifier criteria* to select notes associated either with a given note pad connection or with a given system. A note is associated with a given system if its associated connection was created on that system. The selection criteria record either contains a 12 byte connection identifier, or a 4 byte XCF System ID, or a 1 byte XCF system slot number. The selection criteria record must also indicate whether you want to include persistent notes, nonpersistent notes, or both. If neither persistent notes nor nonpersistent notes are indicated, the request is rejected. Recall that a nonpersistent note is automatically deleted by XCF when the associated connection terminates, but a persistent note is not. Connection identifier criteria are mapped by ixcynote_tSelectByConnectionID.

  The 12 byte connection identifier for a given connection is returned in the answer area (if any) provided when a connection is created. The connection identifier for the connection associated with a note is returned as part of the metadata for the note. Selecting notes associated with a given connection identifier might be useful, for example, when performing cleanup for the connection.

  The connection identifier criteria can also be used to select notes associated with a given system in the sysplex. The system can be identified either by its XCF System ID or by its XCF System Slot Number. If the XCF System ID is specified, only those notes associated with a connection that was created on that specific system instance will be selected. If the XCF System Slot Number is specified, the note is selected if its associated connection resided on some instance of a system that was assigned to the indicated slot number. Selecting notes associated with a given system might be useful, for example, when performing cleanup for a system.

The storage area containing the selection criteria should not be modified while in use by the XCF Note Pad Services. Changing the selection criteria while XCF is processing the request could produce inconsistent or unintended results.

If the specified selection criteria do not have a valid content, XCF rejects the multi-note request. If an answer area is provided, XCF stores detailed information describing the specific problem that was encountered. This information is mapped by ixcynote_tDetailsCriteria. Without this information, it will be difficult to determine why the selection criteria were deemed unsuitable. An answer area is required for a read notes request. An answer area is optional for a delete notes request, but desirable given the potential need for the detailed diagnostic data.

Concurrent request issues

When processing a multi-note request, the XCF Note Pad Services inspect the notes in the note pad to determine whether they meet the prescribed selection criteria. If your program creates, updates, or deletes notes while a multi-note request is being processed, you might need to account for some anomalies that could occur with regard to note selection. For example, the multi-note request might fail to include all the notes that meet the selection criteria, or it might include a given note more than once. If your program needs to reissue a multi-note request one or more times to process all the notes in the note pad (as described below for a read notes request), these anomalies can occur if your program creates, updates, or deletes notes in between the reissued multi-note requests. To understand how these anomalies might arise, you need to understand how the XCF Note Pad Services scan the note pad when selecting notes.

When processing a multi-note request, the notes in the note pad are scanned in the order that they were created. More precisely, the notes in the note pad are maintained in a sorted sequence based on a timestamp taken by the XCF Note Pad Services just before the operation that causes the note to be created is sent to the coupling facility. Thus there is a window between the setting of the created timestamp and the physical creation of the note in the note pad. This window could be observed by a multi-note request. This timestamp remains constant for the life of the note. In particular, the timestamp is not changed when the content of an existing note is replaced. However, if a note is deleted and created anew, the new instance of the note is assigned a new timestamp. With this background, consider the following:

- A given note might be processed more than once if notes are being deleted and recreated while the multi-note request is being processed. Each time a note is newly created, a new created timestamp is set. So as the multi-note request scans the notes in created timestamp order, the timing could be such that it will encounter each newly created instance of the note.

  For example, suppose a note pad contains two notes named A and B created at time T1 and T2 respectively, where T1 < T2. Suppose a multi-note request is issued. The request scans the notes in create time order and therefore processes note A first. It then moves on to note B. While note B is being processed, assume some other thread deletes note A and creates it again at time T3. When the multi-note request finishes processing note B, it moves on to the next note. The next note is the new instance of note A created at time T3. From the perspective of the multi-note request, three notes were encountered: A_{T1}, B_{T2}, A_{T3}. From the perspective of the application, which likely identifies notes by name, it appears as if note A was processed twice. For example, a read request would report both instances of note A (assuming both instances satisfied the selection criteria). If the multi-note request was a delete request, both instances of note A would be deleted.

  Alternatively, suppose that the multi-note request is a read request that completes prematurely after processing note B. Notes A_{T1} and B_{T2} would be reported. If the read request was then reissued to continue reading the remaining notes in the note pad, the new instance of note A (that is, A_{T3}) is reported. Once again, from the perspective of multi-note request processing, three notes were encountered and reported. From the perspective of the application, note A was reported twice.
- Updates to an existing note might not be reported if notes are being replaced while a multi-note read request is being processed. If an existing note is replaced one or more times while a read notes request is being processed, at most one of the instances will be returned by the request. A note is created by a create note request, or by a write request when the note does not exist. Replacing the content of an existing note does not cause the note to be created. The created timestamp is set only when the note is created, not when it is replaced. Thus after a read operation inspects a note, that note will not be inspected again by the ongoing read operation or any subsequent resume of the read request, regardless of how many times the note is replaced.

  For example, suppose a note pad contains two notes named A and B created at time T1 and T2, respectively, where T1 < T2. Suppose note B is being replaced while a read notes request is in progress. Call the original instance B1 and the updated instance B2. Assume both instances of the note satisfy the selection criteria. If the read request encounters note B before the update is made, instance B1 is reported. If the read request encounters note B after the update, instance B2 is reported. Thus, depending on the timing, the read request will either report B1 or B2. From the perspective of the

application, it might appear as if stale content was reported (B1 instead of the latest B2) or it might appear as if an instance of the note was lost (B2 reported but not the prior B1).

Suppose the multi-note read request completes prematurely after reporting instance B1 of note B, and then note B is updated to contain content B2. When the read request is reissued to continue reading the remaining notes, note B (now B2) is not reported. The replacement of instance B1 with the content of instance B2 does not change the created timestamp for note B. Since note B was already reported by the previous read request, the reissued read request does not include note B in the set of notes to be considered since its created timestamp precedes the resume point. From the perspective of the application, it might appear as if stale content (B1) was reported and the latest content (B2) was missed.

- If a note is created while a multi-note request is being processed, the newly created note might not be processed by the multi-note request even though it meets all the selection criteria. For a multi-note read request, the timing could be such that the note would not be reported even when the read request is reissued with the resume token returned by the read request that was active when the note was created. The multi-note request can only observe notes that actually exist in the note pad. But the created timestamps are set by the z/OS system before the note is physically created in the note pad. Thus there could be a delay between when the timestamp is set and when the note is created in the note pad. Thus it is possible for notes to be physically created in a different order than one might expect based on the created timestamps. When this occurs, a multi-note request might not include all the expected notes.

  For example, suppose two notes named A and B are being created at time T1 and T2 respectively, where T1 < T2. However, note B is physically created in the note pad before note A because the create note operations happen to run in the order B then A. Suppose a multi-note request is being processed at the same time that notes A and B are being created. The timing could be such that the multi-note request first encounters note B with created timestamp T2. Note A is then physically created in the note pad. But since the created timestamp T1 for note A is older than T2, note A is positioned ahead of note B in the created timestamp ordering of notes within the note pad. But the multi-note request will continue its scan of the note pad in created time order, and so will only consider notes with created timestamps greater than T2. Thus note A is not processed by the multi-note request.

Depending on the implementation of your program, these anomalies might or might not be possible. Even if they are possible, they might or might not be a problem for your application. If they could be problematic, you might consider designing your application so as to minimize or eliminate these timing windows. For example, you might use various serialization techniques or task structures to ensure that a multi-note request is not being processed while notes are being created, replaced, or deleted. Alternatively, you might consider using related information to detect cases where anomalies might have occurred. You might be able to use note counts, tag values, or instance numbers to make inferences. You might consider reissuing the multi-note request, possibly with modifications, to pick up any notes missed on the first pass. The techniques will vary with the needs of your application and the way in which anomalies might surface based on your implementation.

Read notes

Issue a read notes request to read the collection of notes defined by the specified selection criteria. An answer area must be provided. A buffer area is optional. XCF scans the note pad, inspecting each note to see if it satisfies the selection criteria. Metadata for each of the selected notes is stored in the answer area. The note metadata includes the note tag, instance number, connection identifier, persistence attribute, and note size. If a suitable buffer area is provided, the content of each selected note is stored in the buffer area. If a note has content (not a null note), the metadata in the answer area for that note will also indicate the location of the note content within the buffer area. A valid read request returns to the caller when all the notes have been inspected, or when the storage areas provided for output are filled with as many notes as will fit, whichever comes first.

For a read notes request, your program might need to repeatedly issue the request to fetch all of the selected notes. Such situations occur when the data for the selected notes will not fit in the output storage areas provided by your program (answer area and buffer area). When the read request completes prematurely in this fashion, your program must reissue the request to continue reading the remaining

notes. A resume token is used to maintain the context so that each iteration (reissue) of the read request continues on from where the prior read left off. The number of times that the request needs to be reissued to finish reading the notes will in general decrease as the size of the provided storage areas increases. The duration of each such request will likely increase since more notes can be processed on each call. You might choose to issue a query note pad request to determine the number of notes in the note pad to estimate how much storage would be needed to read all the notes.

Note: All offsets are relative to the start of the answer area.

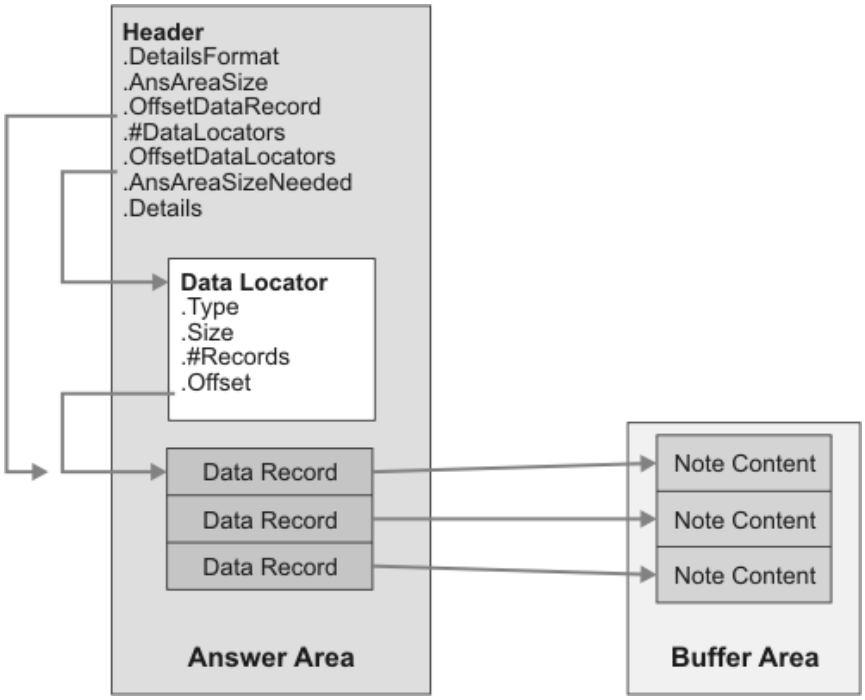


Figure 85. Multi-note answer area with buffer

Take note of the discussion of "Concurrent request issues" on page 743. In particular, the read notes request scans the notes in created timestamp order. If your program concurrently creates notes while the read notes request is being processed, the timing of the interplay between these requests could be such that the read notes request might fail to read a note that satisfies the selection criteria. Your program might need to account for this possibility.

