

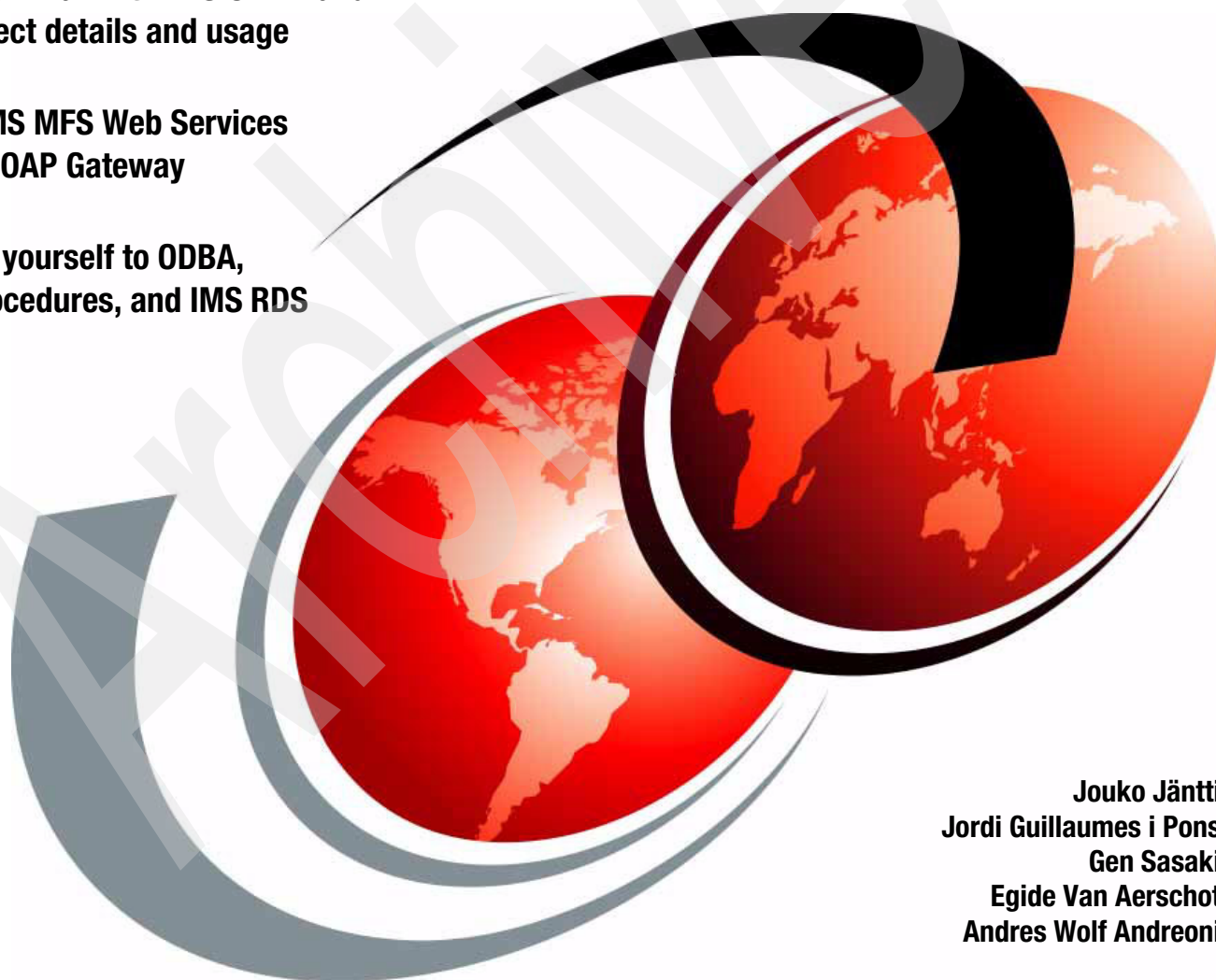
IMS Connectivity in an On Demand Environment:

A Practical Guide to IMS Connectivity

Become familiar with IMS OTMA and
IMS Connect details and usage

Explore IMS MFS Web Services
and IMS SOAP Gateway

Introduce yourself to ODBA,
stored procedures, and IMS RDS



Jouko Jäntti
Jordi Guillaumes i Pons
Gen Sasaki
Egide Van Aerschot
Andres Wolf Andreoni

Redbooks



International Technical Support Organization

**IMS Connectivity in an On Demand Environment: A
Practical Guide to IMS Connectivity**

February 2006

Archived

Note: Before using this information and the product it supports, read the information in “Notices” on page xi.

First Edition (February 2006)

This edition applies to IBM IMS Version 9 (program number 5655-J38) or later for use with the IBM z/OS operating system.

© Copyright International Business Machines Corporation 2006. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	xi
Trademarks	xii
Preface	xiii
The team that wrote this redbook.	xiii
Become a published author	xiv
Comments welcome.	xv
Chapter 1. IMS connectivity in an on demand environment	1
1.1 Addressing the components of the on demand strategy.	2
1.2 IMS in the On Demand Operating Environment	2
1.3 Solutions for IMS connectivity	2
1.4 The organization of this book	4
Chapter 2. Open Transaction Manager Access	7
2.1 OTMA client	8
2.2 OTMA message structure	9
2.3 Commit processing message flows.	12
2.3.1 Commit-then-send (commit mode 0) flow.	12
2.3.2 Send-then-commit message (commit mode 1) flows	13
2.3.3 IMS commit mode 1 message processing	16
2.4 Implementing OTMA	19
2.5 OTMA security issues	21
2.6 Super member support for IMS Connect	23
2.6.1 Super member feature availability.	24
2.6.2 Defining the super member feature.	24
2.6.3 Using the super member feature.	25
2.7 OTMA callable interface	26
2.7.1 OTMA C/I initialization.	27
2.7.2 OTMA C/I security.	27
2.7.3 OTMA C/I restrictions	28
2.7.4 Compiling and binding requirements for OTMA C/I	28
2.7.5 Call functions implemented by OTMA C/I	28
2.8 DSNAIMS stored procedure for OTMA C/I access	29
2.9 WebSphere MQ as an OTMA client	30
Chapter 3. IMS Connect overview	33
3.1 Introduction to IMS Connect	34
3.2 IMS Connect architecture	34
3.3 A brief history and evolution of IMS Connect	37
3.3.1 ITOC: The predecessor to IMS Connect.	37
3.3.2 IMS Connect Version 1.1	38
3.3.3 IMS Connect Version 1.2	38
3.3.4 IMS Connect Version 2.1	39
3.3.5 IMS Version 9 integrated IMS Connect - IMS Connect Version 2.2	39
3.4 IMS Connect clients	41
3.5 IMS Control Center	41
Chapter 4. Configuring IMS Connect.	43

4.1	Introduction	44
4.2	Installing IMS Connect	44
4.3	Configuring IMS Connect	45
4.3.1	IMS Connect start procedure	45
4.3.2	Authorizing IMS Connect and BPE to the APF	46
4.3.3	Updating the program properties table	46
4.3.4	Creating the IMS Connect configuration member	47
4.3.5	Defining IMS Connect security	53
4.3.6	Installing the default user exits into IMS Connect resource library	53
4.4	IMS Control Center support	54
4.4.1	IMS Connect configuration for IMS Control Center support	55
4.4.2	IMS Control Center configuration	55
4.5	Confirming IMS Connect install with the sample Java client	56
	Chapter 5. IMS Connect operations.	63
5.1	IMS Connect REPLY commands	64
5.1.1	CLOSEHWS	64
5.1.2	OPENDS or STARTDS	65
5.1.3	OPENIP or STARTIP	65
5.1.4	OPENPORT or STARTP	66
5.1.5	RECORDER	66
5.1.6	SETRACF	66
5.1.7	SETRRS	67
5.1.8	STOPCLNT	67
5.1.9	STOPDS	67
5.1.10	STOIP	68
5.1.11	STOPPORT	68
5.1.12	VIEWDS	69
5.1.13	VIEWHWS	69
5.1.14	VIEWIP	70
5.1.15	VIEWPORT	71
5.1.16	VIEWUOR	71
5.2	IMS Connect MODIFY commands	72
5.3	IMS Connect BPE commands	73
5.4	IMS command support for IMS Connect and OTMA	74
5.4.1	/DISPLAY OTMA	74
5.4.2	/DISPLAY TMEMBER tmember_name TPIPE tpipes_ID	74
	Chapter 6. Accessing IMS Connect.	77
6.1	IMS Connect in Parallel Sysplex environment	78
6.2	Load Balancer	78
6.3	Virtual IP address (VIPA)	80
6.4	Static VIPA	81
6.5	Dynamic VIPA takeover	81
6.6	Dynamic VIPA takeback	82
6.7	Application-specific dynamic VIPA	83
6.8	Sysplex Distributor	85
6.9	IMS Connect load balancing and failover	87
6.10	Retrieving output messages	88
6.11	The whole picture	90
	Chapter 7. IMS Connect programming model	91
7.1	IMS Connect message structures	92
7.1.1	IMS Request Message (IRM)	92

7.1.2 Request Status Message (RSM)	92
7.1.3 Complete Status Message (CSM)	92
7.1.4 Request Mod Message (RMM)	93
7.2 IMS Connect sample message flows	93
7.2.1 Non-conversational transaction, CM=0, sync level=confirm	93
7.2.2 Non-conversational transaction, CM=1, sync level=none	94
7.2.3 Non-conversational transaction, CM=1, sync level=confirm	94
7.2.4 Conversational transaction, CM=1, sync level=confirm	95
7.2.5 Send-only transaction, CM=0, sync level=confirm	96
7.2.6 The CANCEL TIMER request	97
7.3 Socket connections and settings	98
7.3.1 Persistent sockets	98
7.3.2 Transaction sockets	99
7.3.3 Non-persistent sockets	100
7.4 Asynchronous output support	100
7.4.1 What is asynchronous output?	100
7.4.2 Implementing asynchronous output support	101
7.4.3 SINGLE message control	102
7.4.4 SINGLE WAIT message control	103
7.4.5 NOAUTO message control	103
7.4.6 AUTO message control	104
7.4.7 Purge not deliverable	105
7.4.8 Reroute request	106
Chapter 8. IMS Connect security	109
8.1 General security overview	110
8.2 IMS Connect security	112
8.2.1 Connecting IMS Connect to OTMA	112
8.2.2 User verification	113
8.2.3 User exit security	113
8.2.4 Local option security	113
8.3 OTMA security	113
Chapter 9. IMS Connect user exit support	115
9.1 IMS Connect components and user exits	116
9.2 IMS Connect communication with user exits	116
9.3 User exits supported	118
9.3.1 IMS Connect TCP/IP user message exit (HWSIMSO0 and HWSIMSO1)	119
9.3.2 Sample user message exit (HWSSMPL0 and HWSSMPL1)	119
9.3.3 Difference between HWSIMSO0/SMPL0 and HWSIMSO1/SMPL1	120
9.3.4 IMS Connector for Java user message exit (HWSJAVA0)	121
9.3.5 IMS Connect IMSplex message exits (HWSCSLO0 and HWSCSLO1)	121
9.3.6 Security exit (IMSLSECX)	122
9.3.7 User initialization exit routine (HWSUINIT)	123
9.3.8 Event recording user exit (HWSTECL0)	125
9.4 Message structures between IMS Connect and user exits	127
9.4.1 Input message from client and passed to exit	127
9.4.2 Input message returned from message exit	130
9.4.3 Output message from IMS Connect to IMS Connector for Java client	131
9.4.4 Output message: IMS Connect to non-IMS Connector for Java client	132
9.5 IMS Connect DRU exit for asynchronous output support	135
9.5.1 ALTPCB ISRT message routing flow using OTMA exits	135
9.5.2 How IMS Connect communicates with the DRU exit	137

9.5.3 HWSYDRU0 sample DRU exit	137
9.5.4 Debugging the IMS OTMA exits	138
Chapter 10. IMS Connect diagnostics	141
10.1 IMS Connect recorder trace	142
10.1.1 Enabling IMS Connect recorder trace	142
10.1.2 Starting and stopping the IMS Connect recorder trace	142
10.1.3 Printing out the recorder trace	143
10.1.4 Interpreting the recorder trace printout	143
10.1.5 Example of recorder trace output	145
10.2 IMS Connect traces	148
10.2.1 BPE configuration	148
10.2.2 Formatting incore trace tables	148
10.3 IMS Connect Dump Formatter	149
10.3.1 IMS Connect Dump Formatter activation	149
10.3.2 Accessing the IMS Connect Dump Formatter	150
10.3.3 Using the IMS Connect Dump Formatter	151
Chapter 11. IMS Connect Extensions	155
11.1 Introduction to IMS Connect Extensions	156
11.2 Event collection and reporting	161
11.2.1 Activate event collection	161
11.2.2 Journal management	163
11.2.3 IMS Connect event records	167
11.2.4 Event Collection print utility	170
11.2.5 Recorder trace utility	174
11.2.6 Active session utility	175
11.2.7 IMS Performance Analyzer IMS Connect reports	177
11.2.8 IMS Problem Investigator	184
11.3 Workload management	189
11.3.1 Transaction routing	190
11.3.2 Workload balancing	193
11.3.3 Transaction pacing	194
11.4 Status Monitor	195
11.4.1 System view	196
11.4.2 Port view	197
11.4.3 Form definition	199
11.5 Security	200
11.6 User exits management	201
11.6.1 User exits definition	201
11.6.2 User exits commands	203
11.7 IMS Connect problem determination	205
11.7.1 NODELAYACK issues	205
11.7.2 Incorrect message length	208
11.7.3 Client fails to ACK message	210
11.7.4 Timeout issues	213
11.7.5 Duplicate clients	214
11.8 Highlights of IMS Connect Extensions Version 1 Release 2	216
11.8.1 Status Monitor: Active sessions	217
11.8.2 Programming interface for user applications	218
11.8.3 Primary datastore routing	218
11.8.4 Journal and journal print enhancements	218
11.8.5 Client services exit	219

11.8.6 Enhanced tracing	219
Chapter 12. IMS Connector for Java	221
12.1 J2EE Connector architecture (JCA)	222
12.1.1 System contracts	222
12.1.2 Common Client Interface	223
12.1.3 Resource adapter module	223
12.2 JCA infrastructure and API	224
12.2.1 Connection management	224
12.2.2 Transaction management	227
12.2.3 Other JCA v1.5 items	228
12.2.4 Interaction with EIS	228
12.2.5 Security	230
12.2.6 Summary	231
12.3 Building applications that use IMS Connector for Java	232
12.3.1 Introduction	232
12.3.2 Connection properties	232
12.3.3 Interaction properties	242
12.3.4 Use considerations	247
12.3.5 Summary	255
Chapter 13. IMS Connector for Java rerouting and timeout support	257
13.1 Asynchronous message processing	258
13.2 Messages inserted to ALTPCB	258
13.3 Multiple and timed out IOPCB responses	259
13.3.1 Discarding the non-delivered messages	259
13.3.2 Rerouting the non-delivered messages	261
Chapter 14. Building roll your own clients	265
14.1 Basic structure of a simple IMS Connect client program	266
14.2 IMS Connect message structures	268
14.2.1 The IMS Connect input message	268
14.2.2 The IMS Connect output message	274
14.3 IMS Connect Unicode support	275
14.3.1 Transaction code translation	276
14.3.2 Output message including Unicode data from IMS Connect	277
14.3.3 Message structures for Unicode support	277
14.4 Complete pseudocode samples	279
14.4.1 Commit mode 1 send-receive programming	279
14.4.2 Commit mode 0 send-receive programming	280
14.4.3 Commit mode 0 RESUME TPIPE programming	281
14.5 Detailed code examples	282
14.5.1 C example	283
14.5.2 Java example	291
Chapter 15. IMS Connect client diagnostics	297
15.1 No response from IMS or IMS Connect	298
15.1.1 Hanging clients	298
15.1.2 TCP/IP socket timeouts	299
15.1.3 IMS Connect execution timeouts	300
15.2 Error messages from IMS	302
15.3 Wrong status codes from IMS Connect	302
15.3.1 Duplicate clientID (reason code 56)	303
15.3.2 OTMA protocol error (reason code 36)	305

15.3.3 Other errors	305
15.4 Exceptions in IMS Connector for Java applications	309
15.4.1 Naming (JNDI)-related errors	309
15.4.2 Connection pool-related errors	312
15.5 Diagnosing problems related to sockets	319
15.5.1 IMS Connect and IMS Connector for Java parameters for sockets	319
15.5.2 z/OS UNIX System Services parameters for sockets	320
Chapter 16. IMS MFS Web Services	321
16.1 IMS MFS Web Services introduction	322
16.2 IMS MFS Web Services development process overview	322
16.3 IMS MFS Web Services supported features	323
16.3.1 Supported device types	325
16.3.2 Supported MFS statements	325
16.4 IMS MFS Web Services limitations	325
16.5 Adding operations, messages, and bindings	326
16.6 Creating an enterprise service	326
16.7 Deploying an MFS-based IMS enterprise service	327
Chapter 17. IMS MFS Web Enablement	329
17.1 How does IMS MFS Web Enablement work?	330
17.2 IMS MFS XML Utility	330
17.2.1 Overview of the MFS XML Utility	331
17.2.2 User modes	331
17.2.3 Invoking the MFS XML Utility	332
17.3 IMS MFS Web Enablement runtime support	336
17.3.1 MFS Web Enablement features and functions	337
17.3.2 MFS Servlet	338
17.3.3 MFS Adapter	341
17.4 Installing the instance servlet WAR file	342
17.5 Accessing the deployed instance servlet	343
17.6 Sample MFS style sheets	344
17.7 Instructions to Web-enable IMS Phonebook application	346
17.7.1 Step 1: Parsing the MFS source file with MFS XML Utility	346
17.7.2 Step 2: Generating an instance servlet	347
17.7.3 Step 3: Generating a WAR file	349
17.7.4 Step 4: Configuring WebSphere Application Server	350
17.7.5 Step 5: Deploying the application WAR file	351
17.7.6 Step 6: Invoking the instance servlet	352
17.7.7 Step 7: Invoking the Phonebook application	352
17.7.8 Step 8: Logging out	352
Chapter 18. IMS SOAP Gateway	353
18.1 IMS SOAP Gateway introduction	354
18.2 Making your IMS application a Web service	355
18.2.1 Creating a WSDL file for your IMS application	355
18.2.2 Deploying WSDL and configuring properties with IMS SOAP Gateway	357
18.2.3 Writing the client application	357
Chapter 19. Open Database Access	359
19.1 Accessing IMS databases through the ODBA	360
19.2 The database resource adapter (DRA)	360
19.3 Setting up the DRA and the ODBA interface	361
19.3.1 Creating the ODBA DRA startup table	361

19.3.2	Loading and running ODBA in the z/OS application region	363
19.3.3	Linking application programs	363
19.3.4	Establishing and defining security	363
19.4	Writing ODBA application programs	366
19.4.1	General application program flow	366
19.4.2	Making calls to IMS	367
19.4.3	The application interface block (AIB)	368
19.4.4	DL/I calls in the ODBA application	369
19.4.5	Server program structure and the unit of recovery	372
19.5	Considerations for using ODBA	373
19.5.1	Restrictions	373
19.5.2	Multiple access to IMS subsystems	373
19.5.3	IMS Fast Path resource usage	374
19.5.4	The commit scope change and IMS resource occupancy	374
19.5.5	RRS logging performance	375
19.6	Problem determination	375
19.6.1	Finding the problem	375
19.6.2	IMS initialization errors	375
19.6.3	Running errors	376
19.6.4	The application interface block	376
19.7	IBM-supplied ODBA infrastructures	376
19.7.1	DB2 stored procedure	376
19.7.2	WebSphere Application Server for z/OS and IMS Remote Data Access	377
19.7.3	WebSphere Information Integrator Classic Federation for z/OS	378
19.8	Summary of IBM-supplied ODBA infrastructures	380
Chapter 20.	ODBA from DB2 stored procedures	383
20.1	A short introduction to DB2 stored procedures	384
20.2	DB2 stored procedures' use of ODBA	384
20.3	Sample ODBA using DB2 stored procedures	386
20.3.1	Provided sample jobs	386
20.3.2	Provided sample source codes	388
20.4	Step-by-step instructions for using the sample	388
20.4.1	Step 1: Creating an IMS DRA startup table	389
20.4.2	Step 2: Setting up the DB2 stored procedure address space for ODBA	390
20.4.3	Step 3: Creating the WLM application environment	391
20.4.4	Step 4: Building the stored procedure by DSNTJEJ61	392
20.4.5	Step 5: Defining the IMS environment	393
20.4.6	Step 6: Running the stored procedure by DSNTJEJ62	393
20.4.7	Step 7: Analyzing the output	394
20.5	Commands for ODBA DB2 stored procedure environment	396
20.5.1	IMS commands	396
20.5.2	DB2 commands	397
20.5.3	z/OS Workload Manager commands	398
20.5.4	RRS panel utility	400
20.6	Sample Java client application for ODBA stored procedure	401
Chapter 21.	IMS Remote Database Services	405
21.1	The big picture of the IMS Java environment	406
21.1.1	IMS dependent regions	406
21.1.2	IBM products on the z/OS environment	406
21.2	IMS JDBC interface	407
21.2.1	The layered set of IMS Java class libraries	407

21.2.2	The basic concepts of relational access to hierarchical databases	408
21.2.3	Comparison of DL/I access and IMS JDBC SQL access	409
21.2.4	Supported SQL keywords	410
21.2.5	IMS Java SQL usage	411
21.3	DLIModel utility	420
21.3.1	Example of using the DLIModel utility	422
21.3.2	DLIModel utility plug-in	425
21.3.3	Example of using the DLIModel utility plug-in	426
21.4	Remote Database Services	428
21.4.1	Remote Database Services components	430
21.4.2	Client/server interaction	432
21.4.3	Security	433
21.5	Sample IMS RDS access	434
21.5.1	Step 1: Creating the IMS DRA startup table	435
21.5.2	Step 2: Setting up WebSphere Application Server for z/OS subsystem	435
21.5.3	Step 3: Installing the metadata class for the sample application	446
21.5.4	Step 4: Setting up application server for distributed platforms environment	448
21.5.5	Step 5: Developing the sample application	451
21.5.6	Step 6: Defining the IMS environment	457
21.5.7	Step 7: Running a Web application	458
21.5.8	Problem determination for Remote Database Services	461
21.5.9	Summary of the IMS RDS implementation	465
Appendix A. Sample code: Non-IMS Connector for Java client code		469
	C sample source code	470
	Java sample source code	484
Appendix B. IMS RDS application example		499
	ImsRdsSampleGlobal.java	500
	ImsJavaRdsSample.java	502
	GlobalInput.html	504
	LocalInput.html	505
	Output.jsp	505
Appendix C. Additional material		507
	Locating the Web material	508
	Using the Web material	508
Abbreviations and acronyms		509
Related publications		511
	IBM Redbooks	511
	Other publications	511
	Online resources	512
	How to get IBM Redbooks	513
	Help from IBM	513
Index		515

Notices

This information was developed for products and services offered in the U.S.A.

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 give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

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.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites 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.

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.

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 illustrates 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. 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 IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX 5L™
AIX®
C/MVS™
CICS®
DB2 Connect™
DB2 Universal Database™
DB2®
DFS™
DRDA®
ESCON®

@server®
IBM®
IMS™
Language Environment®
MVS™
OS/2®
Parallel Sysplex®
PowerPC®
POWER™
RACF®

Rational®
Redbooks (logo) ™
Redbooks™
Tivoli®
VisualAge®
VTAM®
WebSphere®
xSeries®
z/OS®
zSeries®

The following terms are trademarks of other companies:

Enterprise JavaBeans, EJB, Java, Java Naming and Directory Interface, JavaBeans, JavaScript, JDBC, JDK, JSP, JVM, J2EE, J2SE, RSM, Solaris, Sun, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

IBM® Information Management System (IMS™) is the IBM premier transaction and hierarchical database management system. Connectivity has always been a priority with IMS. IMS exploits the latest technologies to address customers' requirements for accessing IMS transactions and data. This IBM Redbook is about IMS connectivity.

This book provides a general overview of the IMS Open Transaction Manager Access (OTMA) function and extensive information about IMS Connect and its usage, including a chapter that describes the IMS Connect Extensions product and how you can enhance the IMS Connect operating environment with it.

This book provides a broad understanding of IMS Connector for Java™. We cover some special considerations, such as using the conversational transactions, rerouting, and timeout support, as well as programming roll-your-own clients without using IMS Connector for Java.

We also introduce Open Database Access and provide examples of using it with stored procedures and with IMS Remote Database Services. As for future directions, we also include a chapter about the IMS SOAP Gateway. This book updates and adds to the information in the previous IBM Redbook *IMS e-business Connectors: A Guide to IMS Connectivity*, SG24-6514.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.

Jouko Jäntti is a Senior IT Specialist at IBM Global Services in Finland and also works for the Silicon Valley Laboratory as a member of IMS Worldwide Advocate Team. During the years of 2001 to 2003, he was a Project Leader specializing in IMS with the IBM International Technical Support Organization, San Jose Center. He is the lead author of the IMS-related redbooks listed in "IBM Redbooks" on page 511.

Jordi Guillaumes i Pons works for "la Caixa," a Spanish savings bank, as an IMS systems programmer. He has a bachelor's degree in systems engineering, and 18 years of experience in the IT field, having spent 10 of those developing IMS banking applications and the last three as an IMS systems programmer. His areas of expertise include Java and J2EE™ development, and recently he has been involved in the transition from APPC-based connectivity to IMS Connect and IMS Connector for Java taking place in his company.

Gen Sasaki is an I/T specialist for IBM Japan Systems Engineering Co., Ltd., in Chiba, Japan. He holds a master's degree in Engineering (Mechanics) from the University of Chiba. He provides technical support for IMS and IMS data management tools, with six years of experience. His area of expertise includes IMS Web connectivity.

Egide Van Aerschot has been working for the IBM Program Support Center in Montpellier, France since 1998 and is a member of the New Technology Center team, supporting and providing education for J2EE projects, IBM WebSphere® Business Integration, and connections to established systems. Before his current position, he worked for IBM Belgium as an Account Systems Engineer, responsible for many projects related to transactional processing with major Belgium customers. During this period, he also participated in several residencies in the United States. He graduated from the University of Louvain as a civil engineer.

Andres Wolf Andreoni is an I/T specialist at IBM Global Services in Spain. He has six years of experience in IMS. He holds a degree in Physics from the Universidad Autonoma de Barcelona. He has been assigned full time to “la Caixa,” one of the largest financial entities in Spain. His areas of expertise and responsibilities include IMS installation and maintenance, IMS system management, IMS problem determination, IMS Tools, and IMS related products.

Thanks to the following people for their contributions to this project:

Emma Jacobs
Deanna Polm
International Technical Support Organization, San Jose Center

Rich Conway
Bob Haimowitz
International Technical Support Organization, Poughkeepsie Center

Rose Levin
Ken Blackman
Kyle Charlet
Tram Dinh
Kevin Flanigan
Bill Huynh
Barbara Klein
LeiLei Li
Steve Nathan
James Polo
Judith Smith
Kevin Wang
Jack Yuan
IBM Silicon Valley Laboratory, San Jose, U.S.

Jim Martin
David Mierowski
Edward Breese
Rafael Avigad
Fundi Software

Very special thanks to Suzie Wender for contributing to the writing of Chapter 6, “Accessing IMS Connect” on page 77, as well as to Jenny Hung for providing Chapter 16, “IMS MFS Web Services” on page 321 and Chapter 17, “IMS MFS Web Enablement” on page 329, and to Haley Fung for providing Chapter 18, “IMS SOAP Gateway” on page 353, in addition to their overall support for this redbook project.

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks™ to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

ibm.com/redbooks

- Send your comments in an e-mail to:

redbook@us.ibm.com

- Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. QXXE Building 80-E2
650 Harry Road
San Jose, California 95120-6099

Archived

IMS connectivity in an on demand environment

The On Demand Operating Environment is one that can seamlessly integrate the computing infrastructure—hardware, software, and the related integration services—with vital business applications. It is one that is:

- ▶ **Integrated**

Business applications can interoperate end-to-end across your enterprise. This means integration with other products within IBM and with other products and platforms within the industry.

- ▶ **Open**

You have the flexibility to run the applications and middleware your business needs. This means open standards for portability and ease of development, helping customers programmers and their programs interact smoothly.

- ▶ **Virtualized**

You can improve utilization rates to create cost-efficiencies and maximize your IT investment. This means virtualization for flexibility to grow and expand and the ability to exploit new resources as they become available.

- ▶ **Autonomic computing**

Systems can heal and manage themselves, and you can focus on your business, not your technology infrastructure. This is addressed by autonomic computing, offering ease of use, elimination or reduction in outages, and reducing the education curve for new people.

A new enterprise infrastructure is needed that integrates existing systems and delivers their long-promised business benefits. IMS is addressing all the four components of the on demand strategy. IBM provides support for a variety of connectivity and integration solutions with IMS. This is the area on which we concentrate in this book.

1.1 Addressing the components of the on demand strategy

Connectivity and integration has always been a priority with IMS. IMS has provided solutions that can use workstations or servers to access IMS data. Information can be retrieved from the server system in a two-tier environment or in a three-tier environment. Our strategy here is to support standard connectivity solutions as well as those tailored to the IMS environment.

IMS tooling shipped with IBM WebSphere can provide connectivity to IMS applications and data in addition to other environments, regardless of the tools used or with what they want to connect. For our customers, it is important to maintain their core business applications and provide integration to the new ones. IMS users should remain confident about its viability for new developments, using Java as a way of addressing skills shortages, and complementing IMS with WebSphere.

IMS views integration as a continuing journey and continues to support and enhance new technology for connectivity and e-business enablement into the foreseeable future.

1.2 IMS in the On Demand Operating Environment

The IBM Information Management System (IMS) is the IBM premier transaction and hierarchical database management system, the product of choice for critical online operational applications and data where support for high availability, performance, capacity, integrity, and low cost are key factors. Today, IMS manages the world's mission-critical data and continues as a major player in the on demand world. IMS customers' million instructions per second (MIPS) has been growing rapidly to more than 2.6 million MIPS worldwide, and customer migration to the latest IMS versions has also been growing rapidly, with greater numbers getting into production faster than previous versions. As we move further into the new era of on demand computing, IMS is still helping to lead the way. More than 35 years since the first IMS-ready message for the Apollo space program, IMS along with IBM @server® zSeries® are breaking technology barriers and continue to help lead the industry, even though it is sometimes taken for granted.

IMS is continuing to provide solutions to exploit the latest technologies to address customers' requirements. In exploiting new technologies and balancing priorities to address the increasing demands and sophistication of their customers, IMS customers are at the leading edge. Their customers have been making the highest demands for performance and availability, along with interoperability, flexibility, and support for new, emerging technologies. And IMS has been continuing to provide solutions to address these needs. To extend customers' existing assets in modern on demand architectures, IMS is focusing on enterprise modernization through integration and open access with an on demand service-oriented architecture.

1.3 Solutions for IMS connectivity

IMS provides a variety of solutions for providing access to IMS applications and data as Web services. Figure 1-1 on page 3 shows examples of IMS connectivity solutions for Web services.

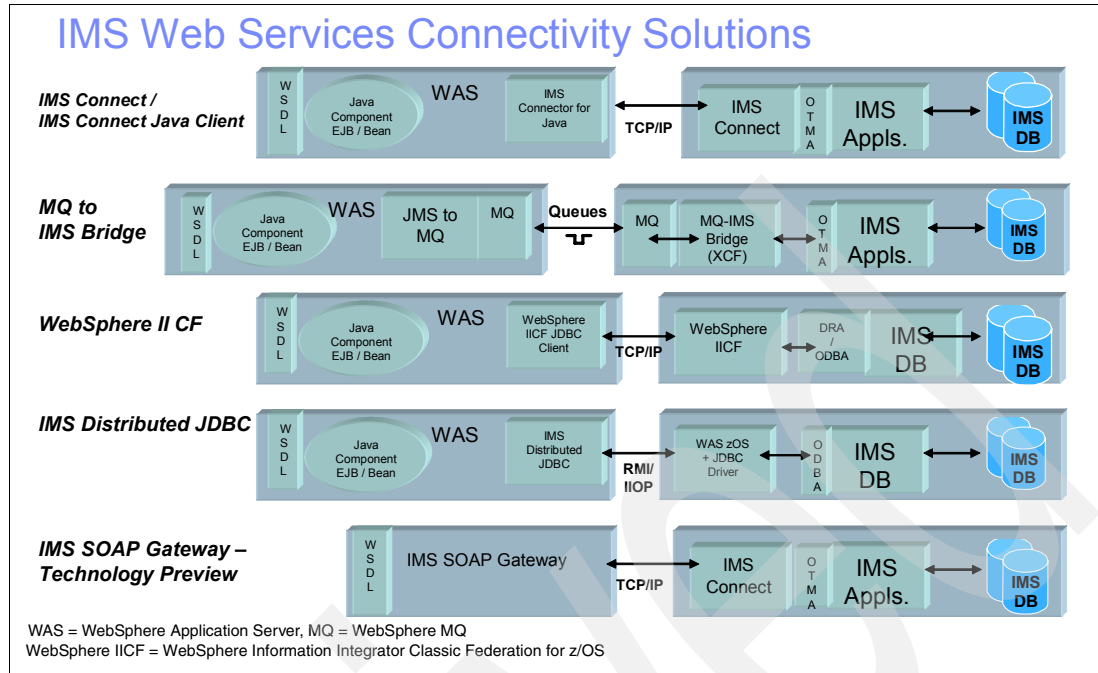


Figure 1-1 IMS connectivity solutions

When customers need rapid response for business transactions and inquiries from many locations using numerous different types of devices, the IMS Connect function provides easy-to-install, easy-to-use, high-performance transparent access to IMS applications and data from practically any application environment, including Linux®. Customers can use their storehouse of IMS applications from IMS Connect client applications to access their IMS and IBM DB2® data from the Internet. When customers need high-speed, transparent universal interchange of information throughout the enterprise, and with partners and customers, IMS provides the ability to store and retrieve XML data natively. IMS also converts non-XML data to XML for interchange and converts it back or stores it natively.

When customers need to deploy new Web-based applications, at the lowest possible cost, to maintain competitive advantage, IMS has state-of-the-art development tools available today to Web-enable and thereby protect customers' long-standing investment in IMS applications, data, and skills. Additionally, IMS customers can now build new applications using Java programming, with XML for universal data interchange, write and test them on workstations and integrate them with existing applications on the host. When customers need to develop a service-oriented architecture to better integrate business processes end-to-end, their IMS applications and IMS transactions can be published on the Internet as Web services, connecting to IMS through Simple Object Access Protocol (SOAP) and Enterprise JavaBeans™ (EJB™) bindings, within customers' service-oriented architecture (SOA), thereby assisting business-to-business application integration. IMS Version 9 is the IMS offering for the On Demand Operating Environment.

The integrated IMS Connect function in IMS Version 9 provides easy-to-install, easy to use, high-performance, high-volume, and secure transparent access to IMS applications and operations from any TCP/IP-supported environment, including Linux. It provides commands to manage the network environment and assist with workload balancing, resulting in better resource utilization. It reduces the design and coding effort for client applications and provides easier access to IMS applications and operations, thereby improving programmer productivity. It can be used with IBM WebSphere and Rational® development tools to quickly transform static Web sites into sources of dynamic Web content, improving marketing

effectiveness and customer service, and to transform IMS transactions into Web services for service-oriented architectures (SOAs), enabling quick response to new customer requirements, business opportunities, and competitive threats. It can be used with DB2 and the IMS Control Center to control both IMS and DB2 operations, improving system availability and operator productivity.

The integrated IMS Connect function of IMS Version 9 can be used to replace the separately priced IMS Connect product offered for earlier IMS versions, simplifying administration and reducing cost. IMS Connect provides one-to-any and any-to-one connectivity. Performance measurements have demonstrated more than 6000 transactions per second with a single IMS Connect instance to a single IMS. This can be greatly increased using parallel IMS Connect instances. Performance measurements have demonstrated more than 22,000 transactions per second (2 billion per day) with IMS on a zSeries Model 990 processor.

1.4 The organization of this book

This book concentrates on the connectivity functions of IMS. In the first part, we provide a general overview of the IMS Open Transaction Manager Access (OTMA) function and extensive information about IMS Connect and its usage. We also include one chapter that describes the IMS Connect Extensions product and how you can enhance the IMS Connect operating environment with that product. The topics in this part include:

- ▶ Chapter 2, “Open Transaction Manager Access” on page 7
- ▶ Chapter 3, “IMS Connect overview” on page 33
- ▶ Chapter 4, “Configuring IMS Connect” on page 43
- ▶ Chapter 5, “IMS Connect operations” on page 63
- ▶ Chapter 6, “Accessing IMS Connect” on page 77
- ▶ Chapter 7, “IMS Connect programming model” on page 91
- ▶ Chapter 8, “IMS Connect security” on page 109
- ▶ Chapter 9, “IMS Connect user exit support” on page 115
- ▶ Chapter 10, “IMS Connect diagnostics” on page 141
- ▶ Chapter 11, “IMS Connect Extensions” on page 155

In the second part of the book, we introduce the IMS Connector for Java and other methods to access IMS transactions. Some special considerations, such as rerouting and timeout support are also covered in this part, as well as programming the clients without using the IMS Connector for Java. The part contains the following chapters:

- ▶ Chapter 12, “IMS Connector for Java” on page 221
- ▶ Chapter 13, “IMS Connector for Java rerouting and timeout support” on page 257
- ▶ Chapter 14, “Building roll your own clients” on page 265
- ▶ Chapter 15, “IMS Connect client diagnostics” on page 297
- ▶ Chapter 16, “IMS MFS Web Services” on page 321
- ▶ Chapter 17, “IMS MFS Web Enablement” on page 329
- ▶ Chapter 18, “IMS SOAP Gateway” on page 353

Next, we introduce ways to access IMS databases using Open Database Access (ODBA) and provide examples of using it with DB2 stored procedures and with IMS Remote Database Services. We discuss these topics in the following chapters:

- ▶ Chapter 19, “Open Database Access” on page 359
- ▶ Chapter 20, “ODBA from DB2 stored procedures” on page 383
- ▶ Chapter 21, “IMS Remote Database Services” on page 405

At the end of the book, we provide sample code for some of the examples described in this book. We also make some code available at the IBM Redbooks FTP site as an additional material for downloading. For more information, see Appendix C, “Additional material” on page 507 and the following URL:

<ftp://www.redbooks.ibm.com/redbooks/SG246794>

Archived

Archived

Open Transaction Manager Access

Open Transaction Manager Access (OTMA) is a function of IMS that was introduced with IMS Version 5. OTMA is a transaction-based, connectionless client/server protocol that provides an access path and an interface specification for sending and receiving transactions and data from IMS.

OTMA is specifically implemented for IMS in an z/OS® sysplex-capable environment. Therefore, the domain of OTMA is restricted to the domain of the z/OS cross-system coupling facility (XCF). XCF is a component of z/OS that provides functions to support cooperation between authorized programs running within a sysplex. OTMA and z/OS applications compose an XCF group, where OTMA and the applications are group members. In a Parallel Sysplex environment, different members can be on different z/OS images.

OTMA is designed to be a high-performance protocol that allows z/OS programs to access IMS applications. The support of large networks is handled through the use of OTMA clients. Because the OTMA connections use internal cross-system communication that is almost comparable to main memory processing, OTMA has very high performance in general.

2.1 OTMA client

OTMA provides the facility for IMS to communicate very efficiently with z/OS applications other than Virtual Telecommunications Access Method (VTAM®). These z/OS applications are called OTMA clients. These OTMA clients can be an IBM written application, an independent software vendor's application, or user-written code. Figure 2-1 shows examples of the OTMA client applications and the corresponding network client applications.

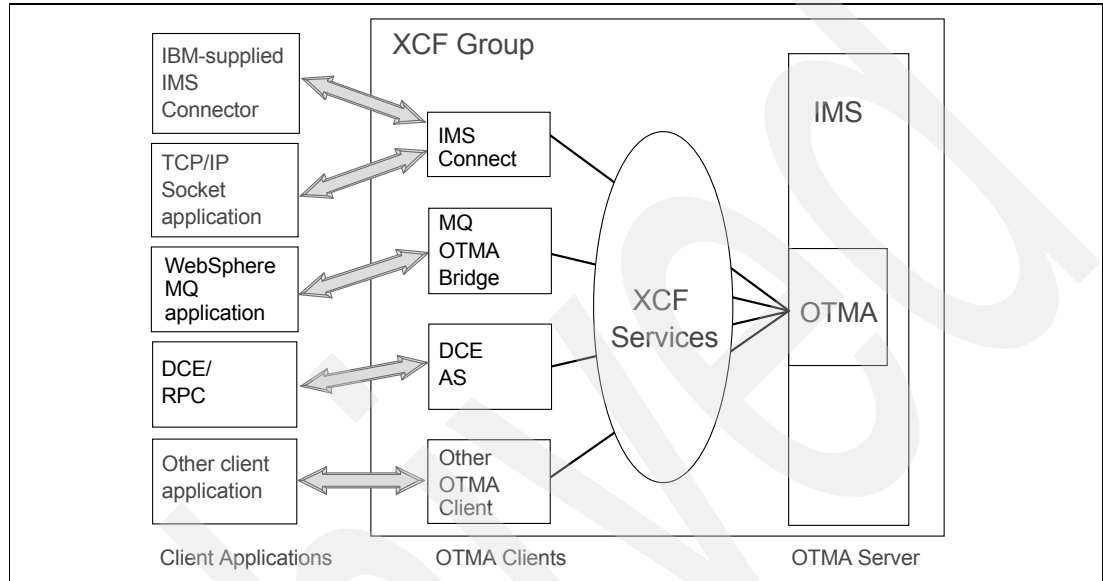


Figure 2-1 Examples of the different OTMA components

IMS Connect is a commonly used OTMA client. The IBM-supplied IMS e-business connectors, such as IMS Connector for Java, are based on the use of IMS Connect. We discuss these connectors in more detail later in this book. The IBM WebSphere MQ product also has the OTMA interface implemented, which we briefly describe in 2.9, “WebSphere MQ as an OTMA client” on page 30, and another example of an OTMA client can be distributed computer environment/remote procedure call (DCE/RPC).

For those who are going to write an OTMA client of their own, there is an application programming interface called the OTMA callable interface (OTMA C/I). It was introduced in IMS Version 6.1. OTMA C/I is an application programming interface (API) that abstracts out the details of OTMA and XCF. Refer to 2.7, “OTMA callable interface” on page 26 for more information about OTMA C/I.

The OTMA interface allows the IMS Transaction Manager to be a server to many different OTMA clients. The OTMA client can be any z/OS program that runs in the same XCF group as IMS and the client acts as an interface between IMS Transaction Manager and the network. By using OTMA, each client can submit transactions or IMS commands to IMS and receive output from IMS applications or from IMS itself. IMS is the server in this configuration, because it can handle many OTMA clients.

The OTMA client handles all device dependencies for a particular network protocol, which can be TCP/IP or Systems Network Architecture (SNA) or any other protocol the client wants to support, for example, CORBA, and IMS Transaction Manager can operate without needing any device-characteristic information. The majority of IMS message processing options, such as non-response mode and response mode transactions, conversational processing, Fast

Path transactions, Multiple Systems Coupling (MSC) processing, and IMS commands, can be implemented with OTMA. OTMA does not support Message Format Service (MFS), although MODname can be provided within the messages. This MODNAME will be placed in the IOPCB MODNAME field so that it can be used by the IMS application.

2.2 OTMA message structure

The OTMA client communicates with IMS by sending and receiving messages. It adds the OTMA message prefix to the input messages and it uses the message prefix to route the output data to the originating device or program. The OTMA message prefix consists of the following sections, shown in Figure 2-2:

- ▶ Message Control Information (MCI), such as the transaction-pipe name, message type, sequence numbers (if any), processing flag, response indicator, and chaining indicator
- ▶ State data, such as map name, synchronization level and commit mode for the transaction, various tokens, and server state
- ▶ Security data, such as the type of the user information, user ID or user token, and various security flags
- ▶ User data, which is any special information needed by the client, or IMS commands to be executed

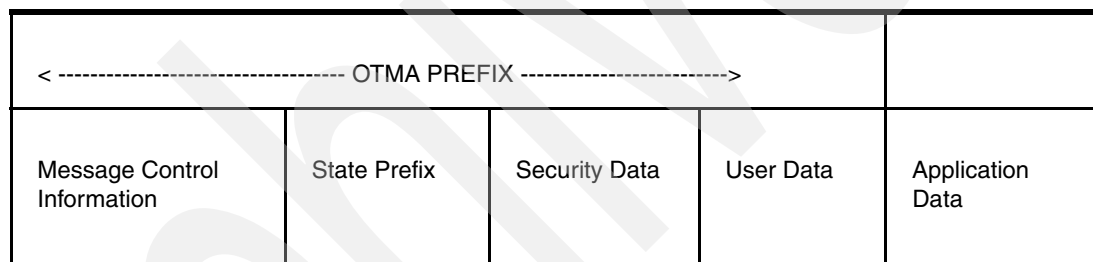


Figure 2-2 OTMA message structure

The message prefix is followed by the application data section of the message, which can be the data sent to the IMS application, or the response sent to the client.

OTMA headers are written into the IMS log record type X'01' for OTMA input messages and into the IMS log record X'03' for responses to the OTMA destination. OTMA messages can be mapped by the DFSYMSG DSECT in IMS.SDFSMAAC. For a detailed description of the message structure, see Chapter 5 *IMS Version 9: Open Transaction Manager Access Guide and Reference*, SC18-7829.

The key parameters that affect the nature of the transaction processing and the message flow are the commit mode, synchronization level, and processing flag. Not all of the combinations of these parameter values are valid. For example, the synchronization level is ignored for the commit-then-send commit mode (sync level *confirm* is always used), and synchronized Tpipes can only be used with commit-then-send mode. The send-then-commit mode must be used for IMS conversational and Fast Path expedited message handling (EMH) transactions.

Here is a brief explanation of these key parameter values:

- ▶ Commit mode (synchronization flag in state data portion of the OTMA headers), as shown in Table 2-1 on page 10

Table 2-1 Values for the synchronization flag in the OTMA header

Content	Value	Meaning
Commit-then-send (commit mode 0)	X'40'	The server commits output before sending. Traditional IMS flow.
Send-then-commit (commit mode 1)	X'20'	The server sends output before committing it, similar to Advanced Program-to-Program Communication (APPC) synchronous flow.

- Synchronization level (in state data portion of the OTMA headers), as shown in Table 2-2

Table 2-2 Values for the synchronization level in the OTMA header

Content	Value	Meaning
None	X'00'	The server application does not request an ACK message when sending output to a client.
Confirm	X'01'	The server sends transaction output with the response requested flag set.
SYNCPT	X'02'	This message is part of a protected conversation under the Resource Recovery Services (RRS) recovery platform, and the resources updated under this conversation uses the two-phase commit protocol.

- Processing flag (in message-control information portion of the OTMA headers), as shown in Table 2-3

Table 2-3 Values for the processing flag in the OTMA header

Content	Value	Meaning
Synchronize Tpipe	X'40'	Input and output sequence numbers are maintained for the transaction pipe.
Asynchronous output	X'20'	The server is sending unsolicited queued data messages.
Error Message Follows	X'10'	An error data message set by the server when sending a NAK.

The logical connection between the OTMA client and IMS is called the *transaction pipe*, or *Tpipe*. The concept of the Tpipe corresponds to the concept of an IMS logical terminal (LTERM) and a physical node connection in an SNA network. Tpipes do not have to be predefined, and the client can create and use as many Tpipes as it needs.

Different OTMA clients can use the same Tpipe name because the structure that is created in IMS to keep messages relies on two names: the client name and the Tpipe name. The client name is the XCF member name of the client that the client uses when it joins the XCF group, and it is provided directly to IMS by the XCF interface.

IMS supports a full-duplex message flow for OTMA messages. If the half-duplex message flow is desired, it should be implemented and maintained by the client.

In the full-duplex message flow, the client associates its transactions with a Tpipe name, and IMS uses the Tpipe name to associate all input and output with a particular client. The transactions and output messages are processed in parallel, and the client can maximize the parallelism by creating its own process for every transaction and output message.

The general message flow from OTMA client, shown in Figure 2-3, to IMS server is as follows:

1. The OTMA client receives the message from the network and submits an IMS transaction or an IMS command through XCF. The IMS transaction code or command is specified in the application data section of the input message. A message prefix is always attached to the input message. It contains the Tpipe name, the commit mode, client and server tokens, and many other flags.
2. The input message is enqueued on the scheduler message block (SMB) for a transaction or is processed by IMS for a command.
3. The LTERM field from the IOPCB contains the Tpipe name specified in the MCI section of the input message, or the LTERM field can be overridden by the destination override specified in the state data section of the input message.
4. The client can also provide a MODNAME in the map name specified in the state data section of the input message. It will be placed in the IOPCB. Similarly, this prefix field will be updated in a reply if the IMS application specifies a new MODNAME in the insert (ISRT) call.
5. The IMS application cannot see the OTMA prefix when it issues the get unique (GU) call.

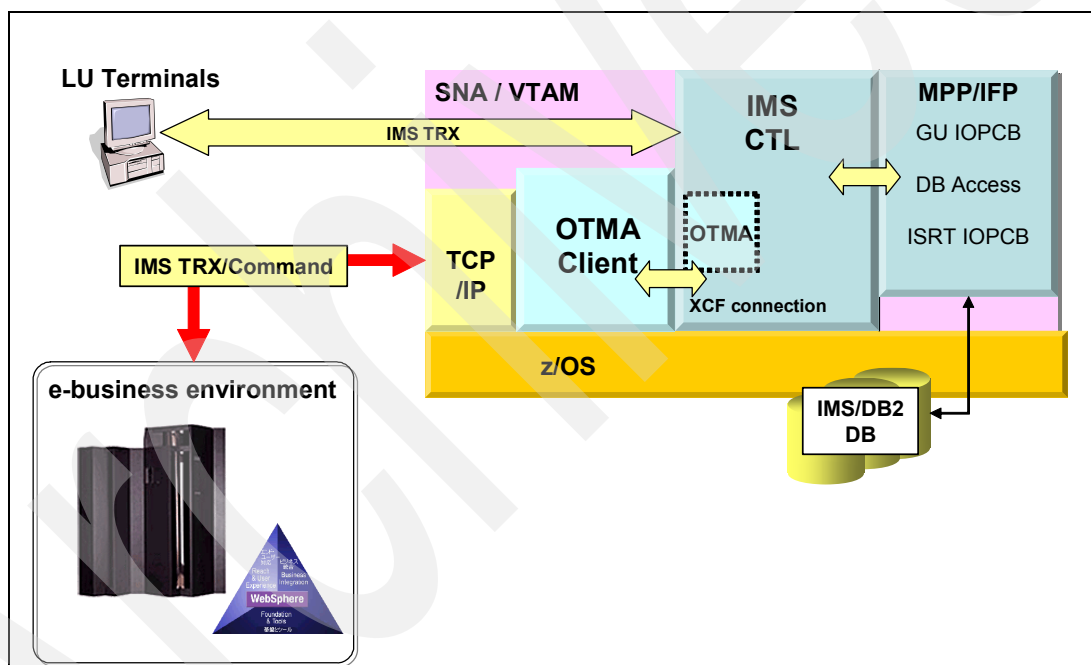


Figure 2-3 General message flows from OTMA client and from LU2 to IMS

6. Output is sent:
 - For IMS commands, output is sent back synchronously, with a few exceptions.
 - For IMS transactions using commit-then-send mode, application output is enqueued on a dynamically created IMS Tpipe structure before it is sent to the client.
 - For IMS transactions using send-then-commit mode, the output is sent synchronously by IMS and bypasses the message queues.
7. The synchronization level determines whether the application requests an acknowledgment from the client when sending send-then-commit mode output. (In commit-then-send mode, acknowledgments are always requested.)

Because XCF cannot guarantee sequential delivery of messages, IMS ensures that incoming segmented messages are ordered correctly.

For a synchronized Tpipe, all output messages are serialized through a single process, and sequence numbers can be assigned to messages for resynchronizing purposes in the event of a failure. The output message is enqueued to a client-specific and dynamically created Tpipe structure before being sent to the client.

2.3 Commit processing message flows

The OTMA client can control how IMS commits transactions by the commit mode value in the state data. There are two possibilities: commit-then-send mode or send-then-commit mode. The fundamental difference between these two commit modes is the way the output to the client is handled. Commit-then-send is also called commit mode 0 (CM0), and with it, IMS commits output before sending it. Send-then-commit mode is called commit mode 1 (CM1), and with it, IMS sends the output to the client before committing the data.

2.3.1 Commit-then-send (commit mode 0) flow

The commit-then-send flow is also known as the standard IMS flow, because it is the way IMS has traditionally worked. The output message is enqueued to the output queue (in the case of OTMA, to the Tpipe structure) and sent after the transaction program has reached its commit point and completed, as shown in Figure 2-4.

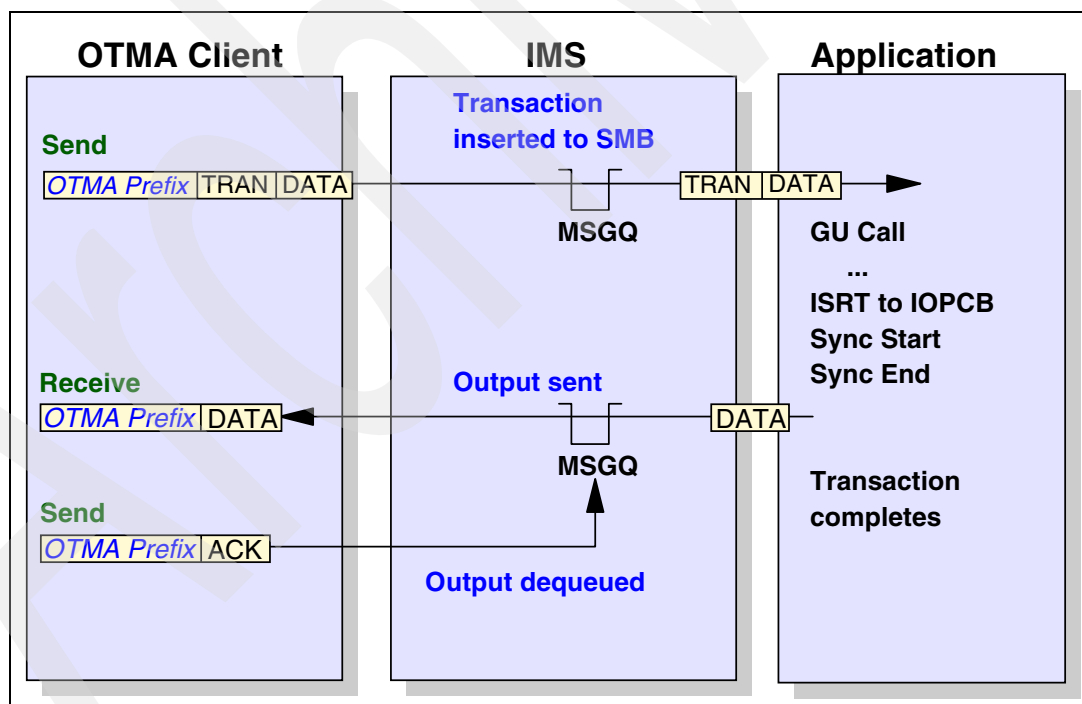


Figure 2-4 Commit-then-send message flow

In order to guarantee that client transactions are processed and that they are processed only once, OTMA provides a protocol for synchronizing transactions. This is done by defining the Tpipe as synchronized in the MCI section of the message and by using commit mode 0. IMS then waits for the response from the client (acknowledgment) before the output is dequeued from the message queue. This provides the message recoverability similar to that provided for Set and Test Sequence Number (STSN) terminals, such as SLUP, 3600, and ISC terminals. If ACK is not received or a NAK is sent by the client, the output message is not dequeued.

Message recoverability is applicable only for commit-then-send mode transactions. Any input messages on the message queue that are marked nonrecoverable are discarded during an IMS restart. Any input messages on the message queue that are marked recoverable are left on the message queue and are eligible for scheduling after the IMS restart completes.

Transactions from OTMA clients are recoverable if the recoverable sequence number in the MCI section of the message prefix is not 0. IMS then forces the transaction to be recoverable even if the TRANSACT macro does not contain the INQUIRY=RECOVER parameter.

2.3.2 Send-then-commit message (commit mode 1) flows

With send-then-commit mode, there are three different levels of synchronization that can be used: *none*, *confirm*, or *sync point*. The message flow is different in all these different cases. With commit-then-send mode, the synchronization level parameter is ignored, and the synchronization level of confirm is always used.

With the send-then-commit flow, the output is not enqueued; instead, it is sent directly to the client before the transaction is committed. Therefore, the output is nonrecoverable in case of a transaction abort. Even though the message is received, the OTMA client should not process the message. After sync point is complete, OTMA will send another output indicating whether the sync point was successful or failed. This is called a deallocate message. If the sync point was successful, a deallocate confirm message is sent, and the OTMA client knows that the transaction has not backed out. If the sync point was not successful (for example, DB2 voted “no” during two-phase commit), OTMA will send a deallocate abort message, and the OTMA client knows that the transaction has backed out. It then has to make the decision of what to do with the output message.

Send-then-commit (sync level=none)

With send-then-commit flow, IMS sends the output message, but does not request or wait for an acknowledgement from the client. After the message is sent, sync point processing continues. Even though the OTMA client gets the message, it should not process the message until the deallocate message is received from OTMA. Figure 2-5 on page 14 shows the message flow for send-then-commit with sync level=none and deallocate confirm message. If the sync point does not complete successfully, OTMA sends the deallocate abort instead of the deallocate confirm message, and then the client discards the output message.

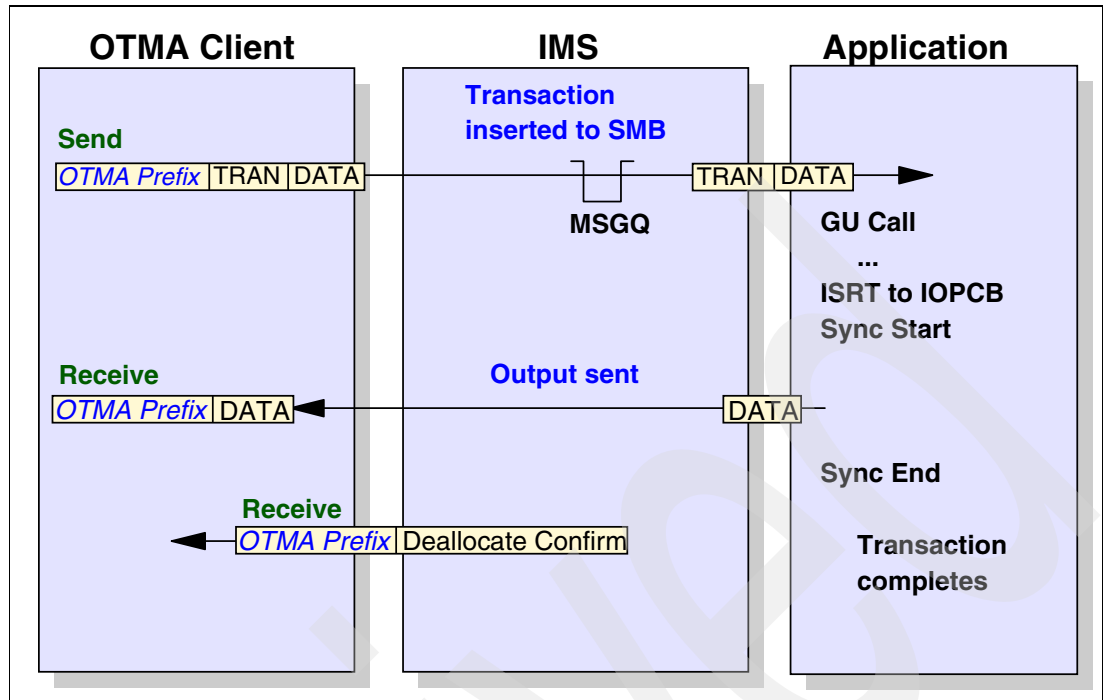


Figure 2-5 Send-then-commit message flow with `sync level=none`, `deallocate confirm`

Send-then-commit (`sync level=confirm`)

If the synchronization level is set to confirm in the state-data section, IMS sends the output message, request, and acknowledgement (ACK) or negative acknowledgement (NAK) before continuing sync point processing. This increases region occupancy.

If an ACK is returned, sync point processing continues. Even though the OTMA client has received the message and returned an ACK, it still does not process the message until the deallocate message is received. Figure 2-6 on page 15 shows the deallocate confirmed flow. Again, if the sync point does not complete successfully, OTMA sends the deallocate abort instead of the deallocate confirm message, and then the client discards the output message.

If a NAK is returned, the transaction will abend U0119. The database updates are backed out, and message DFS555I is sent to the client. The client then sends an ACK for the DFS555I message, and IMS responds with the commit aborted message.

While OTMA is waiting for an ACK/NAK, a `/DIS T MEMBER xxx TPIPE yyy` command shows a status of `WAIT_A` (waiting for ACK). If you enter the `/STOP T MEMBER xxx TPIPE yyy` command, a NAK is generated and the transaction will abend U0119.

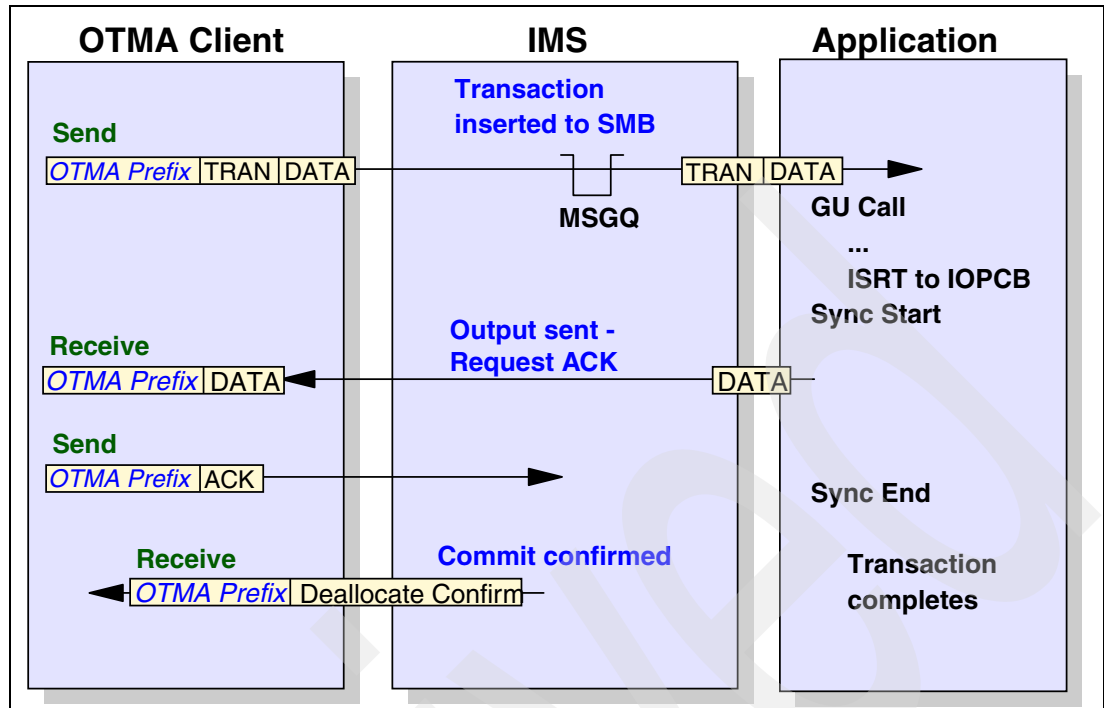


Figure 2-6 Send-then-commit message flow with sync level=confirm

Send-then-commit (sync level=sync point)

Figure 2-7 on page 16, shows the message conceptual flow in a situation where OTMA application programs and OTMA remote application programs participate with IMS in protected conversations with coordinated resource updates. z/OS Resource Recovery Services (RRS) does the global coordination of resources. RRS manages the sync point process on behalf of the conversation participants: the application program and IMS. IMS is acting as local resource manager.

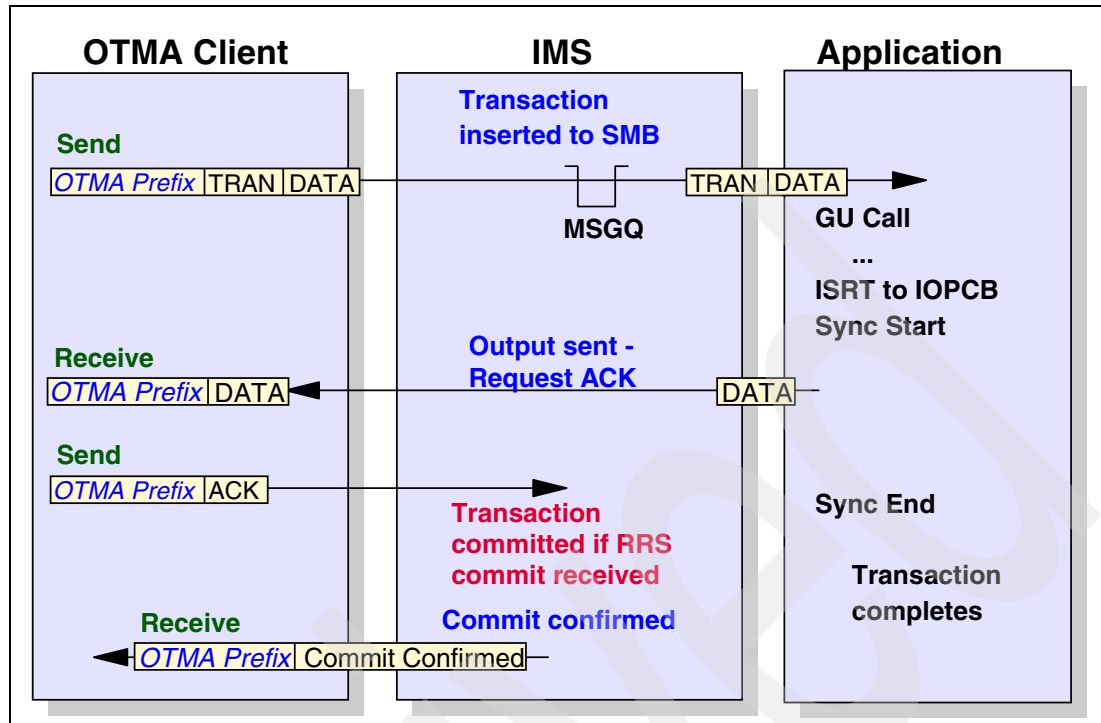


Figure 2-7 Send-then-commit message flow with sync level=sync point

In the resource coordination, two-phase commit (2PC) protocol is used. The two-phase commit protocol is a process involving the participants, the sync point manager, and the resource managers to ensure that, for updates made to a set of resources by the third-party application program, either all occur or none occurs. In simple terms, the application program decides to commit its changes to some resources. This commit is made to the sync point manager who then polls all of the resource managers as to the feasibility of the commit call. This is the preparation phase, often called phase 1. Each resource manager votes *yes* or *no*, and when the sync point manager has gathered all the votes, phase 2 begins. If all votes are to commit the changes, the phase 2 action is commit. Otherwise, phase 2 becomes a backout. System failures, communication failures, resource manager failures, or application failures are not barriers to the completion of the two-phase commit process.

The work done by various resource managers is called a unit of recovery (UOR) and spans from one consistent point of the work to another consistent point, usually from one commit point to another. It is the unit of recovery that is the object of the two-phase commit process.

The OTMA environment requires some additional code, because OTMA is not a resource manager. The additional code needed is an OTMA client, which supplied by IBM or an equivalent. The client indicates to IMS (in the OTMA message prefix) that this message is part of a protected conversation; therefore, IMS and the client are participants in the coordinated commit process as managed by RRS.

2.3.3 IMS commit mode 1 message processing

OTMA processing of commit mode 1 (send-then-commit) input messages can be confusing, especially if program-to-program message switches are involved. Commit mode 1 (CM1) input messages can produce commit mode 0 (CM0) IOPCB output messages, which can require special processing by the OTMA client. CM1 input messages can also produce the message DFS2082 RESPONSE MODE TRANSACTION TERMINATED WITHOUT REPLY

for nonresponse mode transactions. There are also special considerations for IMS conversational transactions.

TIB control block

Each input message has a transaction instance block (TIB) control block associated with it. In IMS Version 8, the TIB is 1,696 bytes in extended private and 150 bytes in the local system queue area (LSQA). For CM0 messages, the TIB is freed when the input message has been put on the IMS message queue. For CM1 messages, the TIB is not freed until a CM1 IOPCB output message or DFS2082 message has been sent. In some cases, neither of these two actions happens, and the TIB can be orphaned and never freed. This fills up storage and can cause the IMS control region to abend. These cases are explained later in this section.

CM1 input with no program-to-program message switches

When a CM1 input message is sent to a transaction, OTMA treats that transaction as a *response* mode even if the transaction is defined as *nonresponse*. If the application does reply to the IOPCB, the output is CM1. If the application does not reply to the IOPCB and does not do a program-to-program message switch, OTMA responds with a DFS2082 RESPONSE MODE TRANSACTION TERMINATED WITHOUT REPLY message.

CM1 input with program-to-program message switch

If the CM1 transaction does do a program-to-program message switch to one or more transactions, those message-switched-to transactions can be scheduled synchronously or asynchronously. At most, only one of the message-switched-to transactions is scheduled synchronously.

Synchronously scheduled message-switched-to transactions

If a message-switched-to transaction is scheduled synchronously, it has the responsibility of replying to the IOPCB or doing a program-to-program message switch to a transaction that is scheduled synchronously and replies to the IOPCB. If the application does reply to the IOPCB, the output is CM1. If the application does not reply to the IOPCB and does not do a program-to-program message switch, OTMA responds with a DFS2082 RESPONSE MODE TRANSACTION TERMINATED WITHOUT REPLY message. This is unlike non-OTMA transactions where only the first transaction scheduled can generate a DFS2082 message. If this transaction does program-to-program message switches, the message-switched-to transactions use the rules described in the following section, and, at most, one of them is scheduled synchronously.

Asynchronously scheduled message-switched-to transactions

If a message-switched-to transaction is scheduled asynchronously, any IOPCB output is CM0. If it does not reply to the IOPCB and does not do program-to-program message switches, there is no DFS2082 message, even if the transaction is defined as a response mode. If an asynchronous transaction does program-to-program message switches, all of the message-switched-to transactions are scheduled asynchronously.

Choosing synchronous or asynchronous

Whether a message-switched-to transaction is scheduled synchronously or asynchronously depends on several factors:

- ▶ IOPCB out from the CM1 input transaction
- ▶ EXPRESS or non-EXPRESS ALTPCB
- ▶ The OTMAASY parameter
- ▶ Response mode or nonresponse mode
- ▶ When the transaction is scheduled

IOPCB output and program-to-program message switches

If the CM1 input transaction (or synchronous message-switched-to transaction) replies to the IOPCB and also does program-to-program message switches, all of its message-switched-to transactions are scheduled asynchronously. The IOPCB output from this transaction is CM1, but the IOPCB output from all of its message-switched-to transactions is CM0.

No IOPCB output and all message switches by EXPRESS ALTPCB

If the CM1 input transaction (or synchronous message-switched-to transaction) does not reply to the IOPCB and does program-to-program message switches using only EXPRESS ALTPCBs, all of its message-switched-to transactions are scheduled asynchronously. There is no CM1 output and no DFS2082 message. OTMA deletes the TIB during phase 2 sync point so that it will not be orphaned.

No IOPCB output and message switches by non-EXPRESS ALTPCBs

If the CM1 transaction (or synchronous message-switched-to transaction) does not reply to the IOPCB and does program-to-program message switches using non-EXPRESS ALTPCBs or a mix of EXPRESS and non-EXPRESS ALTPCBs, the OTMAASY parameter comes into effect. OTMA does not delete the TIB and it can become orphaned.

OTMAASY=N

If OTMAASY=N is specified (or defaulted), *any* message-switched-to transaction (response or nonresponse) is eligible to be scheduled synchronously.

If any message-switched-to transactions schedule before the CM1 input transaction (or synchronous message-switched-to transaction) has completed (these were inserted with EXPRESS ALTPCBs), they are scheduled asynchronously.

The first message-switched-to transaction (response or nonresponse) to be scheduled after the CM1 input transaction (or synchronous message-switched-to transaction) has completed is scheduled synchronously. If this transaction replies to the IOPCB, the output is CM1. If this transaction does not reply to the IOPCB and does not do program-to-program message switches, OTMA returns a DFS2082 message.

All other message-switched-to transactions are scheduled asynchronously, and their IOPCB output is CM0.

OTMAASY=Y

If OTMAASY=Y is specified, only response message-switched-to transactions are eligible to be scheduled synchronously.

If any message-switched-to transactions schedule before the CM1 input transaction (or synchronous message-switched-to transaction) has completed (these were inserted with EXPRESS ALTPCBs), they are scheduled asynchronously.

The first response mode message-switched-to transaction to be scheduled after the CM1 input transaction (or synchronous message-switched-to transaction) has completed is scheduled synchronously. If this transaction replies to the IOPCB, the output is CM1. If this transaction does not reply to the IOPCB and does not do program-to-program message switches, OTMA returns a DFS2082 message.

All other message-switched-to transactions are scheduled asynchronously, and their IOPCB output is CM0.

If all message-switched-to transactions are nonresponse mode or schedule before the CM1 input transaction (or synchronous message-switched-to transaction) ends, there is not a synchronous transaction scheduled and there is no CM1 output and no DFS2082 message. In this case, the TIB is orphaned and never deleted (until an IMS cold start).

Conversational transactions

For an input conversational transaction, only the message-switched-to continuation of the conversation is scheduled synchronously. All other transactions are scheduled asynchronously.

MSC implications

If a CM1 input message is sent through Multiple Systems Coupling (MSC) to a back-end IMS and the transaction there does not reply to the IOPCB, there is not a DFS2082 message sent back to the client.

The OTMA client can hang or time out. The input TIB and ITASK are never freed in the front-end system and are orphaned.

Shared queues implications

In an IMS shared queues environment for IMS Version 6 and IMS Version 7, CM1 input transactions must run on the same IMS on which they are received (the front-end IMS system). For IMS Version 8 and IMS Version 9, CM1 transactions can run on any IMS copy in the IMSplex (an IMSplex is one or more IMS address spaces that work together as a unit). IMS Version 9 has a new parameter (AOS=N) that says that CM1 transactions can only run on the front-end IMS copy (both OTMA and APPC). All of the CM1 IOPCB output is routed back through the front-end IMS using XCF. RRS is used to coordinate the front-end IMS and back-end IMS sync point processing.

In an IMS shared queues environment, all message-switched-to transactions from a CM1 input transaction (or synchronous message-switched-to transaction) must run on the same IMS copy as the original transaction. The only exceptions are IMS conversational transactions, which can run on any IMS copy in the IMSplex. Message-switched-to transactions from an asynchronously scheduled transaction can run on any IMS copy.

If a CM1 transaction runs on a back-end IMS copy, a second TIB is built there. If the TIB becomes orphaned, as described earlier, there are two orphaned TIBs—one in the front end and one in the back end.

2.4 Implementing OTMA

The OTMA feature is part of IMS Transaction Manager, so it is not necessary to do any special installation steps for it. You also do not need any IMS system generation specifications for OTMA.

The following OTMA-related IMS execution parameters can be specified in the IMS startup procedure or in the DFSPBxxx PROCLIB member:

- ▶ **GRNAME=**

GRNAME specifies the name of the XCF group for OTMA. IMS joins that group during the startup of IMS or as a result of the /START OTMA command.

► OTMA= Y/N

OTMA=Y (yes) specifies that the OTMA feature is to be initialized during the IMS startup.

OTMA=N (no) specifies that OTMA is not initialized during the startup. OTMA can be started later with the IMS command /START OTMA if it is not started by the OTMA=Y parameter. The default is N.

► OTMAMD= Y/N

OTMAMD=Y (yes) specifies the member override function of the OTMA Prerouting exit routine (DFSYPX0) is enabled for a transaction initiated from an OTMA client.

OTMAMD=N (no) specifies the member override function of the OTMA Prerouting exit routine (DFSYPX0) is not enabled. The default is N.

► OTMANM=

OTMANM specifies the member name that IMS uses when joining the XCF group. If OTMANM is not specified in an IMS system that is not RSR or XRF, the APPLID from the IMS system definition is used. The APPLID from the IMS system definition can be overwritten by the APPLID1 startup parameter in the IMS procedure or in the DFSPBxxx PROCLIB member. For IMS with RSR or XRF, USERVAR is used by OTMA.

► OTMASE=

OTMASE specifies the default type of OTMA RACF® security after an IMS restart. The default is F. If the default is used, it appears as X in a /DIS OTMA command. The following values are possible:

- C: OTMA RACF security is CHECK. The existing RACF calls are made. IMS commands are checked against the CIMS class. IMS transactions are checked against the TIMS class.
- F: OTMA RACF security is FULL. This is the same as CHECK, but additional security checking is performed in dependent regions.
- N: OTMA RACF security is NONE. No calls to RACF are made.
- P: OTMA RACF security is PROFILE. Each OTMA message defines the level of security checking to be done.

The /SECURE OTMA command overrides the value you specify in the OTMASE keyword.

If you do not specify the OTMASE keyword, IMS retains the OTMA security settings (which are established by the /SECURE OTMA command) after a warm start or an emergency restart.

► OTMASP=Y/N

OTMASP=Y (yes) specifies that if a new Tpipe is created to deliver ALTPCB output, it is a synchronous Tpipe. This is usually used when WebSphere MQ is the OTMA client. IMS Connect does not use synchronous Tpipes.

OTMASP=N (no) specifies that if a new Tpipe is created to deliver ALTPCB output, it is an asynchronous Tpipe. The default value is N.

The OTMA DFSYDRU0 exit can override this parameter by turning on a bit that specifies that a new Tpipe will be synchronous. It must also turn on a second bit to say that valid sequence numbers will be passed for this Tpipe. This is required for WebSphere MQ output to be persistent.

► OTMAASY=Y/N

OTMAASY=Y (yes) specifies that a nonresponse transaction originating from a program-to-program switch be scheduled asynchronously. This parameter is for send-then-commit (commit mode 1) messages only. A DFS2082 message is not issued for a transaction scheduled asynchronously.

OTMAASY=Y can also be used in multiple program-to-program switches to ensure that only the response transaction is scheduled synchronously.

Setting this parameter to Y can help to prevent a race condition that can occur when you do multiple program switches in a send-then-commit transaction and you mix response and nonresponse transactions. OTMAASY ensures that the response transaction, the one most likely to return IOPCB output, is scheduled synchronously.

The default value is OTMAASY= N. See 2.3.3, “IMS commit mode 1 message processing” on page 16 for a more detailed discussion of the OTMAASY parameter.

Checking the status of OTMA

Use the /DISPLAY OTMA command to check the status of OTMA in IMS. Example 2-1 shows that SCSIM9G is the XCF member name (OTMANM) of the OTMA server (that is IMS), it is active (OTMA=Y), and it belongs to the XCF group IMS9EXCF (GRNAME). The OTMASE parameter is set or defaulted to FULL. HWS910G in this example is an XCF member name of an OTMA client that happens to be IMS Connect in this case.

Example 2-1 /DIS OTMA command output

DFS000I	GROUP/MEMBER	XCF-STATUS	USER-STATUS	SECURITY	IMSG
DFS000I	IMS9EXCF				IMSG
DFS000I	-SCSIM9G	ACTIVE	SERVER	FULL	IMSG
DFS000I	-HWS910G	ACTIVE	ACCEPT TRAFFIC		IMSG
DFS000I	*05178/195343*	IMSG			

2.5 OTMA security issues

Use the /SECURE command to control the RACF security level for input from OTMA clients. It is used for administrative control of the IMS environment and as an emergency operations control command to throttle RACF activity, without requiring an IMS shutdown. The /SEC OTMA command is used with the CHECK, FULL, NONE, or PROFILE parameters.

You can use the /DISPLAY OTMA command to show the security level that is currently in effect. Use the OTMASE execution parameter to change the level of security desired at the IMS startup. The default is FULL. The /SECURE OTMA command overrides the value you specify in the OTMASE keyword. If you do not specify the OTMASE keyword, IMS retains the OTMA security settings (which are established by the /SECURE OTMA command) after a warm start or emergency restart.

The parameters in the /SEC OTMA command have the following meanings:

► CHECK

Causes existing RACF calls to be made. IMS commands are checked using the RACF resource class of CIMS. IMS transactions are checked using TIMS.

► FULL

Causes the same processing as the CHECK parameter, but uses additional RACF calls to create the security environment for dependent regions.

- ▶ NONE

Does not call RACF within IMS for security verification.

- ▶ PROFILE

Causes the values in the security-data section of the OTMA message prefix for each transaction to be used.

If RACF is used for security, the XCF group name and the member name (IMSXCF.group.member) have to be defined in the FACILITY class. Note that the SAF interface is used, so if OTMA security is activated, the FACILITY class needs to be defined.

If the IMSXCF.group.member is not defined in the FACILITY class, a client bid is not allowed. This is also a technique used to control OTMA client access to production systems and to prevent an unauthorized OTMA client (OTMA CI) from accessing an IMS system. Use the /SECURE OTMA PROFILE command so that subsequent transactions can be processed according to the values in the security-data section of the OTMA message prefix for each transaction.

If the group name and IMS member name (IMSXCF.group.member) are defined in the FACILITY class and if IMS security is not set to NONE, the user ID in the token for the client bid must be valid (that is, have read access to the profile).

Only transactions or commands that must be protected belong to the TIMS or CIMS classes. If the transaction is not in the TIMS class, or the command is not in the CIMS class, the transaction is allowed, regardless of the option set by the /SECURE OTMA command.

The XCF client must be z/OS APF authorized.

For OTMA applications defined with full security, security is kept until the application ends. Full security also creates an access control environment element (ACEE) in the dependent region. This can be used to enforce security for CHNG calls, AUTH calls, or deferred conversational program-to-program message switches.

The client-bid request for a client includes:

- ▶ Access control environment element (ACEE) aging value
- ▶ Hash table size

The hash table contains the user ID and time stamp (of the last RACINIT command). The table size must be large enough to hold the total number of expected user IDs.

- ▶ The client-user token

The user token is optional, except when RACF security is used, and is identified in the security section of the message prefix.

You can specify “No Security Checking” for the security flag in the security-data section of the message prefix for any transaction.

If RACF security is used, the user token is mandatory for a client bid. If the user token (for a client bid) fails RACF verification, the client receives a NAK message from the server.

If you specify /SECURE OTMA NONE, IMS does not use RACF for security verification, regardless of what security is specified by the class for a client bid or for transactions. If you specify /SECURE OTMA PROFILE (NONE, FULL, or CHECK), IMS checks the message security-data section.

2.6 Super member support for IMS Connect

When the commit-then-send (CM0) protocol is used to send transactions to OTMA from IMS Connect, all asynchronous output is queued to an output Tpipe. These queues are not shared, so to retrieve the asynchronous messages, you must issue a RESUME TPIPE request to the *same* OTMA client (in this case, the same IMS Connect) that sent the original message.

If you are using Sysplex Distributor to connect your clients with IMS Connect, you do not know which IP address will get the request, because Sysplex Distributor can route it to any of the IMS Connect servers it is managing. This means that your RESUME TPIPE request can be routed to a different IMS Connect than the one that generated the asynchronous output. Consequently, you are not able to retrieve those queued messages waiting in the original Tpipe.

In addition, if you are using a configuration with several IMS Connect instances, if one of those instances fails, you have no way to recover the asynchronous messages queued in its Tpipes until you recover the failed IMS Connect.

Figure 2-8 shows an example of such a configuration with two IMS Connect instances, one IMS, and Sysplex Distributor. In this example, a transaction coming from the client can be diverted to IMS Connect A or IMS Connect B. If this transaction generates asynchronous output, it is queued in the queue associated with the IMS Connect where Sysplex Distributor sent the original transaction. If we want to pull this asynchronous message, the client must send a RESUME TPIPE message, which Sysplex Distributor sends again to one of the two IMS Connect instances. We cannot be sure that the RESUME TPIPE will end in the same IMS Connect where the message is queued, so we cannot guarantee the retrieval of the message.

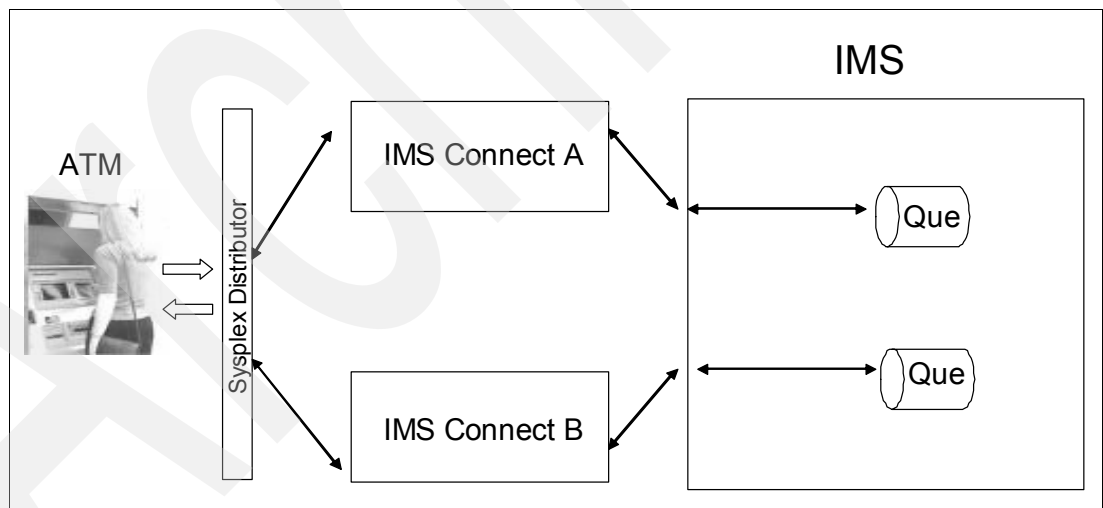


Figure 2-8 Two IMS Connect instances managed by Sysplex Distributor

OTMA introduces the super member concept as a way to solve this problem. A super member is a special OTMA member that can be shared by a set of instances of IMS Connect to handle the CM0 hold queue messages.

Figure 2-9 on page 24 shows the same previous configuration, adding the super member feature. Now, the asynchronous messages are not queued in separate members for each of the IMS Connect instances, but in a new, shared OTMA member (the super member). In this

way, when we want to retrieve the asynchronous messages, each one of the IMS Connect instances is able to reach the super member and thus its message queues.

Therefore, the actual selection of Sysplex Distributor is not relevant. And, if you lose one of the two IMS Connect instances, you are still able to retrieve the asynchronous output using the remaining instance without waiting for the failed instance to recover.

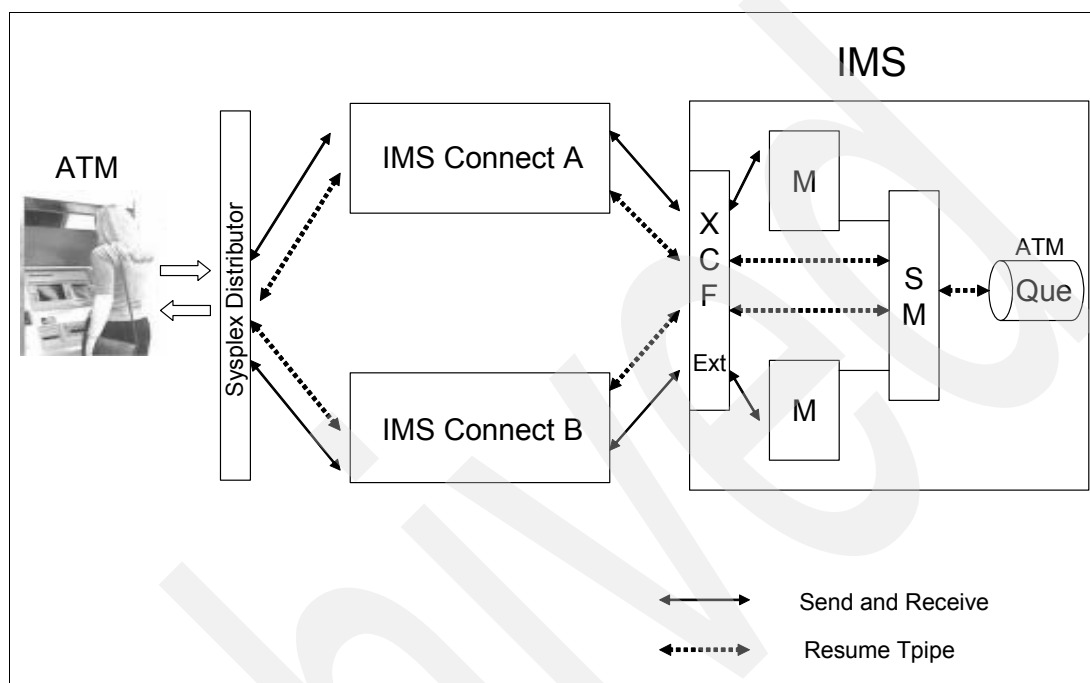


Figure 2-9 Two IMS Connect instances sharing a super member

2.6.1 Super member feature availability

The following IMS OTMA and IMS Connect APARs provide the super member function:

- ▶ IMS Version 8: PK09944
- ▶ IMS Connect V2R2: PK10910
- ▶ IMS Version 9: PK09946
- ▶ IMS Version 9: Integrated IMS Connect, PK10911

2.6.2 Defining the super member feature

To use the super member feature, first you have to define it in the HWSCFGxx member corresponding to each of the IMS Connect instances that will be sharing the super member, as shown in the line marked "1" in Example 2-2.