Premature completion

When reading a collection of notes, your program might need to issue the read notes request more than once to read all of the candidate notes. If the answer area and buffer area (as applicable) are large enough to hold all of the selected notes, only one read request is needed. However, if the supplied storage areas are not large enough to hold all of the selected notes, you must reissue the request to continue reading the remaining notes. A return code of zero implies that the read is complete. A return and reason code indicating *more notes* implies that the read request completed prematurely and needs to be reissued. Each reissued request must provide as input the resume token that was returned by its predecessor request.

Note that when a multi-note read request completes with return code 0, the number of notes reported could be zero. Furthermore, a return and reason code indicating *more notes* does not imply that any of the remaining notes will meet the selection criteria. In effect, *complete* and *more notes* indicate whether XCF has finished inspecting the notes in the note pad.

Resume token

When issuing the IXCNOTE macro to perform a read notes request, you must specify the RESUMETOKEN keyword to name a storage area from which XCF is to retrieve a resume token. Upon return from the XCF

Note Pad Services, a new resume token is stored in this same storage area. The resume token in effect divides the notes in the note pad into two sets, the set of notes that have been considered for selection and the set of notes that have not been considered. For a resume token of zero, the set of notes that have been considered is defined to be empty, and the set of notes not yet considered for selection is defined to be all the notes in the note pad. A program will typically first specify a resume token of zero to have all the notes in the note pad be considered for selection. When that read request returns, XCF updates the resume token with a nonzero value to identify the set of notes in the note pad that were not considered for selection. If this nonzero resume token value is then supplied as input to a subsequent read notes request, only notes from that set will be selected. So in effect, a resume token of zero implies "start from the beginning" and a nonzero token implies "start with the next note".

When specifying a nonzero resume token, only use a value that was stored by a valid read notes request. The resume token is valid for use by the note pad connection that issued the read notes request that stored the token.

When passing a nonzero resume token, your program should in general pass the same selection criteria as was specified when the request was first initiated, and continue doing so for each iteration of the read request until it is complete. However, there is no requirement that the same selection criteria be used when the read request is reissued.

Note that the XCF Note Pad Services also return a nonzero resume token when the multi-note read request completes with return code 0. The resume token in this case divides the note pad into two sets, the set of notes that have been considered for selection (all the notes currently in the note pad) and the set of all notes not yet considered for selection (an empty set). With respect to this resume token, the set of notes not yet considered for selection will become non-empty as new notes are created in the future. If this resume token were to be specified on a subsequent multi-note read request, only the newly created notes would be considered for selection. You could for example issue a sequence of multi-note read requests such that each successor request only reports the notes that were newly created after the predecessor request completed. To achieve this behavior, the resume token returned by each completed predecessor request would be supplied as input to the successor request.

Buffer area and note size

When a buffer area is provided for a multi-note read request, the buffer length must be at least as long as the size of the first note to be read. In general, a multi-note read request keeps storing note content in the buffer area until the buffer is full. Since null notes have no content, an arbitrary number of null notes will fit in the buffer. So when reading multiple notes, it is possible to provide a buffer of length zero. If the first note to be read is a null note, the note will fit in the buffer because it has no content. Thus the read notes request would continue reading additional notes. If the read notes request were to then encounter a note with content, the zero length buffer would be considered full since it is not big enough to hold the content of the next note to be read. The read notes request would then return to the caller with a return and reason code indicating that there are more notes to be read. In contrast, if the read notes request is given a zero length buffer and the first note it encounters has content, the request is rejected with a return and reason code indicating that the buffer is too small.

If the buffer size is nonzero, it must be a multiple of the incremental note size. See “Note pad related limits” on page 748 for more specific details on supported note sizes and the relationship to buffer size required for a multi-note request.

Failure conditions and partial results

If an error condition is encountered while a multi-note read request is being processed, XCF might have stored zero or more note data records in the answer area and the content of the respective notes in the buffer area. However, the answer area header will not contain the information needed to locate these records. It will instead indicate that no data records were returned, despite the fact that the output storage areas might have been updated. In short, partial results are not supported. The resume point identified by the resume token that is stored by the IXCNOTE macro expansion upon return from the XCF Note Pad Services is not changed. If the read notes were to be reissued, these notes would still be candidates for selection.

Delete notes

Issue a delete notes request to delete the collection of notes defined by the specified selection criteria. XCF scans the note pad, inspecting each note to see if it satisfies the selection criteria. Each note that satisfies the selection criteria is deleted. A valid delete request returns when all the notes in the note pad have been inspected.

An answer area is optional. If provided, a count of the number of notes deleted by the request is stored in the answer area upon successful completion. Only the answer area header is needed. A delete notes request does not support "read and delete", so no space is needed in the answer area for note data records and a buffer area is not supported.

Take note of the discussion of “Concurrent request issues” on page 743. In particular, the delete notes request scans the notes in the note pad in created timestamp order. If your program concurrently creates notes while the delete notes request is being processed, the timing of the interplay between these requests could be such that the delete notes request might fail to delete a note that satisfies the selection criteria. Your program might need to account for this possibility.

Deferred delete

For each deleted note, the count of notes in the note pad is decremented by one. As was the case for a single note delete request, the physical deletion of a note and decrement of the note count could be deferred in cases where XCF needs to preserve the maximum tag value of the deleted note. Deferred deletes can occur with note pads that require lifetime tracking of maximum tag values. When applicable, the XCF Note Pad Services preserve the maximum tag value before processing the delete notes request so as to minimize the possibility that the deletion of a note is deferred.

MAXTAG and note selection

For a delete notes request, you can also specify the MAXTAG keyword to control which notes are to be selected. Conceptually, XCF first uses the selection criteria to create a collection of notes, and then removes any notes whose tag value is greater than the indicated MAXTAG value. That is, if SET1 is the collection of notes identified by the selection criteria, and SET2 is the collection of notes whose tag value is less than or equal to the indicated MAXTAG value, then the set of notes to be deleted is the intersection of those two sets.

For example, suppose the note tag values are timestamps and the application uses a timestamp to track periodic checkpoints of its data. The application might need to delete all notes in the note pad that had tag values older than a given checkpoint timestamp (call it *ckpttod*). To do so, it could issue a delete notes request with tag range criteria of MINTAG=0 and MAXTAG=*ckpttod*. Alternatively, it could issue a delete notes request that selected all notes in the note pad and specify MAXTAG=*ckpttod* to limit the delete to those notes that have a tag value less than or equal to the checkpoint timestamp.

As another example, suppose again that note tag values are timestamps and that the application has a checkpoint timestamp *ckpttod*. The application might need to delete all notes written by a given connection prior to the checkpoint timestamp. To do so, it could issue a delete notes request with MAXTAG=*ckpttod* and connection identifier selection criteria that included all notes (both persistent and nonpersistent) associated with the desired connection. The delete notes request would select all notes associated with the indicated connection that had tag values less than or equal to the checkpoint time stamp.

MAXTAG with lifetime tracking of user assigned note tags

If the creator of the note pad requested that XCF track the maximum note tag value ever assigned by the user, use of the MAXTAG keyword for a delete notes request has an additional side effect. After the selected notes are deleted (which includes the case where the set was empty), XCF determines whether the specified MAXTAG value is a new maximum tag value for the note pad. If so, XCF preserves the indicated MAXTAG value as the new maximum note tag value for the note pad.

With a single note delete request, a new tag value can be logically assigned to the note before it is physically deleted. If the newly assigned tag value happened to be a new maximum tag value for the

note pad, that value would be preserved as the new maximum tag value. The MAXTAG keyword provides a similar function for a multi-note delete request in the sense that a new tag value can be logically assigned to a collection of deleted notes. However, there are some important differences between the two behaviors.

For a single note delete request, setting the new logical tag value for the note, deleting the note, and preserving its tag value as the maximum tag value for the note pad (as needed) is effectively an atomic operation. If the note is successfully deleted, the new tag value is guaranteed to be preserved as the new maximum tag value for the note pad (if appropriate). For a multi-note delete request, the deletion of each note and the preservation of its tag value as the maximum tag value for the note pad (as needed) is effectively an atomic operation. However, the setting of the new logical tag value for the collection of deleted notes per the MAXTAG specification is performed as a separate operation after the notes have been deleted. If the notes are deleted, there is no guarantee that the indicated MAXTAG value will be preserved as the new maximum tag value for the note pad. As the result of a failure, the specified MAXTAG value might not be preserved in the note pad.



CAUTION: If your application cannot tolerate a failure to record the specified MAXTAG value as the new maximum tag value when deleting a collection of notes, consider using single note delete requests to delete the notes one at a time. Preservation of the maximum note tag is guaranteed to be atomic when deleting a single note. For example, you might issue a multi-note read request to determine the set of notes to be deleted and then use single note delete requests to delete all but one note in the set. When deleting the last note, set the a new tag value equal to the desired MAXTAG value.

Failure conditions and partial results

If an error condition is encountered while a multi-note delete request is being processed, it could be the case that some, none, or all of the selected notes were deleted.

For example, suppose a delete notes request needs to send two delete operations to the coupling facility to delete all of the selected notes. After the first operation successfully deletes half the selected notes, the second operation could encounter a quiescing condition (or some other failure). The delete notes request completes with a return and reason code indicating that the request failed (in this example, due to a quiescing condition). In this particular case, some but not all of the intended notes were deleted.

If the failure is recognized before the first delete operation is processed, no notes would have been deleted. If the failure was recognized after the last delete operation is processed, all of the selected notes would have been deleted. If a new maximum tag value was to be set per the MAXTAG specification, the designated tag value might not have been set.

If an answer area is provided, the multi-note details (mapped by ixcynote_tDetailsNotes) might not be reported. If they are reported, the count of the number of notes that were deleted might be lower than the number of notes that were actually deleted.

You might need to reissue the delete request to be sure that all of the relevant notes are deleted.

Note pad related limits

Various limits apply depending on the installed level of the XCF Note Pad Services.

Maximum number of note pad connections per address space:
128

Supported note sizes (in bytes):
0 and 1024

Incremental note size:
1024

Maximum number of selection criteria records for multi-note request:
1

Buffer size and single note requests

Given that the maximum supported note size is 1024 bytes, the provided buffer area (if any) must be at least as big as 1024 bytes when reading or deleting a note. Since the application has no ability to specify the maximum size note that the note pad needs to support, XCF assumes that all note pads support 1024 byte notes. As a consequence, should it be the case that the application only uses null notes, it must still provide a 1024 byte buffer when reading what is always known to be a note with no content.

Given that only two note sizes are supported (0 and 1024), the buffer length must be either 0 or 1024 when creating, writing, or replacing a note.

Buffer size and multi-note requests

If the buffer size is nonzero, it must be a multiple of the incremental note size. That is, a nonzero buffer size must be a multiple of 1024 bytes.

Chapter 13. Coupling Facility Accounting and Measuring Services

There are several XCF and XES services that you can use to monitor performance and system utilization. The major differences among IXCMG, IXCQUERY, and IXLMG with regard to coupling facility resources are reviewed here.

- Use **IXCMG** to gather information about the XCF resources in use by the system on which IXCMG is invoked. You can collect detailed information about XCF signaling paths and/or messages pending delivery, as well as summary information about message traffic between systems in the sysplex, and/or message traffic between XCF group members.