Example 2-2 Defining a super member name in HWSCFGxx

```
HWS (ID=IM1ACONN)
TCP/IP (HOSTNAME=TCP/IP,RACFID=STC,MAXSOC=2000,PORTID=(7001),
EXIT=(HWSMPL1))
SMEMBER=(SMEM1),
DATASTORE (ID=IM1A,GROUP=IM1AXCF,MEMBER=HWSIM1A,TMEMBER=SCSIM1A)
DATASTORE (ID=IMSC,GROUP=IM1AXCF,MEMBER=HWSIMSC,TMEMBER=SCSIMS8C)
```

1.

In this example, we use the name SMEM1 for the super member name. If we wanted another IMS Connect instance to be able to retrieve the asynchronous messages generated by transactions originated by IM1ACONN, we simply use the same super member name. OTMA and IMS Connect create the shared queues structure as needed. When a transaction generates asynchronous output, it is queued in the Tpipe associated with the super member, and thus that output will be retrievable by any IMS Connect that uses the same super member name.

Note: Do not use the super member name as the destination of the RESUME TPIPE interactions. You must use the original datastore name. OTMA knows that the datastore is associated with a super member and retrieves the asynchronous output from the correct queue.

The super member implementation adds another interesting feature to OTMA: One OTMA Tpipe running in the shared queues environment is deleted if it is idle for two consecutive IMS checkpoints. This alleviates the storage consumption caused by the creation of a large number of Tpipes.

2.6.3 Using the super member feature

After the super member function is activated by a set of IMS Connects, use the IMS Connect command VIEWHWS to display the super member name. Issue the IMS commands such as /DISPLAY OTMA and /DISPLAY TMEMBER TPIPE to display member information with the super member name. When you issue the /START TMEMBER TPIPE, /STOP TMEMBER TPIPE, or /TRACE TMEMBER TPIPE command with a regular member specified, OTMA expands the command to cover the related super member. These commands can also be issued to an existing super member directly.

For asynchronous output messages created through ALT-PCB processing, the messages are stored into the super member directly. If the input messages are submitted from an IMS Connect that supports the super member, the input parameter list to the DFSYPRX0 and DFSYDRU0 user exits contains a new flag indicating that the message is from a super member supported client. Also, the OTMA state-data prefix pointed to by the input parameter list contains the super member name, if any.

If multiple IMS systems are involved with the super member, those IMS systems must also have IMS shared queues implemented. If there is only one IMS system connected by a set of IMS Connects, shared queues support is not required. Super member functions can also be activated in the supported environments shown in Figure 2-10 on page 26.

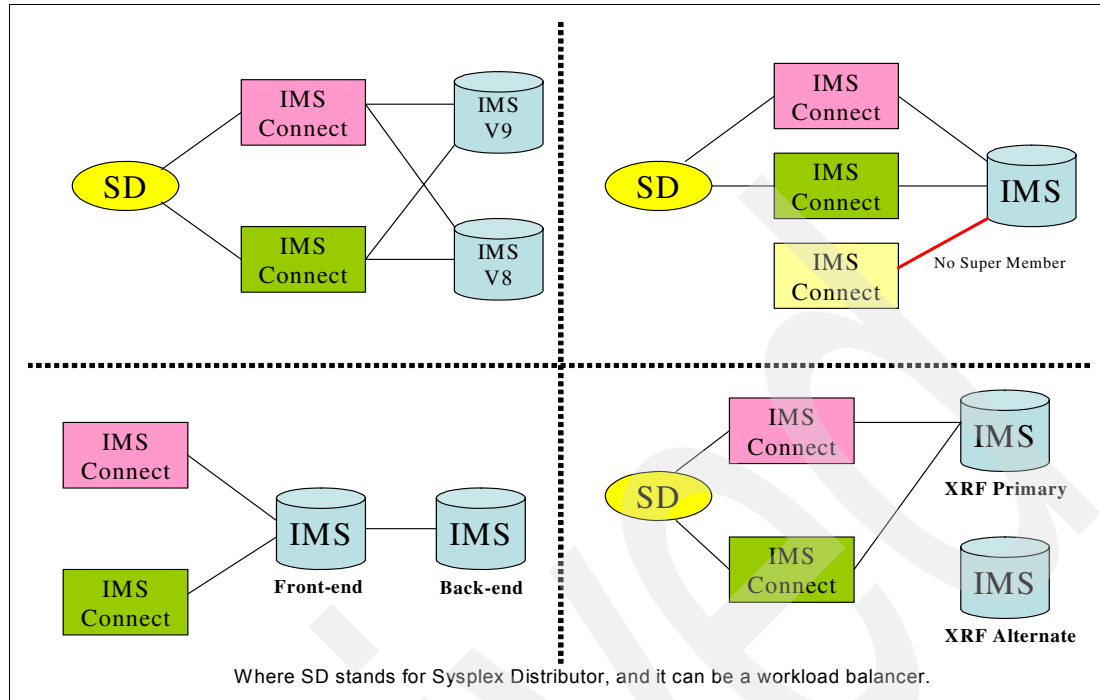


Figure 2-10 Potential environments running with the super member

In addition, the super member solution provides the following enhancements to shared queues customers:

- ▶ The /DISPLAY OTMA command is enhanced to show OTMA members that are created at the shared queues back-end system.
- ▶ OTMA Tpipe clean up can be performed in the shared queues environment.
- ▶ IMS Connect can go to the front-end IMS to retrieve the ALT-PCB output messages created by the shared queues back-end system.

2.7 OTMA callable interface

The IMS OTMA callable interface (C/I) is a function that provides a high-level interface for access to IMS applications from other z/OS subsystems. The interface consists of 10 API calls that can be used to join the IMS/OTMA XCF group, connect to IMS, allocate communication sessions, send IMS transactions and commands, receive output from IMS, close communication sessions, and leave an XCF group.

A benefit of OTMA C/I is that it is easy to use. It is possible for the user to implement their own OTMA client without a callable interface, but it is not necessarily a simple task, because an understanding of the technical protocols of z/OS XCF and IMS OTMA is required. The OTMA callable interface extracts the details of OTMA and XCF. OTMA C/I supports the execution of IMS transactions and commands, and it enables programs running from other z/OS subsystems to connect to multiple IMSs. OTMA C/I API calls can be made from an authorized or unauthorized library, and OTMA C/I can connect to all IMS OTMA releases.

2.7.1 OTMA C/I initialization

OTMA C/I provides a stand-alone program, DFSYSVIO, that must be run after the z/OS IPL to initialize the OTMA C/I. DFSYSVIO invokes DFSYSVC0, one of the OTMA C/I modules. DFSYSVC0 loads and registers the SVC services by an authorized address space running on the same z/OS image as the application programs accessing it.

You must add an entry in the z/OS program properties table (PPT) for the OTMA callable interface initialization program. Complete the following steps to do this:

1. Edit the SCHEDxx member of the SYS1.PARMLIB data set.
2. Add the entry shown in Example 2-3 to the SCHEDxx member.

Example 2-3 SCHEDxx definitions for OTMA callable interface

PPT	PGMNAME(DFSYSVIO)	/* PROGRAM NAME = DFSYSVIO	*/
	CANCEL	/* PROGRAM CAN BE CANCELED	*/
	KEY(7)	/* PROTECT KEY ASSIGNED IS 7	*/
	SWAP	/* PROGRAM IS SWAPPABLE	*/
	NOPRIV	/* PROGRAM IS NOT PRIVILEGED	*/
	DSI	/* REQUIRES DATA SET INTEGRITY	*/
	PASS	/* CANNOT BYPASS PASSWORD PROTECTION	*/
	SYST	/* PROGRAM IS A SYSTEM TASK	*/
	AFF(NONE)	/* NO CPU AFFINITY	*/
	NOPREF	/* NO PREFERRED STORAGE FRAMES	*/

Take one of the following actions to make the SCHEDxx changes effective:

- ▶ Re-IPL the z/OS system.
- ▶ Issue the z/OS SET SCH= command.

Related reading: For additional information about updating the program properties table, see *z/OS V1R6.0 MVS Initialization and Tuning Reference*, SA22-7592.

Example 2-4 shows a sample JCL procedure for running DFSYSVIO.

Example 2-4 Sample JCL for initializing OTMA callable interface

```
PROC RGN=3000K,SOUT=A,
//          PARM1=
//*
//IEFPROC EXEC PGM=DFSYSVIO,
//          REGION=&RGN
//*
//STEPLIB DD DISP=SHR,UNIT=SYSDA,
//          DSN=IMS.SDFSRESL
//SYSPRINT DD SYSOUT=&SOUT
//SYSUDUMP DD SYSOUT=&SOUT
```

2.7.2 OTMA C/I security

A new RACF FACILITY class IMSXCF.OTMACI has to be defined for the OTMA C/I to protect XCF groups from any non-authorized caller. When the RACF resource is defined, RACF RACHECK is invoked before OTMA C/I performs a XCF JOIN. This method protects the access to XCF, the XCF group, and the member. This RACF checking is performed only when a non-authorized caller is using OTMA C/I. Additional security characteristics remain consistent with OTMA.

2.7.3 OTMA C/I restrictions

These restrictions apply to the OTMA C/I:

- ▶ Application program languages other than C and C++ are not currently supported by OTMA C/I. Other languages, such as COBOL, Assembler, and PL/I, will be supported in the future.
- ▶ All OTMA calls must be made in the same state (PSW key, supervisor or problem state, authorized or non-authorized) as the `otma_open` call. For example, if you were authorized when you did the `otma_open` call, you must be authorized for all subsequent calls.
- ▶ The commit-then-send option of IMS OTMA is not supported by OTMA C/I. If IMS generates a commit-then-send output and sends it to an OTMA C/I client, OTMA C/I ignores the output and does not deliver it to the OTMA C/I client.
- ▶ The resynchronization feature of IMS OTMA is not supported.
- ▶ The IMS command `/SECURE OTMA PROFILE` is not currently supported.

2.7.4 Compiling and binding requirements for OTMA C/I

The header file included in the API calling program declares each API invocation and variables used for the invocation. For a C/C++ program using the OTMA callable interface, the C/C++ header file, `DFSYC0.H`, needs to be included in the C/C++ program.

The object stub, `DFSYCRET`, receives all the API invocations and issues a SVC call to perform the requested function. The object stub needs to be available during the link-editing of the API invoking program. `DFSYCRET` is in `SDFSRESL` or `ADFSLOAD` data sets.

2.7.5 Call functions implemented by OTMA C/I

OTMA callable interface implements 10 different functions with the following API calls:

- ▶ `otma_create`
Creates storage structures to support communications, but does not establish a connection with IMS.
- ▶ `otma_open`
Establishes a connection with IMS. Issues an `otma_create` prior to establishing an `otma_open` call.
- ▶ `otma_openx`
Provides the same function as the `otma_open` API, with an added parameter to specify OTMA Destination Resolution User (DRU) exit name routine and special options.
- ▶ `otma_alloc`
Creates an independent transaction session.
- ▶ `otma_send_receive`
Sends to IMS and passes parameters for receive functions.
- ▶ `otma_send_receivex`
Provides the same function as `otma_send_receive` API, with an added parameter to pass OTMA user data.
- ▶ `otma_send_async`
Sends input (transaction or IMS command) only to IMS.

- ▶ `otma_receive_async`
Receives unsolicited or queued output from IMS.
- ▶ `otma_close`
Releases the independent transaction session.
- ▶ `otma_free`
Ends the connection with IMS.

IMS Version 9 Open Transaction Manager Access Guide and Reference, SC18-7829, further explains these API calls.

2.8 DSNAIMS stored procedure for OTMA C/I access

DSNAIMS is a stored procedure that enables DB2 UDB applications to invoke IMS transactions and commands easily, without having to maintain their own connections to IMS. This stored procedure uses the IMS OTMA C/I to connect to IMS and execute the transactions. Figure 2-11 shows the general structure of DSNAIMS stored procedures. When you implement DSNAIMS stored procedure to your DB2 UDB for z/OS environment, you can access an IMS transaction or IMS command by using SQL call interface.

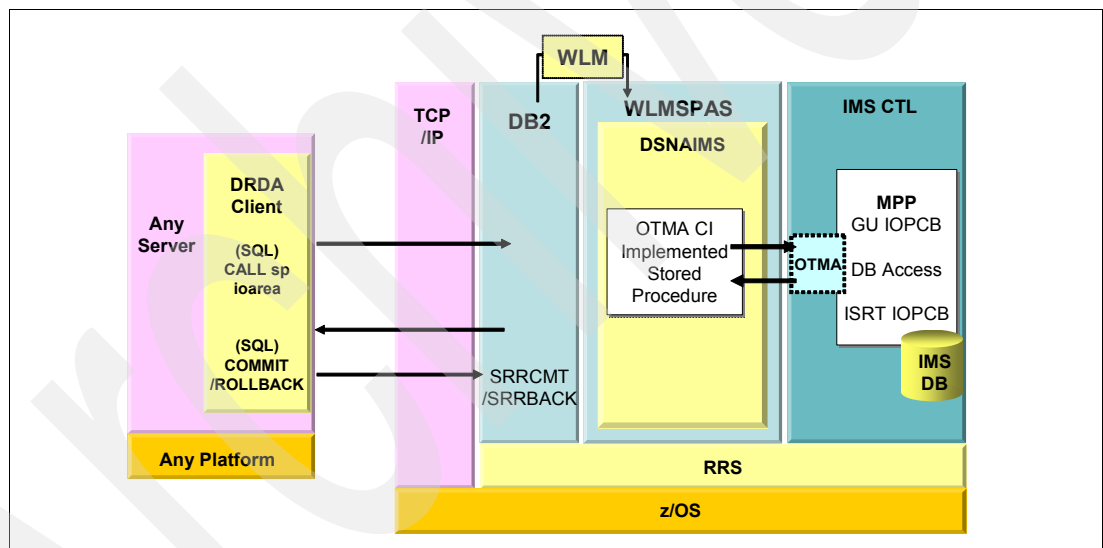


Figure 2-11 DSNAIMS stored procedure for OTMA C/I access

From the OTMA functional point of view, DSNAIMS *can* handle:

- ▶ Send-receive IMS transaction
- ▶ Send-only IMS transaction
- ▶ Receive-only IMS transaction
- ▶ Send-receive two-phase commit IMS transaction (with RRS)
- ▶ IMS command
- ▶ Specifying LTERM/MFS MOD name for I/O PCB mask of the IMS application
- ▶ Specifying Tpipe name (for send-only and receive-only)/DRU exit name/OTMA user data header contents

DSNAIMS *cannot* handle:

- ▶ Specifying OTMA commit mode explicitly
- ▶ IMS conversational transaction
- ▶ IMS multisegment message
(This restriction will be removed by APAR PK07907.)

For more information about the DSNAIMS stored procedure, see the following documents:

- ▶ *DB2 UDB for z/OS Version 8 Installation Guide*, GC18-7418
- ▶ *DB2 UDB for z/OS Version 8 Application Programming and SQL Guide*, SC18-7415
- ▶ *DB2 for z/OS Stored Procedures: Through the CALL and Beyond*, SG24-7083

2.9 WebSphere MQ as an OTMA client

In this section, we provide only a very general overview of connecting IMS with WebSphere MQ, formerly known as MQSeries. For more details, refer to the WebSphere MQ product documentation available at:

<http://www.ibm.com/software/integration/wmq/>

IMS applications can access WebSphere MQ messages in two ways:

- ▶ Using the WebSphere MQ API
The IMS application uses the WebSphere MQ API calls to get and put messages with synchronization point coordination with IMS. Applications are bound with specific WebSphere MQ provided stub modules. For online programs, this requires connecting WebSphere MQ to IMS through an external subsystem (ESS) connection, similar to the IMS to DB2 subsystem connection. For batch, there is no ESS interface, and thus the sync point coordination requires z/OS RRS.
- ▶ As an OTMA client using the WebSphere MQ IMS bridge
The WebSphere MQ IMS bridge is code in the WebSphere MQ queue manager, and it does not require connecting WebSphere MQ to IMS through ESS.

The WebSphere MQ IMS bridge is an OTMA client that ships with WebSphere MQ queue manager code. The IMS bridge communicates with IMS using specially defined queues for taking the messages off the queue and sending them to IMS using the IMS OTMA interface as well as receiving the output messages through the OTMA interface.

As with all OTMAs and OTMA clients, WebSphere MQ (client) and the IMS control region (OTMA server) must be in the same XCF group. One WebSphere MQ queue manager can connect to multiple IMS control regions, and one IMS control region can connect to multiple WebSphere MQ queue managers. WebSphere MQ and IMS can be on different z/OS LPARs in the same Parallel Sysplex®.

When the message arrives in WebSphere MQ, it is sent through XCF to the IMS OTMA interface. The message can be an IMS transaction or an IMS command, but it cannot be an IMS LTERM. IMS puts it on the IMS message queue, and the application does a get unique (GU) call to the IOPCB to retrieve the message. This is very similar to the implicit LU6.2 process. A remote queue manager can send a message to a local queue destined for IMS through OTMA.

The connection is defined in WebSphere MQ CSQZPARM using the OTMACON keyword on the CSQ6SYSP macro. OTMACON has five positional parameters, and it is specified as follows:

OTMACON = (Group,Member,Druexit,Age,Tpipefx)

The parameters have the following meanings:

- ▶ Group: XCF group.
- ▶ Member: WebSphere MQ XCF member.
- ▶ Druexit: IMS exit to format OTMA user data (overrides DFSYDTx), consider a name of DRU0xxxx (xxxx is a MQ queue manager name).
- ▶ Age: How long (in seconds) a user ID from WebSphere MQ is considered previously verified in IMS.
- ▶ Tpipefx: Three-character prefix for Tpipe name to avoid collision with IMS transaction code names.

Member CSQ4ZPRM in data set WMQ.SCSQPROC has default CSQZPARM members that you can use to build your members. For WebSphere MQ, you have to define one or more storage classes with the XCFGNAME and XCFMNAME parameters of the IMS systems to which you will connect. Example 2-5 provides an example for defining the storage class to WebSphere MQ.

Example 2-5 STGCLASS definition for a WebSphere MQ IMS bridge queue

```
DEFINE STGCLASS(IMG) -  
  PSID(02) -  
  XCFGNAME(IMS9EXCF)  
  XCFNAME(SCSIM9G)
```

SCSIM9G in our example is the XCF name of our target IMS system. Note that although XCFGNAME is also specified in the storage class definition, it will not be used, but the one defined in the OTMACON is used. When a STGCLASS is defined for a new IMS, WebSphere MQ does not connect unless WebSphere MQ or OTMA is recycled. Then, for input messages, you need to define local queues referring to the STGCLASS defined as in the previous example. For output messages, you need to define the corresponding reply-to queues.

After startup, WebSphere MQ joins the XCF group defined in the OTMACON parameter. The IMS bridge initiates a client bid resync to each active IMS defined in the STGCLASS macros. When the bid is successful, the IMS bridge opens the bridge queues and messages flow. There are no WebSphere MQ commands to start or stop the IMS bridge. There are IMS commands:

- ▶ /START OTMA - /STOP OTMA
- ▶ /START TMEMBER xxxx TPIPE yyyy - /STOP ...

For more information about defining the WebSphere MQ IMS bridge connection, refer to *WebSphere MQ for z/OS System Setup Guide, Version 5 Release 3.1, SC34-6052*.

Archived

IMS Connect overview

IBM IMS Connect provides connectivity to IMS from any TCP/IP client, enabling you to access IMS resources from a TCP/IP network. IMS Connect supports TCP/IP sockets access to IMS transactions and commands.

IMS Connect provides a general purpose and structured interface for:

- ▶ IMS connectors
- ▶ User-written clients
- ▶ IMS Control Center

This chapter provides an overview of IMS Connect, discussing the following topics:

- ▶ Introduction to IMS Connect
- ▶ IMS Connect architecture
- ▶ A brief history and evolution of IMS Connect
- ▶ IMS Connect clients
- ▶ IMS Control Center

3.1 Introduction to IMS Connect

IMS Connect is an integrated feature of IMS Version 9 that provides high-performance communications for IMS between one or more TCP/IP or local z/OS clients and one or more IMS systems. IMS Connect has the following features:

- ▶ Provides commands to manage the communication environment
- ▶ Assists with workload balancing
- ▶ Supports multiple TCP/IP clients accessing multiple datastore resources
- ▶ Reduces design and coding efforts for client applications
- ▶ Offers easier e-business access to IMS applications and operations with advanced security and transactional integrity

IMS Connect is included as part of IMS Version 9 in the IMS system services function modification identifier (FMID) HMK9900. To provide the same function between one or more TCP/IP clients and one or more IMS Version 7 or IMS Version 8 clients, IMS Connect is also available as one of the IBM DB2 UDB and IMS Tools products, IMS Connect Version 2, Release 2 (program number 5655-K52). IMS Connect Version 2.2 will continue to be supported for IMS Version 7 and IMS Version 8 clients, but future enhancements to the IMS Connect functionality will be made available only with IMS Version 9 or later.

Terminology: By IMS Connect in this redbook, we refer to the integrated IMS Connect function of IMS Version 9 *and* IMS Connect Version 2.2, unless explicitly indicated differently.

The IMS Connect architecture is designed to support any TCP/IP clients communicating with socket calls. IMS Connect supports TCP/IP sockets access to IMS transactions and commands. It does not require modifications to the existing IMS transactions.

IMS Connect also supports the TCP/IP clients using the IMS Connector for Java. The IMS Connector for Java is a collection of Java beans that enable a Java application to communicate data requests through TCP/IP and the IMS Connect to IMS. IMS Connect with WebSphere Development Tooling and the IMS Connector for Java can significantly ease the development of on demand business solutions that access IMS transactions. These solutions can be deployed in IBM WebSphere Application Server, allowing you to use Web applications, Java 2 Platform, Enterprise Edition (J2EE) applications, or Web services to quickly transform static Web sites into sources of dynamic Web content.

IMS Connect with IBM DB2 Version 8, the Universal Database Control Center provides a single graphical user interface to control both IMS and DB2, easing IMS operations.

3.2 IMS Connect architecture

IMS Connect runs in an address space on a z/OS system and performs router functions between TCP/IP clients and local option clients with IMS. Figure 3-1 on page 35 shows an overview of the IMS Connect architecture and introduces the IMS Connect components.

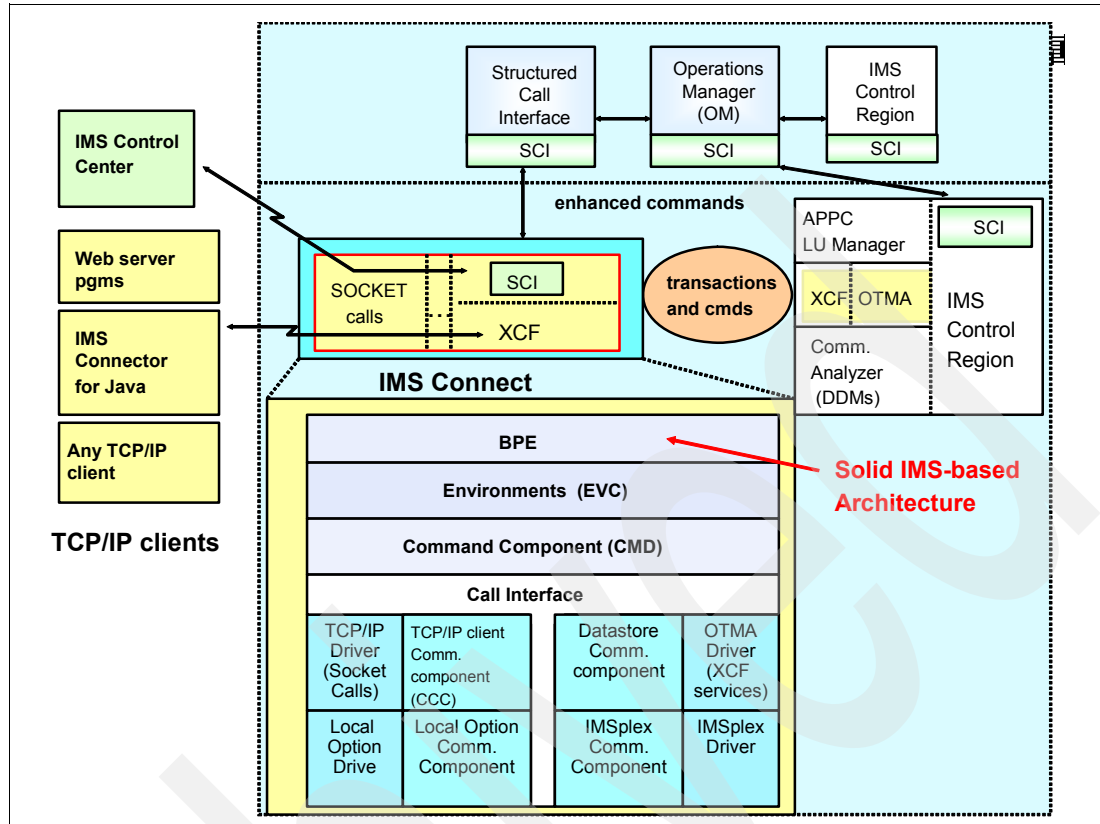


Figure 3-1 IMS Connect architecture

IMS Connect interfaces with Open Transaction Manager Access (OTMA) to provide a direct communication path from clients to IMS applications. OTMA is a feature of IMS that enables any z/OS client (IMS Connect in this case) to access IMS applications through the cross-system coupling facility (XCF) services. For additional information about OTMA, see Chapter 2, “Open Transaction Manager Access” on page 7.

Request messages received from TCP/IP clients, using TCP/IP connections or local option clients using the z/OS program call (PC), are passed to IMS through XCF. Then, IMS Connect receives response messages from IMS and passes them back to the originating TCP/IP or local option clients.

IMS Connect supports TCP/IP clients communicating with socket calls, but it can also support any TCP/IP client that communicates with a different input data stream format. User-written message exits can execute in the IMS Connect address space to convert customer message format to OTMA message format before IMS Connect sends the message to IMS. The user-written message exits also convert OTMA message format to customer message format before sending a message back to IMS Connect. IMS Connect then sends the output to the client.

In addition to TCP/IP client communications, IMS Connect also supports local communication through the local option. This option provides a non-socket (non-TCP/IP) communication protocol for use between IBM WebSphere and IMS Connect in the z/OS environment. Servlets that run in IBM WebSphere Application Server for z/OS and use IMS Connector for Java can communicate with IMS Connect through the local option.

Note: As a feature of IMS Connect, we explain the local option in this redbook but we do *not* recommend that you use it. It does not provide better performance than the TCP/IP hyper sockets and it is not going to be further developed in the future.

IMS Connect also supports TCP/IP connections from the DB2 UDB V8.1 Control Center to exchange IMS Control Center commands and responses. A single IMS Connect can support communication between the IMS Control Center and any IMS within the sysplex. This IMSplex support for enhanced commands requires IMS Operations Manager (OM). The interface between an IMS Control Center and OM uses the Structure Call Interface (SCI).

The IMS Connect configuration supports multiple IMS Connects accessing the same IMS system and a single IMS Connect accessing multiple IMS systems. If the datastore goes down, the status of the datastore is sent to IMS Connect from IMS OTMA through XCF. When the datastore is brought back up and restarted, IMS Connect is notified and automatically reconnects to the datastore if it was originally connected to the datastore before the datastore went down. You do not need to manually reconnect to the datastore.

If you stop and restart TCP/IP, IMS Connect does not automatically reconnect. You have to stop and restart IMS Connect and the IMS Connect clients will have to reconnect.

IMS Connect components

IMS Connect consists of 11 core components. These 11 components are:

- ▶ Client communication component (CCC)
CCC processes communication requests between the front-end drivers and the back-end drivers.
- ▶ Command component (CMD)
CMD processes commands received from the z/OS console operator.
- ▶ Datastore communication component (DCC)
DCC processes communication requests between the back-end drivers and the front-end drivers.
- ▶ Environment component (EVC)
EVC provides IMS Connect startup and termination services. The EVC loads IMS Connect modules, tables, and required storage areas; loads and calls the user message exits and user initialization exits; and terminates IMS Connect.
- ▶ IMS Connect Base Primitive Environment (IMS Connect BPE)
IMS Connect BPE provides common system services to all IMS Connect components.
- ▶ IMSplex communications component (ICC)
ICC processes communication requests between the front-end TIDC driver and the back-end IPDC driver.
- ▶ IMSplex driver (IPDC)
IPDC is a back-end driver and enables IMS Connect to communicate with IMS by using the IMS SCI connection.
- ▶ Local option communication component (LOCC)
LOCC processes communication requests between the front-end PCDC driver and the back-end drivers, OTDC and IPDC.

- ▶ Local option driver (PCDC)
PCDC is a front-end driver and provides the mechanism to communicate with clients by using the local option, which is a non-socket communications protocol.
- ▶ OTMA driver (OTDC)
OTDC is a back-end driver and provides the mechanism to communicate with the IMS datastores by using an XCF connection to IMS OTMA.
- ▶ TCP/IP driver (TIDC)
TIDC is a front-end driver and provides the mechanism to communicate with clients by using a TCP/IP socket connection to the clients.

Figure 3-1 on page 35 shows the layout of each IMS Connect component.

Driver components

IMS Connect uses the driver components (TIDC, PCDC, IPDC, and OTDC) to isolate the core components from the communication software. The TCP/IP driver is used to communicate with TCP/IP clients using the TCP/IP communications protocol. The local option driver (PCDC) is used to communicate with the local option clients. The IMS OTMA driver is used to communicate with the datastores (IMSSs) using the IMS OTMA communications protocol. The IMSplex driver is used to communicate with the IMS Operations Manager (OM) using SCI.

Communication between components takes place using the call interface service. The call interface provides the encapsulation and isolation of structures between the components. Each IMS Connect component provides its own set of functions, which it registers with the call interface. When a component requires that a function is performed by another component, the first component calls the call interface using the following parameters:

- ▶ Component name to which the request is to be forwarded
- ▶ Function the component is to perform
- ▶ Parameters required for the function

The call interface uses a function work element (FWE) to carry information between components.

Base Primitive Environment

The IMS Connect Base Primitive Environment (IMS Connect BPE) is a common system service upon which IMS Connect is built. IMS Connect initializes the IMS Connect BPE in the IMS Connect address space. The IMS Connect BPE provides IMS Connect with these services:

- ▶ Environment
- ▶ Storage
- ▶ Serialization
- ▶ Tracing

3.3 A brief history and evolution of IMS Connect

This section provides the background of the evolution of IMS Connect, describing enhancements included in the versions.

3.3.1 ITOC: The predecessor to IMS Connect

Before the IMS Connect product, IMS TCP/IP connectivity was originally provided in the IMS TCP/IP OTMA Connection (ITOC). ITOC was provided free of charge from the IMS home

page and downloaded the relevant files from the Web for installation. Basically, ITOC had similar functionality to IMS Connect, but the ITOC product service has been discontinued on March 1, 2001. ITOC functionality has been enhanced and repackaged as IMS Connect.

3.3.2 IMS Connect Version 1.1

IMS Connect Version 1.1 includes the following features:

- ▶ SMP/E installability
Improved manageability over ITOC with System Modification Program/Extended (SMP/E) install and maintenance.
- ▶ Formatted dump support
Support to format selected IMS Connect control blocks.
- ▶ Persistent sockets
Improved performance over ITOC with persistent socket support for send-then-commit (commit mode 1).
- ▶ Asynchronous output support
Asynchronous output support with IMS Version 7 or later.
- ▶ User initialization exit support
User exit to enable you to perform initialization and termination processing.
- ▶ Send only capability
Allows the flow connect, send (sendonly) and disconnect.
- ▶ Local option
Support for non-TCP/IP communications (program calls) between IMS Connector for Java with WebSphere Application Server and IMS Connect when IMS Connect and WebSphere Application Server reside in the same z/OS image.
- ▶ Unicode support
Support for sending Unicode application data to an IMS host application capable of dealing with Unicode, such as a Java application running in IMS.
- ▶ ACK/NAK required notification support
Provides client notification that an acknowledgement (ACK) or negative acknowledge (NAK) response for a DFS™ message is required by the client.

Note: IMS Connect Version 1.1 has been at end of service since November 30, 2003.

3.3.3 IMS Connect Version 1.2

IMS Connect Version 1.2 provides the following enhancements over IMS Connect Version 1.1:

- ▶ IMS Connector for Java J2EE runtime support for WebSphere access
WebSphere Adapter support with the addition of the IMS Connector for Java J2EE Connector architecture (JCA) runtime support. You can use it with the IMS Connector for Java of VisualAge® for Java and WebSphere Studio Application Developer Integration Edition.

- ▶ Two-phase commit support in local IBM @server zSeries environments
Allows IMS transactions to participate as a resource in two-phase commit (2PC) external transactions. Recoverable Resource Services (RRS) manages the sync point coordination. WebSphere Application Server for z/OS, IMS Connect, RRS, and IMS must reside in the same z/OS image.
- ▶ Security enhancements
Support for PassTicket and Trusted User.
- ▶ Enhanced timer granularity
Provides a greater level of granularity for time out settings. Each client SEND can specify a different value.
- ▶ Exit enhancement
User message exit limitation relief.
- ▶ Auto reconnect to a recycled IMS
Support to automatically reconnect to an IMS that rejoins the XCF group.
- ▶ Internet Protocol V6 support
Support for larger addressing scheme, Internet Protocol (IP) V6.
- ▶ IMSplex support OM distributed interface
Allows IMS Control Center to issue IMS enhancement commands.

Note: IMS Connect Version 1.2 has been at end of service since April 30, 2005.

3.3.4 IMS Connect Version 2.1

IMS Connect Version 2.1 provides the following enhancements over IMS Connect Version 1.2:

- ▶ PING support
Mechanism to determine availability of IMS Connect.
- ▶ J2EE XA two-phase commit support
J2EE Connector architecture XA two-phase commit for distributed and z/OS environments to expand transactional integrity. XA is a two-phase commit protocol defined by the X/Open protocol. IMS Connector for Java is the required resource adapter.
- ▶ SSL support
Support for the Secure Sockets Layer (SSL) encryption and authentication protocol.

3.3.5 IMS Version 9 integrated IMS Connect - IMS Connect Version 2.2

IMS Connect adds the following enhancements to the previous features:

- ▶ Persistent socket support for commit mode 0
Adds persistent socket support for commit mode `commit_then_send` (commit mode 0). Supports the functions *sendonly*, *resume_tpipe*, and *non-conversational* transactions and a mix of *sendonly*, *resume_tpipe*, and *non-conversational* transactions within a single connection.
This feature is available in IMS Connect Version 2.1 through the APAR PQ80468.

- Purge not deliverable support

Mechanism to remove an output message that cannot be delivered to the remote client from the IMS message queue. It adds new IRM_F3_PURGE(X'04').

Applies to IOPCB replies. Does not apply to commit mode 1, *resume_tpipe*, or *sendonly* requests.

This enhancement needs the IMS APAR PQ87088 for IMS Version 9 and PQ87087 for IMS Version 8. It is available for IMS Connect Version 2.1 by APAR PQ87160.

- New *resume_tpipe* single with wait option

Resume_tpipe identifies a request for asynchronous output messages from IMS, the single type means receive one message and disconnect the socket. A new flag, IRM_F5_SWAIT, allows the wait for a single message if none are currently queued in IMS.

It needs the IMS APAR PQ83639 for IMS Version 9, PQ79040 for IMS Version 8, and PQ78912 for IMS Version 7. This is available for IMS Connect Version 2.1 by PQ80468.

- Cancel timer support

Allows the same endpoint device or another endpoint device to cancel *IMR_Timer* or a HWSCFGxx timer value associated with a prior client application *send* request. A device that sends a cancel timer must connect and specify the same client name as the client name for which the cancel timer is being issued.

Adds the new IRM_F4_CANTIMER with a value of C'C'. Return code X'2C' is returned to the requesting client.

This function is available through the APAR PQ96500 for IMS Connect Version 2.2 and PQ96501 for the integrated IMS Connect function of IMS Version 9. It is also available for IMS Connect Version 2.1 with PQ85126.

- Internal reroute of commit mode 0 (CM0) output

When a commit mode 1 (CM1) transaction does a program switch to a second transaction and both transactions issue an insert to the IOPCB, OTMA sends the output for the first transaction on the port Tpipe CM1 and the output for the second transaction on the port Tpipe CM0. IMS holds the second transaction output and IMS Connect cannot retrieve it, even using a *resume_tpipe* call.

This enhancement allows IMS Connect to verify that all CM0 output has a valid *client_id* as the Tpipe name and that it is set correctly. IMS Connect rejects the output with a *reroute* request to the inputting *client_id*. OTMA removes the port Tpipe and re-queues the output on the *client_id* requested by IMS Connect.

This function does not require your involvement.

You need to have installed APAR PQ96425 for IMS Connect Version 2.2, PQ96428 for IMS Connect integrated function IMS Version 9, or PQ86077 for IMS Connect Version 2.1. In addition, you need APAR PQ84254 for IMS Version 8 and PQ80469 for IMS Version 7.

- Support to allow up to 65,535 sockets

This enhancement extends the TCP/IP sockets from 2,000 to 65,535 per IMS Connect. You can allow up to 65,535 sockets to be connected to IMS Connect, specifying the value of the parameter *maxsoc* on the TCP/IP statement.

APAR PQ90051 in IMS Connect Version 2.2 and APAR PQ91578 for IMS Connect integrated function IMS Version 9 provide this function.

- IMS Connect z/OS commands

Support of modify command support:

F jobname,command

The command verbs are query, update, shutdown, and delete.

- ▶ Support for IMS Connect Extensions

Support for a new product, IMS Connect Extensions (5655-K48), which enhances and augments the services of IMS Connect. We describe this product in more detail in Chapter 11, “IMS Connect Extensions” on page 155.

3.4 IMS Connect clients

Using IMS Connect, you can develop applications as:

- ▶ Roll your own (RYO) client
- ▶ IMS Connector for Java client

A RYO client can be written in any language that supports the TCP/IP interface. You need to know the protocol provided by IMS Connect and OTMA to develop this TCP/IP socket application. The advantage of a RYO client compared with IMS Connector for Java client is that the RYO client has similar functionality to and flexibility of the WebSphere MQ IMS bridge client applications, which support the OTMA protocol and the message routing. We provide examples of building the RYO clients in Chapter 14, “Building roll your own clients” on page 265.

IMS Connector for Java development environment is included in IBM WebSphere Application Developer Integrated Edition and in Rational Application Developer Version 6. The IMS Connector for Java client might have a little less flexibility than the RYO client, depending on your requirements. But you can build the IMS Connector for Java client without in-depth knowledge of the protocol of IMS Connect and OTMA, and you can use utilities that aid programmer productivity. We provide information about using the IMS Connector for Java in Chapter 12, “IMS Connector for Java” on page 221.

3.5 IMS Control Center

You can manage your IMS systems using a graphical interface from a workstation using the IMS Control Center. The IMS Control Center uses the IMS single point of control (SPOC) functions. The IMS Control Center is part of the IBM DB2 Universal Database™ Version 8 Control Center. The DB2 Control Center is available with the IBM DB2 Universal Database (DB2 UDB) Administration Client, Version 8.2.

Using the IMS Control Center, you can view the members of the IMSplex and define groups of members. The Control Center supports both IMS Version 8 and later versions. From the Control Center, you can issue IMS type-2 commands using command windows or wizards, depending on how much assurance you want. Wizards help you build and issue the commands. The results are displayed in the IMS Results window. In addition, using the Command Editor, you can issue both type-1 and type-2 IMS commands.

The IMS Control Center uses IMS Connect as a communication vehicle between the client and IMS Operations manager (OM). Before you can use the IMS Control Center, make sure that you have the correct software at the correct levels installed. The software requirements for the IMS Control Center are:

- ▶ For IMS Version 8:
 - APAR PQ69527 to enable IMS Control Center support.
 - IMS Connect Version 1.2 or later with APARs PQ62379 and PQ70216.

- ▶ For IMS Version 9:
 - Integrated IMS Connect function configured on your system with APAR PQ92398 installed.

For configuring IMS Connect for the IMS Control Center, refer to 4.4, “IMS Control Center support” on page 54.

Configuring IMS Connect

IMS Connect can be configured to meet your availability, capacity, security, and performance requirements:

- ▶ One IMS Connect can connect to multiple IMS control regions in multiple XCF groups.
- ▶ One IMS Connect can have multiple connections to the same IMS copy.
- ▶ One IMS Connect connection can have multiple Tpipes.
- ▶ One IMS control region can connect to multiple IMS Connects.
- ▶ IMS Connect and IMS can be on different LPARs in the same sysplex.

This chapter discusses the configuration and customization processes of IMS Connect, including a section dedicated to the new IMS Control Center support. It also provides a Java sample to verify the IMS Connect installation.

4.1 Introduction

IMS Connect runs as an z/OS job or started task and is controlled by two input files:

- ▶ BPECFGxx

Defines parameters for the Base Primitive Environment (BPE). These parameters are used to control common services such as dispatching, waiting, and tracing.

- ▶ HWSCFGxx

Specifies the environment for IMS Connect. This information is used to define the characteristics of the communication between IMS and TCP/IP.

The TCP/IP client application requests a connection by specifying the host DNS name (resolves to the IP address of the target host), the IMS Connect port number, and the target IMS subsystem name. In an IMS Connect environment, it is called datastore ID, which is the name by which the remote application requests a connection to a specific IMS. The datastore ID is the name that identifies a specific statement in the IMS Connect configuration file, which directs the requests to a specific IMS system. In the HWSCFG file, IMS Connect also has the information related to the OTMA XCF group.

Figure 4-1 shows how IMS Connect uses the HWSCFG file to route the messages.

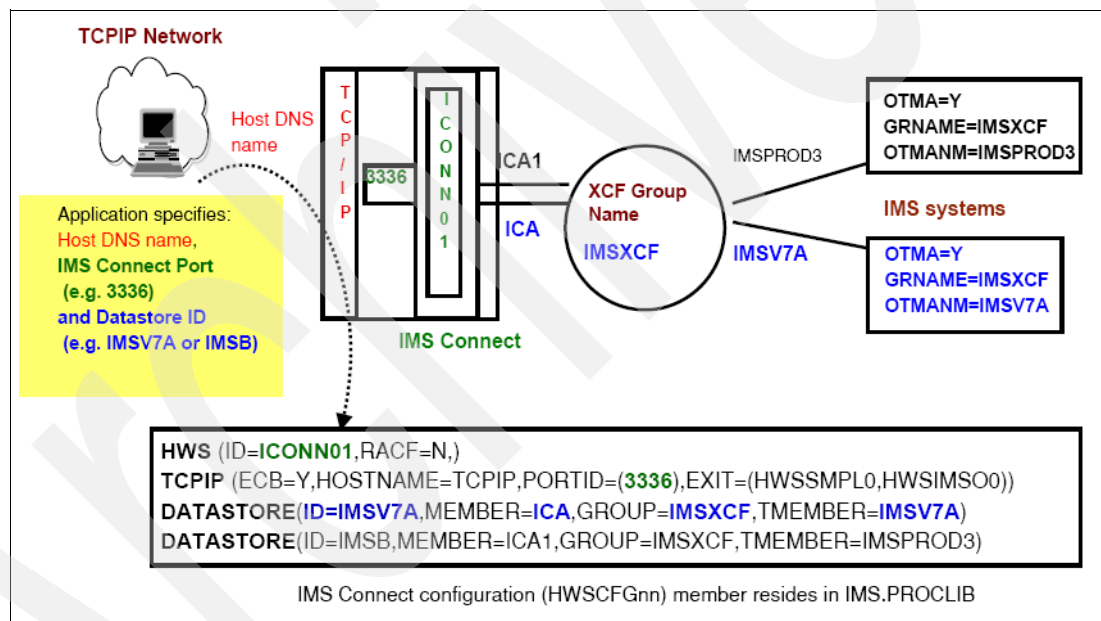


Figure 4-1 IMS configuration HWSCFG member

The other main point of IMS Connect configuration is the installation of the supplied exits and the customization of the user exits. The following sections cover the installation and configuration of IMS Connect.

4.2 Installing IMS Connect

The IMS Connect installation is managed by normal SMP/E RECEIVE, APPLY, and ACCEPT services, and it is the same as other software products that run under a z/OS environment. The IMS Connect software distribution contains sample job control language (JCL) for the installation, which you can modify to meet the requirements of your environment. With IMS

Version 9, the IMS Connect feature is installed as a part of the IMS installation because it comes with the base IMS FMID HMK9900.

If you use IMS Connect Version 2.2, refer to *Program Directory for IBM IMS Connect for z/OS*, GI10-8506, for more information about the IMS Connect installation.

If you use IMS Version 9 integrated IMS Connect, refer *Program Directory for IBM Information Management System Transaction and Database Servers*, GI10-8594, for more information about the IMS Connect installation.

4.3 Configuring IMS Connect

This section describes how to prepare the environment for IMS Connect. To use the information provided in this section, you need a working knowledge of IMS transaction processing, RACF, IMS OTMA, and TCP/IP.

IMS Connect supports communication between one or more TCP/IP clients and IMS systems. IMS Connect uses TCP/IP for communication with clients and IMS OTMA for communication with IMS. It also provides a mechanism to start or stop TCP/IP clients or datastores through the use of commands.

You can configure multiple IMSs on multiple z/OS systems and distribute the client request to the datastores (IMSs). To configure IMS Connect, perform the following actions:

1. Create an IMS Connect job or started task.
2. Authorize the IMS Connect load library with the authorized program facility (APF).
3. Update the program properties table (PPT) in SYS1.PARMLIB. Updating the PPT allows IMS Connect to run in authorized supervisor state and in key 7.
4. Create an IMS Connect configuration member to hold the configuration statements that IMS Connect uses during initialization.
5. Define the IMS Connect security.
6. Install the default user exits into the IMS Connect resident library.

4.3.1 IMS Connect start procedure

You can use the sample JCL in Example 4-1 for the IMS Connect started task procedure. You can also run the IMS Connect as an z/OS job.

Example 4-1 IMS Connect startup JCL

```
//HWS PROC RGN=4096K,SOUT=A,
//      BPECFG=BPECFGHT,
//      HWSCFG=HWSCFG00
// *
//*****
//* BRING UP AN IMS CONNECT *
//*****
//STEP1 EXEC PGM=HWSHWS00,REGION=&RGN,TIME=1440,
//      PARM='BPECFG=&BPECFG,HWSCFG=&HWSCFG'
//STEPLIB DD DSN=HWS.SHWSRESL,DISP=SHR
//      DD DSN=IMS.SDFSRESL,DISP=SHR
//      DD DSN=CEE.SCEERUN,UNIT=SYSDA,DISP=SHR
//      DD DSN=SYS1.CSSLIB,UNIT=SYSDA,DISP=SHR
//      DD DSN=GSK.SGSKLOAD,UNIT=SYSDA,DISP=SHR
//PROCLIB DD DSN=USER.PROCLIB,DISP=SHR
```

```
//SYSPRINT DD SYSOUT=&SOUT
//SYSUDUMP DD SYSOUT=&SOUT
//HWSRCORD DD DSN=HWSRCRD,DISP=SHR
```

If you are using the IMS Connect Version 9 integrated function, the IMS Connect modules reside in the IMS.SDFSRESL library, which is the IMS execution library, and then HWS.SHWSRESL does not exist. If you are using IMS Connect Version 2.2, the IMS Connect modules reside in the HWS.SHWSRESL library, and then the IMS.SDFSRESL library is required only when IMSplex support is used.

IMS Connect requires the CEE.SCEERUN, SYS1.CSSLIB, and GSK.SGSKLOAD libraries (which are the C execution and z/OS system SSL libraries) only when SSL support is used. Many installations that use these libraries place them in the LINKLIST concatenation, in which case, they are not needed in the JCL.

The HWSRCORD data set is the line trace data set. Refer to Chapter 10, “IMS Connect diagnostics” on page 141 for more information.

4.3.2 Authorizing IMS Connect and BPE to the APF

The library in which the IMS Connect modules reside (SDFSRESL or SHWSRESL) must be authorized to the APF. This can be done by editing the appropriate PROGxx member in the SYS1.PARMLIB and issuing the SET PROG=xx command. Coordinate with your z/OS systems programmer to ensure that these libraries are correctly authorized.

4.3.3 Updating the program properties table

Because IMS Connect is executed in supervisor state and key 7, add an entry for it in the z/OS program properties table (PPT) as follows:

1. Add the entry shown in Example 4-2 in the z/OS PPT by editing the SCHEDxx member of the SYS1.PARMLIB data set.

Example 4-2 PPT entry required for IMS Connect

PPT PGMNAME(HWSHWS00)	/*PROGRAM NAME =HWSHWS00 */
CANCEL	/*PROGRAM CAN BE CANCELED */
KEY(7)	/*PROTECT KEY ASSIGNED IS 7 */
SWAP	/*PROGRAM IS SWAPPABLE */
NOPRIV	/*PROGRAM IS NOT PRIVILEGED */
DSI	/*REQUIRES DATA SET INTEGRITY */
PASS	/*CANNOT BYPASS PASSWORD PROTECTION */
SYST	/*PROGRAM IS A SYSTEM TASK */
AFF(NONE)	/*NO CPU AFFINITY */
NOPREF	/*NO PREFERRED STORAGE FRAMES */

Note: Use SWAP only if you are using only TCP/IP communications. If you are using the local option for client communications either by itself or with TCP/IP communications, you have to use NOSWAP.

2. To make the changes effective, do either of the following actions:
 - Re-IPL your z/OS system.
 - Issue the z/OS SET SCH= command.

4.3.4 Creating the IMS Connect configuration member

You have to create a configuration member in your PROCLIB data set for specifying the environment for IMS Connect. IMS Connect uses the information it retrieves from the member to establish communication with IMS and TCP/IP. You can define several configuration members in the partition data set (PDS) to select from during the IMS Connect startup. Specify the member name to be used in the HWSCFG= parameter of the IMS Connect startup JCL (see Example 4-1 on page 45).

IMS Connect configuration statement parameters

You have to specify values for some of the parameters that define the way in which IMS Connect is to communicate with TCP/IP and IMS OTMA in the IMS Connect configuration member. The IMS Connect configuration member contains four types of configuration statements: HWS, TCPIP, DATASTORE, and IMSPLEX. Descriptions of the configuration statements are as follows:

► HWS

The HWS statement specifies one IMS Connect. It includes the following keyword parameters:

– ID

The IMS Connect name. It consists of alphanumeric character data, begins with an alphabetic character, and has a length between 1 and 8 characters.

– RACF

Y (yes), or N (no). Determines whether or not the password and user ID (provided by either the client application or a user exit routine) are passed to RACF for authentication.

This setting can also be changed using the IMS Connect SETRACF command.

If this is set to N, no RACF validation of the user ID is done, and the user ID (if provided) is simply passed to IMS. If this is set to Y, IMS Connect calls RACF to validate the user ID and password combination before passing the request through to IMS. N is the default.

Even if the value is Y, the IMS Connect user exit can set the trusted user flag and tell IMS Connect not to issue the RACF call.

Important: Performance problems might occur if RACF statistics are kept for a user ID.

– RRS

Y (yes), or N (no). Defines if RRS should be enabled. This enables two-phase commit. N is the default.

– XIBAREA

Specifies the number of full words allocated for the XIB user area. Both the user initialization exit routine and the user message exit routines can access and modify the XIB user area.

The default value is 20; the maximum value is 500. If you do not specify value for this parameter, or you specify a value outside of the 20 to 500 range, the system uses the default value of 20.

► TCPIP

The TCP/IP statement defines to IMS Connect the communication with one TCP/IP. Its keyword parameters are as follows:

– HOSTNAME

Name of the TCP/IP host. This is usually the jobname of the TCP/IP z/OS address space.

– ECB

Y (yes), or N (no). Specifies whether to use the TCP/IP exit or event control block (ECB) processing. ECB processing enhances IMS Connect performance by increasing throughput.

When ECB=N is specified (or left blank), IMS Connect executes with TCP/IP driving an IMS Connect exit.

When ECB=Y is specified, IMS Connect executes with TCP/IP driving IMS Connect with the posting of an ECB.

Recommendation: Set ECB=Y for the best performance.

– EXIT

Name of the TCP/IP user message exit that receives control for messages received from and sent to TCP/IP clients. More than one exit can be defined as EXIT=(EZAEXIT,EZBEXIT,EZCEXIT) to a maximum of 254.

This includes the IMS Connect sample exits and the user-written exits.

These user message exits support users other than IMS Connector for Java for OTMA linkage through IMS Connect to IMS. You do not have to include HWSJAVA0 in the EXIT= list. IMS Connect automatically loads the HWSJAVA0 exit, which is shipped with IMS Connect to enable IMS to support the IMS Connector for Java applications.

The user message exits for IMSplex support are HWSCSLO0 and HWSCSLO1 and must be specified here to ensure the activation of the IMS Control Center.

HWSUINIT is an IMS Connect exit but is not a user message exit; you must not add it to the EXIT= parameter. If you add HWSUINIT to the EXIT= parameter, IMS Connect will abend.

– IPV6

Y (yes), or N (no). At IMS Connect startup time, determines whether or not Internet Protocol Version 6 (IPV6) is enabled.

When N is specified or defaulted, IPV4 is used.

If you use IPV6, z/OS Version 1.4 or later is required.

Recommendation: If you are on z/OS Version 1.4 or later, set IPV6=Y even if you are not on IPV6 for better performance.

– MAXSOC

A decimal field set to the maximum number of sockets per IMS Connect that an IMS Connect can open. This value must be a number between 50 and 65535. The default value is 50.

The value specified will result in one socket dedicated to a listen state per port and the remainder available for connections. Therefore, if you specify 80 and have five ports, 75 physical connections can be made. The other five are used for the listen state sockets.

At this point, you have to consider that there are two parameters in the BPXPRMxx member of the SYS1.PROCLIB related to the allowed number of sockets.

MAXSOCKETS sets a limit for the total number on sockets in a system, and MAXFILEPROC sets the maximum number of sockets for any job.

- PORTID

A 1-character to 8-character field to define the TCP/IP ports, or a 5-character field with the value of LOCAL to define the local option connection. For TCP/IP port communications, it specifies the port number or numbers that will bind to the socket (port for IMS Connect on which to listen).

You can define more than one port as PORTID=(9999,8888,7777) to a maximum of 50. Port numbers must be within the range 1 to 65535 and must be selected so that they do not conflict with other ports in the TCP/IP domain.

- RACFID

Default RACF ID for exits. Exits pass this ID to OTMA for security checking if the RACFID has not explicitly been set in the incoming message or by the user exit.

- SSLENVAR

The member name of the SSL initialization file.

- SSLPORT

A 1-numeric character to 8-numeric character decimal field to define Secure Sockets Layer (SSL) ports. For SSL port communication, it specifies the port number that will bind to the socket (port for IMS Connect on which to listen with SSL).

You can define up to 50 ports, which must be numbered within the range of 1 to 65535. These ports must not conflict with any other ports selected in the TCP/IP domain or those selected under the PORTID parameter as basic TCP/IP ports.

Important: At the time of writing this book, there is an open issue with IMS Connect architecture and only *one* SSL port is currently supported. APAR PQ90146 is opened to resolve this problem.

- TIMEOUT

Time interval in hundredths of seconds after which IMS Connect disconnects the client if there is no response from IMS. The maximum value of the timeout is 2147483647 (X'7FFFFFFF') and the default is 0 (which means no timeout).

IMS Connect uses the timeout value to determine the amount of time to wait for a response from IMS that is being sent to the client.

IMS Connect also uses this timeout to disconnect a client socket connection that does not send messages. This timeout value on an IMS Connect read of the client only applies to the wait time between the socket connection and the first input from the client application. The timeout function is not activated between reads but only between the connection and the first IMS Connect read of the client application input.

The client sets a second timeout value in the IMS Request Message (IRM) header field *irm_timer* for use with a *read* to OTMA following a **resume_tpipe** command and the ACK following the *read* for a *resume_tpipe*.

► **DATASTORE**

The DATASTORE statement specifies each datastore with which the IMS Connect communicates through IMS OTMA. You can define multiple DATASTORE cards; each one is an OTMA client.

IMS Connect uses this information to translate the logical datastore name passed by the TCP/IP client into the IMS XCF member name and thus the IMS control region.

The DATASTORE statement keyword parameters are:

– **ID**

The datastore name.

This consists of alphanumeric character data, begins with an alphabetic character, and has a length between 1 and 8 bytes. This ID name cannot be the same as the *tmember* on the IMSPLEX statement.

The IMS Connect client passes to IMS Connect a datastore ID to identify the IMS to receive the message. The datastore ID supplied by the client must match with a datastore name defined to IMS Connect.

For IMS Connector for Java clients, this ID must match the name that is specified in the IMS InteractionSpec for IMS Connector for Java. For non-IMS Connector for Java clients, the ID must match the datastore ID that is placed in the IMS Request Message (IRM) that is sent to IMS Connect.

The IMS Connect user exit can override the datastore ID supplied by the client.

– **GROUP**

The XCF group name for the IMS OTMA. IMS Connect uses this value to join the appropriate XCF groups. This group name must match the XCF group name that you define to the GRNAME in the IMS startup JCL or DFSPBxxx member, because IMS Connect and IMS must be in the same XCF group in order to communicate. Each IMS Connect can join any number of groups.

– **MEMBER**

The XCF member name that identifies IMS Connect in the XCF group specified by the group parameter. This name is the XCF name that IMS uses to communicate with IMS Connect in that XCF group. This XCF member name for IMS Connect must be unique in the datastore definitions for all datastores that are members of the same XCF group.

– **TMEMBER**

The XCF member name for IMS that IMS Connect uses in order to communicate with an IMS in its XCF group. This target member name must match the member name IMS uses when it joins the XCF group. The XCF member name for IMS is specified in the IMS startup JCL by the OTMANM parameter in the IMS startup JCL or DFSPBxxx member. Each datastore definition within an IMS Connect configuration member must contain a unique *tmember* name.

– **RRNAME**

The name of an alternate destination specified in a client reroute request. If this string is not provided, IMS Connect uses HWS\$DEF as the default name.

It must be a string of 1 to 8 uppercase alphanumeric (A through Z, 0 to 9) or special characters (@, #, \$), left-aligned, and padded with blanks. IMS Connect translates lowercase characters to uppercase characters.

The string is terminated by any blank or invalid character. The reroute name is truncated at any invalid character.

- APPL

TCP/IP APPL name defined to RACF in the PTKTDATA statement.

This parameter is optional and will default to blanks. If you are using PassTicket and user message exits, HWSIMSO0, HWSIMSO1, or both, you must specify the APPL on the DATASTORE statement.

- DRU

A 1 to 8 alphanumeric character field. The DRU keyword enables you to specify your own OTMA destination resolution user exit name that is to be passed to OTMA. The DRU exit is required to support asynchronous output to IMS Connect clients. The default is DFSYDRU0, but you can write your own exit.

IMS Connect also provides the sample DRU exit routine (HWSYDRU0).

- IMSPLEX

The IMSPLEX statement specifies each IMS Operations Manager (OM) with which IMS Connect communicates through the IMS Structure Call Interface (SCI). You can have multiple IMSPLEX statements. It is needed if you use the IMS Command Center.

The IMSPLEX statement keyword parameters are as follows:

- MEMBER

This name is passed to the SCI as the name of the IMS Connect that is communicating with the IMS OM through the SCI.

- TMEMBER

Name of the SCI to which IMS Connect communicates. The tmember name consists of alphanumeric character data, begins with an alphabetic character, and has a length between 1 and 5 bytes.

This name must be equal to the name specified in the SCI initialization PROCLIB member IMSPLEX in the parameter NAME.

The tmember name cannot be the same name as the ID name on the DATASTORE statement.

- RUNOPTS

A 1 to 255 character string field that specifies the Language Environment® runtime options to be used to override the IMS Connect default runtime options in support of SSL.

This parameter is optional. It is applicable only to the Language Environment environment for SSL support. IMS Connect passes the default values POSIX(ON),TRAP(OFF,NOSPIE), unless overridden by the RUNOPTS parameter.

Example 4-2 on page 52 shows an example of IMS Connect configuration file for a simple configuration.

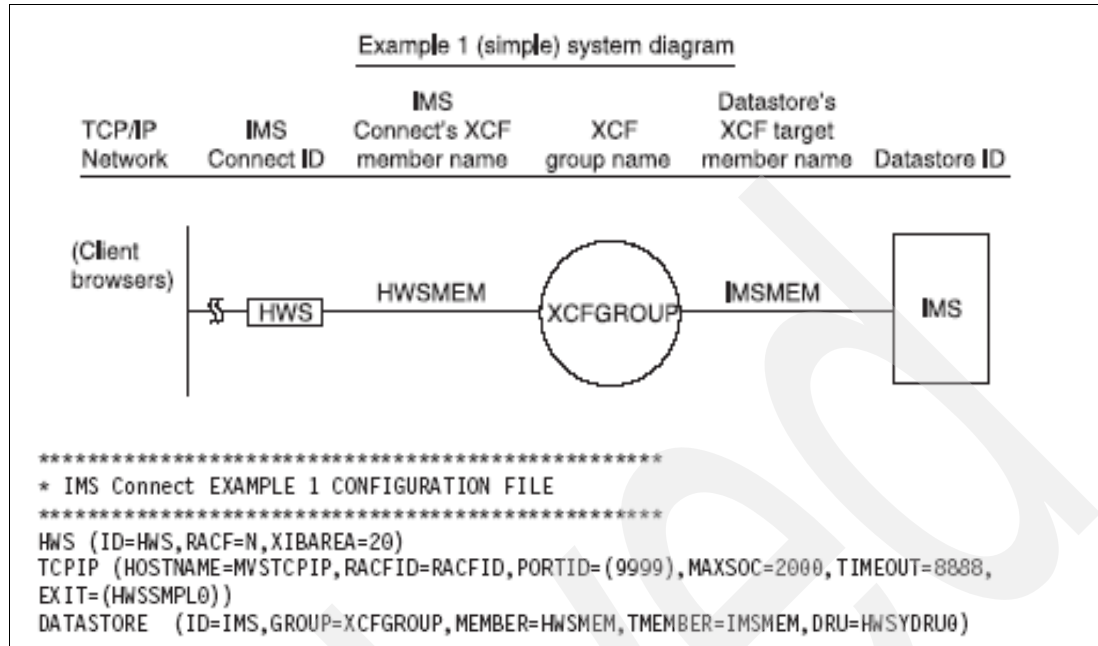


Figure 4-2 Example of a simple configuration HWSCFG member

In this IMS Connect configuration member, the IMS Connect ID is defined as HWS. This IMS Connect is configured to include the ports defined for TCP/IP communications and the IMS Connect group and member names for communication with IMS.

The TCP/IP configuration defines the HOSTNAME as MVSTCPIP, the RACFID as RACFID, the PORTID as 9999, and the EXIT as HWSSMPL0.

The datastore configuration defines the ID as IMS, the GROUP as XCFCGROUP, the MEMBER as HWSMEM, and the TMEMBER as IMSMEM.

IMS Connect Base Primitive Environment configuration

The IMS Connect address space is built on top of the IMS Connect Base Primitive Environment (BPE). Generally, you do not need to work with the IMS Connect BPE. However, you might need to change the default settings for certain IMS Connect BPE functions, such as storage management, internal tracing, dispatching, and other system service functions. IMS Connect supplies a configuration data set member for IMS Connect BPE system service functions that you can modify.

The IMS Connect BPE configuration parameter PROCLIB member defines the IMS Connect BPE execution environment settings for the IMS Connect address space. You specify the PROCLIB member name by coding BPECFG=*member name* on the EXEC PARM= statement in the IMS Connect address space startup JCL (see the IMS Connect startup JCL in Example 4-1 on page 45). Example 4-3 on page 53 shows a sample IMS Connect BPE configuration file. For more information about the BPE environment configuration member and the parameters, refer to *IMS Version 9: IMS Connect Guide and Reference*, SC18-9287.

Example 4-3 Example of a configuration file for IMS Connect BPE

```
*****
*               CONFIGURATION FILE FOR IMS CONNECT BPE               *
*****
LANG=ENU                      /*LANGUAGE FOR MESSAGES */
                               /*(ENU =U.S.ENGLISH)*/

#
#DEFINITIONS FOR IMS CONNECT BPE SYSTEM TRACES
#

TRCLEV=(*,LOW,BPE)            /*DEFAULT TRACES TO LOW */
TRCLEV=(AWE,HIGH,BPE)         /*AWE SERVER TRACE ON HIGH */
TRCLEV=(CBS,MEDIUM,BPE)      /*CTRL BLK SRVCS TRC ON MED */
TRCLEV=(DISP,HIGH,BPE,PAGES=12) /*DISPATCHER TRACE ON HIGH */
                               /*WITH 12 PAGES (48K BYTES)*/

#
#DEFINITIONS FOR IMS CONNECT TRACES
#

TRCLEV=(*,HIGH,HWS)           /*DEFAULT ALL IMS CONNECT TRACES TO HIGH */
TRCLEV=(HWSI,HIGH,HWS)        /*BUT RUN IMS CONNECT TO IMS OTMA TRACE...*/
TRCLEV=(HWSW,HIGH,HWS)        /*AND SERVER TO IMS CONNECT TRACE AT MEDIUM */
```

4.3.5 Defining IMS Connect security

You can start IMS Connect as a job or as a procedure. If IMS is RACF protected, you have to start IMS Connect as a job with the JOB card specifying a valid *user ID* in order to make the connection from IMS Connect to IMS, or you can use the RACF started procedure table.

You use the USERID=*user ID* parameter specified in JOB card of the IMS Connect job JCL as the security vehicle to ensure IMS Connect access to IMS. The user ID must have READ access to IMSXCF.group.member. IMS OTMA provides security for the IMS XCF connection by defining and permitting IMSXCF.group.member in the RACF FACILITY class. For more information about security, see Chapter 2, “Open Transaction Manager Access” on page 7 and Chapter 8, “IMS Connect security” on page 109.

4.3.6 Installing the default user exits into IMS Connect resource library

You have to install the following two exits into your IMS Connect resource library (SHWSRESL) regardless of whether you intend to customize them, because IMS Connect automatically loads these exits when it executes:

- ▶ HWSJAVA0
User message exit for IMS Connector for Java clients
- ▶ HWSUINIT
User initialization exit

You must compile and bind these exits before you execute IMS Connect; otherwise, IMS Connect will not run. If you do not need to customize either of these two exits, you do not need to do anything else with them.

Example 4-4 shows a sample JCL to bind these exits. It uses the following data sets and member names:

- ▶ IMSHWS.SHWSMAC
IMS Connect macro library. If you have IMS Version 9, use the IMS.SDFSMAc library.
- ▶ IMSHWS.SHWSSRC
IMS Connect source library. If you have IMS Version 9, use the IMS.SDFSsRC library.
- ▶ IMSHWS.HWSRESL
IMS Connect resource library. If you have IMS Version 9, use the IMS.SDFSRESL library.
- ▶ Exit name
User exit routine name (HWSJAVA0 or HWSUINIT).

Example 4-4 Sample JCL to install the default user exits

```
//HWSEXIT JOB (ACTINF01),'PGMRNAME',
//          CLASS=A,MSGCLASS=Z,MSGLEVEL=(1,1),REGION=4M
//EXIT01 EXEC PGM=ASMA90,REGION=32M,
//          PARM='DECK,NOOBJECT,SIZE(MAX,ABOVE)'
//SYSLIB DD DSN=SYS1.SDFSMAc,DISP=SHR
//          DD DSN=SYS1.MODGEN,DISP=SHR
//          DD DSN=IMSHWS.SHWSMAC,DISP=SHR
//SYSPUNCH DD UNIT=SYSVIO,DISP=(,PASS),SPACE=(TRK,(1,1,1)),
//          DSN=&&TEXT(exitname)
//SYSPRINT DD SYSOUT=*,
//          DCB=(BLKSIZE=605),
//          SPACE=(605,(100,50),RLSE,,ROUND)
//SYSUT1 DD UNIT=SYSDA,DISP=(,DELETE),
//          DCB=BLKSIZE=13024,
//          SPACE=(CYL,(16,15))
//SYSIN DD DSN=IMSHWS.SHWSSRC(exitname),DISP=SHR
//EXIT02 EXEC PGM=IEWL,
//          PARM='SIZE=(880K,64K),RENT,REFR,NCAL,LET,XREF,LIST,TEST'
//SYSPRINT DD SYSOUT=A
//SYSLMOD DD DSN=IMSHWS.HWSRESL,DISP=SHR
//SYSUT1 DD UNIT=SYSVIO,DISP=(,DELETE),SPACE=(CYL,(10,1),RLSE)
//TEXT DD UNIT=SYSVIO,DISP=(OLD,DELETE),DSN=&&TEXT
//SYSLIN DD *
//          INCLUDE TEXT(exitname)
//          ENTRY exitname
//          NAME exitname(R)
//
```

When you install IMS Connect, the only exits installed to your system are the HWSIMSO0, HWSIMSO1, HWSCSLO0, and HWSCSLO1 user message exits. To customize any of the other exits, modify the exit and then install it into your IMS Connect resource library. Refer to Chapter 9, “IMS Connect user exit support” on page 115 for more information about exits.

4.4 IMS Control Center support

This section describes the steps needed to use the new IMS Control Center support.

4.4.1 IMS Connect configuration for IMS Control Center support

To customize IMS Connect for IMSplex support, perform the following actions:

1. Modify the IMS Connect JCL startup.

If you use IMS Connect Version 2.2, you need to have the IMS.SDFSRESL library in the STEPLIB. If you use the IMS Version 9 IMS Connect integrated function, it is already in the STEPLIB. Example 4-1 on page 45 gives an example of a JCL startup.

2. Customize the IMS Connect configuration member.

You have to modify your HWSCFGxx member to add these new parameters:

- In the parameter EXIT of the TCPIP statement, add the new user message exits for IMSPLEX support, HWSCSLO0, and HWSCSLO1.
- The new IMSPLEX statement specifies each IMSPLEX that IMS Connect communicates with through SCI. This statement defines the access to IMS OM.

Example 4-5 shows an IMS Connect configuration member with IMSPLEX support. IMS Connect in this example connects to IMS Operations Manager (OM) using IMSPLEXG as the member name and PLEXG as the SCI name. PLEXG is the name specified in the SCI initialization PROCLIB member.

Example 4-5 HWSCFG member with IMSPLEX support

```
HWS (ID=MSGCONN,RACF=N,XIBAREA=20)
TCPIP (HOSTNAME=TCPIP,PORTID=(7003,LOCAL),MAXSOC=2000,TIMEOUT=8800,
EXIT=(HWSSMPLO,HWSCSLO0,HWSCSLO1),IPV6=Y)
DATASTORE (ID=MSG,GROUP=IMS9EXCF,MEMBER=HWS910G,TMEMBER=SCSIM9G)
IMSPLEX (MEMBER=IMSPLEXG,TMEMBER=PLEXG)
```

4.4.2 IMS Control Center configuration

In your IMS Control Center, define a new system with the parameters of your IMS Connect. Figure 4-3 shows how to configure an IMS system to connect to IMSG using the IMS Connect of Example 4-5.

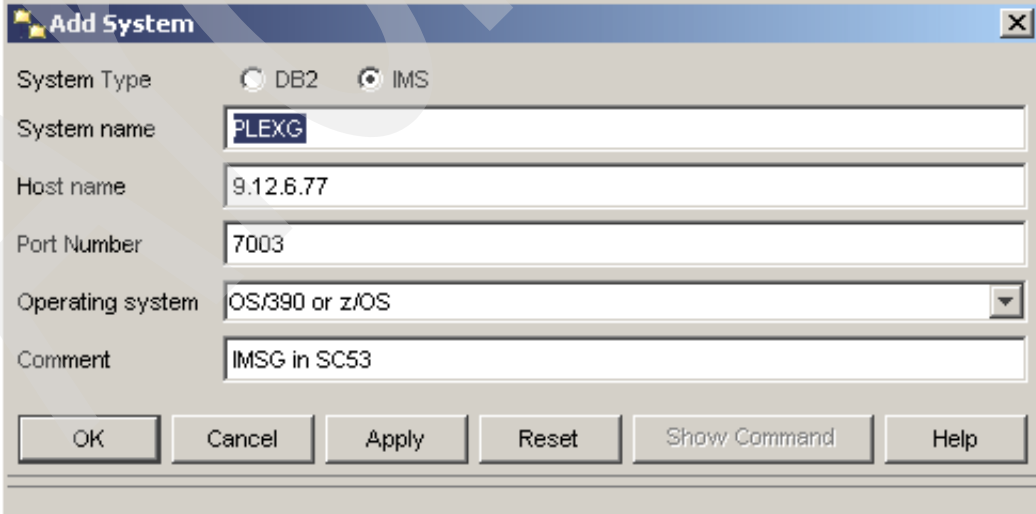


Figure 4-3 Configuring the IMS system in the IMS Control Center

After these steps, you can issue commands to IMSG through the IMS Control Center. Figure 4-4 on page 56 shows the IMS system defined in the previous example, IMSG in PLEXG.

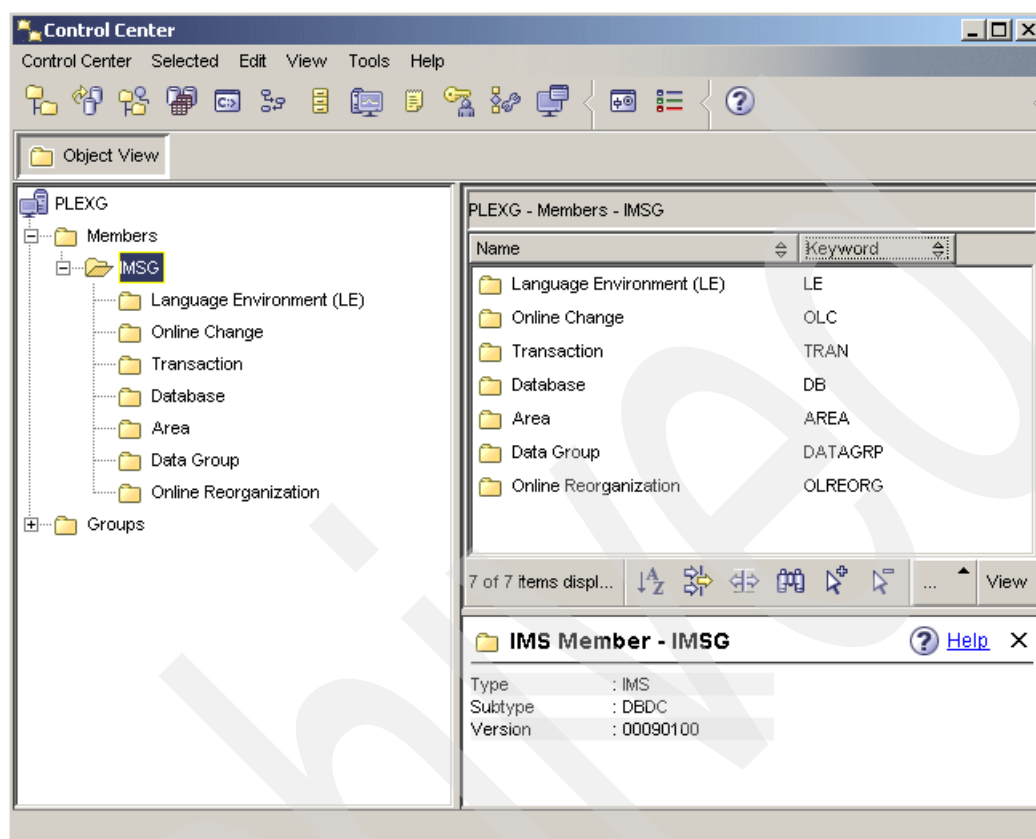


Figure 4-4 IMS system defined in the IMS Control Center

4.5 Confirming IMS Connect install with the sample Java client

You can download the IMS Client for Java program from the following IBM Redbook Web site (see Appendix C, “Additional material” on page 507):

<ftp://www.redbooks.ibm.com/redbooks/SG246794>

You can also download sample client code from the IMS home page and then go to IMS Connect:

<http://www.ibm.com/ims>

The program is shipped with IMS TCP/IP OTMA Connection (ITOC) through the Web, and its purpose is installation verification by sending a transaction to the IMS through ITOC and receiving the message that was returned from the IMS application. This program is designed to use the HWSSMPL0 user exit; therefore, if you install the HWSSMPL0 to IMS Connect, you can also use it for IMS Connect installation verification. However, this program does not support any new functions of IMS Connect (for example, persistent socket connection and asynchronous output support). If you want to verify these functions, you must modify the sample program. Refer to Chapter 7, “IMS Connect programming model” on page 91 for more information about the client programming.

The following steps identify what you need to install and implement to use the Java sample program to confirm your IMS Connect operations.

Installing the IMS Connect sample Java program

This IMS client for Java sample program can be installed on any platform where a Sun™-compatible Java Virtual Machine has been installed. Perform the following steps:

1. Move the JAVASAMP.exe file from the current directory (the directory from where it was extracted) to a directory on an OS/2® or Microsoft® Windows® system where the sample client will be installed or to another temporary directory.

Note: This directory (to which you move the JAVASAMP.exe file) must be located on an OS/2 or Windows drive that supports the long file names used for the Java files.

Optionally, you can enter **JAVASAMP[.exe] -t** at an OS/2 or Windows command prompt. This causes an integrity check of the JAVASAMP.exe ZIP file to execute without extracting any of the files from the ZIP file. If the IMS client for Java sample program needs to be installed on a platform other than on the one where it was expanded, copy the files for it to the platform where the sample program will be installed.

Expand the file JAVASAMP.exe. The expanded files will be placed in the current directory where JAVASAMP.exe is executed.

2. Modify the source code to match your environment.

You must modify the FrameInput.java file to construct input data that matches your environment (host name, port number, transaction, and so on). You need to consider changing the following statements:

- Port IDs: The default files come with port IDs 9999 and 9998. You need to change all occurrences of these in the file to valid port IDs and possibly add options of more (if you require more than two) by adding extra entries, similar to those found in Example 4-6.

Example 4-6 Java sample: FrameInput.java changes for port number

```
groupPort = new CheckboxGroup();
radioButtonPort1 = new Checkbox("9999", groupPort, true);
radioButtonPort1.setBounds(getInsets().left + 312, getInsets().top + 36, 100, 40);
add(radioButtonPort1);

radioButtonPort2 = new Checkbox("9998", groupPort, false);
radioButtonPort2.setBounds(getInsets().left + 312, getInsets().top + 66, 100, 40);
add(radioButtonPort2);
```

- RACFUser: The default list of user IDs come as USER01-4, and the select(0) statement, indicating the first one is brought up by default. You need modify this to reflect valid user IDs for use. Refer to Example 4-7 for an example.

Example 4-7 Java sample FrameInput.java changes for RACF user ID

```
choiceSAFID = new Choice();
choiceSAFID.addItem("SAFUsrID");
choiceSAFID.addItem("USER01");
choiceSAFID.addItem("USER02");
choiceSAFID.addItem("USER03");
choiceSAFID.addItem("USER04");
```

- Similar changes to those identified for the RACFUser are also needed for Tran, DS (datastore ID), GRP (RACF user ID group), Client name (clientID), and HostName (IP address).

The HWSSMPL0 program does not impose any limitations on the number of input and output message segments. The Java client uses multisegment input and output text areas. IMS Connect requires that all active clients have a unique client ID that, for the IMS Client for Java, are taken from the clientID field defined in the FrameInput.java file. Therefore, if you intend to allow multiple IMS Clients for Java to run simultaneously, which is usually the case, you must either modify the FrameInput.java file so that the clientID will be unique for each active client at any given time, or ensure in some other way that the clientID for each active client is unique. For test purposes, be sure that you use a unique clientID for each Java client when you click **Submit**.

3. Go to the command prompt and the directory containing all these expanded Java client source files for the sample program. Enter the **javac *.java** command.

This creates the IMS Client for Java class files in the same directory.

Install the HWSSMPL0 user exit to your IMS Connect resource library. Compile and link-edit HWSSMPL0, and then modify the IMS Connect configuration file to include the HWSSMPL0 user exit in the TCP/IP statement as follows:

```
TCP/IP=(...,EXIT=(HWSSMPL0,...),...)
```

After the IMS Client for Java sample program source code has been compiled, you can run the IMS Client for Java program as a stand-alone Java application by entering the following command at a command line prompt:

```
java ClientLauncher
```

You can also run the IMS Client for Java sample program as an applet, either locally or remotely. If you want to run the applet as a local applet, open ClientLauncher.html as a local file from a Java-enabled Web browser, which allows an applet to execute socket calls.

If you want to run the applet as a remote applet, copy all .class, .gif, and .html files to your Web server's HTML directory and open the Client Launcher (ClientLauncher.html) in a Java-enabled Web browser.

You then receive the panel shown in Figure 4-5 on page 59.

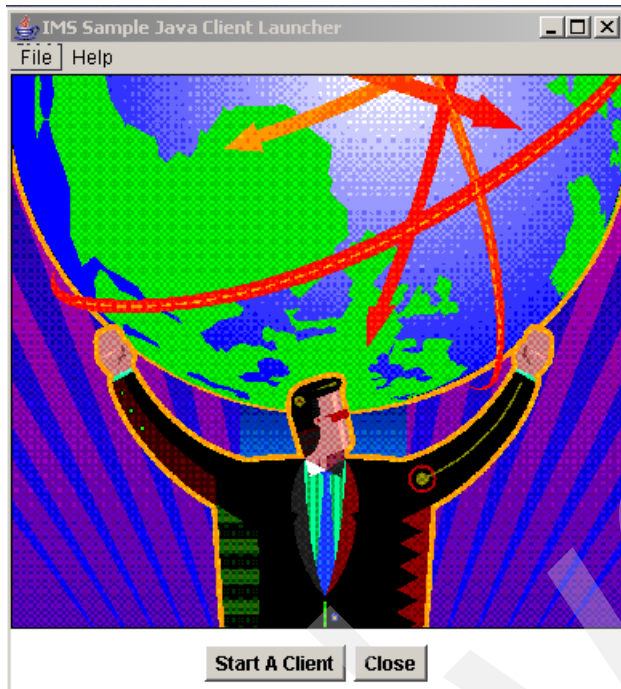


Figure 4-5 Sample Java Client Launcher window

The Client Launcher starts a new transaction input client frame (see Figure 4-6 on page 60) every time you click the **Start A Client** button. If you start more than one client window, the newer windows hide the existing client window. All of these client frame windows run independently, so you must close each window individually if you are running the IMS Client for Java as an applet. If you are running the IMS Client for Java as a Java application, you can either close the client frame windows individually (as you must do if IMS Client for Java is running as an applet) or all at the same time by closing the original window (the Client Launcher running as an application). When you close the Client Launcher application, all client frame windows that were generated by that application close automatically.

The screenshot shows a window titled "Transaction Data" with two main sections: "INPUT DATA" and "OUTPUT DATA".

INPUT DATA:

- HostName: 9.12.6.77
- Transaction: PART
- Client ID: (empty)
- SAFID: (empty)
- Sync Level: NONE
- Password: (empty)
- Input Text: PART 3003806
- Port: ☒ 7003, ☐ 7002
- DataStore ID: IMSG
- GROUP: (empty)
- Commit Mode: 1 - SEND THEN COMMIT

OUTPUT DATA:

- TRAN: PART
- MOD: (empty)
- RC = (empty)
- RS = (empty)
- OUTPUT TEXT:


```
Part..... 3003806; Desc..... SWITCH
Proc Code..... 74; Inv Code..... 2
Make Dept..... 12-00; Plan Rev Num...
Make Time..... 63; Comm Code..... 5
*CSMOKY*
```

Buttons at the bottom: Send, Receive, Ack, Nack, Resume Tpipe, Send Only, Disconnect, Close.

Figure 4-6 Java transaction input client frame

The IMS Client for Java program provides pull-down lists and a text area for supplying the information for a transaction request message that will be used to contract a message to be sent to IMS Connect. The values associated with the list items can be changed. The functions of the eight buttons at the bottom of the IMS Client for Java window are:

- ▶ **Send**
Connects to IMS Connect if the TCP/IP socket connection does not exist, and sends a transaction request message.
- ▶ **Receive**
Requests output from the current transaction request.
- ▶ **Ack**
Sends a message acknowledging (accepting) the transaction output that was just received.
- ▶ **Nack**
Sends a message rejecting the transaction output that was just received, and then automatically receives and displays the abnormal termination message from IMS Connect.
- ▶ **Resume Tpipe**
Reconnects to IMS Connect to receive asynchronous output only.
- ▶ **Send Only**
Connects to IMS Connect if the TCP/IP socket connection does not exist, and sends a message containing a non-response transaction request to IMS Connect.
- ▶ **Disconnect**
Sends requests to deallocate TCP/IP socket connection to IMS Connect, and then automatically receives and displays abnormal termination messages from IMS Connect.

► Close

Closes the IMS Client for the Java program window.

The IMS Client for Java program is intended to enable users to manually execute simple transactions such as those that are part of the IMS INSTALL/IVP sample application. The program is designed to enable you to manually start multiple clients if wanted. However, when using multiple clients simultaneously, you must ensure that each client uses a unique clientID. Otherwise, if you try to send messages to IMS from two clients, both of which use the same clientID, you will get a duplicate client error for the second client, assuming that the first client is still active when the second client attempts to connect (sends a message) to IMS Connect.

To run the PART transaction, type the transaction code and data in the *input text* window. Figure 4-6 on page 60 shows the input for a PART transaction and the output received from IMS.

Note: The IMS Client for Java program is sample code and is not intended to represent a robust program suitable for production environments. As a result, it is relatively easy to put the IMS Client for Java program into states where, for example, it cannot continue a conversation. Another specific example is when a conversation using one of the IMS INSTALL/IVP transactions has ended, and the sample program does not have the ability to recognize that the conversation has ended. As a result, it attempts to process the Send button event as the next iteration of the already ended conversation. Because the conversation has ended, a Java exception is thrown. In this case, and in other similar cases, it is usually sufficient to click the **Disconnect** button and start over. In other cases, it might be necessary to close the client that is not responding correctly and start a new client in its place.

Archived

IMS Connect operations

You can interact with IMS Connect using two types of commands:

- ▶ Commands entered as system replies against the IMS Connect started task
- ▶ Commands entered as MODIFY instructions against the IMS Connect started task (new with IMS Connect Version 2.2 and the integrated IMS Connect function of IMS Version 9)

Reply and modify command families are functionally equivalent. You can choose the set that is more appropriate for your environment and needs. In addition, you can also issue IMS commands and BPE commands related to the functions of IMS Connect. In this chapter, we cover all these types of commands.

5.1 IMS Connect REPLY commands

You can issue the following commands using a REPLY against the IMS Connect started task:

- ▶ CLOSEHWS
- ▶ OPENDS or STARTDS
- ▶ OPENIP or STARTIP
- ▶ OPENPORT or STARTPT
- ▶ RECORDER
- ▶ SETRACF
- ▶ SETRRS
- ▶ STOPCLNT
- ▶ STOPDS
- ▶ STOPIP
- ▶ STOPPORT
- ▶ VIEWDS
- ▶ VIEWHWS
- ▶ VIEWIP
- ▶ VIEWPORT
- ▶ VIEWUOR

All of these IMS Connect commands must be immediately preceded on the command line of the z/OS system console according to the reply number of the outstanding IMS Connect reply message. *IMS Version 9: IMS Connect Guide and Reference*, SC18-9287, fully documents these commands.

The following sections list all the commands with their parameters and an example of the output from each. All the following examples are based on the IMS Connect configuration file in Example 5-1.

Example 5-1 Example of IMS Connect configuration file

```
HWS (ID=IMSGCONN,RACF=N,XIBAREA=20)
TCPIP (HOSTNAME=TCPIP,PORTID=(7003,LOCAL),MAXSOC=2000,TIMEOUT=8800,
EXIT=(HWSSMPLO,HWSCSL00,HWSCSL01),IPV6=Y)
DATASTORE (ID=IMSG,GROUP=IMS9EXCF,MEMBER=HWS910G,TMEMBER=SCSIM9G)
```

5.1.1 CLOSEHWS

The CLOSEHWS command terminates IMS Connect. The parameters for this command are:

- ▶ QUIESCE

Specifies that termination is to end all client and datastore connections in a controlled manner. If no parameter is specified for CLOSEHWS, this parameter is used by default.

All work that is currently in progress, or that is queued for processing, is completed before IMS Connect is terminated.

- ▶ FORCE

Specifies that termination is to end all client and datastore connections immediately, which forces any IMS applications that are executing for the connected clients to abnormally terminate.

Example 5-2 on page 65 shows the output of this command.

Example 5-2 An example of the CLOSEHWS command

```
R074,CLOSEHWS
*075 HWSC0000I *IMS CONNECT READY*  MSGCONN
HWSS0770I LISTENING ON PORT=LOCAL    TERMINATED; M=PSCH
HWSP1415I TCP/IP SOCKET FUNCTION CALL FAILED; F=ACCEPT4 , R=-1, E=1152, M=SDCO
HWSS0770I LISTENING ON PORT=7003     TERMINATED; M=SSCH
HWSS0781I TCPIP COMMUNICATION FUNCTION CLOSED; M=SOCC
HWSD0260I DS=MSG    TRANSMIT THREAD TERMINATED; M=DXMT
HWSD0260I DS=MSG    RECEIVE  THREAD TERMINATED; M=DREC
HWSM0560I IMSPLEX=PLEXG    TRANSMIT THREAD TERMINATED; M=OXMT
HWSM0560I IMSPLEX=PLEXG    RECEIVE  THREAD TERMINATED; M=OREC
HWSD0282I COMMUNICATION WITH DS=MSG    CLOSED; M=DSCL
HWSM0582I COMMUNICATION WITH IMSPLEX=PLEXG    CLOSED; M=DSCL
HWSM0580I IMSPLEX COMMUNICATION FUNCTION CLOSED; M=DOC3
HWSC0020I IMS CONNECT IN TERMINATION
BPE0007I HWS  BEGINNING PHASE 1 OF SHUTDOWN
BPE0008I HWS  BEGINNING PHASE 2 OF SHUTDOWN
BPE0009I HWS  SHUTDOWN COMPLETE
HWSL0101I HWS  CLEANUP SUCCESSFUL
```

5.1.2 OPENDS or STARTDS

The OPENDS and STARTDS commands are equivalent and can be used to start the communication between IMS Connect and a datastore. The parameter for this command is:

► **datastore_id**

Specifies the name of the datastore, as defined in the HWSCFGxx configuration member.