  If you are using coupling facility structures for XCF signaling, then IXCMG will return information about XCF's use of those structures for signaling.
- Use **IXCQUERY** to receive general or detailed information about coupling facilities and/or coupling facility structures defined in the active CFRM policy. The coupling facility information includes that which you would find in the CFRM policy, such as coupling facility identifier, size of the dump space, the number of systems connected along with each system's identifier, and structure names and structure allocation status. Similarly, structure information includes that which you would find in the structure definition section of a CFRM policy, having to do with structure name, size, and preference and exclusion lists.
- Use **IXLMG** to gather system-related information, such as configuration data, usage statistics, and subchannel utilization and coupling facility-related information, such as coupling facility structure limits, structure controls, cast-out class data, and storage class data.

Each of these services has its own place in helping to manage your sysplex with a coupling facility. Information about IXLMG follows. See “Obtaining Tuning and Capacity Planning Information” on page 111 for additional information about IXCMG and “Using the IXCQUERY Macro” on page 98 for IXCQUERY.

Using IXLMG

Installations that use a coupling facility need data for capacity planning and for tuning. RMF provides this type of information for each coupling facility attached to the sysplex. The data gathered can be used to monitor how effectively the coupling facility is being utilized and to indicate possible performance constraints in a sysplex environment. See *z/OS Resource Measurement Facility User's Guide* for a description of the RMF Coupling Facility Activity Report.

The IXLMG macro is the mechanism by which authorized routines can collect information from individual systems and from coupling facilities. The programs that use IXLMG may be, but are not required to be, connectors to a structure in the coupling facility. The information provided by the IXLMG macro is mapped by the IXLYAMDA macro.

Specifying the information level

The Accounting and Measurement Data Area (IXLYAMDA) supports several levels of information that IXLMG returns. Certain coupling facility and structure requests might provide data that was not returned when the IXLMG service was first made available. For these request types, you can specify the level of information you want with the AMDALEVEL parameter on IXLMG. The AMDALEVEL parameter is available with version 1 of the IXLMG macro. The system returns base AMDA information when you specify AMDALEVEL=0 on your request; the system returns level *n* AMDA information when you specify AMDALEVEL=*n* on your request. Be aware of the type of output that you are requesting and be able to process it correctly. Specifying a larger level might result in larger IXLYAMDA records.

The list in “Types of information available” on page 752 lists the IXLYAMDA mappings that support the various levels of IXLYAMDA information. A structure name to which a number is appended will contain all

© Copyright IBM Corp. 1994, 2023

the information contained in the lower-level structures plus, optionally, additional information pertaining to the information request type. See the IXLYAMDA macro in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for a description of the information returned.

Types of information available

With the IXLMG macro, you specify whether you want usage information reported by facility (either all coupling facilities or a particular coupling facility) or for a single named coupling facility structure. You can also request control information for a structure and measurement information for each coupling facility. Control and measurement information, when requested, is retrieved from the coupling facility itself.

The IXLYAMDA macro provides the following mappings that are related to the type of information requested:

IXLYAMDAREA
Data area that contains header information used to determine the scope of data returned by IXLMG. The information includes:

- The total length of the output data area needed to contain all the requested information. This length includes the area for the records that were already returned on this call.
- The total number of entries of all kinds included in the record.
- Version number of the IXLYAMDA information.

IXLYAMDHD
Header record that is returned for all entry mapping types. The information includes:

- Type of entry
- Length of entry
- Address of next entry.

IXLYAMDCF and IXLYAMDCF1
Coupling facility usage and control information, which includes:

- Configuration data
- Accounting and measurement data
- Control information
- If applicable, pointer to the first CF remote facility record

IXLYAMDSLL and IXLYAMDSLL1
List structure limit information

IXLYAMDSLC and IXLYAMDSLC1
Cache structure limit information

IXLYAMDCFMI
Coupling facility capacity measurement information entry (the header to set up an array of elements mapped by IXLYAMDCFMINFO)

IXLYAMDCFMINFO
Coupling facility capacity measurement information element

IXLYAMDCFRF
Coupling facility remote facility entry

IXLYAMDSTRL, IXLYAMDSTRL1, IXLYAMDSTRL2, IXLAMDSTRL3, and IXLAMDSTRL4
List structure usage and control information. Following the header information are:

- Configuration data
- List measurement data
- List control structure information

IXLYAMDADUP
Structure asynchronous duplexing information (provided only when at least level-2 information is requested)

IXLYAMDSTRC, IXLYAMDSTRC1, IXLYAMDSTRC2, IXLYAMDSTRC3, and IXLYAMDSTRC4
Cache structure usage and control information. Following the header information are:

- Configuration data
- Cache measurement data
- Control information

IXLYAMDSCSC and IXLYAMDSCSC1
Storage class information (cache structure only)

IXLYAMDSCOC
Cast-out class information (cache structure only) (the header to set up an array of elements mapped by IXLYAMDCFMINFO).

IXLYAMDSCOCSTATS
Cast-out class information element

IXLYAMDSC and IXLYAMDSC1
Subchannel information

IXLYAMDSSCC
Structure copy controls information

IXLYAMDCFCP
Coupling facility channel path header information

IXLYAMDCFCPINFO
Coupling facility channel path information

IXLYAMDSSCM
Storage-class memory information for a structure

If the specified structure is in the process of structure rebuild or duplexing rebuild, IXLMG returns information for both the old structure and the new structure.

The layout of the IXLYAMDA information is depicted in Figure 86 on page 754. If you specified AMDALEVEL=*n* on the IXLMG macro, you will receive level-*n* IXLYAMDA records.

The IXLYAMDSTRL1 and IXLYAMDSTRC1 records contain additional new information.

- Count of “CF-to-CF link not available” conditions, per structure
- Count of “execution suppressed” conditions, per structure
- Count, sum of times, and sum of squares of times, for peer wait for subchannel conditions in which a request is waiting for a peer subchannel to become available for use, without any reserved subchannel held for the operation to the current structure. (System-managed duplexing rebuild uses a “peer subchannel” to drive the second half of a duplexed request to the associated coupling facility.)
- Count, sum of times, and sum of squares of times, for peer wait with reserve conditions in which a request is waiting for a peer subchannel to become available for use, with a reserved subchannel held for the operation to the current structure. The subchannel is not available for use during this time.
- Count, sum of times, and sum of squares of times, for peer wait for completion conditions in which a subchannel for a duplexed operation is waiting for the completion of the operation to the other structure instance. The subchannel is not available for use during this time.

For each coupling facility specified on the IXLMG request, the following information could potentially be returned. The symbol “...” indicates that there could be more than one of that type of entry. For example, IXLMG could return a AMDSTRL entry for each eligible list structure in a coupling facility.

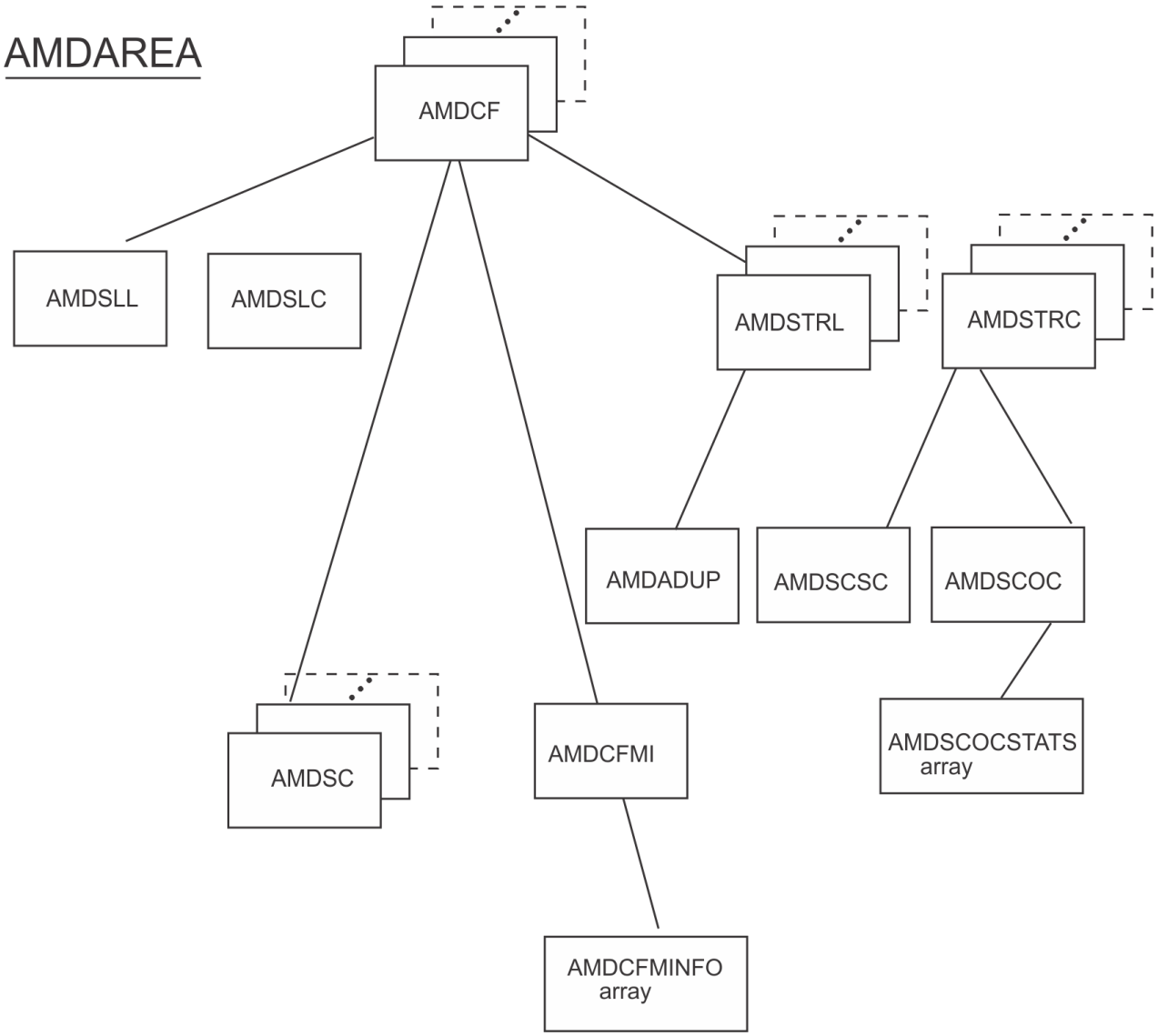


Figure 86. Layout of IXLYAMDA

For a complete description of IXLYAMDA, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Defining an Output Area

When you code the IXLMG macro, you specify where you want the information placed (with the DATAAREA parameter) and the length of the area (with the DATALEN parameter). If the size of the area is too small to contain the requested measurement data, MVS returns as much data as can fit in the area you provided. MVS also sets a reason code (IXLRSNCODEMOREDATA) to indicate that more data is available. The proper size for the data area is returned in the IXLYAMDA header. Note that if you provide a larger area for the requested data, subsequent invocations of IXLMG return the latest information from the coupling facility, which may differ slightly from the original data returned when the area was too small.