Example 5-3 shows the output of this command.

Example 5-3 An example of the OPENDS command

```
R 79, OPENDS MSG
*080 HWSC0000I *IMS CONNECT READY*  MSGCONN
HWSL0290I CONNECTED TO DATASTORE=MSG    ; M=DSC1
```

5.1.3 OPENIP or STARTIP

The OPENIP or STARTIP commands can be used to start or reestablish the communication between IMS Connect and the IMSplex that contains the Operations Manager (OM) instance that is connected to SCI and will be used to issue commands using the IMS Control Center.

The parameter for this command is:

► **imsplex_id**

Identifies the IMSplex. This name must be defined to IMS Connect through the configuration member HWSCFGxx and must match the T MEMBER that is defined in the IMSplex configuration statement.

Example 5-4 shows the output of the STARTIP command.

Example 5-4 An example of the STARTIP command

```
R 102, STARTIP PLEXG
*103 HWSC0000I *IMS CONNECT READY*  MSGCONN
HWSM0590I CONNECTED TO IMSPLEX=PLEXG    ; M=OSC1
```

5.1.4 OPENPORT or STARTP

The OPENPORT and STARTP commands can be used to reestablish IMS Connect communication with TCP/IP to enable listening on TCP/IP ports. The parameter for this command is:

- ▶ portid

Identifies the number of the port to be opened. For the local option port, specify a port ID value of LOCAL.

Example 5-5 shows the output of this command.

Example 5-5 An example of the OPENPORT command

```
R 82, OPENPORT 7003
*083 HWSC0000I *IMS CONNECT READY*  MSGCONN
HWS0790I LISTENING ON PORT=7003      STARTED; M=SOC2
```

5.1.5 RECORDER

The RECORDER command can either open or close the recorder trace file. This file is a reusable file and requires an IDCAMS VSAM REPRO utility to dump the contents for analysis. It is used primarily to diagnose problems with the IMS Connect user exits. The parameters for this command are:

- ▶ OPEN

Opens the recorder trace.

- ▶ CLOSE

Closes the recorder trace.

Example 5-6 shows the output of this command.

Example 5-6 An example of the RECORDER command

```
R 83, RECORDER OPEN
*084 HWSC0000I *IMS CONNECT READY*  MSGCONN
HWSR0880I RECORDER OPENED; M=RCDR
R 84, RECORDER CLOSE
*085 HWSC0000I *IMS CONNECT READY*  MSGCONN
HWSR0890I RECORDER CLOSED; M=RCDR
```

5.1.6 SETRACF

The SETRACF command can turn the RACF flag and any subsequent RACF user ID checking on or off. The RACF= parameter in the HWS startup parameter sets this at IMS Connect startup. The parameters for this command are:

- ▶ ON

Turns on RACF security.

- ▶ OFF

Turns off RACF security.

Example 5-7 on page 67 shows the output of this command.

Example 5-7 An example of the SETRACF command

```
R 85, SETRACF ON
*086 HWSC0000I *IMS CONNECT READY*  MSGCONN
R 86, SETRACF OFF
*087 HWSC0000I *IMS CONNECT READY*  MSGCONN
```

5.1.7 SETRRS

The SETRRS command enables or disables communication between IMS Connect and RRS.

The parameters for this command are:

- ▶ ON
Turns on communication with RRS.
- ▶ OFF
Turns off communication with RRS.

Example 5-8 shows the output of the SETRRS command.

Example 5-8 An example of the SETRRS command

```
R 323, SETRRS OFF
*324 HWSC0000I *IMS CONNECT READY*  MSGCONN
R 324, SETRRS ON
*325 HWSC0000I *IMS CONNECT READY*  MSGCONN
```

5.1.8 STOPCLNT

The STOPCLNT command terminates communication with a client using a specific TCP/IP port. The parameters for this command are:

- ▶ portid
Identifies the number of the port whose client communication is to be stopped.
- ▶ clientid
Identifies the name of the client, obtained from the VIEWHWS command.

Example 5-9 shows the output of this command (HWS8PBTM is the client name, in this case automatically generated by IMS Connector for Java).

Example 5-9 An example of the STOPCLNT command

```
R 88, STOPCLNT 7003 HWS8PBTM
*089 HWSC0000I *IMS CONNECT READY*  MSGCONN
HWS0761I TCPIP COMMUNICATION WITH CLIENT=7003  _HWS8PBTM STOPPED;
M=SCCM
```

5.1.9 STOPDS

The STOPDS command immediately terminates communication between IMS Connect and the datastore. The parameter for this command is:

- ▶ datastore_id
Specifies the name of the datastore, as defined in the HWSCFGxx configuration member.

Example 5-10 shows the output of this command.

Example 5-10 An example of the STOPDS command

```
R 078, STOPDS IMSG
HWSO0260I DS=IMSG      TRANSMIT THREAD TERMINATED; M=DXMT
HWSO0260I DS=IMSG      RECEIVE  THREAD TERMINATED; M=DREC
HWSO0202W FWE FUNCTION=COMMERR , FAILED FOR DS=IMSG      , COMMAND=STOPDS  IN PROGRESS;
M=DSCM
HWSO0284I COMMUNICATION WITH DS=IMSG      STOPPED;
M=DSCM
```

5.1.10 STOPIP

The STOPIP command stops communication between IMS Connect and the IMSplex that contains the OM that is connected to SCI and used to send commands entered through the IMS Control Center. The parameter for this command is:

► **imsplex_id**

Specifies the name of the IMSplex. This name must be defined to IMS Connect through the configuration member HWSCFGxx.

Example 5-11 shows the output of the STOPIP command.

Example 5-11 An example of the STOPIP command

```
R 99, STOPIP PLEXG
HWSM0560I IMSPLEX=PLEXG  TRANSMIT THREAD TERMINATED; M=OXMT
*102 HWSC0000I *IMS CONNECT READY*  MSGCONN
HWSM0560I IMSPLEX=PLEXG  RECEIVE  THREAD TERMINATED; M=OREC
HWSM0502W FWE FUNCTION=COMMERR , FAILED FOR IMSPLEX=PLEXG  ,
HWSM0584I COMMUNICATION WITH IMSPLEX=PLEXG  STOPPED;
M=DSCM
```

5.1.11 STOPPORT

The STOPPORT command immediately terminates listening on a TCP/IP port. The parameter for this command is:

► **portid**

Identifies the number of the port to be stopped.

Example 5-12 shows the output of this command.

Example 5-12 An example of the STOPPORT command

```
R 81, STOPPORT 7003
HWSO1415E TCP/IP SOCKET FUNCTION CALL FAILED; F=ACCEPT4 , R=-1, E=1152, M=SDCO
*082 HWSC0000I *IMS CONNECT READY*  MSGCONN
HWSO770I LISTENING ON PORT=7003      TERMINATED; M=SSCH
```

5.1.12 VIEWDS

The VIEWDS command displays the current status of the specified IMS system (datastore). The parameter for this command is:

► **datastore_id**

Specifies the name of the datastore, as defined in the HWSCFGxx configuration member.

Figure 5-13 shows an example of the output of this command.

Example 5-13 An example of the VIEWDS command

```
R 89, VIEWDS IMSG
HWSC0001I  DATASTORE=IMSG      STATUS=ACTIVE
HWSC0001I  GROUP=IMS9EXCF MEMBER=HWS910G
HWSC0001I  TARGET MEMBER=SCSIM9G
HWSC0001I  RACF APPL NAME=
```

This display shows the following information:

- The datastore name of IMSG (it also happens to be the IMSID).
- IMS9EXCF is the name of the XCF group, which must be the same group of which the IMS system is a member.
- HWS910G is the XCF member name of IMS Connect within the XCF group.
- SCSIM9G is the XCF member name of the target IMS system within the XCF group. This is the same as the OTMANM parameter in the IMS startup parameters.

5.1.13 VIEWHWS

The VIEWHWS command displays the current status of the OTMA connection. Example 5-14 shows an example of the output of this command.

Example 5-14 An example of the VIEWHWS command

```
R 91, VIEWHWS
HWSC0001I  HWS ID=IMSGCONN RACF=N
*092 HWSC0000I *IMS CONNECT READY*  IMSGCONN
HWSC0001I  MAXSOC=2000 TIMEOUT=8800
HWSC0001I  RRS=N      STATUS=REGISTERED
HWSC0001I  VERSION=910 IP-ADDRESS=009.012.006.077
HWSC0001I  DATASTORE=IMSG      STATUS=ACTIVE
HWSC0001I  GROUP=IMS9EXCF MEMBER=HWS910G
HWSC0001I  TARGET MEMBER=SCSIM9G
HWSC0001I  RACF APPL NAME=
HWSC0001I  IMSPLEX=PLEXG      STATUS=ACTIVE
HWSC0001I  MEMBER=IMSPLEXG TARGET=PLEXG
HWSC0001I  PORT=7003      STATUS=ACTIVE
HWSC0001I  CLIENTID USERID  TRANCODE STATUS      SECOND  CLNTPORT IP-ADDRESS
HWSC0001I  HWS9MBBI USER01  PART      RECV      6        3116 009.001.039.119
HWSC0001I  TOTAL CLIENTS=1  RECV=1 CONN=0 XMIT=0 OTHER=0
HWSC0001I  PORT=LOCAL      STATUS=ACTIVE
HWSC0001I  NO ACTIVE CLIENTS
```

This display shows the following characteristics:

- This IMS Connect is known as HWS910G.
- RACF security within IMS Connect is turned off.

- ▶ The datastore name is IMSG (it also happens to be the IMSID).
- ▶ IMS9EXCF is the name of the XCF group, which must be the same group of which the IMS system is a member.
- ▶ HWS910G is the XCF member name of the IMS Connect within the XCF group.
- ▶ SCSIM9G is the XCF member name of the target IMS system within the XCF group. This is the same as the OTMANM parameter in the IMS startup parameters.
- ▶ PLEXG is the XCF member name of the IMSplex that this IMS Connect will send commands on behalf of the IMS Control Center.
- ▶ The port number is 7003, as defined to IMS Connect.
- ▶ The following information concerns the client that is connecting to the IMS Connect:
 - Client
HWS9MBBI is the client name, which is set in the IRM header (IRM_CLIENTID field) by the client (this client name has been generated automatically by IMS Connector for Java).
 - USERID
USER01 is USERID, which is set in the IRM header (IRM_RACF_USERID field) by the client.
 - Tran Code
PART is the transaction code, which is submitted by the client.
 - Status
RECV (waiting for input from client) and WFCM (waiting for confirmation from IMS) is the status of the client's thread. In this case, IMS Connect is waiting for an ACK/NAK message from the client for a send-then-commit (confirm) message confirmation.
 - Seconds
The number of seconds (6) that the client has been in the specified status.
 - Client port
3116 is a random number that TCP/IP generates to represent a connection from the client.
 - IP address
009.001.039.119 is the IP address used by the connection of the client to IMS Connect.

5.1.14 VIEWIP

The VIEWIP command displays the current activity for the IMSplex.

This command parameter is:

- ▶ `imsplex_id`
Species the name of the IMSplex for which information is to be displayed. This is an optional parameter, and if specified, it must match the ID parameter of the IMSplex configuration statement in the HWSCFGxx member.

Example 5-15 shows the output of the VIEWIP command.

Example 5-15 An example of the VIEWIP command

```

9.18.22 STC27345 R 104, VIEWIP PLEXG
19.18.22 STC27345 HWSC0001I IMSPLEX=PLEXG STATUS=ACTIVE
19.18.22 STC27345 HWSC0001I MEMBER=IMSPLEXG TARGET=PLEXG

```

This displays shows:

- ▶ PLEXG is the name of the IMSplex defined in the ID parameter of the IMSplex configuration statement in the HWSCFGxx member.
- ▶ The status of the IMSplex is ACTIVE. It can also be NOT ACTIVE or DISCONNECT.
- ▶ IMSPLEXG is the name of the member as defined in the MEMBER parameter of the IMSplex configuration statement in HWSCFGxx.
- ▶ PLEXG is the target member name, that is, the member of the IMSplex to which IMS Connect has connected. It is defined in the TMEMBER parameter of the IMSplex configuration statement in HWSCFGxx.

5.1.15 VIEWPORT

The VIEWPORT command displays the current status of communication between IMS Connect and the specified port. The parameter for this command is:

- ▶ portid
Identifies the number of the port to be displayed.

Example 5-16 shows the output of this command.

Example 5-16 An example of the VIEWPORT command

```
R 162, VIEWPORT 7003
HWSC0001I  PORT=7003      STATUS=ACTIVE
*163 HWSC0000I *IMS CONNECT READY*  MSGCONN
HWSC0001I      NO ACTIVE CLIENTS
```

This display shows the following characteristics:

- ▶ The port number is 6001, as defined to IMS Connect.
- ▶ There are no currently active clients, but the connection is active.

5.1.16 VIEWUOR

The VIEWUOR command displays the current status of a specific unit of recovery identifier (URID) or all the URIDs in IMS Connect.

This command expects to get one parameter:

- ▶ UORID
Specifies the 16-byte character identifier of a specific unit of recovery, or the keyword ALL to request to display all UORs.

Example 5-17 shows the output of the VIEWUOR command.

Example 5-17 An example of the VIEWUOR command

```
R 320, VIEWUOR ALL
HWSC0050I URID=BD293FE57E5FAA5C0000000601060000
*321 HWSC0000I *IMS CONNECT READY*  MSGCONN
HWSC0050I  STATE=IN_FLIGHT          FMID=57415344
HWSC0050I  GTRID_L=39 BQUAL_L=28
HWSC0050I  XID=000000000000000B700000004F5B61706
HWSC0050I  XID=253293601DBA22945AEF45D538B7931A
HWSC0050I  XID=73657276657231F5B61706253293601D
HWSC0050I  XID=BA22945AEF45D538B7931A000000049D
HWSC0050I  XID=569D5600000000000000000000000000
```

```
HWSC0050I  XID=00000000000000000000000000000000
HWSC0050I  XID=00000000000000000000000000000000
HWSC0050I  XID=00000000000000000000000000000000
HWSC0050I  TOTAL UOR=1 INDOUBT=0 INBACKOUT=0 INCOMMIT=0 OTHER=1
```

The display shows:

► URID

The 16-byte character identifier of a specific unit of recovery.

► STATE

State of the UR. It can be one of the following values:

- IN_RESET
- IN_FLIGHT
- IN_STATE_CHECK
- IN_PREPARE
- IN_DOUBT
- IN_COMMIT
- IN_BACKOUT
- IN_END
- IN_ONLY_AGENT
- IN_COMPLETION
- IN_FORGET
- FORGOTTEN

Refer to *IMS Version 9: IMS Connect Guide and Reference*, SC18-9287.

► XID

The X/open identifier identifies the distributed transaction, used by the X/Open architecture.

► TOTAL UOR

The total number of UORs in any state.

► INDOUBT

The total number of UORs in IN_DOUBT state.

► INBACKOUT

The total number of UORs in IN_BACKOUT state.

► INCOMMIT

The total number of UORs in IN_COMMIT state.

► OTHER

The total number of UORs in any other state.

5.2 IMS Connect MODIFY commands

Starting with IMS Connect Version 2.2 and with the integrated IMS Connect function of IMS Version 9, you can issue commands to IMS Connect using the operating system MODIFY interface. The syntax of the MODIFY command is:

```
F jobname,command
```

Where jobname is the name of the IMS Connect started task, and command is the string of characters you want IMS Connect to execute.

This interface supports the use of wildcard characters:

- * matches zero or more characters.
- % matches exactly one character.

This wildcard characters are the same ones that you use in TSO and PDF when you specify a member or data set names.

The commands you can issue using the MODIFY interface are functionally equivalent to the ones you can issue using REPLY. The commands responses are also the same.

Table 5-1 shows the equivalence between the REPLY syntax and the MODIFY commands.

Table 5-1 Equivalence between REPLY and MODIFY commands

REPLY command	MODIFY command
CLOSEHWS	SHUTDOWN MEMBER OPTION({QUIESCE FORCE})
OPENDS/STARTDS	UPDATE DATASTORE (name1,name2,...) START(COMM)
OPENIP/STARTIP	No equivalent command
OPENPORT/STARTPT	UPDATE PORT(name1,name2,...) START(COMM)
RECORDER OPEN RECORDER CLOSE	UPDATE MEMBER TYPE(IMSCON) START(TRACE) UPDATE MEMBER TYPE (IMSCON) STOP(TRACE)
SETRACF ON SETRACF OFF	UPDATE MEMBER TYPE(IMSCON) SET (RACF(ON)) UPDATE MEMBER TYPE(IMSCON) SET (RACF(OFF))
SETRRS	No equivalent command
STOPCLNT	DELETE PORT NAME(port) CLIENT(client1,client2,...)
STOPDS	UPDATE DATASTORE (name1,name2,...) STOP(COMM)
STOPIP	No equivalent command
STOPPORT	UPDATE PORT(name1,name2,...) STOP(COMM)
VIEWDS	QUERY DATASTORE NAME(name1,name2,...) [SHOW(ALL)]
VIEWHWS	QUERY MEMBER TYPE(IMSCON) [SHOW(ALL)]
VIEWIP	No equivalent command
VIEWPORT	QUERY PORT NAME(port1,port2,...) [SHOW(ALL)]
VIEWUOR	QUERY UOR NAME(urid1,urid2,...) [SHOW(ALL)]

5.3 IMS Connect BPE commands

You can use IMS Connect Base Primitive Environment (BPE) commands for internal tracing. You can only invoke IMS Connect BPE commands through the z/OS MODIFY command. Refer to *IMS Version 9: IMS Connect Guide and Reference*, SC18-9287, for information about IMS Connect BPE commands.

5.4 IMS command support for IMS Connect and OTMA

This section describes the IMS commands that can be used to check the status of IMS Connect.

5.4.1 /DISPLAY OTMA

The /DISPLAY OTMA command displays the XCF information, status, and security level. Example 5-18 shows an example of the output of this command.

Example 5-18 An example of the /DIS OTMA command

R 328, /DIS OTMA					
DFS000I	GROUP/MEMBER	XCF-STATUS	USER-STATUS	SECURITY	IMSG
DFS000I	IMS9EXCF				IMSG
DFS000I	-SCSIM9G	ACTIVE	SERVER	FULL	IMSG
DFS000I	-HWS910G	ACTIVE	ACCEPT TRAFFIC		IMSG
DFS000I	*05165/133257*	IMSG			

This example shows the OTMA XCF group name of IMS9EXCF, with a number of members of this group. SCSIM9G is the IMS system itself, and HWS910G is the XCF member name of the IMS Connect, which is connecting to SCSIM9G.

5.4.2 /DISPLAY TMEMBER tmember_name TPIPE tpipe_ID

The /DISPLAY TMEMBER command can be used to display the status of the current transaction member status for OTMA clients and servers.

The first example (Example 5-19) simply shows all the members in the XCF group, which is very similar to the /DIS OTMA command output.

Example 5-19 An example of the /DIS TMEMBER ALL command

R 329, /DIS TMEMBER ALL					
DFS000I	GROUP/MEMBER	XCF-STATUS	USER-STATUS	SECURITY	IMSG
DFS000I	SCSIM9G	ACTIVE	SERVER	FULL	IMSG
DFS000I	HWS910G	ACTIVE	ACCEPT TRAFFIC		IMSG
DFS000I	*05165/133605*	IMSG			

Next, this command (Example 5-20) also shows all the Tpipe connections and queue count statistics for a particular OTMA XCF member (HWS910G). If IMS Connect sends transactions using the send-then-commit protocol, the port number (in this case, 7003) is used as the Tpipe name. If IMS Connect sends messages using the commit-then-send protocol, the client name (in this case, CLIENT01/CLIENT02) is used as the Tpipe name. You can also see one Tpipe name (HWSDLMT7) generated automatically by an IMS Connector for Java shareable connection and another one (10913924) generated automatically by the HWSSMPL0 exit.

Example 5-20 An example of the /DIS TMEMBER tmember_name TPIPE ALL command

R 330, /DIS TMEMBER HWS910G TPIPE ALL					
DFS000I	MEMBER/TPIPE	ENQCT	DEQCT	QCT STATUS	IMSG
DFS000I	HWS910G				IMSG
DFS000I	-HWSDLMT7	3	2	1	IMSG
DFS000I	-CLIENT01	6	0	6	IMSG
DFS000I	-7003	0	0	0	IMSG
DFS000I	-CLIENT02	140	138	2	IMSG
DFS000I	-10913924	7	4	3	IMSG
DFS000I	*05165/133835*	IMSG			

Note: ENQCT/DEQCT fields show the *output message's* enqueue and dequeue counts. Therefore, the Tpipe for the send-then-commit message shows zero in these fields, because with the send-then-commit message flow, the output message is not enqueued. The QCT field shows the messages waiting to be retrieved in the hold asynchronous queue of each Tpipe.

Archived

Archived

Accessing IMS Connect

After the e-business application is developed and deployed, the next area to consider is the requirement to support growth and availability and the concept of a single point-of-presence to the Internet on behalf of multiple back-end servers. There are a number of mechanisms that can be used to balance workloads across servers and provide fail-over capabilities. When implementing these solutions, however, it is important to determine which IMS Connect and associated IMS systems are part of this configuration and, if there are several, whether any of the IMS systems accessible by these methods can accept or subsequently process or route any transaction requested.

This chapter describes different ways to access IMS Connect and balance the workload in the sysplex or Parallel Sysplex environment. The mechanisms presented in this chapter are focused on those that are applicable to the use of TCP/IP sockets and IMS Connect. Specifically, the topics include IP load balancing and high availability using functions such as the Load Balancer of the IBM WebSphere Edge Components, virtual IP address (VIP), and the Sysplex Distributor. Additionally, we include a topic that describes the capability of IMS Connect to route/reroute and receive messages to and from different IMS systems.

6.1 IMS Connect in Parallel Sysplex environment

You can have a number of IMS Connect systems available, any of which can be connected to one or more IMS systems. Figure 6-1 shows an example of such a configuration.

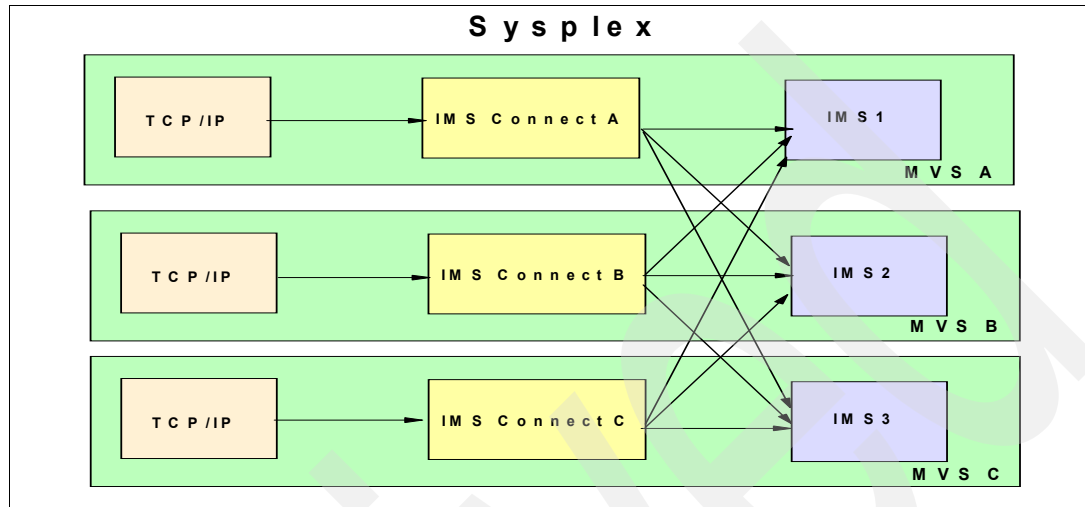


Figure 6-1 The connections between IMS Connect and IMS systems in Parallel Sysplex

Note that IMS Connect can either reside on the same z/OS image as its target IMS or can cross a system boundary assuming that the environment is sysplex-enabled. The assumption in this discussion is that the servers that participate in load balancing and fail-over mechanisms are part of a *cluster* of servers that together provide sufficient processing power and capability to support the growing demands of the client base. Furthermore, because the environment on which IMS executes is a mainframe host, the term *cluster* in this document specifically refers to a *sysplex* that is defined as a group of loosely coupled z/OS images. For example, a sysplex can be multiple physical hosts connected by ESCON® channels or it can be several LPARS within a physical host. The sysplex can be either a *base sysplex* that uses CTC links or a *Parallel Sysplex* that uses a coupling facility to send data between the images. The cluster, therefore, from a connectivity perspective is a set of servers that run the same application and can provide the same contents to its clients.

6.2 Load Balancer

Connection requests to IMS Connect can be short-lived, as is the case for transaction sockets and non-persistent sockets, or they can be long-lived, as is the case for persistent sockets. This is important to understand because the decision for load balancing, or picking a specific server instance to process the work, occurs during the upfront connection phase and not during the data transfer of individual messages.

The Load Balancer is a function delivered with IBM WebSphere Edge Components and consists of five functions that can be used separately or together. These are the Dispatcher, content-based routing (CBR) for HTTP and HTTPS, the Site Selector, the Cisco CSS Controller, and the Nortel Alteon Controller. The Dispatcher component distributes the load it receives to servers contained in a cluster. More specifically, for IMS Connect, the Dispatcher support in the Load Balancer can be used to intercept connection requests and attempt to balance traffic by choosing and then forwarding the request to a specific server, for example, a specific instance of IMS Connect, in the sysplex. This mechanism, commonly known as *IP spraying*, also allows for scalability and failover. The use of the Load Balancer Dispatcher

helps maximize the potential of a sysplex because it provides a solution that can automatically find new servers as they are enabled and added to the sysplex. It can also detect a failed server and route new connection requests only to the available servers. This type of capability was originally implemented and delivered as the Network Dispatcher in IBM networking hardware such as the 2216 and the 3745 MAE. It evolved into the Network Dispatcher capability of WebSphere Edge Server, which can be implemented on platforms such as IBM AIX® 5L™, Microsoft Windows, Sun Solaris™, and Linux and is now the delivered as the Load Balancer of the WebSphere Edge Components.

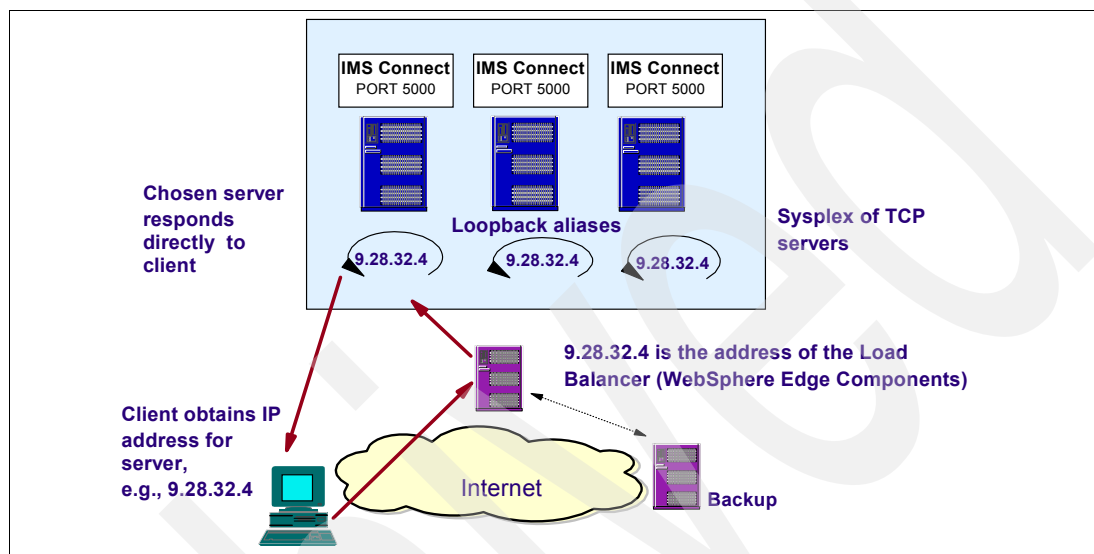


Figure 6-2 Communication flow using the Load Balancer Dispatcher function

In this environment, clients send their connection requests and data to a special IP address that is defined as a cluster address to the Load Balancer Dispatcher. This same address is further defined as a loopback alias address on all the sysplex IP stacks that contain copies of the target application server, IMS Connect in this case. When a request resolves to this special address, the Load Balancer selects one of the back-end servers and forwards the packet to the appropriate port and server.

The selection of a server, or load balancing of requests, can be controlled through several mechanisms. It can be based on simple round-robin scheduling to available servers, or it can use more sophisticated techniques. These can be based on the type of request (HTTP, FTP, Telnet, and so on), or an analysis of the load on the servers, or even through an algorithm based on weights assigned to each server.

After a server is selected, the connection request and all subsequent data packets on that connection are routed to that server. Because IP packets contain the originating IP address of the requesting client, the server can reply directly to the client without sending the output back through the Load Balancer. This is where the loopback alias address is required. Because the original connection request and all inbound packets specify the special IP cluster address of the Load Balancer, by protocol, all reply packets must also be sent by the same IP address. The definition of the loopback alias on the back-end servers allow the reply packets to carry this same address.

To support high availability and to prevent a single point of failure, the Load Balancer function also supports a backup capability and redundancy. Two Load Balancers can be configured with connectivity to each other, the same clients, and a cluster of servers. A heartbeat function exists that enables each to detect the failure of the other along with a synchronization mechanism for the applicable databases (connectivity tables, reachability tables, and so on)

along with logic to elect the active Load Balancer and a mechanism to perform fast IP takeover to switch from the active to the standby.

For more information about the Load Balancer and the WebSphere Edge Components, see the IBM Redbook *WebSphere Application Server V6 Scalability and Performance Handbook*, SG24-6392, or the document *Concepts, Planning, and Installation for Edge Components Version 6.0*, GC31-6855, which is available from the Edge Components Information Center at:

<http://www.ibm.com/software/webservers/appserv/ecinfocenter.html>

In addition to providing a load balancing capability through the Load Balancer's Dispatcher function, IBM has also collaborated with other network technology partners to provide robust load balancing solutions for sysplex. An example of this is the Cisco System MultiNode Load Balancing (MNLB) capability, which provides similar functionality. For information, see:

<http://www.ibm.com/servers/eserver/zseries/networking/technology.html>

6.3 Virtual IP address (VIPA)

The next area to consider is high availability. In TCP/IP networks, a primary area of concern is the outage of an IP network interface, more commonly known as an IP address. This is the network access point to the TCP/IP stack. By default, the failure of an interface prevents access to and from applications using that IP address. When the interface that fails is that of a *client* TCP/IP stack, the impact is the isolation of the client application from the network. However, if the interface that fails is that of a *server*, the impact is greater, because it affects access to all server applications on that stack.

A solution to the larger impact of a server IP interface failure is a concept called virtual IP address or VIPA, which eliminates the host server applications' dependence on a specific network interface. VIPA allows for the definition of a *virtual* IP address that does not correspond to any physical interface, but instead it is associated with the stack as a whole. The VIPA appears to be on a separate subnetwork with the stack itself as a gateway to that subnetwork. The use of VIPA presumes that a TCP/IP stack has multiple physical links or interfaces. Client packets that are targeted to the VIPA are routed to one of the available physical interfaces. If that interface fails, the connection continues to remain intact, and subsequent packets continue to be routed to the VIPA address using one of the other active interfaces.

The implementation of VIPA can address the failure of a specific interface on a single TCP/IP stack (static VIPA), the failure of the entire stack and all its interfaces (dynamic VIPA takeover/takeback), or the failure of the server application on a specific host (application-specific dynamic VIPA).

6.4 Static VIPA

Static VIPA is the term used to refer to the first and simplest implementation of VIPA. It supports failover from one network interface to another on a single stack, as shown in the Figure 6-3.

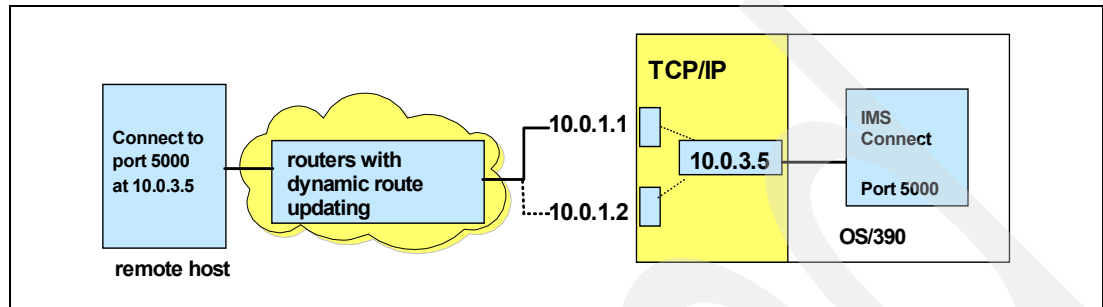


Figure 6-3 Static VIPA

Note that the client application references the target server using the VIPA or a host name that resolves to the VIPA. In this example, it is 10.0.3.5. The real network interfaces 10.0.1.1 and 10.0.1.2 appear to be intermediate hops. When the connection request is made, one of the network interfaces, such as 10.0.1.1, is chosen for the connection and all subsequent data transfer. Only in the event of a physical interface failure associated with 10.0.1.1 will the traffic be routed to 10.0.1.2. This rerouting will be done non-disruptively without the client application receiving any notification of a connection failure.

The definition and activation of VIPA is done through configuration statements in the *hlq.profile.tcip* file. IMS Connect is unaware of VIPA. It simply connects to a TCP/IP stack and relies on the stack to perform the appropriate network function.

Static VIPA only addresses nondisruptive fault tolerance in the case of a specific network interface failure on a stack. If the stack itself fails, VIPA fails. To move the VIPA to a different stack a manual, issue the **vary obey** command.

6.5 Dynamic VIPA takeover

To support recovery of a failed TCP/IP stack and to automate the movement of the VIPA to a surviving backup stack, the VIPA support was enhanced to include a function called dynamic VIPA takeover. The use of this capability presumes that server application instances and IMS Connect instances exist on the backup stack and can serve the clients formerly connected to the failed stack.

In the example shown in Figure 6-4 on page 82, the dynamic VIPA IP address 10.0.3.5 is defined as having a *home* stack on TCPIPA and a *backup* on TCPIPB.

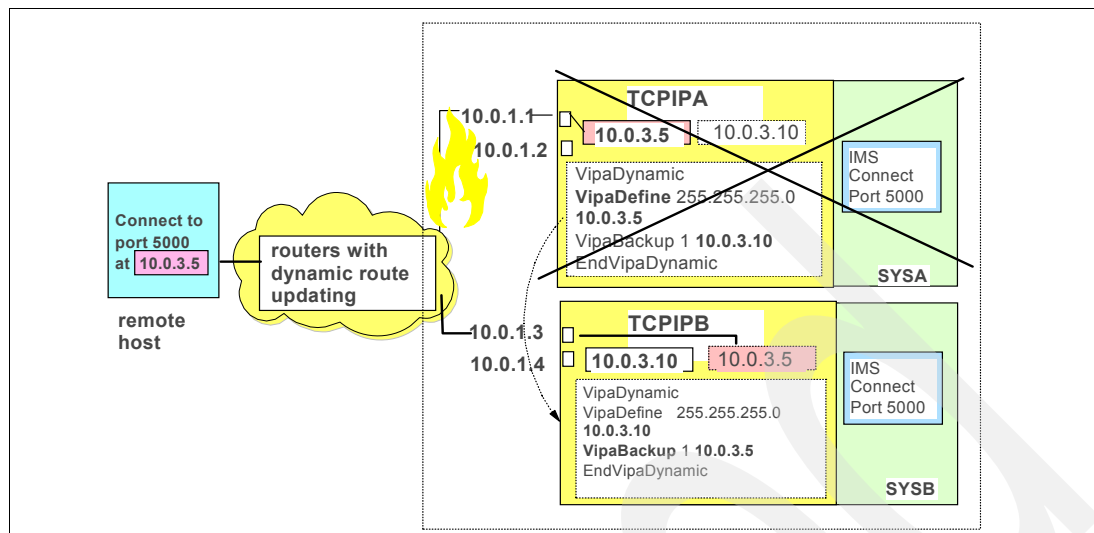


Figure 6-4 Dynamic VIPA takeover

Likewise, TCPIPB is defined as the primary for 10.0.3.10 and TCPIPA as the backup. Both stacks share information regarding the dynamic VIPAs through the use of XCF messaging services. Therefore, each TCP/IP stack is aware of all the dynamic VIPA addresses and the associated primary and backup order.

If a stack or its underlying z/OS system fails, all other stacks in the sysplex are informed of the failure. The VIPA is automatically moved to the backup stack, which receives information regarding the connections from the original stack. All new connection requests to the VIPA are processed by the backup, which becomes the new active. Instances of the server applications, such as IMS Connect systems, listening on the same ports are automatically started if they are not already active on the backup. Additionally, the network routers are informed about the change. From a client application perspective, a connection failure is received when the primary stack fails but the client can immediately resubmit a new connection request that will be processed by the backup (new active) stack.

6.6 Dynamic VIPA takeback

The corollary to VIPA takeover support is VIPA takeback, which is the ability of a reactivated primary or home stack to reestablish ownership of the VIPA. There are two choices for the takeback support that can be defined, as shown in Table 6-1.

Table 6-1 Dynamic VIPA takeback

VIPADEFine	---MOVEable	--- IMMEDIATE---	address_mask	-----	IP address
VIPADEFine	---MOVEable	--- WHENIDLE---	address_mask	-----	IP address

By default, after the primary is reactivated, the VIPA is reacquired and new connection requests are immediately routed to it. Connections that are still active on the backup remain there, and the backup sends information about these still active connections to the primary. Because subsequent data packets associated with all connections using the VIPA flow to the primary, the primary has the responsibility to detect where the data should be sent. If the data is associated with a connection owned by the primary, it is sent to the appropriate server application on its stack. If, however, the data is associated with an old connection that is still associated with the backup, the primary sends the data to the backup stack.

6.7 Application-specific dynamic VIPA

Up to this point, references to a VIPA have implied a direct association with and an ownership by a TCP/IP stack. Access to a specific server application using the VIPA requires that processes be in place to either start the application after the VIPA is activated on that stack, or to have an active instance of the same application (same port) on each stack.

Another VIPA capability is a function called application-specific dynamic VIPA. This enables a server application to register its own unique VIPA with the TCP/IP stack and establish its ownership of the address. The application server can then be moved around the sysplex without affecting the clients that know it by name or address. The name and address do not change even when the specific application server instance physical location changes. Because the application instance is active on only one image of the sysplex, and therefore one TCP/IP stack at a time, the other images and TCP/IP stacks provide a cold standby of the service. This is of value for environments that do not need or require multiple concurrent instances of the same application server, but still need the high-availability profile that can be provided with dynamic VIPA.

To support this capability, the configuration statements in `hlq.profile.tcpip` must permit activation of the specific dynamic VIPA within a subnet range that is defined to ensure that unwanted IP addresses are not created. This is done through the specification of a *viparange* statement, as shown in Figure 6-5 on page 84.

Additionally, the application server must request the initiation of this special IP address. This can be done in one of three ways:

- ▶ Code in the application itself that issues a `bind()` to that specific IP address.
This is not applicable to IMS Connect because the existing code is written to bind to any IP address (non-specific) that is provided by the stack itself.
- ▶ Specification of the `BIND` keyword in the `PORT` statement of the TCPIP profile.
This option is the preferred method for IMS Connect. This method invokes the TCP/IP Server Bind Control function that intercepts the IMS Connect non-specific `bind()` request and converts it to a specific bind request to the IP address specified in the `PORT` statement. Figure 6-5 on page 84 shows this option.
- ▶ Invocation of an authorized program that issues the `SIOCSVIPA IOCTL()` command.
This is an alternative to the previous solution if the `PORT` statement cannot be modified. To use this alternative, add a step in the IMS Connect startup JCL to invoke a special TCP/IP utility called `MODDVIPA`. The utility is in the `TCPIP.SEZALINK` library.

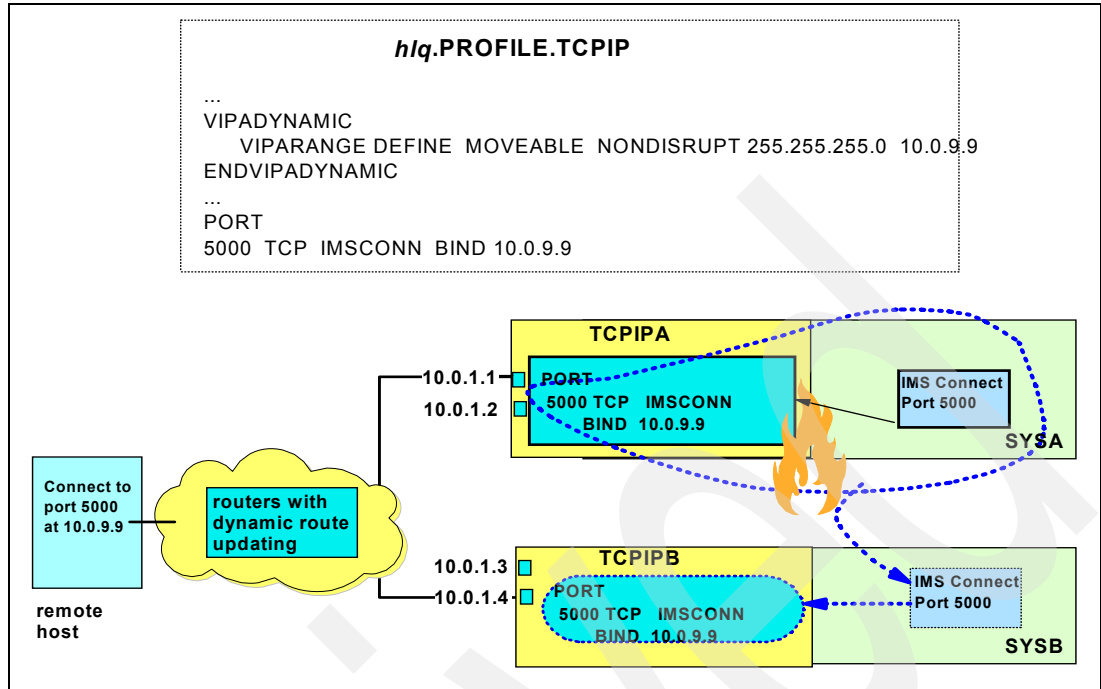


Figure 6-5 Application-specific dynamic VIPA

In the example in Figure 6-5, when IMS Connect is activated on SYSA, its unique VIPA 10.0.9.9 is activated on TCPIPA as defined by the configuration statements in the TCPIP profile. Remote programs that request a connection to port 5000 and IP address 10.0.9.9 are routed to IMS Connect on SYSA. In the event of a failure, such as failure of the entire SYSA image, IMS Connect can be restarted on SYSB. When this new instance of IMS Connect opens port 5000 on TCPIPB, the dynamic VIPA is automatically moved. Remote applications that again issue the same connection requests to port 5000 and IP address 10.0.9.9 are transparently routed to the IMS Connect instance on SYSB. For more information, refer to the IBM Redbook *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 5: Availability, Scalability, and Performance*, SG24-6517, and the following Web site:

<http://www.ibm.com/servers/eserver/zseries/networking/vipa.html>

The previous example assumes that there is a procedure in place to automatically restart IMS Connect on SYSB. This restart can be done with automated operator procedures or through manual operator intervention. The z/OS environment provides a recovery function, called the Automatic Restart Manager (ARM), that can automatically restart applications that have failed on either the same or a different LPAR. To take advantage of this capability, the application must be coded to issue the ARM API (IXCARM). IMS Connect does not issue the ARM API and does not provide the capability to do so.

To resolve the issue of having to modify applications to take advantage of ARM, IBM developed a program called ARMWRAP that issues the ARM calls on behalf of the application. After this code is downloaded from the Web, the IMS Connect JCL can be modified to call the function. Figure 6-6 on page 85 shows how this is done.

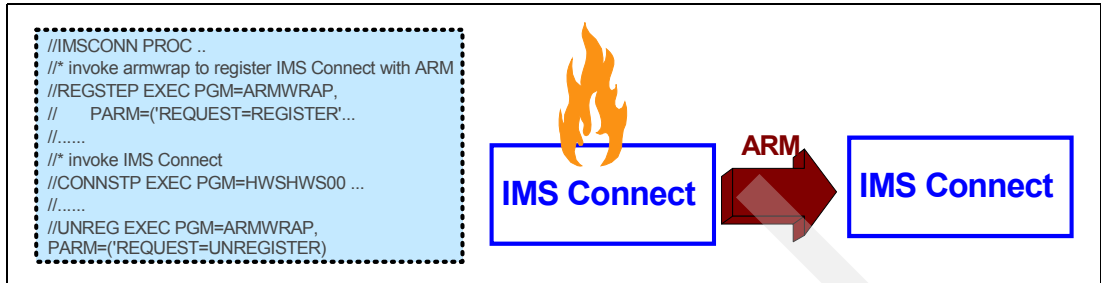


Figure 6-6 Implementing ARMWRAP with IMS Connect

This example shows that the IMS Connect procedure is wrapped in a step ahead of the actual invocation of IMS Connect to register the process to ARM and a step after IMS Connect terminates to unregister. If IMS Connect fails during execution, the ARM definitions are invoked and automatically restart IMS Connect based on the policy defined.

For more information about ARMWRAP, see:

<http://www.redbooks.ibm.com/redpapers/pdfs/redp0173.pdf>

To download the ARMWRAP code, access the following site:

<ftp://www.redbooks.ibm.com/redbooks/REDP0173/>

Figure 6-7 show the combination of ARMWRAP and application-specific dynamic VIPA from a definition perspective.

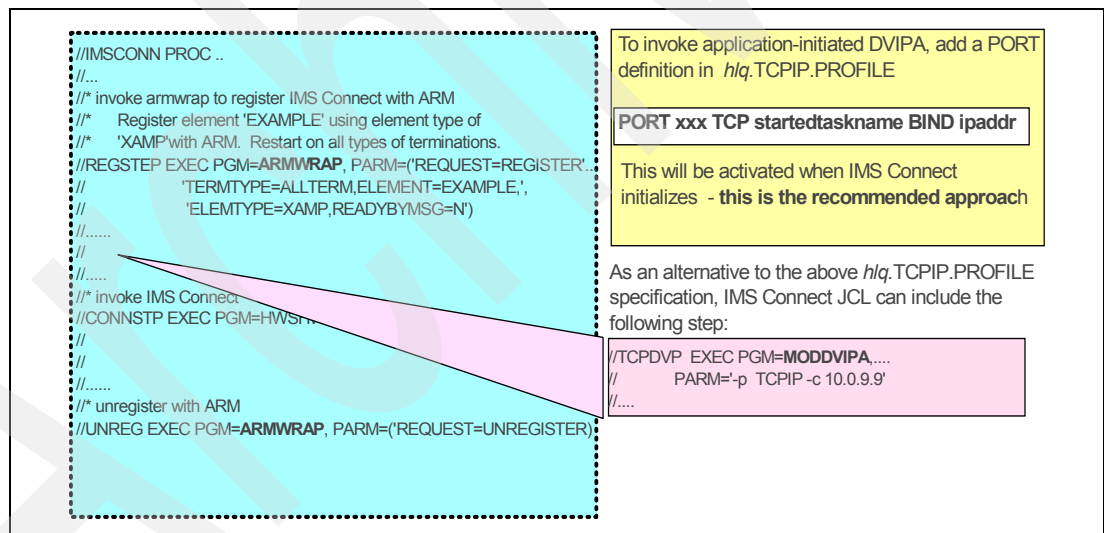


Figure 6-7 IMS Connect JCL with ARMWRAP and application-specific dynamic VIPA support

6.8 Sysplex Distributor

The Sysplex Distributor is an enhancement to sysplex IP support and is a function that resides on a sysplex host. It answers the requirement for a single network-visible or single point-of-presence IP address for the sysplex cluster and also provides a complete solution to workload balancing and high availability. In terms of the concepts that have just been discussed, Sysplex Distributor provides a load balancing type of function with the benefits of dynamic VIPA all rolled up into a single host-based solution.

As a sysplex function, it removes the configuration limitations associated with the Load Balancer of the WebSphere Edge Components (XCF links, rather than LAN connections, are used between the distributing stack and the target servers) and further removes the requirement of specific hardware in the WAN. It also enhances the dynamic VIPA capability and takes advantage of the takeover/takeback support for its own distributing and backup stacks. Figure 6-8 demonstrates the concept.

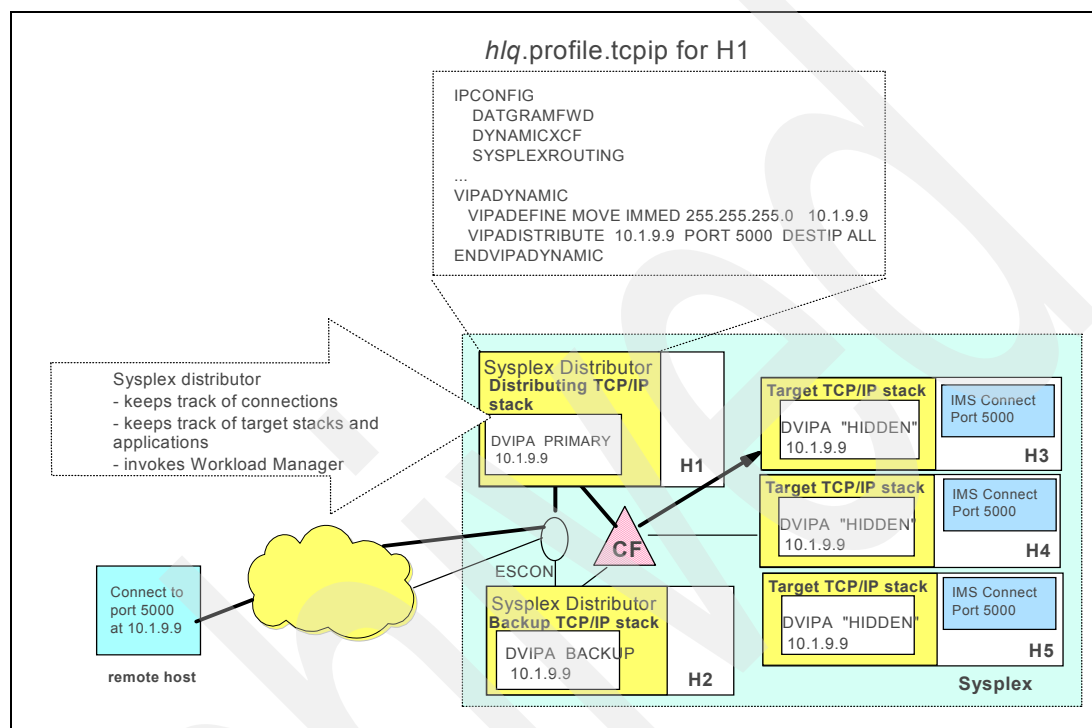


Figure 6-8 The Sysplex Distributor environment

As part of the implementation, the Sysplex Distributor is configured with a distributing stack on one of the sysplex images (H1) and a backup stack on another image (H2). The other images (H3-H5) can be configured as secondary backups. Only the active distributing stack takes the responsibility of advertising the dynamic VIPA (DVIPA) outside the sysplex.

The stacks in the sysplex communicate with each other using XCF services. H1 detects which stacks in the sysplex have active listening ports (5000 for IMS Connect) that are defined as part of the DVIPA environment. The distributing stack builds a table to keep track of server information and also tracks all connection requests associated with the DVIPA.

When an inbound connection request arrives, the distributing stack selects an available target sever with a listening socket and uses XCF services to send the connection to the selected stack. The connection is established between the remote client and the target server, in this case H3. The distributing stack on H1 updates its connection table with the information. This allows H1 to know where to route subsequent data packets that are sent on that connection. When the connection terminates, H3 notifies H1 that the connection no longer exists so that H1 can update its table.

If H1 fails, all participating stacks in the sysplex are notified of the failure. Dynamic VIPA takeover is activated, and H2 defines itself as the new distributing stack. All other participating stacks H3-H5 send their connection information regarding the DVIPA to H2 to facilitate the rebuilding of the connection table. Existing connections between remote clients and IMS Connect systems on H3-H5 are not affected. Data flows are simply rerouted through H2.

After H1 is reactivated, dynamic VIPA takeback begins. Again, this is nondisruptive to existing connections. H2 sends its connection information to H1 so that it can rebuild its table and resume its activities.

In this environment, the only time that remote clients see a connection failure is when the back-end host running IMS Connect fails. This is because that host is the terminating point of the failure. Failures of other hosts or stacks, including the distributing stack, are transparent. When a connection failure is received, the remote client can immediately resend a new connection request. As long as an IMS Connect is active on a listening port, the request can be satisfied.

6.9 IMS Connect load balancing and failover

After a message destination has been resolved to a particular host system and IMS Connect, the next set of configuration decisions deals with connectivity between IMS Connect and the IMS systems in the sysplex. IMS Connect can be partnered with a single IMS, or it can be given access to several as shown previously in Figure 6-1 on page 78.

When provided with access to multiple IMS systems, code can be added to the IMS Connect user message exits to perform load balancing and failover. The code to do this has to be user-written unless the environment includes a product such as IMS Connect Extensions (5655-K48) that provides an interface that facilitates this capability. Regardless of whether code has to be written or a product is used, IMS Connect provides a datastore table that keeps track of the status, active or inactive, of all the IMS systems that are defined in the HWSCFG file. The table is updated as events occur, such as whenever a member of the group, IMS, or IMS Connect joins or leaves. When a message reaches IMS Connect, the appropriate user message exit is invoked. All user message exits have access to the datastore table and can take action based on the information. Figure 6-9 describes two possibilities.

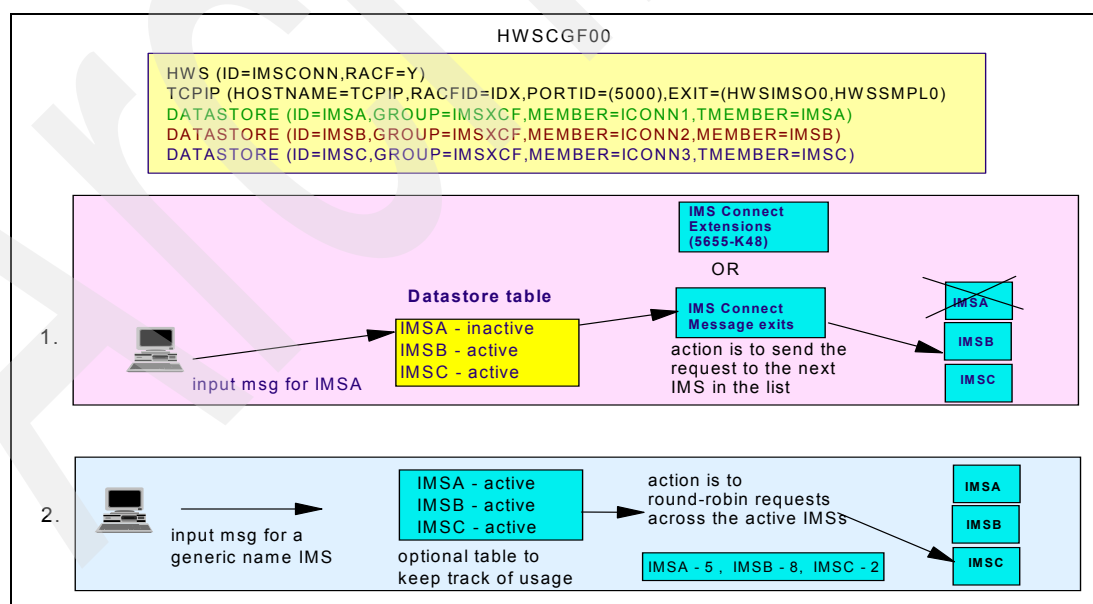


Figure 6-9 Using the datastore table

Be aware that the examples in Figure 6-9 assume that the input messages can be accepted by any of the IMS systems. In other words, the systems are cloned and data sharing is implemented; or an IMS routing mechanism such as Multiple Systems Coupling (MSC) has

been configured to send the message to the appropriate IMS; or IMS shared message queue support has been enabled. The examples are only two of several possibilities that can be explored:

1. In the first example, the IMS Connect user message exit is enhanced to verify that the IMS system, as defined by the datastore ID, is available. If the target IMS is available the exit makes no changes to the destination. If not, the exit can implement a fail-over solution by routing the message to another IMS system with an active status.
2. If an IMS Connect system can reach multiple IMSs, it is up to the client program to specify a target datastore ID. This can be an unwieldy requirement to impose on client applications. This example provides a generic resource capability. The client programs are written to specify a generic datastore ID, IMS, that is intercepted by a user message exit and then changed to a valid value that is marked active in the datastore table. Additionally, the exit can implement a load balancing algorithm such as a round-robin of the requests.

The IMS Connect user message exits, therefore, provide an interface that enables a creative programmer the opportunity to code a solution that answers both failover and load balancing needs. Refer to *IMS Version 9: IMS Connect Guide and Reference*, SC18-9287, for more information about the datastore table and the user message exits.

6.10 Retrieving output messages

The next area of consideration for IMS Connect is the retrieval of output messages. There are two supported application commit protocols that a remote application can use when communicating with IMS Connect that control not only the interaction with IMS but also impact how output messages are to be treated:

- Send-then-commit (CM1)

Remote applications that specify the use of this protocol in the IRM_F2 flag are written to send an input message and wait for the output reply on the same connection. The IOPCB reply messages using the CM1 protocol are sent prior to sync point processing. This type of interaction is easily incorporated into the load balancing, sysplex distribution, and so on, environment. IMS Connect simply delivers the output message to the waiting remote application using the same connection that was established on the inbound request.

- Commit-then-send (CM0)

Remote applications can also use the alternative CM0 protocol to send and receive messages. If the remote application chooses to wait for a reply, the IOPCB message is delivered as a result of the completion of IMS sync point processing on the same connection that was used for the input message. However, as documented in Chapter 2, “Open Transaction Manager Access” on page 7, IMS Connect also supports asynchronous or unsolicited output from IMS using the CM0 application protocol. The output messages, either those resulting from ALTPCB calls or any IOPCB replies that were not delivered to a waiting application, are maintained in a special hold queue in IMS until a remote TCP/IP application sends a special RESUME TPIPE request to IMS Connect to retrieve the message.

The CM0 output messages in IMS are queued to a message queue construct that is identified by Tmember and Tpipe and, therefore, associated with a specific IMS Connect instance. As a result, there is a potential consideration when using any of the load balancing or sysplex distribution mechanisms. To retrieve the message, the remote program will need to establish a connection through the appropriate IMS Connect to the actual IMS system that queued the message. This can be a challenge because there is no easy mechanism for a remote program to discover the required connection path, that is, a

specific IMS Connect to a specific IMS. Additionally, the remote programs might not want to know specific connection paths to IMS because that would negate the value of using load balancing and distribution mechanisms. Figure 6-10 illustrates this point.

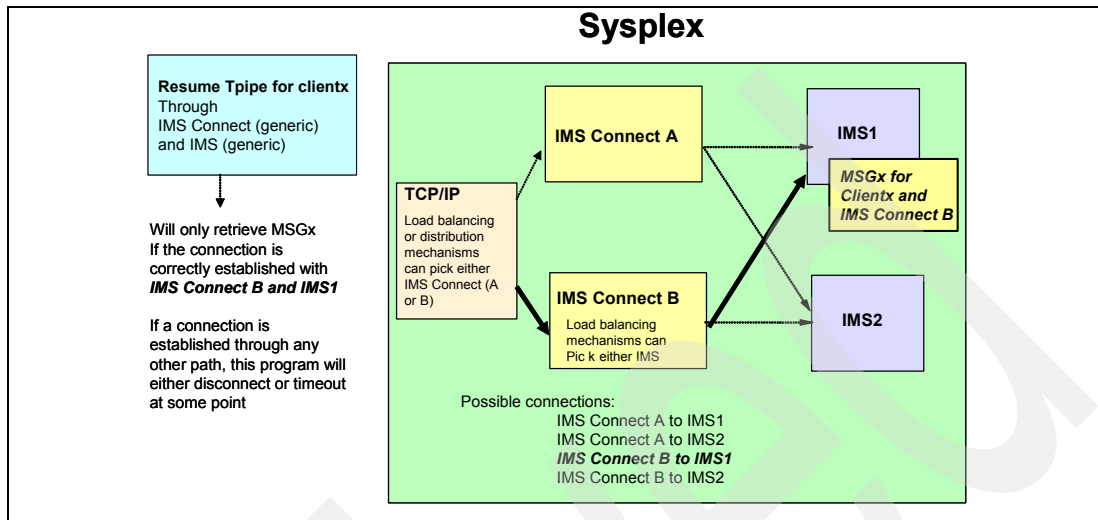


Figure 6-10 RESUME TPIPE using load balancing or distribution mechanisms

The OTMA *super member* function resolves this issue. Refer to 2.6, “Super member support for IMS Connect” on page 23 for more information about prerequisites and implementation information. The super member capability allows a RESUME TPIPE request to retrieve CM0 output across all combinations of IMS Connect and IMS systems. If multiple IMS systems are involved, those IMS systems must also have IMS shared queues implemented. If there is only one IMS system but multiple IMS Connects, shared queues support is not required. See Figure 6-11.

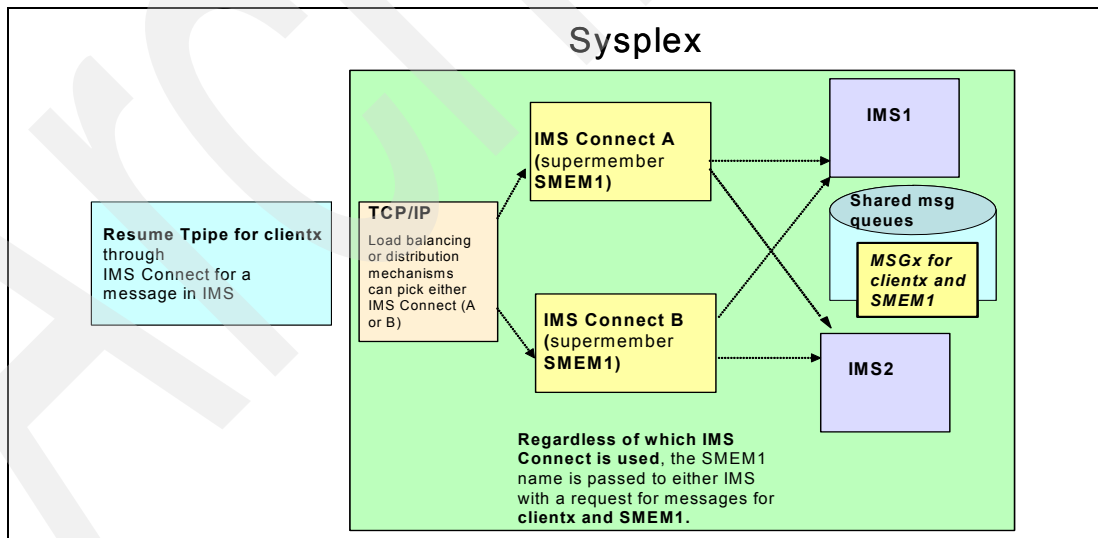


Figure 6-11 RESUME TPIPE with super member support

As shown in Figure 6-11, a RESUME TPIPE request for clientx can be routed through any load balancing or distribution mechanism to either IMS Connect A or IMS Connect B. Both systems are identified to IMS1 and IMS2 as their unique XCF member names and the global super member name of SMEM1. The request to retrieve the output message for Tpipe clientx can be sent to either IMS1 or IMS2 because both have access to the shared

queues and, more specifically, to all the messages under the shared queues construct for SMEM1 and Tpipe clientx.

6.11 The whole picture

The use of all the capabilities mentioned in the previous sections is what enables IMS Connect and IMS to be configured in a single point-of-presence environment to the Internet. More importantly, this type of configuration allows the flexibility required for workload growth without requiring modifications to the remote applications to discover new instances of the back-end servers. Figure 6-12 shows the environment from the remote host all the way to IMS.

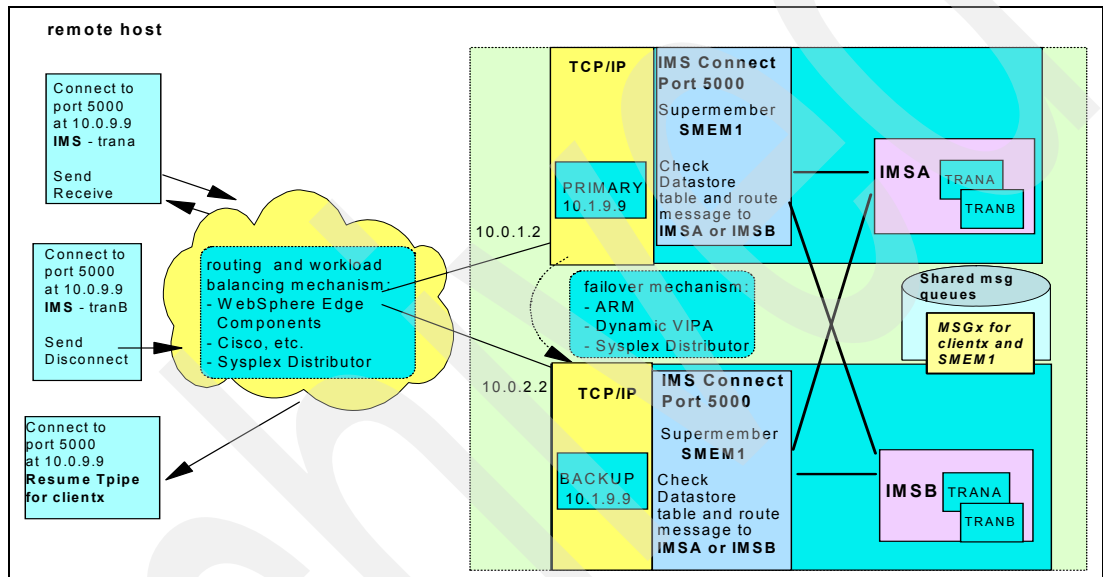


Figure 6-12 End-to-end options

IMS Connect programming model

IMS Connect receives data from a TCP/IP client, performs basic editing and translation, invokes security, and prepares the message in the OTMA format. Response messages from IMS are also prepared into a format that the TCP/IP client understands. Before you start coding the your own TCP/IP client programs, you must know the following information:

- ▶ The message structures that are allowed by IMS Connect and user exits
- ▶ The relationship between the client application and IMS application, OTMA commit mode, and IMS transaction mode
- ▶ The socket types supported by IMS Connect
- ▶ The asynchronous output function supported by IMS Connect

This chapter describes how a TCP/IP to IMS connection is established and how the client and server exchange application data. The descriptions in this chapter are generic. If you are interested about how to program an IMS Connect client, refer to Chapter 12, “IMS Connector for Java” on page 221 if you are using IMS Connector for Java, or to Chapter 14, “Building roll your own clients” on page 265 if you are writing your own client.

7.1 IMS Connect message structures

IMS Connect communicates with OTMA through an XCF session using the OTMA message headers. See Figure 7-1. Clients that use TCP/IP socket calls as their communication vehicle can design a user exit routine that runs with IMS Connect to convert messages between the following formats:

- Convert the client message format to OTMA message format
- Convert the IMS response, in an OTMA message format, to a client message format

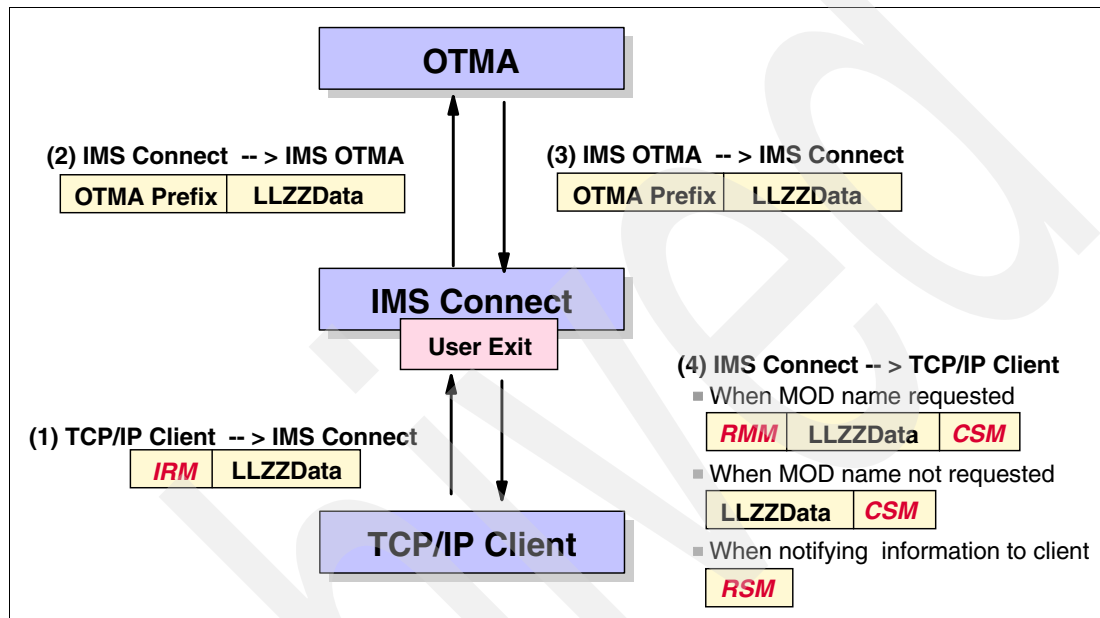


Figure 7-1 Message flow between TCP/IP client and IMS (OTMA)

These conversions enable the client to retrieve IMS data through a TCP/IP connection. Therefore, clients that communicate with IMS Connect must follow the message structure that is predefined by the user exit routine. The message structures consists of the standard IMS messages (LLZZData) and the TCP/IP message structures.

7.1.1 IMS Request Message (IRM)

IMS Connect expects all client messages that it receives to start with a 4-byte total length field, followed by an IMS Request Message (IRM) prefix. The TCP/IP client communicates with IMS by sending an IRM as the first message segment to IMS Connect. The input data stream consists of the IRM, immediately followed by all segments of application data.

7.1.2 Request Status Message (RSM)

IMS Connect returns the Request Status Message (RSM™) as the structure of an output message if IMS Connect or the message exit determines that an error occurred. The RSM contains a return and reason code indicating the type of status.

7.1.3 Complete Status Message (CSM)

IMS Connect returns the Complete Status Message (CSM) as the last structure of an output message if the input message is processed successfully. It does not generate an end-of-message (EOM) segment.

7.1.4 Request Mod Message (RMM)

IMS Connect returns the Request Mod Message (RMM) as the first structure of an output message, if the Message Format Service (MFS) message output descriptor (MOD) name is requested and the data output is present.

Refer to Chapter 9, “IMS Connect user exit support” on page 115 and Chapter 14, “Building roll your own clients” on page 265 for more information about IMS Connect message structures.

7.2 IMS Connect sample message flows

This section describes several sample IMS Connect message flows, including the client calls and IMS Connect actions, for conversational and non-conversational transactions. The sample flows use the following OTMA protocols:

- ▶ Commit-then-send (commit mode 0) and sync level=confirm with ACK
- ▶ Send-then-commit (commit mode 1) and sync level=none
- ▶ Send-then-commit (commit mode 1) and sync level=confirm with ACK

Refer to 2.3, “Commit processing message flows” on page 12 for more information about OTMA protocols.

7.2.1 Non-conversational transaction, CM=0, sync level=confirm

Figure 7-2 shows a sample message flow under the following conditions:

- ▶ The transaction is non-conversational.
- ▶ Commit mode is commit-then-send (confirm).
- ▶ Client sends ACK after it receives the message.

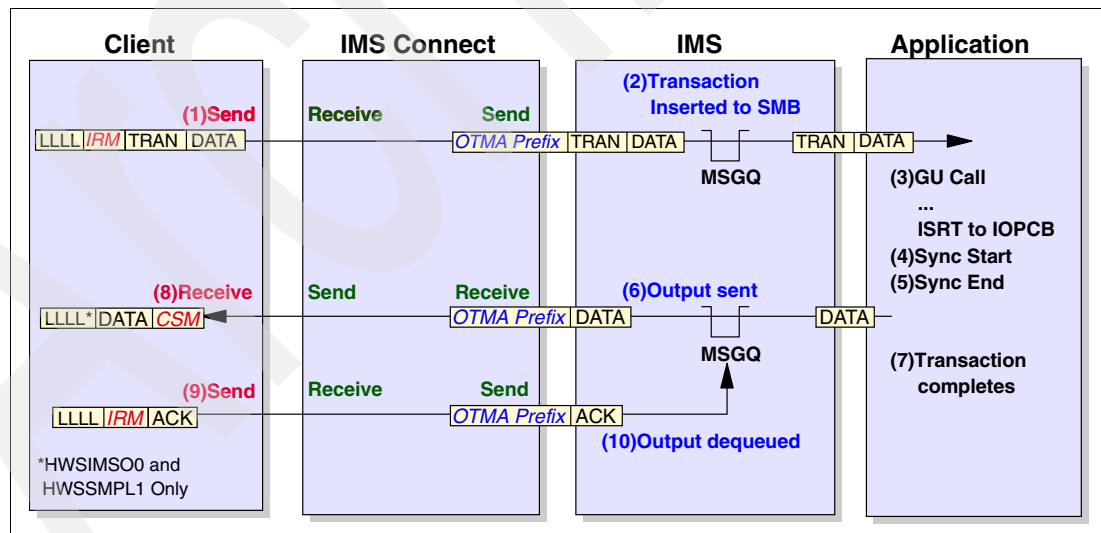


Figure 7-2 Commit-then-send message flow for non-conversational protocols

The sample flow shown assumes the following items:

- ▶ Commit-then-send is specified in the IRM_F2 field of the IRM header.
- ▶ The synchronization level is specified as *confirm* in the IRM_F3 header.
- ▶ The Tpipe is not synchronized.
- ▶ Both input and output messages are enqueued.

7.2.2 Non-conversational transaction, CM=1, sync level=none

Figure 7-3 shows a sample message flow under the following conditions:

- ▶ The transaction is non-conversational.
- ▶ Commit mode is send-then-commit (none).

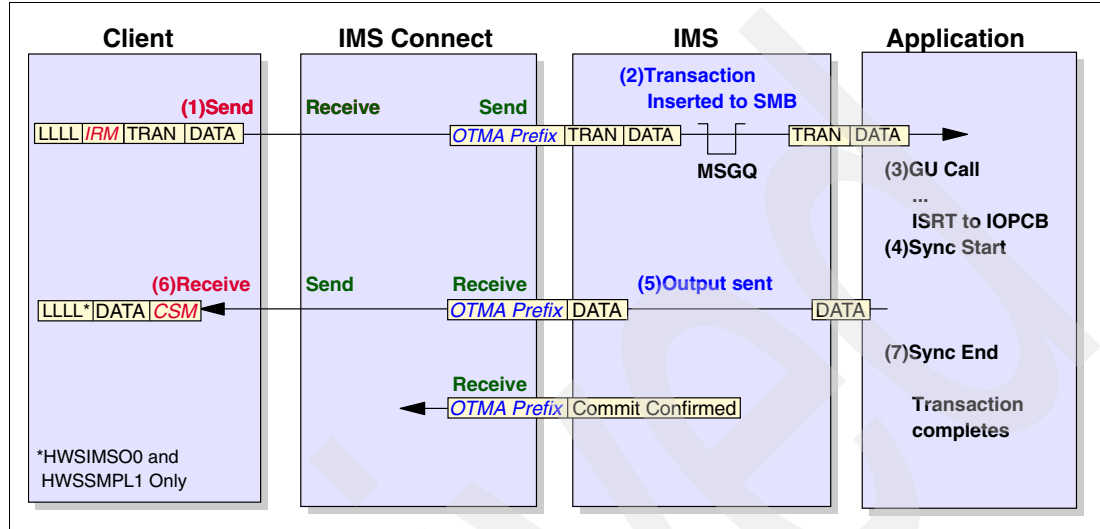


Figure 7-3 Send-then-commit (none) message flow for non-conversational transaction

The sample flow shown assumes the following items:

- ▶ Send-then-commit is specified in the IRM_F2 field of the IRM header.
- ▶ The synchronization level is specified as *none* in the IRM_F3 header. Therefore, IMS does not request a response (ACK/NAK) when sending output.
- ▶ The Tpipe is not synchronized.
- ▶ The output message is not enqueued; it is sent directly to the client before data is committed.

7.2.3 Non-conversational transaction, CM=1, sync level=confirm

Figure 7-4 on page 95 shows a sample message flow under the following conditions:

- ▶ The transaction is non-conversational.
- ▶ Commit mode is send-then-commit (confirm).
- ▶ Client sends ACK after it receives the message.

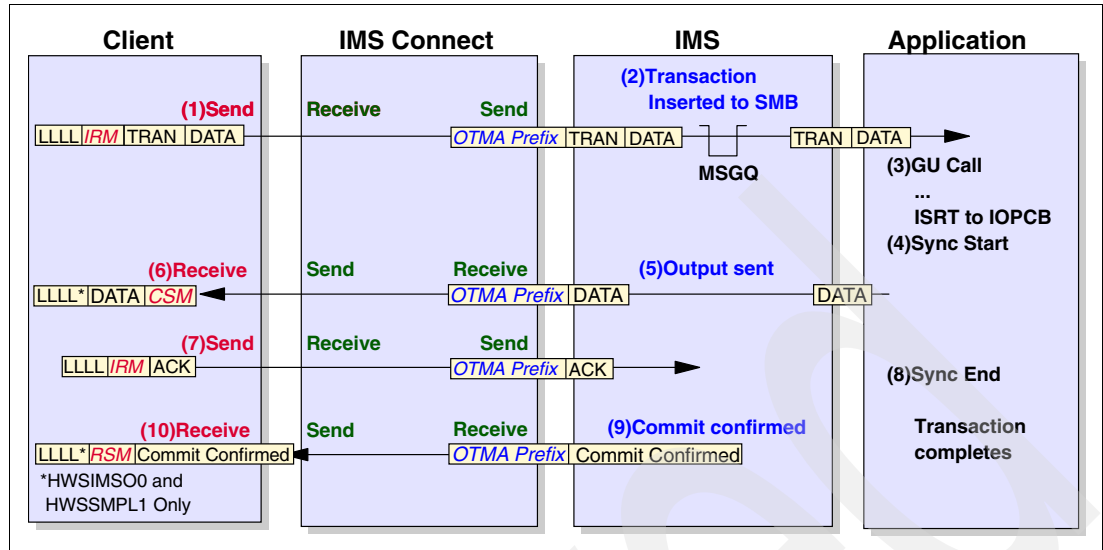


Figure 7-4 Send-then-commit (confirm) message flow for non-conversational transaction

The sample flow shown assumes the following items:

- ▶ Send-then-commit is specified in the IRM_F2 field of the IRM header.
- ▶ The synchronization level is specified as *confirm* in the IRM_F3 field of the IRM header. Therefore, IMS requests a response (ACK/NAK) when sending output.
- ▶ The Tpipe is not synchronized.
- ▶ The output message is not enqueued; it is sent directly to the client before data is committed.

7.2.4 Conversational transaction, CM=1, sync level=confirm

Figure 7-5 on page 96 shows a sample message flow under the following conditions:

- ▶ The transaction is conversational and terminated from the program successfully.
- ▶ Commit mode is send-then-commit (confirm) with ACK.

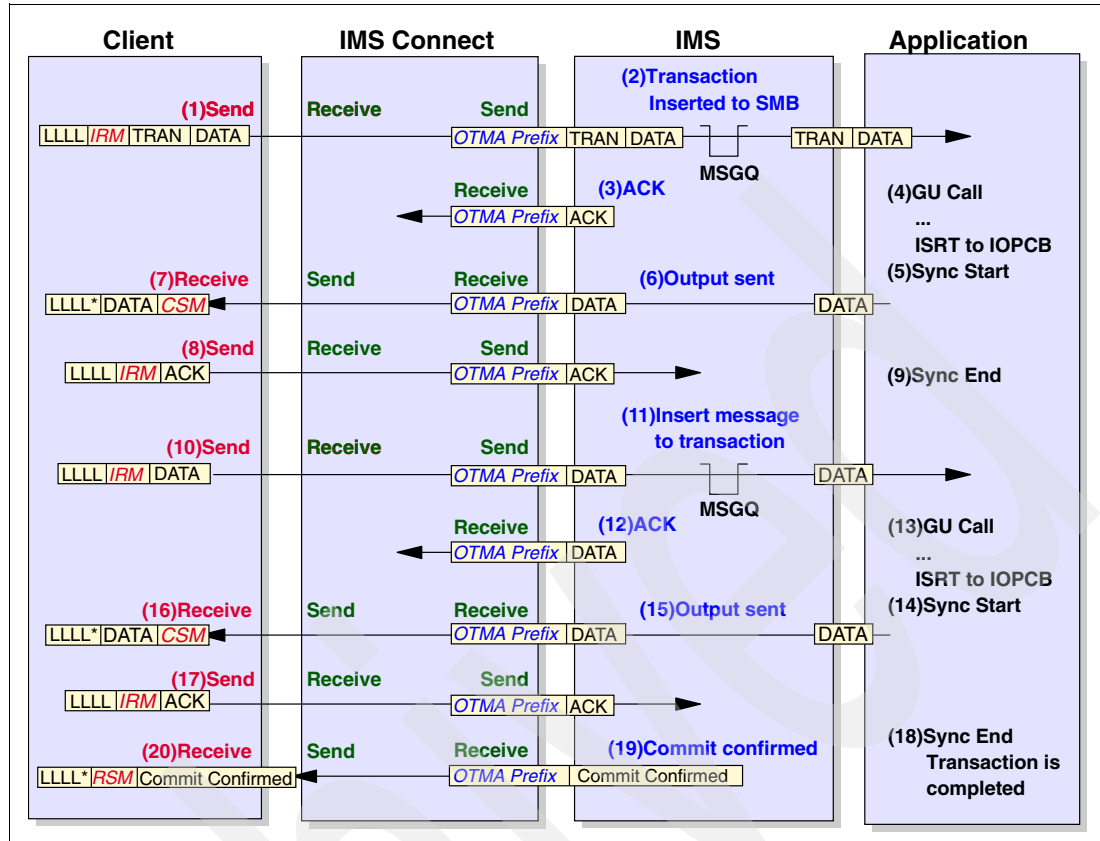


Figure 7-5 Send-then-commit (confirm) message flow for conversational transaction

The sample flow shown assumes the following items:

- Send-then-commit is specified in the IRM_F2 field of the IRM header.
- The synchronization level is specified as *confirm* in the IRM_F3 field of the IRM header. Therefore, IMS requests a response (ACK/NAK) when sending output.
- The Tpipe is not synchronized.
- The output message is not enqueued; it is sent directly to the client before data is committed.

7.2.5 Send-only transaction, CM=0, sync level=confirm

Figure 7-6 on page 97 shows a sample message flow under the following conditions:

- The transaction is in non-response mode and terminated successfully without an output message.
- Commit mode is commit-then-send (confirm) with ACK.

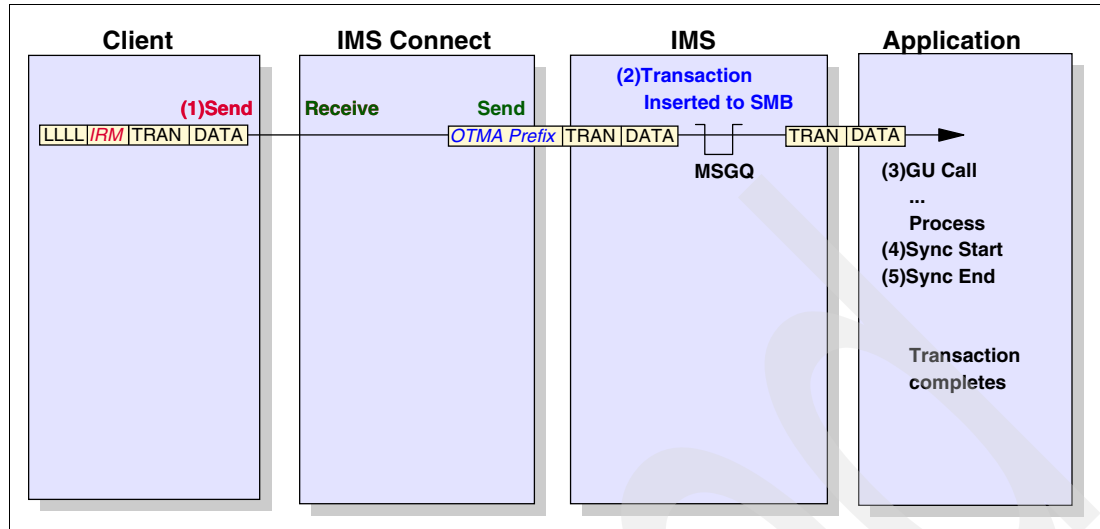


Figure 7-6 Commit-then-send message flow for send-only transaction

The sample flow shown assumes the following items:

- ▶ Commit-then-send is specified in the IRM_F2 field of the IRM header.
- ▶ The synchronization level is specified as *confirm* in the IRM_F3 field of the IRM header.
- ▶ The message type is specified as a send-only transaction in the IRM_F4 field of the IRM header. Therefore, the client does not expect any output from IMS.
- ▶ The Tpipe is not synchronized.

7.2.6 The CANCEL TIMER request

You can find yourself in a situation in which you are waiting for an IMS Connect response and, for some reason, the RECEIVE function fails. One such case is if a socket timeout is in effect, and the socket timeout value is less than the IMS Connect execution timeout specified using IRM_TIMER. Another case is an interrupted RECEIVE call due to an I/O error.

In both cases, if you are using specific clientIDs and you try to send another message to IMS Connect, you will get an OTMA error because that the clientID is still active on IMS Connect. IMS Connect will refuse to use the same clientID until the original interaction completes or times out.

If you need to free that clientID so that you can continue using it without waiting for the timeout to occur, you can send an IRM with a value of C in IRM_F4 and no transaction data. This enables you to send transactions using that clientID again. If the first interrupted interaction completes after that, its output will be considered not deliverable, and so:

- ▶ If the interrupted interaction was uses commit mode 1 and a synchronization level other than none, IMS abends the transaction with U0119, and the output message is discarded. If the synchronization level is none, the transaction is committed and its output is discarded.
- ▶ If the interrupted transaction uses commit mode 0, IMS queues the output in the asynchronous hold queue associated with Tpipe named after the clientID.

7.3 Socket connections and settings

IMS Connect provides three kinds of client connection protocols, which are called sockets. Sockets define how IMS Connect manages client connections and when IMS Connect sends a disconnect. The three socket types are:

- ▶ Persistent
- ▶ Transaction
- ▶ Non-persistent

The client program controls the socket settings, and the IMS Connect user message exits and the user initialization exit enforce socket settings. The client selects the socket connection type by setting a flag in IRL in the IRL_SOCT field.

Table 7-1 shows the relationship between OTMA the commit mode/user exit and the socket connection type. Notice that the provided exits (HWSSMPL0/1, HWSIMSO0/1, and HWSJAVA0) *do not support* non-persistent sockets. In IMS Connect Version 2.2 and later, IMS Connector for Java uses only persistent sockets.

Table 7-1 Socket mode

	Persistent	Transaction	Non-persistent
IRL_SOCT value	X'10'	X'00'	X'40'
Send-then-commit	Available	Available	Available
Commit-then-send	Available	Available	Not available
HWSIMSO0/HWSIMSO1	Available	Available	Not available
HWSSMPL0/HWSSMPL1	Available	Available	Not available
HWSJAVA0	Available	Not available	Not available

The user message exits determine the socket type and then move the socket type information to the OMUSR_FLAG1 field of OTMA header user data section.

Note: The IRL_SOCT flag must be set for each message that is sent to IMS Connect and must be set for all messages that are associated with a single transaction to the same socket type. If you do not, unexpected results can occur.

7.3.1 Persistent sockets

A persistent socket is a connection between the client and IMS Connect that remains connected until either the client or IMS Connect specifically makes a disconnect request. A persistent socket can exist across multiple transactions.

There are two ways that the client can force a termination:

- ▶ By sending IMS Connect a disconnect request.
- ▶ By changing the socket type to *transaction socket* for the last transaction entered, such as a log-off transaction.

IMS Connect can also terminate the connection when an error occurs. IMS Connect user message exits HWSIMSO0, HWSIMSO1, HWSSMPL0, HWSSMPL1, and HWSJAVA0 support the use of persistent sockets. IMS Connector for Java also supports the use of persistent sockets. Both the send-then-commit and commit-then-send messages can use this protocol. Figure 7-7 shows a sample message flow of a persistent socket connection (send-then-commit, sync level=confirm with ACK, non-conversational transaction).