- **Handling the IXLRSNCODEMOREDATA Reason Code**

  The IXLRSNCODEMOREDATA reason code indicates that the DATAAREA you provided is too small to contain all the requested data. You can reissue the IXLMG macro using the value returned in IXLYAMDAAREA_TLEN (total length of output data area needed to contain all the requested information) as the length of your data area. However, as noted, the IXLMG information returned is a snapshot of the current environment — which might change between one invocation of IXLMG and the next. (For example, additional coupling facilities might have been added or removed from the sysplex, thus

changing the number of coupling facility records in the output data area.) You must provide code to handle the IXLRSNCODEMOREDATA reason code in case the length of the record(s) you are requesting ever changes.

- **Retrieving Information from the Output Data Area**

  The output data area mapped by IXLYAMDA can contain one or more instances of many different types of records, depending on your IXLMG request. To help you reference each of the record types, the data area contains fields indicating the length of the record type and pointers to the next entry for the same record type. You must use these fields to index through the data area in case the length of the record(s) you are requesting ever changes. Using the DSECT length of a particular record type is not recommended because the length might have been changed since your program was assembled.

Programming Considerations

Depending on the type of information requested, IXLMG might reference the CFRM active policy. Multiple IXLMG requests could result in a large amount of I/O to the CFRM couple data set, which in turn, could generate a noticeable loss of system performance. When designing an application such as a sysplex monitoring tool that uses the IXLMG macro, be aware of the performance effect of multiple macro invocations.

Specifying the Information To Be Returned by IXLMG

The amount of information that IXLMG returns depends on:

- Whether the system on which IXLMG is invoked has a configured connection to the coupling facility for which information is requested, and
- Whether the coupling facility contains one or more structures with active XES connectors on the system from which IXLMG is invoked.

You can specify that you want either coupling facility information (with or without the associated statistics gathered from the coupling facility) or coupling facility structure information.

Coupling Facility Information

You can request information about all coupling facilities that are attached to the system on which the IXLMG macro is issued or about a specific coupling facility that is connected to the system on which IXLMG is issued. If you specify a coupling facility by name (CFNAME), the data returned includes information about all allocated structures within the coupling facility as well as the coupling facility information.

The statistics gathered from the coupling facility (requested by specifying the HWSTATISTICS=YES parameter) include structure control information and coupling facility measurement data. A structure ID can be specified to limit structure processing to that specific structure ID, otherwise all allocated structures are processed. If you do not want this information, you must explicitly code HWSTATISTICS=NO. If you want the statistics gathered from the coupling facility to include only the coupling facility measurement data but not the structure control information, code HWSTATISTICS=CF.

Note that the number of accesses to the coupling facility for data gathering might degrade a system's performance. If you need primarily coupling facility statistics as opposed to individual structure statistics, you should use HWSTATISTICS=CF, which will generate fewer accesses to the coupling facility than HWSTATISTICS=YES.

The coupling facility information returned includes:

- If HWSTATISTICS=YES
  - AMDCF and AMDCF1 include configuration data, accounting and measurement data, and control information.
  - AMDSLL and AMDSLL1 (list structure limits) and AMDSLC and AMDSLC1 (cache structure limits) are returned.

  - AMDSTRL and AMDSTRL1 -n (list structure entry) and AMDSTRC and AMDSTRC1 -n (cache structure entry) are returned for all structures in the coupling facility regardless of whether there's an active XES connection to that structure on this system. AMDSTRL, AMDSTRL1 -n, AMDSTRC, and AMDSTRC1 -n include configuration data, measurement data, and control information for all structures.

    Structure usage information in AMDSTRL, AMDSTRL1 -n, AMDSTRC, and AMDSTRC1 -n is available only for structures that have an active XES connection to that structure on this system.
  - AMDSC and AMDSC1 (subchannel information) include configuration and contention data.
  - AMDCFMI and AMDCFMINFO (measurement information array) includes measurement information elements.
- If HWSTATISTICS=NO
  - AMDCF and AMDCF1 include configuration data, accounting and measurement data, and control information.
  - AMDSLL and AMDSLL1 (list structure limits) and AMDSLC and AMDSLC1 (cache structure limits) are returned.
  - AMDSTRL and AMDSTRL1 -n (list structure entry) and AMDSTRC and AMDSTRC1 -n (cache structure entry) are returned only for structures that have an active XES connection to that structure on this system. AMDSTRL, AMDSTRL1, AMDSTRC, and AMDSTRC1 include configuration data and measurement data.

    AMDSTRL, AMDSTRL1 -n, AMDSTRC, and AMDSTRC1 -n do not contain structure control information.
  - AMDSC and AMDSC1 (subchannel information) include configuration and contention data.
  - AMDCFMI and AMDCFMINFO (measurement information array) are not returned.
- If HWSTATISTICS=CF
  - AMDCF and AMDCF1 include configuration data, accounting and measurement data, and control information.
  - AMDSLL and AMDSLL1 (list structure limits) and AMDSLC and AMDSLC1 (cache structure limits) are returned.
  - AMDSTRL and AMDSTRL1 -n (list structure entry) and AMDSTRC and AMDSTRC1 -n (cache structure entry) are returned only for structures that have an active XES connection to that structure on this system. AMDSTRL, AMDSTRL1 -n, AMDSTRC, and AMDSTRC1 -n include configuration data and measurement data.

    AMDSTRL, AMDSTRL1 -n, AMDSTRC, and AMDSTRC1 -n do not contain structure control information.
  - AMDSC and AMDSC1 (subchannel information) include configuration and contention data.
  - AMDCFMI and AMDCFMINFO (measurement information array) include measurement information elements.
- If HWSTATISTICS=YES or CF **and** AMDALEVEL=1 with CFLEVEL=10 or higher, the following information pertains to remotely connected coupling facilities:
  - AMDCF and AMDCF1 includes an additional field to report the remotely connected coupling facilities for each subject CF. The remotely connected coupling facility records are chained together in a linked list in the IXLYAMDA answer area, so that information for any number of remotely connected CFs connected to a given coupling facility can be returned.
  - AMDCFRF contains information about each remotely connected coupling facility:
    - Node descriptor (ND)
    - System identifier (SYID) that designates the coupling facility's ownership by a particular sysplex.
    - Coupling facility name (CFNAME) — if available, otherwise binary zeros.
    - Path group size (PGS) — the number of currently active paths (CF-to-CF links) connecting this coupling facility to the remote coupling facility.

- CF-to-CF signal counter information. (CF-to-CF signals are those signals exchanged over the CF-to-CF links to synchronize the execution of duplexed commands in system-managed duplexing rebuild.)
- CF-to-CF signal service time information, including sum of times and sum of squares of times for signal service times and delay times.

  Note that a remotely connected coupling facility entry for a coupling facility at CFLEVEL=10 will specify the sum of signal service times in IXLYAMDCFRF_SSTFM. For a coupling facility at CFLEVEL=11 or higher, the same information may also appear in IXLYAMDCFRF_SSTFME. For CFLEVEL=11, the SSTRM field was extended to 64-bits to prevent the value from wrapping in a given data-gathering interval. To determine which field to use, test IXLYAMDCFRF_SSTFME. If it is non-zero, use that value; otherwise obtain the value from IXLYAMDCFRF_SSTFM.
- Chpid type information for each currently active path in the path group.

Coupling Facility Structure Information

You can request information about a single named structure (STRNAME) that is allocated in a coupling facility attached to the system on which the IXLMG macro is issued. The data returned includes:

- AMDCF (AMDCF1)
- AMDSLL (AMDSLL1) and AMDSLC (AMDSLC1)
- AMDSTRL (AMDSTRL1 -n) or AMDSTRC (AMDSTRC1 -n), depending on whether the structure is a list or cache structure. AMDSTRL (AMSDTRL1 -n) or AMDSTRC (AMDSTRC1 -n) includes structure control information.

  AMDSTRL (AMDSTRL1 -n) or AMDSTRC (AMDSTRC1 -n) contain structure usage information only for structures that have an active XES connection to that structure on this system.
- AMDSC (AMDSC1)
- AMDCFMI and AMDCFMINFO are not returned.
- For a cache structure only,
  - If cast-out class information was requested, AMDSCOC and AMDSCOCSTATS are returned.
  - If storage class information was requested, AMDSCSC (AMDSCSC1) is returned.
- If HWSTATISTICS=YES or CF **and** AMDALEVEL=1 with CFLEVEL=10 or higher
  - AMDCF and AMDCF1 includes an additional field to report the remotely connected coupling facilities for each subject CF. The remotely connected coupling facility records are chained together in a linked list in the IXLYAMDA answer area, so that information for any number of remotely connected CFs connected to a given coupling facility can be returned.

    Information in the coupling facility remotely connected (CFRF) record includes:
    - Node descriptor (ND)
    - System identifier (SYID) that designates the coupling facility's ownership by a particular sysplex.
    - Coupling facility name (CFNAME) — if available, otherwise binary zeros.
    - Path group size (PGS) — the number of currently active paths (CF-to-CF links) connecting this coupling facility to the remote coupling facility.
    - CF-to-CF signal counter information. (CF-to-CF signals are those signals exchanged over the CF-to-CF links to synchronize the execution of duplexed commands in system-managed duplexing rebuild.)
    - CF-to-CF signal service time information.
    - Chpid type information for each currently active path in the path group.

Chapter 14. Dumping Services for Coupling Facility Structures

Two MVS services exist specifically to support the dumping of coupling facility structures. The first, IHABLDP, lets you build a parameter list to be passed as input to SDUMPX, the SVC Dump macro. Using the IHABLDP macro is the only way to inform SDUMPX of the specific structure information you want included in the dump.

The syntax of SDUMPX is described in *z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU*. The syntax of IHABLDP is described in *z/OS MVS Programming: Sysplex Services Reference*.

The second MVS service, IXLZSTR, lets you to extract specific structure information from an SVC dump in an IPCS environment. See *z/OS MVS IPCS User's Guide* for guidance in working in an IPCS environment.

The syntax of IXLZSTR is described in *z/OS MVS Programming: Sysplex Services Reference*.

MVS also provides a set of macros that you can use to map the coupling facility structure data in the dump data set, where the structure information resides in IPCS COMPDATA spaces.

Using the IHABLDP Macro

The IHABLDP macro builds a parameter list to be passed as input to the SDUMPX macro. SDUMPX allows you to request structure information from a coupling facility

You build the parameter list by multiple invocations of the IHABLDP macro. For each structure that you specify in the parameter list:

- You can specify that you want a range of information (for example, a range of cast-out classes for a cache structure), or
- You can specify that you want specific options included (such as the lock table entries associated with a list structure).

You begin building the parameter list with the TYPE=INITIAL parameter; you end its construction with the TYPE=ENDLIST parameter. The parameter list is mapped by the IHASDSTR macro.

The size of the area in which you build the parameter list depends on the amount of structure information requested. For the best utilization of space within the parameter list, you should group all the range and option requests for a single structure together. The information for each structure will be dumped in the order it is specified in the parameter list.

Once the parameter list is built, you can specify it as input to SVC Dump by specifying its address on the SDUMPX STRLIST keyword.

Using the IXLZSTR Macro

Use the IXLZSTR macro in an IPCS environment to retrieve coupling facility structure information from a dump containing the data. The macro builds a parameter list to specify the requested information, calls the access service, and then returns the requested information in an answer area that you provide.

Requesting Structure Information

To determine what structure information is in a dump, issue the IXLZSTR macro requesting summary data. IXLZSTR returns the names and types of all structures for which there is information in the dump. From there, you can request the appropriate type of data depending on the structure type (for example, storage classes for a cache macro).

© Copyright IBM Corp. 1994, 2023

Receiving Information Returned by the IXLZSTR Macro

When IXLZSTR returns the requested information in the answer area that you provide, the first entry is a header record that describes the contents of the area. The header contains:

- The number of entries
- The length of the entry
- Whether the structure is a list or a cache structure
- Information pertinent to the request.

The remainder of the answer area contains one or more entries for the requested information. The header information is mapped by the STRBHEADER mapping in IXLZSTRB. Other information that the answer area might contain, depending on the request, is mapped by additional mappings in IXLZSTRB as well as by the following macros:

IHAARB
Associated request block

IXLYDCAC
Dumping cache structure controls

IXLYDCCC
Dumping cast-out class controls

IXLYDDIB
Dumping information block mappings. Includes the following:

- Lock table entry
- List entry control block
- Directory information block
- List user control block
- Local cache control block
- Event monitor controls block.

IXLYDEQC
Dumping event queue controls

IXLYDLC
Dumping list header controls and the list monitor table entries found in the list controls

IXLYDLCC
Dumping local cache controls

IXLYDLIC
Dumping list structure controls

IXLYDLUC
Dumping list user controls

IXLUDSCC
Dumping storage class controls

IXLYSTRC
Partial dump reason code constants.

Using Component Data in the Dump Data Set

When coupling facility structure data is written to the dump data set, the data is organized into several different COMPDATA spaces. Each COMPDATA space contains a specific type of data. Figure 87 on page 761 provides a diagram of the types of coupling facility structure data available in the dump data set.

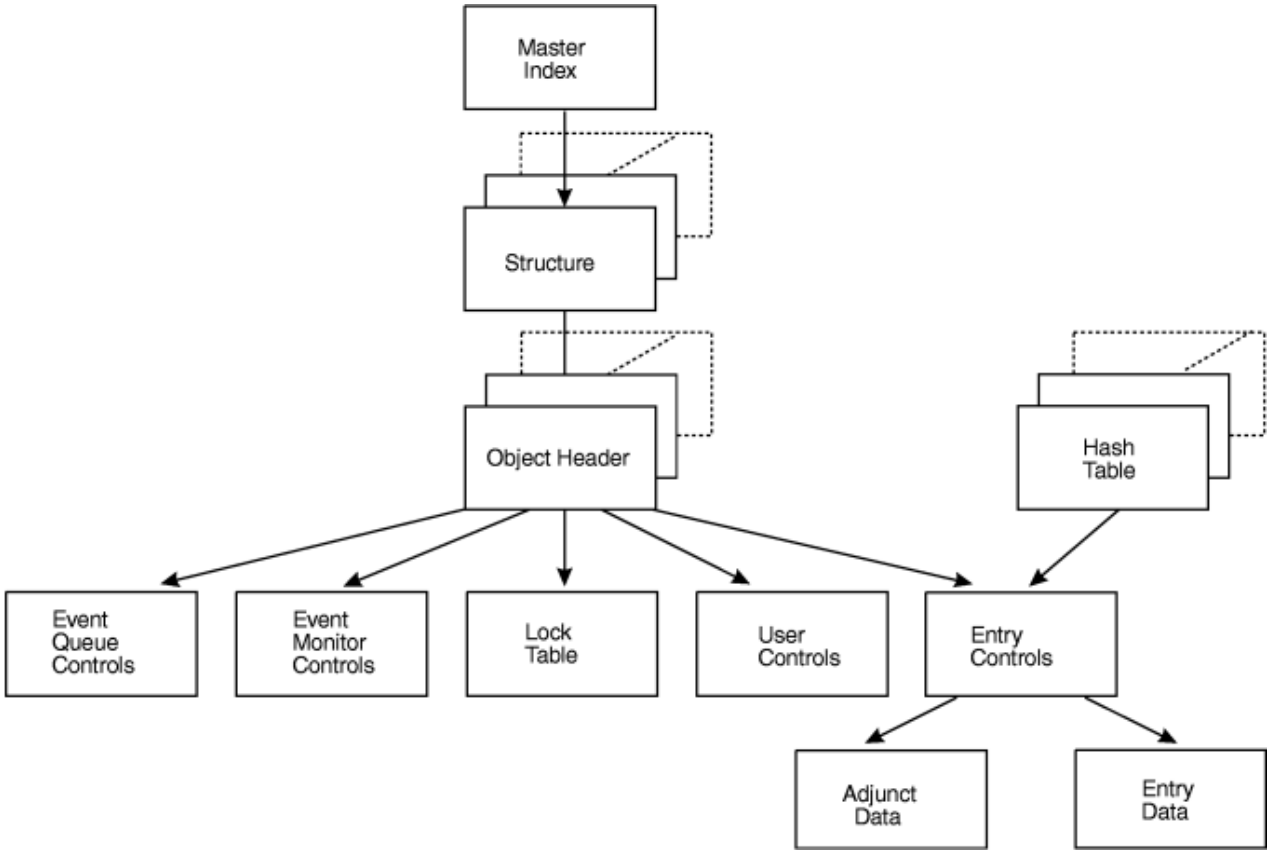


Figure 87. Format of Coupling Facility Structure Data in Dump Data Set

Within the dump data set there is one master index COMPDATA space that identifies the structures in the dump and provides an index into the other COMPDATA spaces for the structure. The diagram shows that for a structure identified in the master index, there may be one or more structure, object header, and hash table COMPDATA space records.

The name of the master index COMPDATA space is CFD0000I. The naming convention for the other COMPDATA spaces containing coupling facility structure data allows you to index through each of the other types. Names are of the format CFDxxyy_, where:

- CFD is the component prefix
- xx is the sequence number of the space. All sequence numbers start at 00.
- yy is the structure number that appears in the master index entry for the structure
- _ is an alphabetic character indicating the type of data in the COMPDATA space.

To advance to the next type of COMPDATA space for a structure, increment the xx part of the COMPDATA space name by one.

All of the records in the COMPDATA spaces start at address X'1000'. The hash table compdata space is the only type not pointed to by another COMPDATA space. To access the hash table COMPDATA space, use its name (CFDxxyyH) and address (X'1000').

Table 49 on page 761 lists each COMPDATA space by name and describes the contents of each.

Table 49. Coupling Facility Structure COMPDATA Space Descriptions

Data Type	Name	Description
Master Index	CFD0000I	Contains the index of all coupling facility structures that were requested to be dumped.

Table 49. Coupling Facility Structure COMPDATA Space Descriptions (continued)		
Data Type	**Name**	**Description**
Structure	CFDxxyyS	Associated with each structure that is listed in the master index. Summarizes the contents of this space.
Object Header	CFDxxyyO	Contains the object headers for all of the objects that were dumped for the structure. Objects are castout classes, storage classes, list numbers, lock tables, and user controls.
Hash Table	CFDxxyyH	Provides a way to get to the entries in classes or list numbers by entry name or by entry identifier.
Lock Table	CFDxxyyL	Contains all the nonzero lock table entries from the lock table, if one was defined for the structure.
Event Monitor Controls	CFDxxyyE	Contains all the event monitor controls for the structure.
Event Queue Controls	CFDxxyyQ	Contains all the event queue controls for the structure.
User Control	CFDxxyyU	Contains user control information for the structure.
Entry Control	CFDxxyyC	Contains control information about the entries that were dumped for castout classes, storage classes, or list numbers.
Adjunct Data	CFDxxyyA	Contains the adjunct data for all the entries that have adjunct data associated with them.
Entry Data	CFDxxyyD	Contains the entry data for all the entries that have entry data associated with them.

Associating Macros with the Data Types

The data in the COMPDATA spaces can be mapped by several macros. The following describes each compdata space type and the associated macros.

Master Index

Master index entries, sorted alphabetically by structure name, are mapped by **IXLYCOMP** (COMPINDEX mapping). There is one entry for every structure in the dump. The information in each entry includes:

- Name of the structure
- An indicator specifying a reason why the structure was not dumped, if applicable (reason codes are defined in **IXLYSTRC**)
- A structure number by which you can identify different COMPDATA spaces with the structure
- A pointer to the structure trailer.

The structure trailer is mapped by **IXLYCOMP** (COMPSTRTRL mapping). It is available for each structure in the dump unless the dump data set is full or an I/O error occurred. The information in the structure trailer includes:

- An indicator specifying whether the requested structure information was dumped completely or partially
- An indicator specifying a reason why the structure was dumped partially (reason codes are defined in **IXLYSTRC**)
- Flags to indicate whether lock table entries and user controls were dumped for the structure.

Structure

For each structure in the master index, there may be one or more structure COMPDATA spaces associated with the structure. A structure COMPDATA space can consist of up to four parts.

- The structure dump space header, which appears in each structure COMPDATA space, is mapped by **IXLYCOMP** (COMPSTRHDR mapping). The information in the dump space header includes:
  - A pointer to the dump header for a given structure
  - A pointer to the object map index within the structure COMPDATA space.
- The dump header, which appears only in the first structure COMPDATA space for a structure, is mapped by **IHADWHDR**. The dump header includes the following:
  - Information about the dump of the structure and the structure controls associated with the structure
    - **IXLYDCAC** maps cache structure controls
    - **IXLYDLIC** maps list structure controls
  - The associated request block, mapped by **IHAARB**, which contains the list of objects and ranges that were requested for the structure.
- The object map index is mapped by **IXLYCOMP** (COMPSTROBJMAPINDEX mapping). The information includes:
  - A list of pointers to the beginning of each object that was dumped for the structure
  - The minimum value and maximum value of the identifier for each object.
- The object map, which can span more than one structure COMPDATA space, is mapped by **IXLYCOMP** (COMPSTROBJMAP mapping). For each object, the information includes:
  - The object type and identifier
  - A pointer to the object header in the object header COMPDATA space
  - A sequence number (xx) to identify an object header COMPDATA space with the object.

Object Header

Contains the object headers for each object that was dumped for the structure. Each object header entry is mapped by **IHADWOBH** and contains the following information:

- Status of the object
- Controls that are associated with the object
- A pointer to the appropriate object COMPDATA space for the first entry dumped for the object
- A sequence number (xx) to identify the object COMPDATA space with the object.

The following macros map the control information associated with the object:

- List header controls — **IXLYDLC**
- List user controls — **IXLYDLUC**
- Local cache controls — **IXLYDLCC**
- Castout class controls — **IXLYDCCC**
- Storage class controls — **IXLYDSCC**
- Event monitor controls — **DEMC** mapping in **IXLYDDIB**
- Event queue controls — **IXLYDEQC**

If the object is a lock table, the sequence number and the address of the first entry dumped for the lock table appear in the object header. If the object is the user controls for a structure, the sequence number and the address of the first entry dumped for the user controls appear in the object header.

Hash Table

Provides a way to access entries in classes or list numbers by entry name or entry identifier.

- The hash table header, mapped by **IXLYCOMP** (COMPHASHTABLEHDR mapping), indicates the number of slots that are in the hash table and points to the hash table slot array.
- The hash table slot array, mapped by **IXLYCOMP** (COMPHASHSLOTARRAY mapping), is an array of pointers to the lists of hash elements
- A hash table element, mapped by **IXLYCOMP** (COMPHASHELEM mapping), contains the following information for each entry on the list:
  - An indicator to specify whether the element corresponds to an entry name or entry identifier
  - A pointer to the appropriate entry control COMPDATA space associated with the hash table element
  - A sequence number (xx) to associate the entry control COMPDATA space with the element.