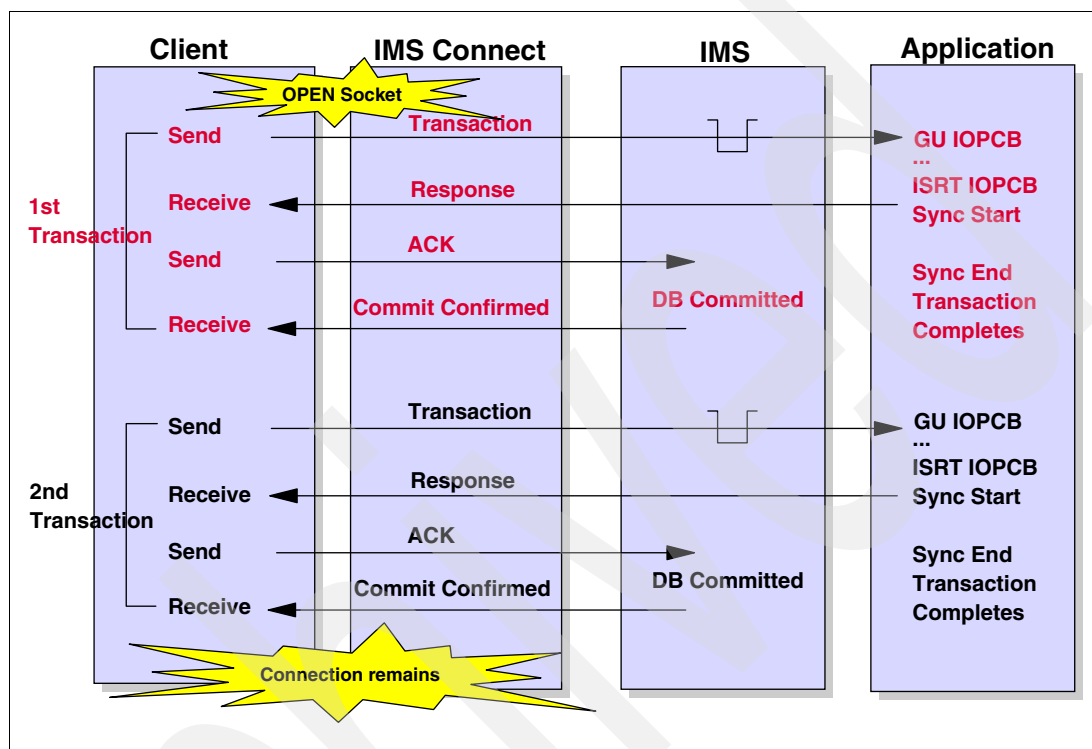


Figure 7-7 A sample message flow of persistent socket (CM=1, sync level=confirm)

7.3.2 Transaction sockets

A transaction socket is a connection between the client and IMS Connect that remains connected for a single transaction or IMS conversation. The connection can be terminated only by IMS Connect, either when IMS itself terminates, or when an error occurs. Figure 7-8 on page 100 shows a sample message flow of a transaction socket connection (send-then-commit, sync level=confirm with ACK, non-conversational transaction).

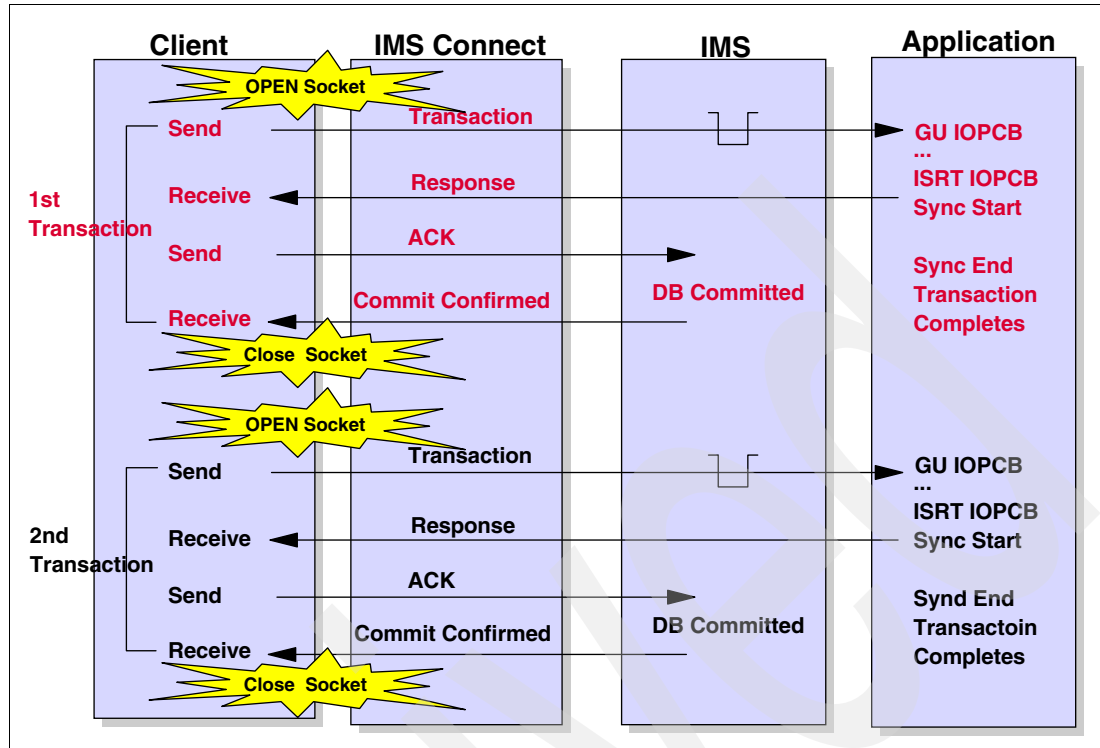


Figure 7-8 A sample message flow of transaction socket (CM=1, sync level=confirm)

7.3.3 Non-persistent sockets

A non-persistent socket maintains a connection for a single input-and-output pair to IMS Connect. IMS Connect terminates the connection after sending the output to the client for non-conversational and conversational transactions. If three exchanges of input and output occur, a disconnect is issued three times, one for each output from IMS Connect. None of the IBM-supplied user message exits or IMS Connector for Java support non-persistent sockets.

7.4 Asynchronous output support

IMS Connect can manage asynchronous output by not allowing it to flow while a transaction is being processed. This section describes the IMS Connect asynchronous output support for IMS Version 9 or later.

7.4.1 What is asynchronous output?

There are several types of asynchronous output:

- ▶ Output that is sent to a client from an IMS application using the ALTPCB.
- ▶ Output that is sent to a client from an IMS application using a second or posterior output to the IOPCB.
- ▶ Any commit-then-send (commit mode 0) output that is sent to the client for which the client or IMS Connect sends a NAK in response to the output message.

OTMA inserts these messages to the *special queue*, which is used to hold commit-then-send output that is commit-then-send output or alternate program communication block (PCB) output for an OTMA client. Output messages in the queue are not delivered until the client makes a request. IMS Connect can send the RESUME TPIPE command to tell IMS to deliver one or all queued messages on the special queue for Tpipe. However, the option specified in the command can be used to request how IMS can hold and deliver the message.

IMS Connect communicates the presence of asynchronous output to the client in one of the following ways:

- ▶ By returning the flag CSM_AMSG (binary '10000000') in the CSM_FLG1 field in the Complete Status Message (CSM).
- ▶ By returning the flag RSM_AMSG (binary '10000000') in the RSM_FLG1 field in the Request Status Message (RSM).

Restriction: Asynchronous output is only supported in IMS V7 and later releases. It is not operational when IMS Connect is communicating with IMS V6.

7.4.2 Implementing asynchronous output support

When you receive the asynchronous output message, you must implement the following actions to your client application:

1. Issue a CONNECT command.
2. Issue a TCP/IP SEND of an OTMA RESUME TPIPE command, immediately followed by a TCP/IP READ function from the primary client application.
3. Issue a TCP/IP SEND of an ACK or NAK response on the receipt of the output message.
4. Repeat steps 2 and 3 until either all messages have been received, or until the end user has received all of the messages that they want.
5. Issue a DISCONNECT command.

Asynchronous output message functions are controlled by information that is passed in the IRM and then set in OTMA header by the user message exit. There are three types of asynchronous output message controls: single, noauto, and auto. These control functions are supported by IMS V7 or later. To choose the type of message control, the client code sets the IRM field IRM_FLG5 to be one of the following values:

- ▶ IRM_F5_ONE, which retrieves a single message (single).
- ▶ IRM_F5_NOAUTO, which retrieves all messages that have been queued (noauto). After all messages are received, a DISCONNECT and CONNECT, RESUME TPIPE must be issued to reactivate the asynchronous output retrieve function.
- ▶ IRM_F5_AUTO, which retrieves all messages that have been queued, then retrieves any additional messages that are queued later (auto).

HWSSMPL0 and HWSIMSO0 user message exits default to the NOAUTO type of asynchronous output message management. The code that activates the selection from the IRM_F5 field is commented out of the HWSSMPL0 user message exit.

For RESUME TPIPE requests, you must set values shown in Table 7-2 on page 102.

Table 7-2 IRM header value for RESUME TPIPE request

Function	IRM field	For SINGLE	For NOAUTO	For AUTO
Retrieval type	IRM_F5	Binary '.... 0001'	Binary '.... 0000'	Binary '.... 0010'
Socket type	IRM_SOCT	Transaction socket (X'00') or persistent socket (X'10')		
Commit mode	IRM_F2	Commit-then-send (X'40')		
Sync level	IRM_F3	Confirm (X'01')		
Message type	IRM_F4	RESUME TPIPE (Character 'R')		
Timer setting	IRM_TIMER	See the following description		

Timer value (IRM_TIMER) setting

The IRM_TIMER value is the wait value to wait for a RECEIVE issued from the client following a RESUME TPIPE, or an ACK to the RECEIVES following the RESUME TPIPE. The IRM_TIMER value is set on the RECEIVE call, not on the RESUME TPIPE call. You can set the IRM_TIMER field to be either a label or a decimal value, as shown in the Table 7-3.

Table 7-3 IRM header value for IRM_TIMER

IRM_TIMER value	Wait value
IRM_TIME_QS (X'00')	0.25 second (default value)
IRM_TIME_ZERO (X'E9')	No wait
IRM_TIME_FF (X'FF')	Wait indefinitely
X'01' through X'19'	A range of 0.01 second to 0.25 second

Recommendations: Use IRM_TIME_FF only with the auto asynchronous output message management type (IRM_F5_AUTO). Because of the wait *indefinitely* condition, use IRM_TIMER_FF and IRM_F5_AUTO together only when a dedicated output device is available. Using these two IRM options together causes the client to act as though it were a persistent socket (even though it is not), or to act like a never-ending transaction.

7.4.3 SINGLE message control

Figure 7-9 on page 103 shows a sample message flow of the SINGLE message control option. When using this option (by setting the IRM_F5 to IRM_F5_ONE field), the client can receive only a single message.

Using the SINGLE message control option forces the following sequence of events to occur:

1. Client issues the CONNECT function.
2. Client issues the RESUME TPIPE function.
3. Client issues the RECEIVE function.
4. Client sends an ACK to IMS Connect.
5. IMS Connect disconnects the socket from the host end.
6. Client issues the DISCONNECT function.

If the client responds with a NAK rather than an ACK, the message that has been NAKed is put back on the OTMA asynchronous hold queue and can be retrieved again later. IMS Connect terminates the socket as described in event 5.

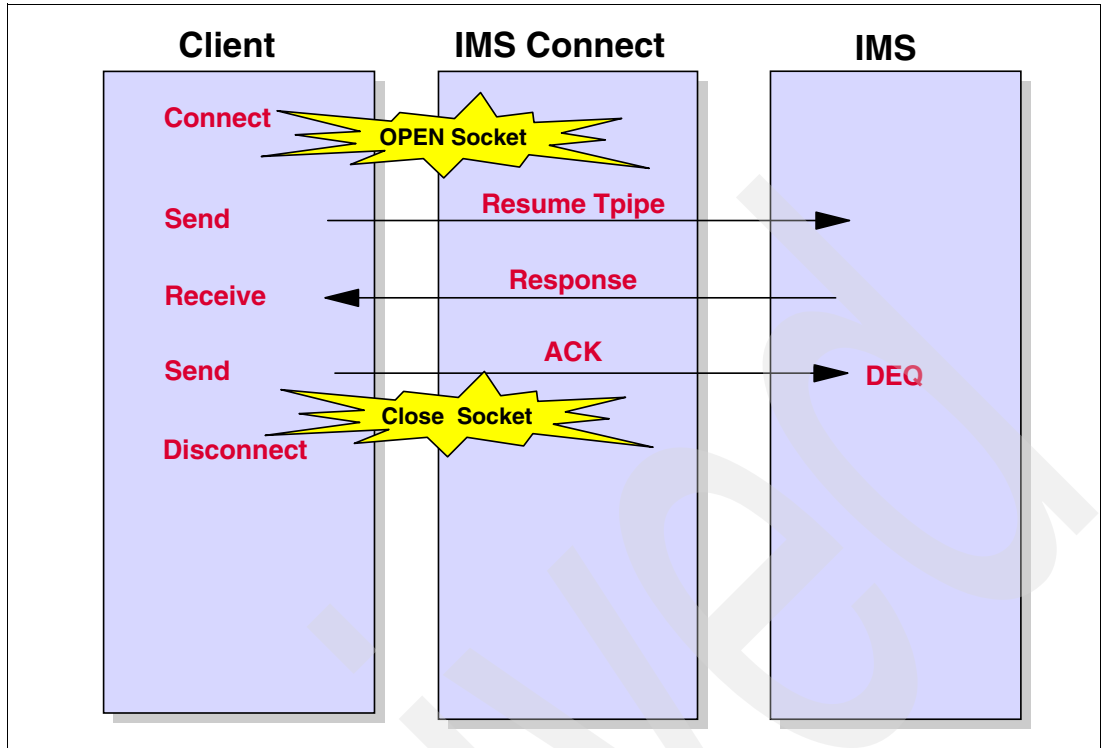


Figure 7-9 A sample message flow of the *SINGLE* message control option

If there is no message waiting to be retrieved, IMS Connect waits for the timeout set using `IRM_TIMER` and sends the client an RSM with the timeout return code. Because IMS Connect does *not* send any message to the client not present in the queue at the time the first `Receive()` call is issued, you should specify a small timeout value to avoid the client being blocked waiting for the timeout. Compare this behavior of the *SINGLE* option against the *SINGLE WAIT*, described in the following section.

7.4.4 SINGLE WAIT message control

The flow for the *SINGLE WAIT* option is similar to the one for the *SINGLE* option. The difference is that if you issue a `RESUME TPIPE` with the *SINGLE WAIT* option and there is no message waiting to be retrieved, IMS Connect waits until the time set by `IRM_TIMER` before sending a timeout response. Therefore, you can make your client wait for a message to arrive without the need of polling the Tpipe.

Specify a reasonably high timeout value when you use this option, or even `X'FF'` (wait forever), if you are writing a specialized client to process the asynchronous messages.

7.4.5 NOAUTO message control

Figure 7-10 on page 104 shows a sample message flow of the *NOAUTO* message control option. When using this option (by setting the `IRM_F5` to `IRM_F5_NOAUTO` field), the client can receive all of the messages on the OTMA asynchronous queue.

Using the *NOAUTO* message control option forces the following sequence of events to occur:

1. Client issues the `CONNECT` function.
2. Client issues the `RESUME TPIPE` function.
3. Client issues the `RECEIVE` function.

- Using the NOAUTO message control option, the client can always terminate by issuing a DISCONNECT function after sending an ACK to IMS Connect. If the client responds with a NAK rather than an ACK, the message that has been NAKed is put back on the OTMA asynchronous hold queue and can be retrieved again later. IMS Connect terminates the socket as described in event 6.

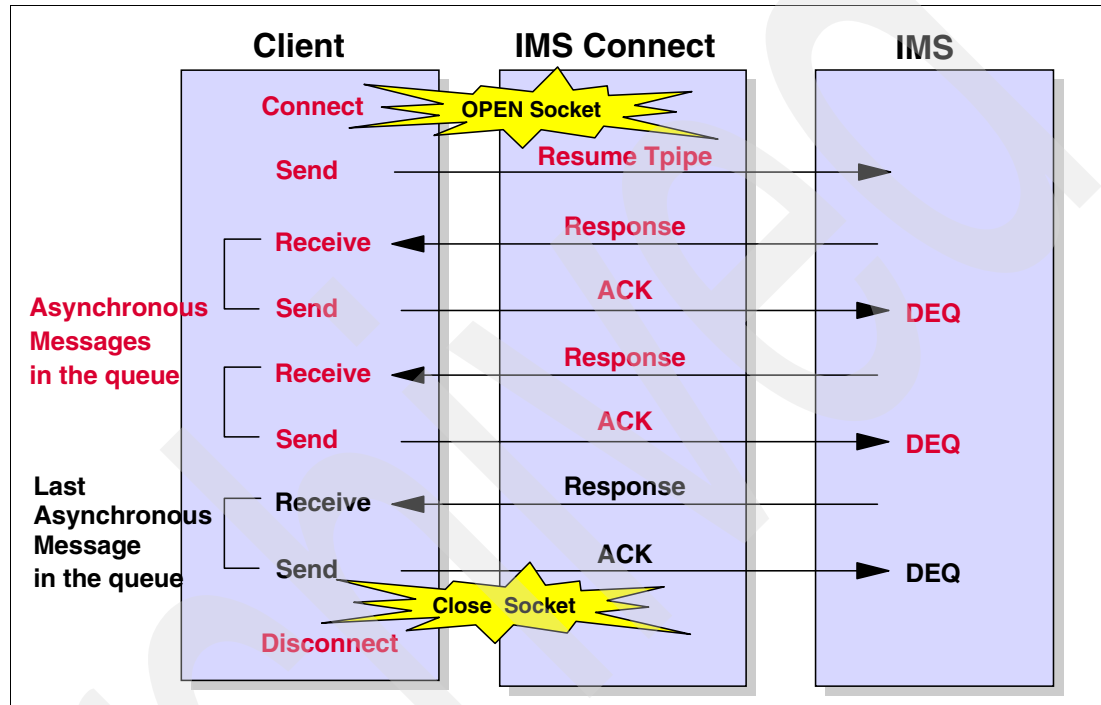


Figure 7-10 A sample message flow of the NOAUTO message control option

7.4.6 AUTO message control

Figure 7-11 on page 105 shows a sample message flow of the AUTO message control option. When using the AUTO message control option (by setting the IRM_F5 to IRM_F5_AUTO field), the client can receive all of the messages on the OTMA asynchronous queue, and any messages that are placed on the OTMA asynchronous queue after the current messages are all removed.

Using the AUTO message control option forces the following sequence of events to occur:

1. Client issues the **CONNECT** function.
2. Client issues the **RESUME TPIPE** function.
3. Client issues the **RECEIVE** function.
4. Client sends an **ACK** to **IMS Connect**.
5. Client repeats events 3 and 4.

If all the messages have been removed from the queue, event 3 remains active (that is, in a receive state) until the user-specified timer supplied in the IRM has expired. IMS Connect then terminates the socket.

Recommendation: If event 3 or event 5 receives a disconnect of the socket, the client should disconnect and then wait for a time interval before repeating events 1-5.

Using the AUTO message control option, the client can always terminate by issuing a DISCONNECT function after sending an ACK to IMS Connect.

If the client responds with a NAK rather than an ACK in events 3 or 5, the message that has been NAKed is put back on the OTMA asynchronous hold queue, IMS Connect terminates the socket, and then those messages can be retrieved again later.

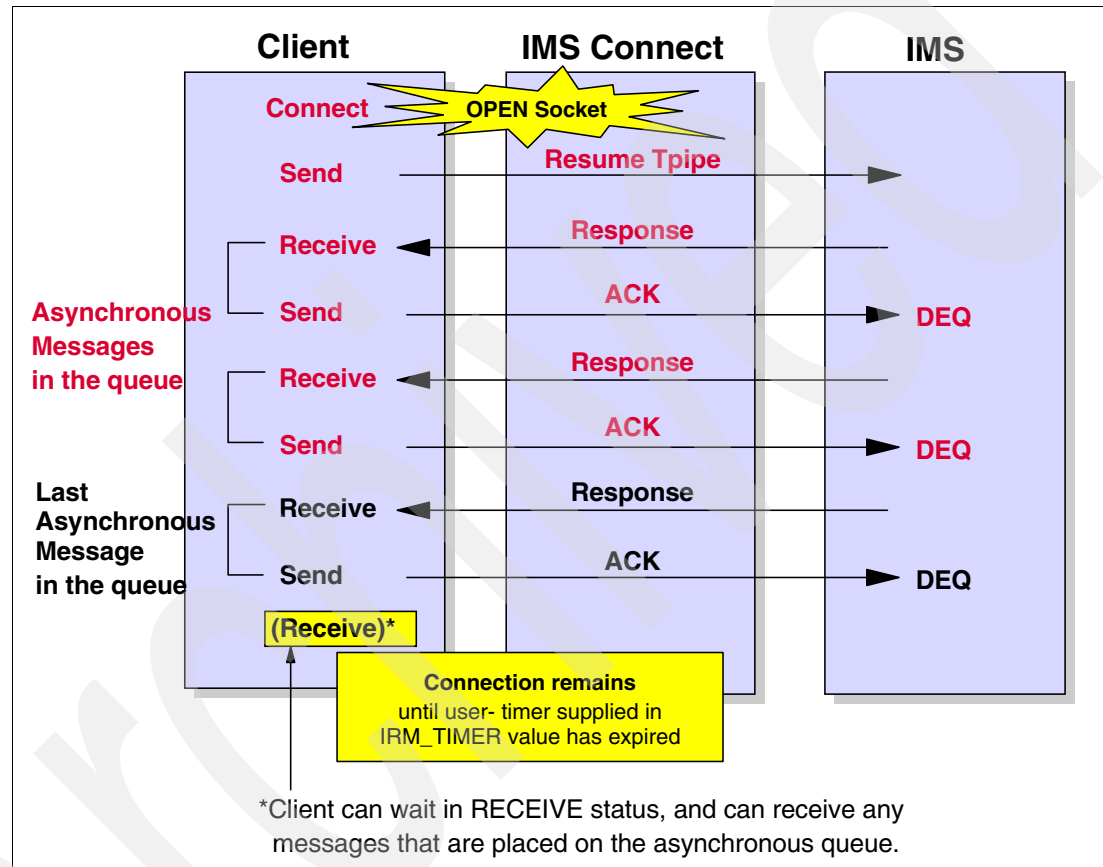


Figure 7-11 A sample message flow of the AUTO message control option

7.4.7 Purge not deliverable

There are situations in which you will not be interested in the retrieval of the asynchronous output messages. If you do not retrieve those messages, they add up in the queues and, finally, you will need to purge those queues to get rid of the unwanted messages.

To avoid this situation, if you know from the beginning that you will not retrieve any asynchronous output for a specific interaction, you can enable the purge not deliverable output feature, using the X'04' value of the IRM_F3 parameter, or the equivalent attribute of the IMS Connector for Java connection object.

If you set the purge not deliverable option for a commit mode 1 transaction that does a program switch to a second transaction and the first transaction does an ISRT to the IOPCB,

the purge not deliverable option applies to any ulterior ISRTs to IOPCB made by the second or subsequent transactions.

If a commit mode 0 transaction sent with this option active does a program-to-program switch to another transaction *and* inserts to the IOPCB, the IOPCB output from the second and subsequent transactions will be discarded.

If purge not deliverable is set in the inputting message, and if IMS Connect can send the response to the client and gets a NAK, the output is *not* purged.

Purge not deliverable is not supported for commit mode 1 output (notice that the output from second or posterior transactions in a program to program switch chain when the first transaction has responded is commit mode 0 output), SENDONLY transactions, or RESUME TPIPE interactions. It is not supported for commit mode 1 because there is no queuing of commit mode 1 responses. It does not make sense for SENDONLY because there is no response to purge, and it is not applicable to RESUME TPIPE by its own definition.

7.4.8 Reroute request

If a client application sends a commit mode 0 transaction that generates output that is not deliverable, and purge not deliverable is not in effect, that output gets enqueued in the asynchronous hold queue of the client Tpipe.

This default behavior might not be convenient for several reasons:

- ▶ The client might be using the IMS Connector for Java shareable persistent connections feature, which implies that the clientIDs are automatically generated at random.
- ▶ You might want to use a centralized procedure to retrieve and process all the asynchronous output.

In the first case, there are strong restrictions if you need to issue a RESUME TPIPE to a clientID that is generated automatically by IMS Connector for Java. Briefly, you can only do a RESUME TPIPE if you use the same connection object that sent the transaction that originated that asynchronous output. That means that you must associate RESUME TPIPE interactions with normal SEND interactions, and it is possible that you do not or cannot do that.

In second first case, if you are using several application-managed clientIDs, you must take the appropriate measures to be sure that a specific clientID is not being used twice at the same time, including the case when SEND and RESUME TPIPE interactions are being issued simultaneously by the regular client activity and the specialized RESUME TPIPE process. That means that you must design a serialization procedure, which can become quite sophisticated when you take into account the possibility of running more than one client server or several server clones (if using WebSphere Application Server for z/OS in scalable mode).

IMS Connect lets you request that any undelivered asynchronous output request be routed to a different clientID, which can be processed, if you want, by a specialized process. To use this feature, you must:

- ▶ Specify the value X'08' for IRM_F3. This enables the reroute request.

Note: If you specify both X'04' and X'08' values, meaning purge not deliverable output and reroute request, purge not deliverable supersedes reroute request. That means that you will *lose* your asynchronous output.

- ▶ Optionally, specify an alternate client name using the IRM_ARCH value to X'01' and moving the name to IRM_REROUT_NM.
- ▶ If you do not specify an alternate client name, IMS Connect tries to use the value specified for the RRNAME parameter in the IMS Connect configuration member.
- ▶ If you did not set a value for RRNAME, IMS Connect uses the default value “HWS\$DEF”.

Note: Even beginning with HWS, you *can* use IMS Connector for Java to retrieve the messages queued under HWS\$DEF using a dedicated persistent connection. That is an exception to the rule, which states that the HWSxxxxx Tipes can only be queried using the same shareable persisted connection that was used to create the asynchronous output.

Archived

IMS Connect security

Security deserves increased focus and attention when your IMS systems are Web enabled. Prior to IMS Version 5, the extent of the IMS security domain primarily involved only the IMS systems and their related SNA connections. An exception to this was Advanced Program-to-Program Communication (APPC), which opened up direct connectivity to IMS from remote SNA clients. For these connections, the security environment had to be enhanced to consider the remote client programs and the associated access to those programs.

In IMS Version 5, the introduction of OTMA opened up another area of connectivity from a variety of non-SNA clients, including IMS Connect. With the explosive growth in the requirement to access established mainframe data through the Internet, IMS Connect has moved into the forefront of providing connectivity into IMS from the TCP/IP environment. Therefore, security administration can no longer just reside at the mainframe host level, but also encompasses other platforms and levels of computing.

TCP/IP connectivity offers multiple solutions to connect to a host system. No matter what type of TCP/IP client tries to connect to IMS, each client must do the following actions:

- ▶ Prove their right origin.
- ▶ Declare who they are.

This chapter primarily focuses on the security aspects of IMS Connect itself, but also touches on security measures that are currently available in OTMA and IMS.

8.1 General security overview

A security system must provide the following features:

- Authentication

The security system must be able to determine if a client or a person is who or what it declares to be.

- Authorization

The security system must be able to determine if a client or person has the right to perform an action or to access specific information.

- Secure communication

The security system must be able to ensure that the transmitted information can only be seen by the correct destination.

When we take a look at an IMS Connect-based application, we can see that there are different components that can provide one or more of these features to the overall system. The specific components and the assignment of the different security tasks to them will depend on each development, but as a general rule, we can identify the components and possible security responsibilities shown in Table 8-1.

Table 8-1 Components of an IMS Connect application and security roles

Component	Possible security roles	Notes
Stand-alone (RYO) client	Anyone. Because this is your code, you can put in anything.	<ul style="list-style-type: none">► Try to do as much authentication as possible at the client level. Because you will not have an application server doing authentication for you, if you do not authenticate at the client level, you have to do it at IMS Connect level, which is expensive in terms of resources.► Perhaps you will want to try to implement SSL support in your client so that you will be able to establish a secure, trusted communication with IMS Connect. If you do not do that, you must be sure that your client systems are trusted.
J2EE client	Anyone. Because this is your code, you can put in anything.	<ul style="list-style-type: none">► Follow the J2EE guidelines regarding security, using the container-provided security features whenever possible.► If you need fine-grained authorization controls beyond the J2EE role-based model, you need to perform some programmatic security checking in your EJB or servlet code.
WebSphere Application Server	<ul style="list-style-type: none">► User authentication, using JAAS and SSLv3 at client level.► Client system authentication, using SSLv3.► Authorization, using the J2EE role-based security model.► Secure communication, using SSL, both between WebSphere Application Server and its clients and between WebSphere Application Server and IMS Connect.	<ul style="list-style-type: none">► You should authenticate your users using JAAS.► If you need to authenticate your client machines, you can use the client authentication mechanism provided by SSLv3 and TLSv1.

Component	Possible security roles	Notes
IMS Connect	<ul style="list-style-type: none"> ▶ User authentication, enabling RACF support. ▶ Client system authentication, using SSLv3 and TLSv1. ▶ Authorization, using the IMS Connect security exit. ▶ Secure communication between IMS Connect and the client systems, using SSL. 	<ul style="list-style-type: none"> ▶ Enabling RACF is expensive in terms of resources, because each incoming message generates a RACF call. ▶ IMS Connect Extensions provide a security token caching mechanism that alleviate the resource consumption caused by enabling RACF support. ▶ SSLv3 can be used to authenticate the client systems, so you can avoid the risk of having a <i>trojan client</i> in your network. ▶ You can also set up an internal firewall to protect the IMS Connect ports, so only specific IP-identified clients can connect to these ports.
IMS	Transaction-level authorization, enabling OTMA security. Note that IMS does not do authentication, because it will not verify any passwords.	<ul style="list-style-type: none"> ▶ Notice that IMS does <i>not</i> perform authentication. It ignores any password that you provide. If you rely on RACF-based authorization at the IMS level, ensure that you can trust the user ID that the client sends in the input messages.
IMS application	Anyone. Because this is your code, you can put in anything, based on any part of the input message or the transaction attributes.	<ul style="list-style-type: none"> ▶ Under an OTMA environment, the client can set any attribute of a transaction. For example, if you wrote transactions that LTERM name-based restrictions, you should be aware that the LTERM can be set to any value unless you have restricted it at the user exit level. This means that if you have LTERM-based restrictions, you are implicitly trusting the client application to send the appropriate LTERM name for each “real” user. ▶ There is no IMS sign-on for OTMA-originated transactions. The user ID will be the one sent by the client.

Figure 8-1 on page 112 shows an example of end-to-end security using the components described in Table 8-1 on page 110. In this case, the different elements take the following actions (roles):

1. The user logs on the application using a browser, invoking a secure (SSL) page that prompts the user for the user ID and password.
2. WebSphere Application Server authenticates the user ID and password using the Java Authentication and Authorization Service (JAAS) services.
3. The user, when logged on, sends a request to the application.
4. The server checks if the user, previously authenticated, is authorized to perform the requested action, using the intrinsic, server-based J2EE authorization services, which are also based on JAAS.
5. The application uses the JCA adapter provided by IMS Connector for Java to send a transaction to IMS Connect.
6. The adapter establishes communication with the IMS Connect server, using SSLv3. Both the server and client machines authenticate each other using their respective server and client certificates, so a secure, enciphered communication pipeline is established between WebSphere Application Server and IMS Connect.
7. The adapter sends the transaction through this secure pipeline, including the previously authenticated user ID, but not the password (which is unknown, because the server has not saved it after authenticating the user).

8. IMS Connect does not check the user ID against RACF. Because the pipeline with the JCA adapter is secure and has been authenticated, it trusts the user ID sent by the application and passes it to OTMA.
9. IMS uses RACF to check if the user ID is authorized to run the transaction. If the authorization is correct, it queues the transaction and, eventually, schedules the application program to execute it.

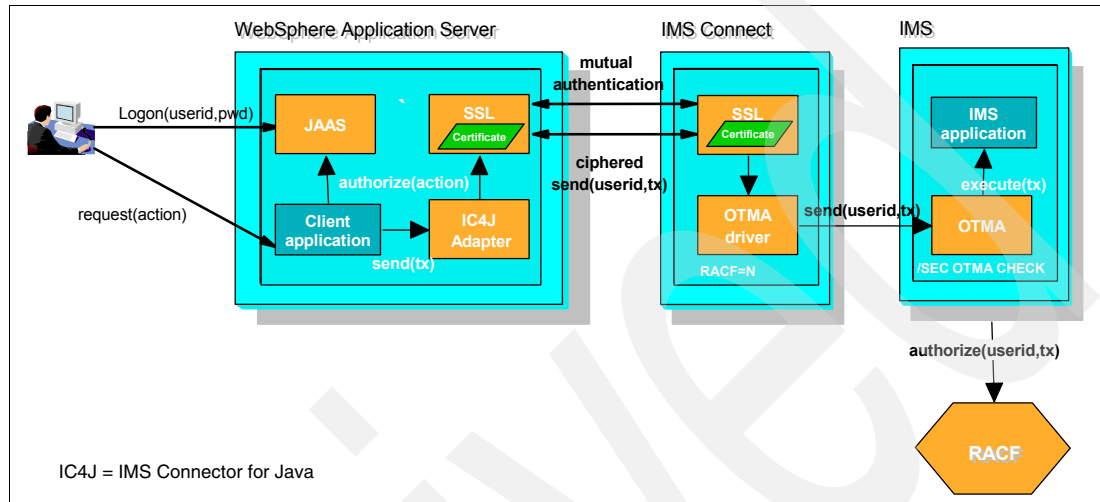


Figure 8-1 End-to-end security using IMS Connector for Java, IMS Connect, and OTMA

If you do not want to enable SSL for performance reasons, you can get nearly as much security (but not privacy) using a firewall between WebSphere Application Server and IMS Connect, making sure that only the designated IP addresses can open a connection to the IMS Connect ports.

8.2 IMS Connect security

IMS Connect allows security support by checking RACF. Security information is passed from clients in the IRM header or OTMA header, and IMS Connect calls the RACROUTE REQUEST=VERIFY macro to verify the user ID and password. In this section, we discuss the IMS Connect security function.

8.2.1 Connecting IMS Connect to OTMA

Before IMS Connect can send any messages to IMS through OTMA for processing, IMS Connect has to identify itself as an OTMA client. This is achieved by issuing an OTMA command of the type client bid, passing the required security data to OTMA for verification. If your IMS is RACF protected, the user ID passed by IMS Connect in the client bid must have READ access to the FACILITIES class entry `IMSXCF.xcf_group_name.xcf_member_name (client)` in RACF. If rejected by RACF, the client bid is denied and IMS Connect cannot connect to IMS as an OTMA client.

8.2.2 User verification

After IMS Connect has successfully joined the XCF group and connected as an OTMA client to IMS, you have the option to let IMS Connect do RACF user ID and password verification of each client on a per-message basis. This facility is driven by the RACF=Y | N parameter, as specified in the HWSCFG configuration file. See 4.3.4, “Creating the IMS Connect configuration member” on page 47 for an example.

You can modify the RACF status by using the IMS Connect command SETRACF=ON | OFF. See 5.1.6, “SETRACF” on page 66 for an example. The user ID and password can be set up in one of two places:

- ▶ The originating client can build and send the security data as part of the message that is sent to IMS Connect through TCP/IP.
- ▶ The user message exit that gets driven after IMS Connect received the complete message from the TCP/IP client.

After IMS Connect receives control back from the user message exit and the RACF= option is set to Y, IMS Connect issues the RACF call to verify the user ID and password. If not authorized, the message is rejected and sent back to the originating client.

Note: IMS Connect Extensions provide RACF identification and authorization caching. This can dramatically improve IMS Connect performance. See Chapter 11, “IMS Connect Extensions” on page 155 for details.

8.2.3 User exit security

The last option available with IMS Connect security is available in the user exit that is driven by IMS Connect after the complete message has been received from the TCP/IP client. This option is really open ended to such a degree that the user exit can perform any data manipulation, or checking that it wants to do, which can include RACF verification or any other security verification that the author of the exit wants to implement and execute. This is totally separate from the optional user ID and password verification performed by IMS Connect, as discussed previously in this chapter.

8.2.4 Local option security

To configure security for the local option using RACF, you must add HWS.IMSConnect_name as the SAF FACILITY class name (whether you configured security with the IMS Connect configuration member or the SETRACF command). IMSConnect_name is how IMS Connect is defined in the ID parameter of the HWS statement in the IMS Connect configuration member. The resource that must access IMS Connect is WebSphere Application Server, and UPDATE authority is required to update the RACF profile.

8.3 OTMA security

OTMA provides various levels of security checking that can be implemented, which is independent of the optional RACF verification that can be performed by IMS Connect. OTMA verifies security in two instances:

1. When a client bids to connect
2. When an input message from a client is processed

Client bids to connect

When a client (in our case, IMS Connect) sends a client bid command to OTMA (to connect), OTMA has to verify that the client is sufficiently authorized to connect to OTMA. The data passed in the security-data section of the client bid message is used by RACF to verify the client's authorization. The `IMSXCF.xcf_group_name.xcf_member_name (client)` must be defined to RACF in the FACILITY class. IMS Connect adheres to this protocol, as discussed previously (see 8.2.1, "Connecting IMS Connect to OTMA" on page 112).

Processing an input message from the client

When OTMA processes an input message, whether it is command or transaction input, security is driven by the:

- ▶ Status as set by the OTMASE= parameter in DFSPBxxx member
- ▶ Status as set by the IMS /SECURE OTMA command
- ▶ Security data in the OTMA header

If the OTMA security status is PROFILE, security verification can vary on a transaction-by-transaction basis, driven by the settings in the security section of the OTMA header. We recommend using security at the PROFILE, because security requirements can vary on a per-transaction basis. This provides the flexibility for an installation to implement customized security profiles for each transaction or command separately.

If OTMA security status is CHECK, security checking of commands and transactions is performed subject to the appropriate definitions of the commands and transactions in the security profile. If RACF is used for security, commands and transactions that must be verified have to be defined to the appropriate CIMS and TIMS classes. Failure to state these definitions causes the command or transaction to be allowed by default.

If OTMA security status is FULL, the same security checking as in CHECK is performed, but additional RACF calls are used internally by IMS to build and maintain the security environment on dependent regions.

If OTMA security status is NONE, it causes OTMA security checking to be bypassed, irrespective of what is set up in the security data section of the OTMA header, whether security data is present or not.

Refer to the 2.5, "OTMA security issues" on page 21 for further information about OTMA security.

Note: The user exits, DFSCCMD0 and DFSCTRNO, if present, will still be invoked, irrespective of the status of the /SECURE OTMA command. This provides the facility to drive any user-written security from the appropriate exits if wanted.

IMS Connect user exit support

IMS Connect is essentially middleware or a gateway that acts as an intermediate communication vehicle between the TCP/IP client-initiated messages and the datastore at the host (IMS, through OTMA). Because each client message can be unique in design, format, processing, and security requirements, data manipulation of such messages into a minimum level of IMS Connect-recognizable format is essential for the ability of IMS Connect to act as the go between for both the client and the datastore.

To achieve this minimum level of recognizable format, IMS Connect provides support for user exits to receive and manipulate messages from both the originating client and the IMS server. These exits can be customized to perform various functions such as reformatting, translation, and security verification. They can be regarded as plug-ins to IMS Connect, so the central or kernel of the IMS Connect logic is isolated from the message manipulation exits.

IBM delivers prewritten clients and their associated user exits for simplified development. However, customers are also able to create and develop their own TCP/IP clients, with their associated user exits, according to their installation requirements and needs.

9.1 IMS Connect components and user exits

IMS Connect itself has many functions to perform and is broken down into various components (see Figure 9-1). The compartmentalization of the different functions provides the platform to divide IMS Connect into the different components and keep the functions isolated and modular. Because IMS Connect performs several distinct communication functions, TCP/IP client communication, IMS OTMA communication, and IMSplex communication, the exits are involved in all instances, because message manipulation is a factor in both directions.

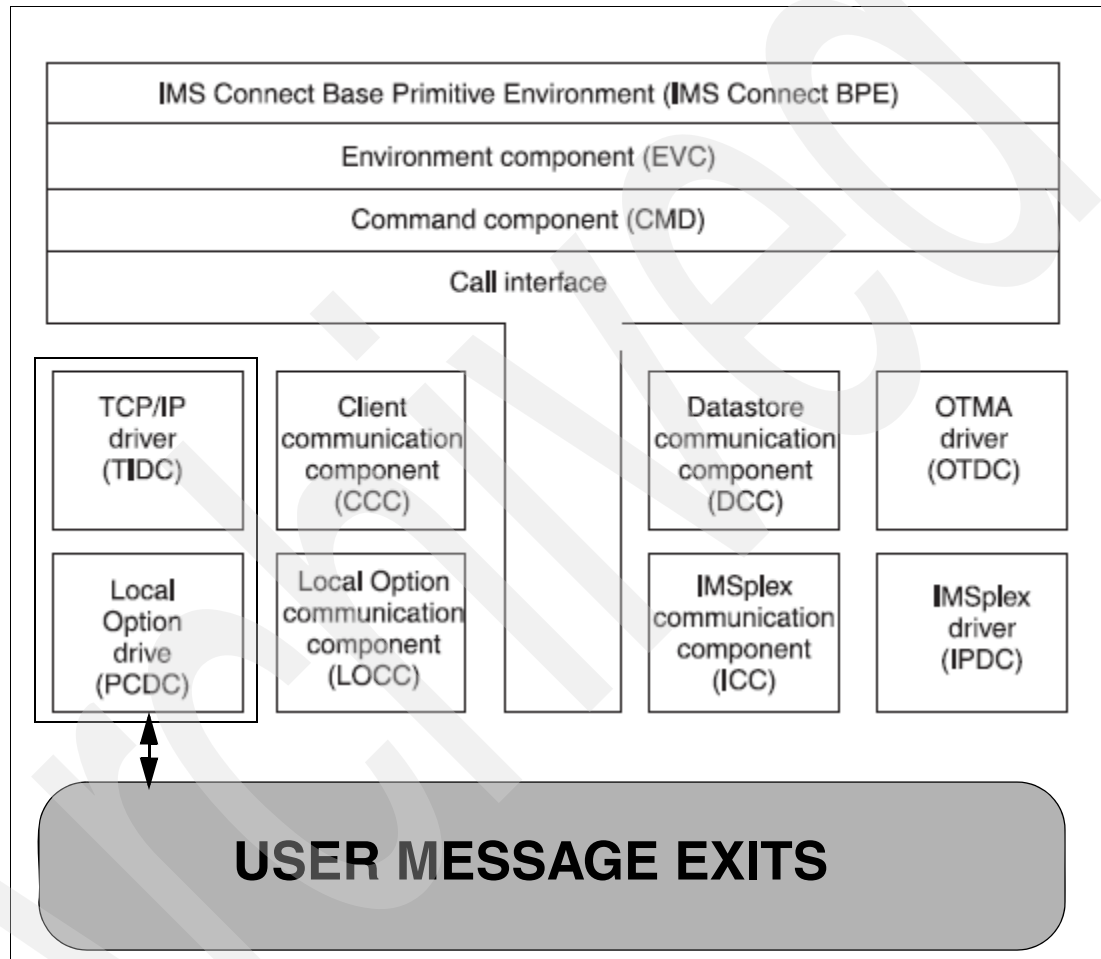


Figure 9-1 User exits are invoked on incoming and outgoing messages

9.2 IMS Connect communication with user exits

The purpose of the user exits in IMS Connect is to format and possibly modify the incoming or outgoing messages.

Commonalities among all user exits

Any user exit that is associated with an input message and needs to be invoked by IMS Connect to process the message must be defined in the EXIT= parameter in the HWSCFG configuration file. A maximum of 254 exits can be defined here.

Restriction: Do not define the IBM-supplied exits (HWSUINIT and HWSJAVA0) in the HWSCFG configuration file. IMS Connect automatically loads the HWSJAVA0 exit, and HWSUINIT is an IMS Connect exit but is not a user message exit.

With the exception of HWSCSLO0 and HWSCSLO1, which are used to support the IMS Control Center, IMS Connect invokes the defined user exits in the following instances:

- ▶ During IMS Connect initialization time.
- ▶ After receiving the complete message from the client and prior to passing it to OTMA.
- ▶ After receiving control back from the user exit during the input phase and if IMS Connect detects an error in the output buffer setup by the user exit.
- ▶ After receiving the IMS response from OTMA and prior to passing the output message to the client.
- ▶ During IMS Connect termination.

IMS Connect does *not* invoke a different exit for each of the previous instances, but expects an exit to be able to process each of these phases. This is dictated by an option specified in the parameter list passed to the exit. These options are:

- ▶ INIT (initialization)
- ▶ READ (input phase)
- ▶ EXER (error on input phase)
- ▶ XMIT (output phase)
- ▶ TERM (termination)

HWSCSLO0 and HWSCSLO1 are a bit different, because they deal with IMSplex communication. These two exits are provided by IBM as object code only (OCO) and cannot be customized, so we do not provide more information about these exits in this chapter.