Lock Table
Contains all of the nonzero lock table entries from the lock table, if applicable, that was written to the dump data set. Each lock table entry is mapped by **IXLYDDIB** (DLTE mapping), and includes the following:

- Index of the lock table entry
- Contents of the lock table entry

User Control
Contains the user control information about all connected users to the structure. The information in mapped by **IXLYDDIB** (DLUCB mapping for a list structure and DLCCB mapping for a cache structure).

Event Monitor Controls
Contains the event monitor controls information for all connected users to the structure. The information is mapped by the DEMC mapping of IXLYDDIB and includes the following:

- Connection identifier
- Whether the EMC is queued to this connector's event queue
- List number with which the EMC is associated
- List entry key of the sublist with which the EMC is associated
- User notification control data.

Event Queue Control
Contains the event queue control information for all connected users to the structure. The information is mapped by IXLYDEQC and includes the following:

- Connection identifier
- Whether the list transition exit is to be driven when a list transition occurs
- Whether event queue monitoring is in effect
- Vector index associated with the event queue
- Counts of EMCs queued to the event queue and event queue transitions.

Entry Control
Contains control information about entries that were dumped for cast-out classes, storage classes, or list numbers. For each structure in the master index, there may be one or more entry control compdata spaces.

- The entry control header, mapped by **IXLYCOMP** (COMPENTRYCNTL mapping), includes the following information for each entry that was dumped for castout classes, storage classes, or list numbers.
  - Status data about the items that were dumped
  - Pointers to the entry's adjunct data in the adjunct COMPDATA space and entry data in the entry data COMPDATA space, if applicable
  - Length of the entry data, if applicable

  - Sequence numbers (xx) to identify the adjunct COMPDATA space and the entry data COMPDATA space with the entry
- The entry controls associated with the entry are mapped by **IXLYDDIB** (DDIL mapping for a list structure and DDIC mapping for a cache structure). Entry controls that were located in storage-class memory at the time of the dump are identified by a placeholder DDIB that contains a storage-class memory token rather than actual entry data.

Adjunct Data
Contains the 64 bytes of adjunct data for each of the entries that have associated adjunct data. There is no mapping for this data.

Entry Data
Contains the entry data for each of the entries that have associated entry data. The length of the entry data is defined by the COMPENTRYCNTLENTRYDATALEN field in the COMPENTRYCNTL mapping. There is no mapping for this data.

See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for a description of the macros used to map the information in the COMPDATA spaces.

Chapter 15. Documenting your Coupling Facility Requirements

Having designed and coded an application that uses the coupling facility, you must now provide the users of the application with a set of guidelines for setting up the coupling facilities in their installation. For each coupling facility structure, you must supply requirements about the characteristics of the structure and the coupling facility in which it is to reside. System programmers will use the structure and coupling facility information you provide to set up the installation's coupling facility resource management (CFRM) policies and to establish run time procedures for their operations staff. *z/OS MVS Setting Up a Sysplex* describes these and other tasks that the system programmer must complete.

Specifying the Coupling Facility Structure Requirements

When setting up the CFRM policy, the installation must be aware of the following structure attributes:

- Name of the structure
- Size of the structure (which might include both an initial size and a maximum size)
- Names of other structures with which your structure should not share the same coupling facility

Other attributes of which the user should be aware are whether the structure is persistent, whether it can be rebuilt or altered, whether it supports system-managed processes, whether it has a requirement for a specific level of coupling facility, and the connectivity requirements of the application. Also, when threshold monitoring is active, if the structure is expected to have a percent full threshold above the default value of 80%, this too should be specified in the CFRM policy.

The user also must be aware of the characteristics of the coupling facility in which the structure might be allocated. See “Specifying the coupling facility requirements” on page 769.

Naming the Structure

If your application has hardcoded a structure name, the installation must provide the name in the CFRM policy. If you derive the name from another source or query the policy to determine the name, indicate that fact to the user.

Determining the Structure Size

The size of the structure is installation-specific in most cases. You must supply some means of calculating an initial structure size, and if appropriate, a method for tuning the structure size after it is in use by your application. To determine structure size, the following methods are available.

- The CFSizer web-based structure sizing tool.
- The IXLCSP structure computation service.

To use either method, you must first identify and document:

- The fixed attributes of the structure.

  Document the attributes that do not vary with workload or environmental configuration, such as whether your list or cache structure has adjunct areas, whether a lock structure has record data, the size of data elements, or whether the structure supports alter. Depending on the application design, attributes such as the number of list headers in a list structure, the number of lock table entries in a lock or serialized list structure, the data element size, or the maximum number of data elements per list or directory entry might be fixed or variable.
- The inputs necessary to describe the application’s workload, including instructions on how to obtain the values.

© Copyright IBM Corp. 1994, 2023

These inputs depend on the nature of the application. For a messaging application, for example, they would include the size and arrival rate of the messages, and perhaps information about the number of senders and recipients. For a structure that uses list entries to store token information of some kind, the inputs would likely include the rate of events that generate the stored information. For each input, document its source. For dynamic inputs, that might be RMF records, application-specific display or query outputs, logs, and so on. Environmental inputs might come from parmlib members, procedure specifications, policy information, and so on.

- The relationship between the inputs describing the workload and the variable contents of the structure.

  Document how to convert the input information into counts of structure objects such as list entries and data elements. For example, in a messaging application there might be a one-to-one relationship between messages and list entries, but the number of data elements per list entry might vary with average message size.

To size a structure for a specific workload, convert the workload description to an estimate of the structure object counts. Then, supply the counts and fixed attributes as input either to CFSizer or IXLCSP.

- If using CFSizer, see the OEM input page.
- The IXLCSP service accepts as input the structure attributes and object counts and returns the structure size appropriate to the CFLEVEL of the target facility. The input parameters that are passed as input to IXLCSP map directly to the parameters specified on IXLCONN.

If you are defining a new structure for an externally available IBM product, send an email to the point of contact that is listed at the bottom of the CFSizer welcome page to have your structure formally supported by the CFSizer tool.

Providing an Exclusion List

The exclusion list in a CFRM policy is a list of structure names with which a particular structure is not to share the same coupling facility. If your structure has high activity, you should state that fact, so that the installation does not place the structure in a coupling facility with another high activity structure. Another example of using the exclusion list is to separate a backup copy of a structure from its original instance to avoid a single point of failure.

Understanding the Persistence Attribute

The installation needs to know how you handle your structure when there are no active connectors to it. Console messages might require that an operator take some action against the structure, so it is important that the installation understands the structure's use.

A structure that you define as persistent will remain allocated in the coupling facility even when there are no active connectors to it. To delete a persistent structure from a coupling facility, the operator must issue a SETXCF FORCE,STRUCTURE command.

Specifying the Rebuild and/or Alter Attribute

The installation needs to know whether your structure can be rebuilt at another location and whether have it can have its size and/or entry-to-element ratio altered. Operator commands allow the installation to initiate these structure rebuild and structure alter actions and to disallow the structure alter action. The installation also can specify in its CFRM policy whether MVS is to initiate a structure rebuild if a certain percentage of connectors lose connectivity to the structure. The installation also can specify in its CFRM policy whether MVS is to initiate a structure rebuild if a certain percentage of connectors lose connectivity to the structure.

If your structure can be rebuilt, the installation must ensure that there is coupling facility space available for the rebuild. If the structure is eligible for a system-managed rebuild, there is the additional requirement that at least two coupling facilities at CFLEVEL=8 or higher are listed in the CFRM policy preference list for the structure. If the structure is eligible for a system-managed duplexing rebuild, there is the additional requirement that at least two coupling facilities at CFLEVEL=11 or higher are listed in the CFRM policy preference list for the structure.

Providing Connectivity Requirements

The installation needs to know what level of connectivity each system in the sysplex must have to your application's structure in a coupling facility. If all systems in the sysplex must be connected to the structure (IXLCONN CONNECTIVITY=SYSPLEX), the installation must be aware that your application will fail unless they have configured their systems and coupling facilities accordingly. If your application requires that the structure be allocated in the coupling facility that provides the best global connectivity to systems in the sysplex (IXLCONN CONNECTIVITY=BESTGLOBAL), the installation must attempt to configure their sysplex with the systems most important to the application attached to the same coupling facility and with the highest SFM weights.

An additional requirement for the installation to understand is the rebuild protocol that your application follows with regard to the connectivity requirement. If your connectivity requirement is CONNECTIVITY=SYSPLEX, the rebuild will not be successful until a sysplex-connected coupling facility is available. Therefore, to allow for rebuild that is necessitated by a loss of connectivity or in which LOCATION=OTHER has been specified, the preference list for the structure in the CFRM policy must contain the names of at least two fully-connected coupling facilities.

Specifying the coupling facility requirements

When setting up the CFRM policy, the installation also must be aware of certain coupling facility attributes. If your structure has the following coupling facility characteristics, you must document them:

- Nonvolatility — the requirement that a coupling facility must be able to preserve the structure storage over a utility power failure.
- Failure-independence — the requirement that a coupling facility be in a separate configuration from the system accessing it, thus eliminating a single point of failure.
- Coupling facility level — the requirement that a coupling facility have a certain level of coupling facility control code (CFCC).

Additionally, the application must document whether it supports system-managed protocols. If so, the installation must provide at least two coupling facilities at CFLEVEL=8 or higher to support system-managed rebuild. For system-managed duplexing rebuild, at least two coupling facilities at CFLEVEL=11 or higher must be provided. For system-managed asynchronous duplexing, at least two coupling facilities at CFLEVEL=21 or higher must be provided.

Knowing these requirements enables the installation to correctly specify a preference list of coupling facilities in which your structure can reside. The system uses the preference list and the SFM system weights when attempting to allocate the structure. The system chooses the first coupling facility in the preference list that meets the following requirements:

- Has connectivity to the system that is trying to allocate the structure (depending on the application's connectivity specifications)
- Has a CFLEVEL equal to or greater than the requested CFLEVEL or with a CFLEVEL that supports system-managed processes if the application specified ALLOWAUTO=YES. Note that a CFRM policy DUPLEX specification of ASYNC or ASYNCONLY will cause a requested CFLEVEL of at least CFLEVEL=21.
- Has available space greater than or equal to the requested structure size
- Meets the volatility requirement requested
- Meets the failure-independent requirement requested
- Does not contain a structure in the exclusion list.

If no coupling facility in the preference list meets all these requirements, then the system uses the following priorities to attempt to allocate the structure:

- Without the exclusion list requirement
- Without the failure-independent requirement
- Without the volatility requirement
- In a coupling facility that meets or exceeds the CFLEVEL requirement and has the most available space.

Summarizing Your Requirements

You should document your structure and coupling facility requirements with the installation planning information that you provide for your application.

Appendix A. Accessibility

Accessible publications for this product are offered through IBM Documentation (www.ibm.com/docs/en/zos).

If you experience difficulty with the accessibility of any z/OS information, send a detailed message to the Contact the z/OS team web page (www.ibm.com/systems/campaignmail/z/zos/contact_z) or use the following mailing address.

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
United States

© Copyright IBM Corp. 1994, 2023

Notices

This information was developed for products and services that are offered in the USA or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

This information could include missing, incorrect, or broken hyperlinks. Hyperlinks are maintained in only the HTML plug-in output for IBM Documentation. Use of hyperlinks in other output formats of this information is at your own risk.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Site Counsel
2455 South Road

© Copyright IBM Corp. 1994, 2023

Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or

reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at ibm.com®/privacy and IBM's Online Privacy Statement at ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at ibm.com/software/info/product-privacy.

Policy for unsupported hardware

Various z/OS elements, such as DFSMSdfp, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those

products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

Programming Interface Information

This book documents intended programming interfaces that allow the customer to write programs to obtain services of z/OS.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and Trademark information (www.ibm.com/legal/copytrade.shtml).

Index

A

accessibility
  contact IBM 771
assistive technologies 771
Async Duplex Established phase 285, 314
asynchronous duplexing 634
automatic restart management
  introduction 175

C

cache services
  asynchronous IXLCACHE operation
    suspending task while waiting for completion 679
    testing for completion 679
cache services (IXLCACHE)
  adjunct area
    purpose 360
  algorithm for storage reclaim
    default algorithm 381
    storage reclaim overview 381
    user-defined algorithm 381
  allocation of a cache structure
    introduction 367
  answer area
    answer area validity 397
    defining 395
    information returned on CASTOUT_DATA request 431
    information returned on CROSS_INVAL request 450
    information returned on DELETE_NAME request 445
    information returned on PROCESS_REFLIST request 459
    information returned on READ_COCLASS request 467
    information returned on READ_COSTATS request 471
    information returned on READ_DATA request 420
    information returned on READ_STGSTATS request 474
    information returned on RESET_REFBIT request 461
    information returned on SET_RECLVCTR request 457
    information returned on UNLOCK_CASTOUT request 437
    information returned on WRITE_DATA request 410
  asynchronous IXLCACHE operation
    IXLFCOMP macro 388
  asynchronous operation
    MODE parameter 387
  asynchronous request
    completion notification methods 387
    overview 386
    specifying 386
cache services (IXLCACHE) *(continued)*
  buffer
    design consideration 391
    location 358
    managing the buffers 369
    purpose 358
    selection 388
    selection on CASTOUT_DATA request 430
    selection on PROCESS_REFLIST request 458
    selection on READ_COCLASS request 466
    selection on READ_COSTATS request 469
    selection on READ_DATA request 419
    selection on READ_DIRINFO request 463
    selection on UNLOCK_CASTOUT request 436
    selection on WRITE_DATA request 409
    storage key 395
  cache structure
    adjunct area 360
    allocation overview 367
    changing a data item 371
    characteristics 360
    connection overview 367
    data entry 360
    directory 359
    elements 359
    managing resources 380
    measuring resource usage 386
    purpose 358
    relationship of elements 359
    structure life-span 368
  cache system
    cache structure 358
    changing a cached data item 371
    data access, overview 369
    data management, overview 369
    directory-only cache method 365
    elements of a cache system 357
    local cache buffer 358
    local cache vector 359
    maintaining data consistency 373
    managing cache structure resources 380
    managing local cache buffers 369
    managing the local cache vector 368
    methods of using 363
    permanent storage 359
    store-in cache method 363
    store-through cache method 364
    updating permanent storage 372
  cast-out class
    defined 361
    described 385
    maximum number defined 407
    specification on READ_COCLASS request 466
    WRITE_DATA request 407
  cast-out lock
    defined 361
    unlocking 386

cache services (IXLCACHE) *(continued)*
cast-out lock *(continued)*
WRITE_DATA request 406
cast-out process
and storage reclaim 385
assigning cast-out classes 385
CASTOUT_DATA request. 427
defined 361
described 372, 385
establishing 385
CASTOUT_DATA request 427
changed and unchanged data item
described 371
changed data
defined 362
changed versus unchanged data item
on WRITE_DATA request 405
complete exit 357, 474
connect token
specifying 369
connect token (CONTOKEN) 369
connection identification (CONTOKEN) 369
connection to a cache structure
connection life-span 368
introduction 367
CROSS_INVAL request 449
data consistency
maintaining 373
data element
number in a data entry 408
size 408
data entry
characteristics 360
purpose 360
size 408
data item
accessing 369
casting-out 385
changing in a cache structure 371
defined 362
determining validity 377
IBM serialization recommendation 377
identifying to cache structure 370
maintaining data consistency 373
managing data access 377
managing data item storage 384
managing storage reclaim 383
parity assignment 407
serializing data access 377
DELETE_NAME request 442
deregister interest in a data item 375
deregistration of interest in a data item
defined 362
directory
information returned on READ_DIRINFO request 463
purpose 359
directory entry
purpose 359
UNLOCK_CASTOUT request's affect on 437
directory-only cache method
considerations for using 365
description 365
IXLCACHE services typically used 365
cache services (IXLCACHE) *(continued)*
directory-only cache method *(continued)*
updating permanent storage 373
directory-only usage summary 365
elements of a cache structure
adjunct area 360
characteristics 360
data entry 360
directory 359
elements of a cache system
cache structure 358
local cache buffer 358
local cache vector 359
permanent storage 359
exit 474
introduction 357
invalidate a data item 375
invalidation of a data item
defined 362
IXLFCOMP macro
used with IXLCACHE macro 388
IXLVECTR macro
changing the size of the local cache vector 377
determining data item validity 377
maintaining data consistency 373
local cache vector
changing the vector size 377
IXLVECTR macro 368
location 359
managing 368
purpose 359
role in maintaining data consistency 373
methods of using a cache system
directory-only cache 365
store-in cache 363
store-through cache 364
mode of operation
described 386
overview of usage by cache method 365
pageable storage
user or system provided 394
parity of a data item
described 407
permanent storage
purpose 359
updating 372
premature request completion
CROSS_INVAL request 450
DELETE_NAME request 398
READ_COCLASS request 467
READ_COSTATS request 470
READ_DIRINFO request 464
RESET_REFBIT request 460
UNLOCK_CASTOUT request 436
priority of IXLCACHE request 357
process identifier
CASTOUT_DATA request 430
UNLOCK_CASTOUT request 436
WRITE_DATA request 406
PROCESS_REFLIST request 457
READ_COCLASS request 465
READ_COSTATS request 468
READ_DATA request 417
READ_DIRINFO request 461

cache services (IXLCACHE) *(continued)*
READ_STGSTATS request 472
reason code 357
reclaim
defined 362
reclaim processing
activating on SET_RECLVCTR request 456
deactivating on SET_RECLVCTR request 456
default algorithm 381
described 381
example scenarios 454
managing data item storage 383
PROCESS_REFLIST request 383
user-defined reclaim algorithm 381
with SET_RECLVCTR request 454
register interest in a data item
CASTOUT_DATA request 429
defined 362
local cache vector relationship 373
on CASTOUT_DATA request 429
on READ_DATA request 418
on WRITE_DATA request 397, 405
READ_DATA request 418
VECTORINDEX parameter 373
WRITE_DATA request 403
request identification (REQID) 369
request identifier
specifying 369
request identifier (REQID) 369
request processing overview 386
RESET_REFBIT request 459
resources
assigning storage classes 381
managing 380
measuring usage 386
using storage classes 381
restart a request
CROSS_INVAL request 450
DELETE_NAME request 398
READ_COCLASS request 467
READ_COSTATS request 470
READ_DIRINFO request 464
RESET_REFBIT request 460
UNLOCK_CASTOUT request 436
return code 357
serialization 357
serializing data access
IXLLOCK macro 377
SET_RECLVCTR request 452
shared data
IBM serialization recommendation 377
managing data access 377
serializing data access 377
statistics
cast-out class on READ_COSTATS request 468
storage class on READ_STGSTATS request 472
storage class
assignment on READ_DATA request 419
assignment on WRITE_DATA request 408
defined 362
described 381
directory-only usage 381
maximum number defined 408, 419
reading statistics on READ_STGSTATS request 472
cache services (IXLCACHE) *(continued)*
storage class *(continued)*
specification on PROCESS_REFLIST request 459
specification on SET_RECLVCTR request 454
statistics returned on READ_STGSTATS request 472
using 381
storage key for buffers 395
storage reclaim
default reclaim algorithm 381
described 381
managing data item storage 383
user-defined reclaim algorithm 381
store-in cache method
changing a cached data item 371
considerations for using 363
description 363
IXLCACHE services typically used 365
updating permanent storage 372
store-in usage summary 365
store-through cache method
changing a cached data item 371
considerations for using 364
description 364
IXLCACHE services typically used 365
updating permanent storage 372
store-through usage summary 365
synchronous operation
MODE parameter 387
synchronous request
overview 386
specifying 386
unchanged and changed data item
described 371
unchanged versus changed data item
on WRITE_DATA request 405
UNLOCK_CASTOUT request 434
UNLOCK_CO_NAME request 439
user-defined data in cache structure
described 408
valid data
defined 363
valid data item
described 373
validate a data item 373
validation of a data item
defined 363
validity of a data item
testing with IXLVECTR macro 377
vector entry
specifying on CASTOUT_DATA request 429
specifying on READ_DATA request 418
specifying on WRITE_DATA request 397, 405
vector index
registering interest in a data item 403, 418, 429
WRITE_DATA request 402
CASTOUT_DATA request
answer area information returned 431
buffer method selection 430
process identifier 430
specifying data for cast-out 429, 430
summary 431
connection services
overview 203
Connection services 203

connection to a structure 222
contact
  z/OS 771
coupling facility
  failure-independence 215
  level 209
  storage allocation 219
  storage increment 221
  structure ID limit 221
  using IXLFORCE to delete objects 350
coupling facility statistics
  gathering with IXLMG 751
coupling facility structure
  definition 197
CROSS_INVAL request
  answer area information returned 450
  identifying data to cross-invalidate 449
  restarting 450
  summary 450

D

DELETE_NAME request
  answer area information returned 445
  identifying data to delete 443
  restarting 398
  summary 445

E

ENF event code 35
  issued after structure alter 332
  purpose 250
  when to listen for 250
  when to use 252
event exit 333

F

feedback xxvii

G

group user routine
  coding 85, 94
  events that cause XCF to schedule 77
  purpose 17
  skipping of events 81

I

introduction to cache services 357
IXCARM
  ASSOCIATE 182
  DEREGISTER 181
  READY 181
  REGISTER 179
  WAITPRED 181
IXCCREAT macro
  using 22, 24, 25
IXCDELET macro
  using 117
IXCJOIN macro
  using 22, 24, 25
IXCJOIN macro *(continued)*
  using 22, 24, 25
IXCLEAVE macro
  using 117
IXCMG macro
  using 111
IXCMOD macro
  using 69
IXCMSGIX macro
  illustration of use 43
  message user routine to invoke IXCMSGIX 52
  using to receive a message 43
IXCMSGOX macro
  using to send a message 29
IXCQUERY macro
  programming considerations 104
  using 98
IXCQUIES macro
  using 117
IXCSETUS macro
  example 25
  using 25
  using for members on different systems 26
IXCTERM macro
  using 117
IXCYAMDA mapping macro
  information mapped 111
IXCYGEPL mapping macro
  information mapped 88
IXCYQUAA mapping macro
  information mapped 104
IXCYSEPL mapping macro
  information mapped 73
IXLADUPX macro 688
IXLALTER macro 204, 333
IXLCACHE operation
  purging 680
IXLDISC macro 346
IXLFCOMP macro 679
IXLFORCE macro 350
IXLLIST operation
  purging 680
IXLLOCK macro 625
IXLLSTC service 615
IXLLSTC: List Structure Control Services
  Monitoring a List 618
IXLLSTE service 599
IXLLSTM service 605
IXLMG macro
  programming considerations 755
IXLPURGE macro 680
IXLRT macro 672
IXLRT operation
  purging 680
IXLUSYNC macro 344
IXLVECTR macro
  list notification vector
    changing size 680
    checking list state 681
    testing a range of list notification vector entries 682
    testing whether a list is full or not-full 682
  list notification vectors 680
  local cache vector
    changing size 683

IXLVECTR macro *(continued)*
local cache vector *(continued)*
checking the state of a range of data items 684
checking validity of local cache buffer 683
IXLYAMDA macro
information mapped 752
IXLYLMI macro 565

K

keyboard
navigation 771
PF keys 771
shortcut keys 771

L

list services
answer area validity 535
asynchronous IXLLIST operation
suspending task while waiting for completion 679
testing for completion 679
IXLLSTC request 615
IXLLSTE request 599
IXLLSTM request 605
MONITOR_EVENTQ request 574
list services (IXLLIST)
adjunct area described 483
answer area
conditions when valid 534
DELETE request 559
DELETE_ENTRYLIST request 564
DELETE_MULT request 561
DEQ_EVENTQ request 582
LOCK request 569
MONITOR_LIST request 573
MONITOR_SUBLIST request 577
MONITOR_SUBLISTS request 579
MOVE request 555
READ request 539
READ_EMCONTROLS request 580
READ_EQCONTROLS request 581
READ_LCONTROLS request 565
READ_LIST request 546
READ_MULT request 549
WRITE request 534
WRITE_LCONTROLS request 568
asynchronous IXLLIST operation
IXLFCOMP macro 516
MODE parameter 515
buffer
design consideration 527
selection 524
storage key 530
comparative lock value 517
connection ID described 520
connection name described 520
connection token described 520
contention described 518
data element described 483
data entry described 483
DELETE request 557
DELETE_ENTRYLIST request 557, 562
list services (IXLLIST) *(continued)*
DELETE_MULT request 557, 560
DEQ_EVENTQ request 582
entry ID described 489
entry key described 489
entry name described 489
event monitor controls
diagram 484
exit
complete exit 514, 583
list transition exit 514, 588
notify exit 514, 585
full list structure 590
KEYREQTYPE 491
list controls
described 511
list limit described 511
reading 511, 565
writing 511, 567
list cursor
controlling 496
described 494
initialization 494
set to zero 503
list cursor described 494
list entry
creating 533
deleting 557
described 483
moving 550
reading 537, 541, 547
referencing 489
updating 533
writing 531
list entry controls 511
list entry controls described 483
list entry key 491
list entry version
changing 524
using 523
list entry version number described 490
list header described 483
list monitoring
IXLVECTR macro 570
list monitoring information for a specific list 565
list notification vector 570
list transition exit 570, 588
list structure
concepts 482
parts 483
lock
held by system 517
LOCK request 568
lock table described 516
LOCKOPER parameter 568
multiple requests 521
ownership information 520
persistent 522
protocols 519
reconnection with persistent lock 522
recovery 521
recovery of persistent lock 522
LOCK request 568
lock table 484

list services (IXLLIST) *(continued)*
MONITOR_LIST request 570
MONITOR_SUBLIST request 576
MONITOR_SUBLISTS request 576
MOVE request
types 550
with create 554
with read 554
with write 554
without a data operation 553
notify exit 521
pageable storage
user or system provided 530
READ request 537, 538
READ_EMCONTROLS request 580
READ_EQCONTROLS request 581
READ_LCONTROLS request 565
READ_LIST request 537, 541
READ_MULT request 537, 547
serialized list structure
asynchronous lock request 521
conditional lock request 518
contention described 518
described 516
diagram 483
example of use 517
lock states 517
lock stealing 521
LOCKCOMP parameter 517
LOCKDATA parameter 520, 521
locking functions 517
locking protocols 519, 520
notify exit 519, 521
recovery of lock 521
REQDATA parameter 520, 521
unconditional lock request 518
storage key for buffers 530
summary of functions 486
synchronous IXLLIST operation
MODE parameter 515
WRITE request 531
WRITE_LCONTROLS request 567
list services (IXLLSTE, IXLLSTM, IXLLSTC) 595
local cache vector 368
lock cleanup and recovery service (IXLRT) 672
lock services (IXLLOCK)
ALTER request
completion 651
return and reason codes 651
connection identifier 649
contention
definition 628
handling 628
specifying user-defined protocols 629
description 625
exit routine 654
exit routine coding
complete exit 657
contention exit 659
general requirements 655
notify exit 668
hashing algorithm
analyzing 640
defining 639
lock services (IXLLOCK) *(continued)*
OBTAIN request
completion 650
return and reason codes 650
PROCESSMULT request
return and reason codes 654
record data entry
current number available 641
maximum number available 641
recovery planning 643
RELEASE request
completion 652
return and reason codes 652
resource definition 625
resource request
definition 625
resource request queue
composite state 626
definition 626
using IXLRT 672
using IXLSYNCH 670
lock structure
concepts 634
lock table
description 635
parts 634
record data entry
description 641
using 641
resource name length attribute 239

M

message user routine
coding 52, 59
purpose 16
Monitoring a List
Understanding List Monitoring Notification Delay 619
Monitoring a List by Keyrange Values
Understanding List Monitoring Notification Delay 621

N

navigation
keyboard 771

P

permanent status recording
definition 12
obtaining 22
persistence of a structure 208
PROCESS_REFLIST request
answer area information returned 459
buffer method selection 458
identifying data items to reference 458
storage class specification 459
summary 459

R

READ_COCLASS request
answer area information returned 467

READ_COCLASS request *(continued)*
buffer method selection 466
cast-out class specification 466
data item specification 466
restarting 467
returned information format 466
summary 467
READ_COSTATS request
answer area information returned 471
buffer method selection 469
cast-out class specification 468
restarting 470
returned statistics format 469
summary 471
READ_DATA request
answer area information returned 420
buffer method selection 419
data item name specification 418
data item specification 418
registering interest in a data item 418
specifying data to be read 419
specifying no data to be read 420
storage class assignment 419
summary 420, 421
READ_DIRINFO request
buffer method selection 463
identifying entries to read 462
restarting 464
returned information format 463
summary 464
READ_STGSTATS request
answer area information returned 474
statistics returned 472
storage class specification 472
summary 474
Rebuild Event Timeline 298
reclaim processing with IXLCACHE macro 452
RESET_REFBIT request
answer area information returned 461
identifying data items to unreference 460
restarting 460
summary 461
resource name length attribute
of lock structure 239

S

sending to IBM
reader comments xxvii
server exit routine
coding 167
for client/server processing 167
SET_RECLVCTR request
answer area information returned 457
reclaim vector activation 456
reclaim vector deactivation 456
reclaim vector specification 454
storage class specification 454
summary 457
shortcut keys 771
signaling services
illustration of sending and receiving message 27
receiving a message using the IXCMSGIX macro 26, 43
sending a message using the IXCMSGOX macro 26
status user routine
coding 70, 77
purpose 16
using 63
structure
monitoring repopulation 229
persistence 208
summary of changes xxix, xxx
synchronous update service (IXLSYNCH) 670
sysplex
definition 7
obtaining information
capacity planning 111
tuning 111
sysplex services for data sharing
benefits of use 197
concepts and terminology 198
coupling facility structure
types 199
programming considerations 200
programming features 199
system-managed asynchronous duplexing 227
System-Managed Duplexing Rebuild Timeline 323
System-Managed Rebuild Timeline 323

T

trademarks 776

U

UNLOCK_CASTOUT request
affect on directory entry 437
answer area information returned 437
buffer method selection 436
cast-out lock specification 435
initializing elements in cast-out list 435
process identifier 436
restarting 436
summary 438
UNLOCK_CO_NAME request
affect on directory entry 441
answer area information returned 441
buffer method selection 440
cast-out lock specification 440
initializing a name element 440
process identifier 440
summary 442
user interface
ISPF 771
TSO/E 771
user state field
changing value 25
definition 15
initializing 23
Using the Automatic Restart Manager 194

W

working with structures 285, 314
WRITE_DATA request
answer area information returned 410
buffer method selection 409

WRITE_DATA request *(continued)*
cast-out class 407
cast-out lock 406
changed state specification 405
data item name specification 405
parity specification 407
process identifier 406
registering interest in a data item 403, 429
specifying data to be written 409
storage class assignment 408
summary 410, 411
unchanged state specification 405
user-defined data 408
vector entry assignment 397, 405, 418

X

XCF (cross-system coupling facility)
active member state 12
capacity planning
obtaining information 111
communicating between members 10
concepts 7
created member state 12
defining a member to XCF
through IXCCREAT macro 22
through IXCJOIN macro 22
disassociating a member from XCF
through IXCDELET macro 117
through IXCLEAVE macro 117
through IXCQUIES macro 117
through IXCTERM macro 117
failed member state 13
group
definition 7
maximum number 24
name 15
obtaining information 98
group services
definition 9
group user routine
coding 85, 94
events that cause XCF to schedule 77
purpose 17
skipping of events 81
information
group 98
member 98
obtaining 97
obtaining through IXCMG macro 111
obtaining through IXCQUERY macro 98
sysplex 98
IXCCREAT macro
using 22, 24, 25
IXCDELET macro
using 117
IXCJOIN macro
using 22, 24, 25
IXCLEAVE macro
using 117
IXCMG macro
using 111
IXCMOD macro
using 69
XCF (cross-system coupling facility) *(continued)*
IXCQUERY macro
using 98
IXCQUIES macro
using 117
IXCSETUS macro
example 25
using 25
using for members on different systems 26
IXCTERM macro
using 117
IXCYAMDA mapping macro
information mapped 111
IXCYGEPL mapping macro
information mapped 88
IXCYQUAA mapping macro
information mapped 104
IXCYSEPL mapping macro
information mapped 73
macro
address space restrictions 18
summary of XCF 18
table summarizing for XCF 20
maximum number of groups and members 24
member
association 17
attributes 11
defining to XCF 22
definition 8
disassociating from XCF 116
maximum number 24
name 15
obtaining information 98
state 12
token 15
member association 17, 23
member data field 23
member state
active 12
created 12
failed 13
illustration 14
not-defined 14
quiesced 13
member termination
address space 119
system 120
task 118
member token 15
message response collection
specifying 23
message user routine
purpose 16
multisystem application
definition 7
design considerations 8
example of designing and implementing 120
multisystem environment
definition 7
not-defined member state 14
notifying members of changes 77
permanent status recording
definition 12
obtaining 22

XCF (cross-system coupling facility) *(continued)*
quiesced member state 13
services
categories 8
group 9
signaling 10
status monitoring 11
signaling services
definition 10
status field
updating 69
status monitoring services
definition 11
events other than normal processing 68
illustration 65
normal processing 64
requesting 63
summary of important concepts 68
using a status user routine 63
status user routine
coding 70, 77
purpose 16
using 63
status-checking interval
changing 69
changing through IXCMOD macro 69
setting 69
sysplex
definition 7
obtaining information 98
system cleanup
specifying 23
tuning
obtaining information 111
user routine
group 17
identifying on IXCJOIN macro 23
message 16
message user 17
status 16
user state field
changing through IXCSETUS macro 25
changing value 25
definition 15
initializing 23
XCF client/server services
communication
requesting 133



Product Number: 5650-ZOS

SA23-1400-50