All exits are expected to function in a reentered environment and are given control in key 7 and in the addressing mode with which the exit was linked. The parameter list passed to the exit is always in storage located above the 16 MB virtual storage line, so all exits should be able to perform 31-bit addressing at the minimum, even if the author of such an exit wants to restrict the exits location to RMODE 24.

IMS Connect provides a 1 KB buffer work area with each invocation of an exit. The purpose of the work buffer is to ensure that the reentered environment is maintained and to provide a unique storage area where registers can be saved. Although the general purpose register 13 points to an IMS Connect save area at entry to an exit, an automatically pre-allocated follow-on save area is not provided. This means that the exit must provide its own save area in the 1 KB work buffer and backchain this unique save area to the IMS Connect save area. The exit can, on entry to the exit, save the status of the general purpose registers in the IMS Connect save area, as pointed to by general purpose register 13, and subsequently restore them from there when the exit returns control to IMS Connect. IMS Connect supports assembler-written user exits only. Table 9-1 shows the common register contents on entry to all user exits.

Table 9-1 Register contents on entry to user exits (all options)

Register	Contents on entry
1	Pointer to parmlist, as mapped by HWSEXPRL.
13	IMS Connect save area.
14	Return address of IMS Connect.

Register	Contents on entry
15	Entry point address of user exit. The entry point name and the load module name, as described in the HWSCFG configuration file, must be the same.

Table 9-2 shows the common register contents at exit to all user exits.

Table 9-2 Register contents on return from user exit (all options)

Register	Contents on return
1	Pointer to parmlist, as mapped by HWSEXPRL, which includes the return code
13	IMS Connect save area
14	Return address of IMS Connect
15	Not used

For more information about the communication between user exits and IMS Connect, refer to *IMS Version 9: IMS Connect Guide and Reference*, SC18-9287.

9.3 User exits supported

IBM currently delivers 10 supported user exits. Six of these exits are message exits, which are also associated with predefined IRM_ID identifiers in IRM header. IBM provides source code for three of these exits, and object code only for the others. Table 9-3 summarizes all of the user exits.

Table 9-3 User exits summary

User exit name	IRM_ID	Purpose
HWSIMSO0	*IRMREQ*	IBM-delivered user exit in support of any TCP/IP client that conforms to the message format dictated by this exit. Format=OCO only. IMS Version 9 is the last version that supports this exit. We recommend that you migrate to HWSSMPL1.
HWSIMSO1	*IRMRE1*	IBM-delivered user exit in support of any TCP/IP client that conforms to the message format dictated by this exit. It has a full word length field proceeding the message. Format=OCO only. IMS Version 9 is the last version that supports this exit. We recommend that you migrate to HWSSMPL1.
HWSSMPL0	*SAMPLE*	IBM-delivered sample user exit for user modification. This exit is based on HWSIMSO0. Format=Object and source. We recommend that you migrate to HWSSMPL1 because the new function will no longer be added to HWSSMPL0.
HWSSMPL1	*SAMPL1*	IBM-delivered sample user exit for user modification. This exit is based on HWSIMSO0. It has a full word length field proceeding the message. Format=Object and source.
HWSJAVA0	*HWSJAV*	IBM-delivered sample user exit in support of the IMS Connector for Java client. Format=Object and source.

User exit name	IRM_ID	Purpose
IMSLSECX	N/A	Security exit. IBM supplied with TCP/IP for z/OS. Format=OCO only.
HWSUINIT	N/A	Enables you to perform your own processing during IMS Connect initialization and termination. This exit receives control at initialization and termination time. Format=Object and source.
HWSTECL0	N/A	Enables you to customize IMS Connect to support event recording. Format=Object and source.
HWSCSLO0	N/A	IMSplex support exit. Enables the IMS Connect capability to communicate with IMS Operations Manager (OM). This is necessary to use the IMS Control Center. Format=OCO only.
HWSCSLO1	N/A	Same as HWSCSLO0.

Note: 'HWSxxxxx' exit name is reserved as the prefix for IRM_ID.

9.3.1 IMS Connect TCP/IP user message exit (HWSIMSO0 and HWSIMSO1)

HWSIMSO0 and HWSIMSO1 exits are generic exits to process messages from any TCP/IP client. These TCP/IP exits are shipped with IMS Connect and they should be bound into the IMS Connect load library (SDFSRESL/SHWSRESL). You must use these TCP/IP exits instead of the one shipped by TCP/IP. The installation must place the IMS Connect load library that contains the IMS Connect supplied exit (HWSIMSO0 and HWSIMSO1) in front of the TCP/IP load library. The HWSIMSO0 and HWSIMSO1 exits are shipped as object code only (OCO).

The IMS Connect version bundled with IMS Version 9 is the last one that will provide the HWSIMSO0 and HWSIMSO1 exit routines. You should not use these exits. We recommend that you migrate to HWSSMPL1 as soon as you can. For details, see the description of HWSSMPL0 and HWSSMPL1 exits in the following section.

9.3.2 Sample user message exit (HWSSMPL0 and HWSSMPL1)

HWSSMPL0 and HWSSMPL1 are generic exits to process messages from any TCP/IP client. These sample user message exits perform the same functions as the IMS Connect HWSIMSO0 and HWSIMSO1 exits, which cannot be modified and are considered deprecated. You can modify the source code for the HWSSMPL0 or HWSSMPL1 exit, which are supplied with the IMS Connect installation.

Details

These exits have the following characteristics:

- ▶ Commit mode
Default=send-then-commit (can be overridden in IRM header extension).
- ▶ Sync level
Default=none (can be overridden in IRM header extension).

- ▶ Translation
Exit performs the translation from ASCII to EBCDIC (input phase) and from EBCDIC to ASCII (output phase). For Unicode support, refer to 14.3, “IMS Connect Unicode support” on page 275.
- ▶ OTMA header
Builds OTMA header structures (input phase) and strips them off (output phase).
- ▶ Input message structure passed to HWSSMPL0 and HWSSMPL1 exits
Refer to Table 9-9 on page 128. See also Table 14-1 on page 269.
- ▶ Input message structure returned from HWSSMPL0 and HWSSMPL1 exits
Refer to Table 9-10 on page 130.
- ▶ Output message structure passed to HWSSMPL0 and HWSSMPL1 exits
Refer to Table 9-12 on page 132.
- ▶ Output message structure returned from HWSSMPL0 and HWSSMPL1 exits
Refer to Table 9-16 on page 134, Table 9-17 on page 135, and Table 9-18 on page 135.
See also 14.2.2, “The IMS Connect output message” on page 274.

9.3.3 Difference between HWSIMSO0/SMPL0 and HWSIMSO1/SMPL1

The HWSSMPL1 message exit (and its predecessor, HWSIMSO1) sends a full word length field preceding the output message sent to the client. We recommend that any customer using HWSIMSO0 and HWSSMPL0 migrate to this new user message exit, which includes the total message length header (LLLL) preceding the output message to the client.

This is the only difference between exits HWSIMSO0 and HWSSMPL0 and HWSIMSO1 and HWSSMPL1. Providing different message ID values in the IRM field IRM_ID of *SAMPL1* and *IRMRE1* enables all four user message exits to be present at the same time and allows for an easier migration to the user message exit HWSSMPL1.

HWSSMPL0 and HWSIMSO0 currently are capable of sending the following three message structures to the client, one of the three at a time:

```
LLZZDATA1 LLZZDATA2 ... CSM
RMM LLZZDATA1 LLZZDATA2 ... CSM
RSM
```

HWSSMPL1 and HWSIMSO1 are capable of sending the following three structures to the client, one of the three at a time:

```
LLLL LLZZDATA1 LLZZDATA2 ... CSM
LLLL RMM LLZZDATA1 LLZZDATA2 ... CSM
LLLL RSM
```

Note: LLLL is the total length of the output messages.

Migrating to HWSSMPL1 considerations

If you have modified HWSSMPL0, there will be few, if any, changes required to move to the new user message exit HWSSMPL1. You can also move the changes from HWSSMPL1 to your modified copy of HWSSMPL0.

Basically, the following actions are required to use the new exit:

- ▶ Add HWSSMPL1 to EXIT= in the IMS Connect configuration file.

- ▶ Modify the client application code to send the value of `"*SAMPL1*"` for HWSSMPL1 rather than `"*IRMREQ*"` or `"*SAMPLE*"` in the IRM field `'IRM_ID'`.
- ▶ Modify the client application code to handle the LLLL (total length field) that is now being sent to the client application.

Note: The sample code provided in Appendix A, "Sample code: Non-IMS Connector for Java client code" on page 469 and described in 14.5, "Detailed code examples" on page 282 uses the HWSSMPL0 user message exit.

9.3.4 IMS Connector for Java user message exit (HWSJAVA0)

The purpose of the HWSJAVA0 exit is to process input from the IMS Connector for Java client. This IMS Connector for Java client exit is shipped with IMS Connect, and it should be bound into the installation load library (SDFSRESL/SHWSRESL). This exit does not perform a translation or build to the OTMA headers. Both the translation and insertion or deletion of the OTMA header is done by the IMS Connector for Java client server. HWSJAVA0 is supplied as source code and can be modified.

Details

The HWSJAVA0 exit has the following characteristics:

- ▶ Commit mode
Default=send-then-commit (using the appropriate methods of the IMSConnectionSpec object, which translates into different IRM header settings).
- ▶ Sync level
Default=CONFIRM for CM0 and NONE for CM1 (*cannot* be overridden in the IRM header extension).
- ▶ Translation
None.
- ▶ OTMA header
It does *not* build the OTMA header structures. The OTMA header structures are built by the IMS Connector for Java client library.
- ▶ Input message structure passed to HWSJAVA0 exits
Refer to Table 9-8 on page 127.
- ▶ Input message structure returned from HWSJAVA0 exits
The IMS Connector for Java exit returned message format is the same message format of the passed message format.
- ▶ Output message structure passed to and returned from HWSJAVA0 exits
IMS Connector for Java output messages are not passed to the exit. Refer to Table 9-11 on page 131.

9.3.5 IMS Connect IMSplex message exits (HWSCSLO0 and HWSCSLO1)

The purpose of the HWSCSLO0 and HWSCSLO1 exits is to provide IMSplex support to enable the IMS Control Center. These routines are provided in object code, and you cannot modify or replace them.

Details

These exits have the following characteristics:

- ▶ Commit mode
Default=send-then-commit (can be overridden in IRM header extension).
- ▶ Sync level
Default=none (can be overridden in IRM header extension).
- ▶ Translation
The exits translate ASCII to EBCDIC and build the required message structure containing the required internal headers for messages received from the client. HWSSCLO0 performs EBCDIC to ASCII translation and removes the internal headers from messages transmitted to the client. HWSSCLO1 does not perform any translation but it does remove the headers.
- ▶ Input message structure passed to HWSSCLO0 and HWSSCLO1 exits
Refer to Table 9-9 on page 128.
- ▶ Output message structure returned from HWSSCLO0 and HWSSCLO1 exits
Refer to Table 9-16 on page 134, Table 9-17 on page 135, and Table 9-18 on page 135.

9.3.6 Security exit (IMSLSECX)

The purpose of this exit is to perform user authentication for the other user exits. You must provide a security exit (or use the TCP/IP exit, IMSLSECX) if any security checking is to be done by the message exit. Due to the many options available for security, and because most installations have their own specific security method, no sample security exit is provided. The call to RACF is performed by IMS Connect if RACF parameters are provided in the OTMA header when the message exit returns the message.

The name of the security exit called by HWSSMPL0, HWSSMPL1, HWSIMSO0, or HWSIMSO1 is IMSLSECX. You can change the name of the security exit called by HWSSMPL0 or HWSSMPL1 and supply and define it in the HWSSMPL0 and HWSSMPL1 message exit by changing the EXTRN IMSLSECX to a name of your choice. If you require a different security exit in HWSSMPL0 or HWSSMPL1, you must provide the new security exit name.

You must also provide the name of the security exit called by HWSJAVA0 and define it in the HWSJAVA0 message exit.

The parameter list for user security exit

The following is the list and order of parameters being passed to the security exit, IMSLSECX. The order of the parameters is fixed for the exits supplied by IMS Connect (HWSSMPL0, HWSSMPL1, HWSIMSO0, and HWSIMSO1).

- ▶ Address of full word client's IP address
- ▶ Address of halfword client's port
- ▶ Address of 8-character string IMS transaction
- ▶ Address of halfword data type (data type setting: 0=ASCII, 1=EBCDIC)
- ▶ Address of full word length of user data
- ▶ Address of user-supplied data
- ▶ Address of full word set by security exit
- ▶ Address of full word set by security exit

- ▶ Address of RACF user ID
 - If blanks are returned (in the field pointed to) from the security exit, the RACF fields in the OTMA security header are not set.
 - The address points to a field containing blanks.
- ▶ Address of RACF group ID
 - The address points to a field containing blanks.

9.3.7 User initialization exit routine (HWSUINIT)

IMS Connect provides a user initialization exit routine, HWSUINIT. This routine enables you to perform customized initialization tasks during IMS Connect startup and customized termination tasks during IMS Connect shutdown.

Important: The HWSUINIT user initialization exit routine shipped with IMS Connect does not do any processing. Modify HWSUINIT only if you want to use it.

How IMS Connect communicates with HWSUINIT

HWSUINIT contains two subroutines: INIT and TERM. When IMS Connect starts, HWSUINIT loads and gives control to the INIT subroutine. When IMS Connect shuts down, HWSUINIT gives control to the TERM subroutine.

HWSUINIT contains two of its own user control blocks: XIB and XIBDS. The HWSXIB and HWSXIBDS DSECTs map the XIB and XIBDS user control blocks. The message exit routines in the INIT, READ, XMIT, TERM, and EXER subroutines can also use the XIB and XIBDS user control blocks.

Example 9-1 shows the DSECT of the XIB control block user area (from the HWSXIB macro).

Example 9-1 The DSECT of XIB control block

HWSXIB	DSECT	Exit	Interface Block

* XIB Header *			

XIB_HEADER	DS	0D	
XIB_EYE	DS	CL4'XIB'	EYECATCHER
XIB_DATASTORES	DS	A	DataStore list address
XIB_UFLD_CNT	DS	F	User field count
	DS	6F	Reserved for ITOC
XIB_HDR_LEN	EQU	*-HWSXIB	XIB header fixed length

* XIB User areas *			
* The number of user area words is specified by the XIBAREA parameter *			
* of the HWS statement in the configuration file. The default is 20 *			
* fullwords if it is not specified. The number is also stored in *			
* XIB_UFLD_CNT field in the XIB header above. *			

XIB_USERAREA	DS	0F	Start of user area

The XIB user control block contains a fixed length header section and a variable length user area. You cannot modify the fixed header section. You can only modify the user area.

You specify the size of the XIB control block user area, in full words, with the XIBAREA parameter (in the HWS statement of the IMS Connect configuration file). Refer to 4.3.4, “Creating the IMS Connect configuration member” on page 47 for more information about the IMS Connect configuration file.

Example 9-2 shows the DSECT of the XIBDS control block user area (from the HWSXIBDS macro).

Example 9-2 The DSECT of XIBDS control block

HWSXIBDS	DSECT	Exit	Interface Block Data Store entry
XIBDS_NAME	DS CL8		Data store name
XIBDS_STATUS	DS X		Data store status
XIBDS_INACTIVE	EQU X'00'		Data store not active
XIBDS_ACTIVE	EQU X'01'		Data store active
XIBDS_DISC	EQU X'02'		Data store disconnected @PQ77003
XIBDS_FLAG	DS X		Data store entry flags
XIBDS_LAST_ENTRY	EQU X'80'		Last entry in list
	DS XL2		Reserved
XIBDS_USER	DS XL4		User field
XIBDS_LEN	EQU *-HWSXIBDS		XIB data store entry length
	MEND		

The XIBDS user control block represents an entry in a list of datastores that are defined in the configuration file. The second word in the fixed header area of the XIB user control block points to the datastore list. The XIBDS user control block is 16 bytes long. Each datastore list entry contains the datastore name, the datastore status (active or inactive), a flag byte, and a 4-byte field that you can use to store any kind of data. The last entry is indicated by a value of x'80' (hexadecimal) in the flag byte. The number of entries in the list is equal to the number of datastores defined in the IMS Connect configuration file.

HWSUINIT usage

You can modify the HWSUINIT routine to display a specific message when IMS Connect starts up or shuts down.

In addition, because the XIBDS user control block keeps track of the IMS Connect datastore status, you can enable any user message exit (HWSSMPL0, HWSSMPL1, or HWSJAVA0 and user written exit) to take action based on the status of one or more of the IMS Connect datastores. Figure 9-2 on page 125 shows a sample of the HWSUINIT usage. For example, before a user message exit passes a client message to an IMS Connect datastore for processing, you can have the user message exit query the XIBDS control block area for the status of the target datastore. If the target datastore is not active, you can enable the user message exit to switch to an active datastore by modifying the datastore name in the message header. Or, your client sends a generic datastore name in IRM header, and the user message exit determines which IMS is most appropriate for processing and modifies the datastore name to action round-robin requests across the active IMSs.

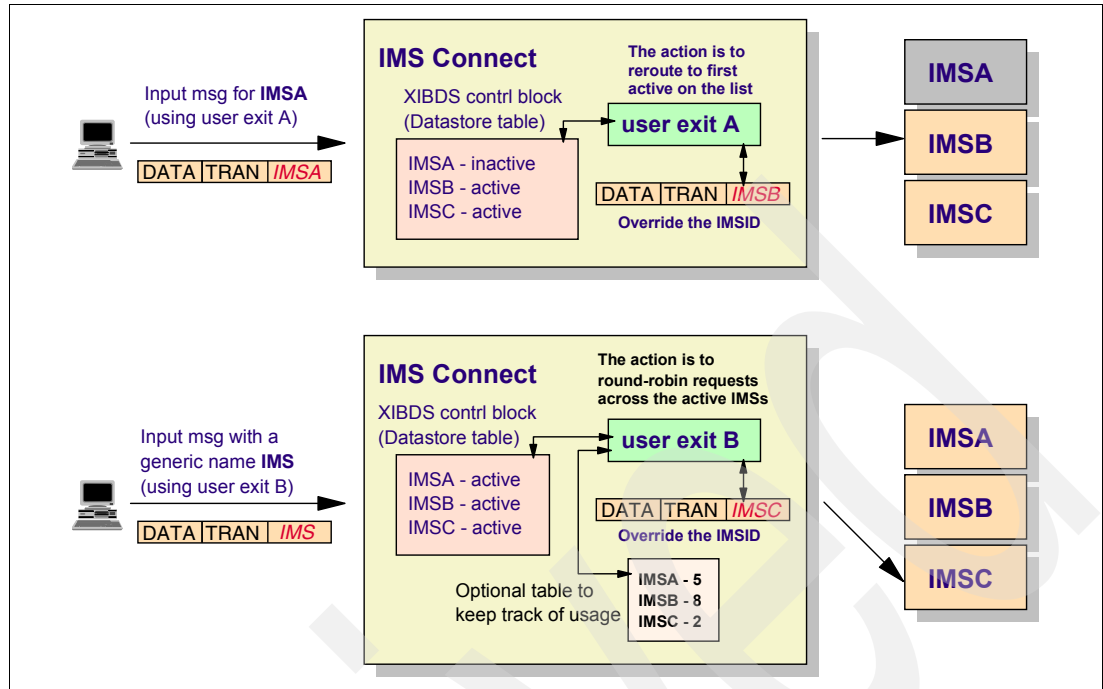


Figure 9-2 A sample of HWSUINIT usage

Table 9-4 shows the common register contents on entry to HWSUINIT.

Table 9-4 Register contents on HWSUINIT entry

Register	Contents on entry
1	Pointer to a parmlist: <ul style="list-style-type: none"> ► +0: XIB address ► +4: Function to perform (INIT or TERM) ► +8: 1 KB buffer for exit to use
14	Return address of IMS Connect
15	Entry point address to HWSUINIT

Table 9-5 show the common register contents at exit to HWSUINIT.

Table 9-5 Register contents on HWSUINIT exit

Register	Contents on entry
0-14	Restored
15	<ul style="list-style-type: none"> ► 0: Completed successfully ► 1 to 7: Warning, but IMS Connect initialization continues ► 8 or higher: Force IMS Connect termination

9.3.8 Event recording user exit (HWSTECL0)

The purpose of the event recording user exit (HWSTECL0) is to support event recording. IMS Connect can be customized to pass event data to this load module, which stores all trace and event notifications through a recording routine and can be used by any event recording function. Some examples of the events that can be recorded include:

- TCP/IP read/write

- ▶ RACF calls
- ▶ OTMA send/receive
- ▶ User exit calls
- ▶ Session errors
- ▶ Two-phase commit events

Modifying the HWSTECL0 user exit

Although IMS Connect provides a sample HWSTECL0 user exit, you must modify it, using standard user exit development guidelines, if you want to receive event data from IMS Connect. The following steps describe how to customize, modify, and reinstall the HWSTECL0 exit:

1. Insert your changes to the source code provided in the AHWSSRC source library.
2. Assemble the exit. The exit and its associated macro files are members of the partitioned data set into which you receive the AHWSSRC data set.
3. Bind (link-edit) the output from the assembled job to create a load module named HWSTECL0.
4. Bind (link-edit) HWSTECL0 into the IMS Connect resource library (SDFSRESL or SHWSRESL). IMS Connect loads the module from the resource library during initialization.

Table 9-6 shows the contents of the registers at entry of the event recording routine.

Table 9-6 Registers at event recording entry

Register number	Contents and meaning
R1	Address of the Event Recording Parameter List (ERPL).
R13	Address of one save area. The event recording routine must preserve the integrity of this save area.
R14	Caller's return address.
R15	Entry point of the event recording routine, taken from EICB after initialization of the event recording interface.

Table 9-7 shows the expected contents of the registers at the return of the event record routine.

Table 9-7 Registers at event recording exit

Register number	Contents and meaning
R0	Reason code associated with any non-zero return codes.
R1	When R1 is not equal to zero, it contains the address of a message providing additional information about initialization of trace and event recording.
R15	Return code: <ul style="list-style-type: none"> ▶ 0=event recording was successful. ▶ 4=event recording is not active; event was not recorded. ▶ 16=event recording was not successful. See the reason code for additional information. An error message is present if r1 is not zero.

For more information about the event recording user exit, refer to *IMS Version 9: IMS Connect Guide and Reference*, SC18-9287.

9.4 Message structures between IMS Connect and user exits

The following tables show details of these different message structures.

9.4.1 Input message from client and passed to exit

Input messages from the client consist of IMS Connector for Java and non-IMS Connector for Java message structures.

IMS Connector for Java message structure, type 1

Table 9-8 shows the input message format supported by IMS Connect from an IMS Connector for Java client.

Table 9-8 IMS Connector for Java message structure, type 1

Field	Length (bytes)	Meaning
IIII	4 (binary)	Length of entire message, including III field.
The first 28 bytes represent the fixed IRM header.		
IRM_LL	2 (binary)	Length of IMS Connector for Java interface header, including LLZZ field.
IRM_ARCH	1 (binary)	Architectural level: <ul style="list-style-type: none">► X'00' Base support► X'01' Required space for IRM_REROUT_NM field.
IRM_F0	1 (binary)	Reserved (set to binary zeros).
IRM_ID	8 (character)	Char value of *HWSJAV*.
Reserved	4 (binary)	Reserved (set to binary zeros).
IRM_F5	1 (binary)	Binary value for input message type. In case of HWSJAVA0, set to X'11000000'.
IRM_TIMER	1 (character)	N/A for HWSJAVA0.
IRM_SOCT	1 (binary)	Socket connection type. The client can set this value as follows: <ul style="list-style-type: none">► X'00' Transaction socket► X'10' Persistent socket
IRM_RSV02	1 (binary)	Reserved for future use. Set to binary zeros.
IRM_CLIENTID	8 (character)	Char value of a unique clientID.
End of IRM header.		
OTMA header	466	See OTMA DSECT member HWSOMPFX in IMS.SDFSMACT library for a description.
LL	2 (binary)	Length of data segment.
ZZ	2 (binary)	Reserved (set to binary zeros).
DATA	n	User data.
OTMA CTL headers	20	See OTMA DSECT member HWSOMPFX in IMS.SDFSMACT library for a description.
LL	2 (binary)	Length of data segment.

Field	Length (bytes)	Meaning
ZZ	2 (binary)	Reserved (set to binary zeros).
DATA	n	User data.
Repeat OTMA CTL header, LL, ZZ, DATA for all segments.		
OTMA CTL headers	20	See OTMA DSECT member HWSOMPFX in IMS.SDFSMACT library for a description.
LL	2 (binary)	Length of data segment.
ZZ	2 (binary)	Reserved (set to binary zeros).
DATA	n	User data.

Non-IMS Connector for Java message structure, type 2

Table 9-9 shows the input message format supported by IMS Connect from a non-IMS Connector for Java client. In this section, we provide a description of this structure from the point of view of an IMS Connect systems programmer. If you are interested in a description for a client applications programmer, refer to 14.2, “IMS Connect message structures” on page 268.

Table 9-9 Non-IMS Connector for Java message structure, type 2

Field	Length (bytes)	Meaning
IIII	4 (binary)	Length of entire message, including III field.
The first 28 bytes represent the fixed IRM header.		
IRM_LL	2 (binary)	Length of IMS Connector for Java interface header, including LLZZ field.
IRM_ARCH	1 (binary)	Architectural level: <ul style="list-style-type: none"> ► X'00' Base support ► X'01' Required space for IRM_REROUT_NM field
IRM_F0	1 (binary)	Reserved (set to binary zeros).
IRM_ID	8 (character)	Char value of the identifier of the user exit: <ul style="list-style-type: none"> ► *IRMREQ* for HWSIMSO0 ► *IRMRE1* for HWSIMSO1 ► *SAMPLE* for HWSSMPL0 ► *SAMPL1* for HWSSMPL1 ► *HWSJAV* for HWSJAVA0
Reserved	4 (binary)	Reserved for future use.
IRM_F5	1 (binary)	Input message type and RESUME TPIPE processing: <ul style="list-style-type: none"> 1000 : OTMA headers built by client 0100 : Translation done by client 0000 : OTMA header built and translation done by user exit (default) 0000 : No auto flow of messages 0001 : Single message 0010 : Auto flow of messages 0100 : No auto flow of messages
IRM_TIMER	1 (binary)	Receive after ACK/RESUME TPIPE wait time.

Field	Length (bytes)	Meaning
IRM_SOCT	1 (binary)	Socket connection type. The client can set this value as follows: <ul style="list-style-type: none"> ▶ X'00' Transaction socket ▶ X'10' Persistent socket
IRM_RSV02	1 (binary)	Reserved for future use. Set to binary zeros.
IRM_CLIENTID	8 (character)	Char value of a unique clientID.
End of fixed IRM header.		
The following definition is for use with the HWSIMSO0 and HWSSMPL0 exits. The user installation can provide its own exit and structure the following items as required by the user exit. The following items should be considered. This example lists only some of the items you can use. You might want to include fields that are used only by the user exit, or other items that can be passed in the OTMA headers, such as the MID name.		
IRM_F1	1 (binary)	Specify if the MFS MOD name is to be returned: <ul style="list-style-type: none"> ▶ X'00' User requests no MOD name to be returned. ▶ X'80' User requests MOD name to be returned.
IRM_F2	1 (binary)	Commit mode flag: <ul style="list-style-type: none"> ▶ X'40' commit-then-send (commit mode 0) ▶ X'20' send-then-commit (commit mode 1)
IRM_F3	1 (binary)	Sync level flag: <ul style="list-style-type: none"> ▶ X'00' Sync level=None ▶ X'01' Sync level=Confirm ▶ X'02' Sync level=Sync Point ▶ X'04' Purge not deliverable request ▶ X'08' Reroute not deliverable request
IRM_F4	1 (character)	It specifies if the client is sending: <ul style="list-style-type: none"> ▶ A=ACK - Positive acknowledgment ▶ N=NAK - Negative acknowledgment ▶ D=DEALLOCATE - Deallocate connection ▶ R=RESUME - RESUME TPIPE ▶ S=SENDONLY - Send only ▶ C=CANCEL TIMER - Cancel pending timer ▶ Blank (X'40') - No request for acknowledgment or deallocation
IRM_TRANCOD	8 (character)	Character value of transaction code.
IRM_IMSDESTID	8 (character)	Character value of datastore ID.
IRM_LTERM	8 (character)	Character value of LTERM override name.
IRM_RACF_USERID	8 (character)	Character value of RACF user ID.
IRM_RACF_GRPNAME	8 (character)	Character value of RACF group name.
IRM_RACF_PW	8 (character)	Character value of PassTicket/password.
IRM_APPL_NM	8 (character)	RACF APPL name defined to RACF on the PTKDATA definition (not supported in HWSIMSO0 or HWSIMSO1).
IRM_REROUT_NM	8 (character)	Reroute name for the client reroute request.
The following is the data structure for all non-IMS Connector for Java clients.		
LL	2 (binary)	Length of data segment.

Field	Length (bytes)	Meaning
ZZ	2 (binary)	Reserved (set to binary zeros).
DATA	n	User data.
LL	2 (binary)	Length of data segment.
ZZ	2 (binary)	Reserved (set to binary zeros).
DATA	n	User data.
Repeat LL, ZZ, DATA for all segments.		
LL	2 (binary)	Length of data segment.
ZZ	2 (binary)	Reserved (set to binary zeros).
DATA	n	User data.
LL	2 (binary)	End of message (set to X'0004').
ZZ	2 (binary)	Reserved (set to binary zeros).

9.4.2 Input message returned from message exit

This section describes input messages from the message exit of IMS Connector for Java and non-IMS Connector for Java message structures.

IMS Connector for Java message structure, type 1

The IMS Connector for Java exit output message format that is supported by IMS Connect is the same message format of the input message. See Table 9-8 on page 127.

Non-IMS Connector for Java message structure, type 3

Table 9-10 shows the output message format supported by IMS Connect from the supplied HWSIMSO0 and HWSSMPL0 exits (non-IMS Connector for Java client exits). Variable length OTMA headers are supported, and therefore, the OTMA header length can be other than 466 bytes. The following example contains 466 bytes as used by the supplied exits.

Table 9-10 Non-IMS Connector for Java message structure, type 3

Field	Length (bytes)	Meaning
The first 64 bytes represent the fixed BPE header.		
IIII	4 (binary)	Length of entire buffer.
CHAIN PTR	4 (binary)	Chain pointer to next BPE header.
STORAGE TYPE	8 (character)	Storage type.
TYPE ACCESS	4 (character)	Type access.
SUBPOOL	1 (binary)	Subpool number.
RESV	43	Reserved.
End of BPE header.		
Start of user data for this BPE header.		

Field	Length (bytes)	Meaning
OTMA header	466	See OTMA DSECT member HWSOMPFX in IMS.SDFSMACT library for a description.
LL	2 (binary)	Length of data segment.
ZZ	2 (binary)	Reserved (set to binary zeros).
DATA	n	User data.
Repeat LL, ZZ, DATA for all segments.		
LL	2 (binary)	Length of data segment.
ZZ	2 (binary)	Reserved (set to binary zeros).
DATA	n	User data.
YY	2 (binary)	ZERO indicates end of user data for preceding BPE header.
Start of user data for next BPE header.		
BPE header	64	See first BPE header for layout.
OTMA CTL header	32	See OTMA DSECT member HWSOMPFX in IMS.SDFSMACT library for a description.
LL	2 (binary)	Length of data segment.
ZZ	2 (binary)	Reserved (set to binary zeros).
DATA	n	User data.
Repeat LL, ZZ, DATA for all segments.		
LL	2 (binary)	Length of data segment.
ZZ	2 (binary)	Reserved (set to binary zeros).
DATA	n	User data.
YY	2 (binary)	ZERO indicates end of user data for preceding BPE header.

Restriction: The length of data from one BPE header to the next BPE header cannot exceed 32 K, excluding the BPE header and the OTMA header.

9.4.3 Output message from IMS Connect to IMS Connector for Java client

Output messages from IMS Connect consist of IMS Connector for Java message structures.

IMS Connector for Java message structure

Table 9-11 shows the message format from IMS Connect to the client. IMS Connector for Java output is *not* passed to HWSJAVA0 exit.

Table 9-11 IMS Connector for Java message structure (IMS Connect to client)

Field	Length (bytes)	Meaning
IIII	4 (binary)	Total message length.
ID	8 (character)	Char value of *HWSJAV*.

Field	Length (bytes)	Meaning
OTMA header	466	See OTMA DSECT member HWSOMPFX in IMS.SDFSMA library for a description.
LL	2 (binary)	Length of data segment.
ZZ	2 (binary)	Reserved (set to binary zeros).
DATA	n	User data.
OTMA CTL headers	20	See OTMA DSECT member HWSOMPFX in IMS.SDFSMA library for a description.
LL	2 (binary)	Length of data segment.
ZZ	2 (binary)	Reserved (set to binary zeros).
DATA	n	User data.
Repeat OTMA CTL header, LL, ZZ, DATA for all segments.		
OTMA CTL headers	20	See OTMA DSECT member HWSOMPFX in IMS.SDFSMA library for a description.
LL	2 (binary)	Length of data segment.
ZZ	2 (binary)	Reserved (set to binary zeros).
DATA	n	User data.

9.4.4 Output message: IMS Connect to non-IMS Connector for Java client

Output messages from IMS Connect consist of non-IMS Connector for Java message structures.

The message format from IMS Connect to the exit

Table 9-12 shows the message format from IMS Connect to the exit.

Table 9-12 Non-IMS Connector for Java message structure (IMS Connect to exit)

Field	Length (bytes)	Meaning
OTMA header	466	See OTMA DSECT member HWSOMPFX in IMS.SDFSMA library for a description.
LL	2 (binary)	Length of data segment.
ZZ	2 (binary)	Reserved (set to binary zeros).
DATA	n	User data.
OTMA CTL headers	20	See OTMA DSECT member HWSOMPFX in IMS.SDFSMA library for a description.
LL	2 (binary)	Length of data segment.
ZZ	2 (binary)	Reserved (set to binary zeros).
DATA	n	User data.
Repeat OTMA CTL header, LL, ZZ, DATA for all segments.		
OTMA CTL headers	20	See OTMA DSECT member HWSOMPFX in IMS.SDFSMA library for a description.

Field	Length (bytes)	Meaning
LL	2 (binary)	Length of data segment.
ZZ	2 (binary)	Reserved (set to binary zeros).
DATA	n	User data.

The message format from exit to the client

The following tables show the message format from the exit to the client. This section describes these message structures from the point of view of an IMS Connect systems programmer. If you are looking for a description more appropriate for a client applications programmer, refer to 9.4, “Message structures between IMS Connect and user exits” on page 127.

Request Mod Message (RMM)

IMS Connect returns the Request Mod Message (RMM) as the first structure of an output message if the MFS MOD name is requested and the data output is present. See Table 9-13.

Table 9-13 Request Mod Message format

Field	Length	Meaning
LL	2 (binary)	Length of RMM message
ZZ	2 (binary)	Reserved (set to binary zeros)
ID	8 (character)	Char value of *REQMOD*
MOD	8 (character)	Char value of the requested MFS MOD name

Complete Status Message (CSM)

IMS Connect returns the Complete Status Message (CSM) as the last structure of an output message if the input message is processed successfully. It does not generate an end-of-message (EOM) segment. See Table 9-14.

Table 9-14 Complete Status Message format

Field	Length	Meaning
LL	2 (binary)	Length of CSM message.
CSM_FLG1	1 (binary)	Flag byte 1: Binary '10000000' Asynchronous message queued. Binary '01000000' Conversational output message. Binary '00100000' ACK/NAK required (see the Tip following Table 9-15).
Reserved	1 (binary)	Reserved (set to binary zeros).
CSM_ID	8 (character)	Char value of *CSMOKY*.

Request Status Message (RSM)

IMS Connect returns the Request Status Message (RSM) as the only structure of an output message if IMS Connect or the message exit determined an error occurred. This is valid for IMS command output. The RSM contains a return and reason code indicating the type of status. See Table 9-15 on page 134.

Table 9-15 Request Status Message format

Field	Length	Meaning
LL	2 (binary)	Length of RSM message.
RSM_FLG1	1 (binary)	Flag byte 1: <ul style="list-style-type: none"> Binary '10000000' Asynchronous message queued. Binary '01000000' Conversational output message. Binary '00100000' ACK/NAK required (see the Tip following this table).
Reserved	1 (binary)	Reserved (set to binary zeros).
RSM_ID	8 (character)	Char value of *REQSTS*.
RSM_RETCOD	4 (binary)	Return code.
RSM_RSNCOD	4 (binary)	Reason code.

Tip: ACK/NAK required flag

With sync level=confirm, some IMS messages (for example, DFS065 TRAN/LTERM STOPPED) do not require an ACK. The ACK/NAK flag applies to all messages, not only the IMS error messages. This allows the client code to interrogate CSM_FLG1 and RSM_FLG1 to determine if an acknowledgement (ACK/NAK) response is required for all messages received.

The message format from exit to the client

The following tables show the message formats from IMS Connect to the exit. Table 9-16 shows the case when the MFS MOD name is requested and data is being sent.

Table 9-16 Message format when MOD name requested and data being sent

Field	Length (bytes)	Meaning
IIII	4 (binary)	Total length of the output message (HWSIMSO1/HWSSMPL1 only).
RMM header	20	See RMM layout in Table 9-13.
LL	2 (binary)	Length of data segment.
ZZ	2 (binary)	Reserved (set to binary zeros).
DATA	n	User data.
Repeat LL, ZZ, DATA for all segments.		
LL	2 (binary)	Length of data segment.
ZZ	2 (binary)	Reserved (set to binary zeros).
DATA	n	User data.
CSM	12	See CSM layout in Table 9-14.

Table 9-17 on page 135 shows the case when MFS MOD name is not requested and data is being sent.

Table 9-17 Message format when MOD name not requested and data being sent

Field	Length (bytes)	Meaning
IIII	4 (binary)	Total length of the output message (HWSIMSO1/HWSSMPL1 only).
LL	2 (binary)	Length of data segment.
ZZ	2 (binary)	Reserved (set to binary zeros).
DATA	n	User data.
Repeat LL, ZZ, DATA for all segments.		
LL	2 (binary)	Length of data segment.
ZZ	2 (binary)	Reserved (set to binary zeros).
DATA	n	User data.
CSM	12	See CSM layout in Table 9-14.

Table 9-18 shows the case when an error is detected or IMS Connect sends unsolicited output to client.

Table 9-18 Error detected or sent unsolicited output to client

Field	Length (bytes)	Meaning
IIII	4 (binary)	Total length of the output message (HWSIMSO1/HWSSMPL1 only).
The following structures are same between HWSIMSO0/SMPL0 and HWSIMSO1/SMPL1.		

9.5 IMS Connect DRU exit for asynchronous output support

An OTMA destination resolution (DRU) exit is required to support asynchronous output that is generated by an IMS application that does an insert (ISRT) to an alternate program communication block (PCB). IMS Connect provides a sample OTMA DRU exit named HWSYDRU0. You can either modify the HWSYDRU0 exit to work with your installation, or provide your own DRU exit.

9.5.1 ALTPCB ISRT message routing flow using OTMA exits

Two optional OTMA output routing exits are provided in IMS for application output to the ALT-PCB:

► DFSYPRX0

The prerouting exit, performs an initial search for the output destination.

► DRU exit

The destination resolution exit, determines the final destination for an OTMA output message. (The default name is DFSYDRU0.)

With these two exits, the IMS ALT-PCB output message can be directed to a non-OTMA destination or to any OTMA client. Figure 9-3 on page 136 illustrates the relationship between the two OTMA user exits and how IMS handles the return codes from the exits.

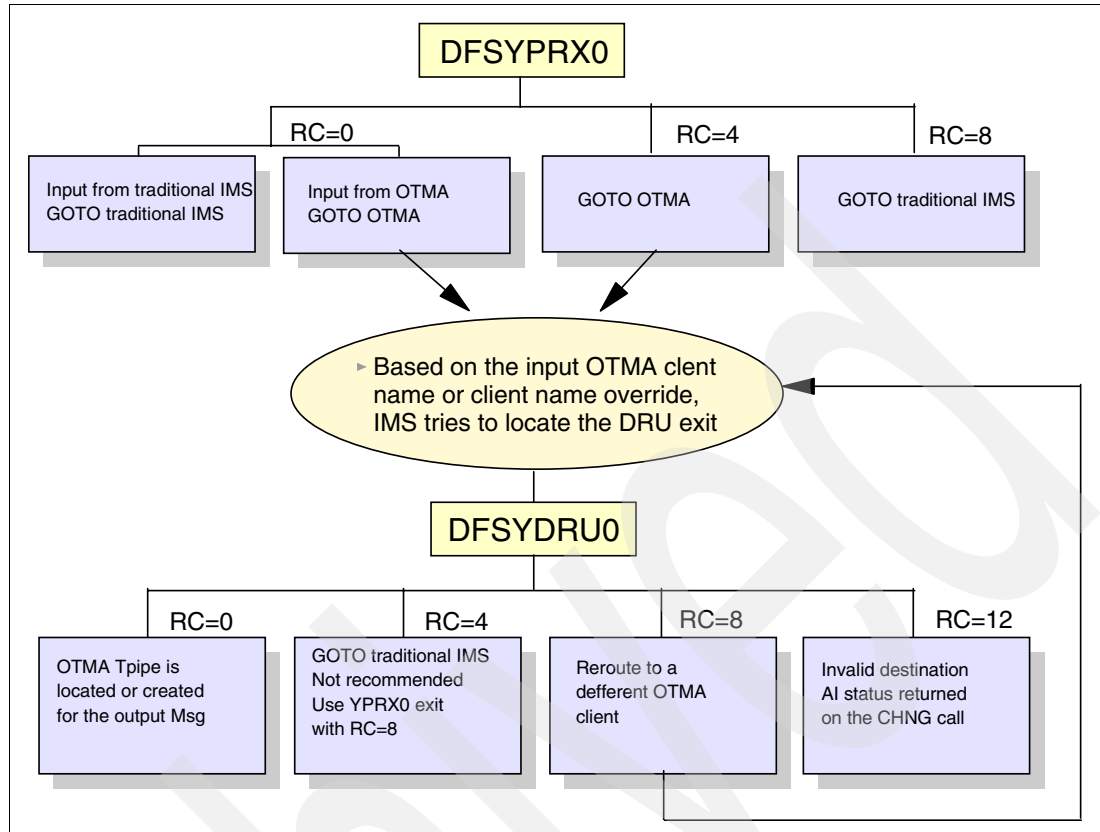


Figure 9-3 Two-phase destination resolution flow

Phase 1: Prerouting process

In the first phase, IMS searches for an initial destination for an ALT-PCB output message to determine if the OTMA route or a traditional IMS route should be used. The processing algorithm varies, depending on whether or not the OTMA DFSYPX0 user exit exists in the IMS system.

If a DFSYPX0 user exit is in the IMS system, the exit can specify either the traditional IMS or the OTMA route regardless of the origination of the input message. The exit is not invoked if the destination is a system console or master terminal operator (MTO).

If the transaction was entered from a non-OTMA LTERM and you want to route the output to an OTMA destination, you must specify the name of the OTMA client in the appropriate field of the parameter list. If the transaction was entered from an OTMA client, the contents of that field are ignored *unless* you have set the OTMAMD initialization parameter to Y in the DFSPBxxx PROCLIB member. See the description of DFSYPX0 in *IMS Version 9: Customization Guide*, SC18-7817, for details.

Phase 2: Final destination resolution

In the second phase, IMS determines or changes the destination of an ALT-PCB output message. If the OTMA route was specified during the prerouting process, and if the DRU exit is available in the IMS system, it can override the final destination if that destination is not a transaction (SMB). A new OTMA transaction pipe can be created to send the output message. The output message can also be rerouted to a different OTMA client by specifying

the client name. Previously, that was the only way to reroute the asynchronous output from one OTMA-originated transaction to a different OTMA client, but now this function can be done in DFSYPRX0. When OTMAMD is set to Y, DFSYPRX0 can and should make all routing decisions. Refer to APAR PQ33996 for more details.

For more information about the OTMA prerouting exit and destination resolution exit, refer to *IMS Version 9: Customization Guide*, SC18-7817.

9.5.2 How IMS Connect communicates with the DRU exit

In IMS Connect, the LTERM name is analogous to the unique clientID name for commit-then-send message flow. To clarify whether a destination is for IMS Connect (through OTMA), IMS uses a prerouting exit routine (DFSYPRX0) that can specify where IMS should look to resolve the destination names. In this case, IMS needs to look at the IMS Connect clientIDs, because the DRU exit cannot determine whether the message should be directed to OTMA client (IMS Connect) or traditional IMS processing (ISRT to traditional LTERM or P to P switching). Determining the destination for an OTMA (IMS Connect client) message requires two phases:

1. DFSYPRX0 is called to determine the initial destination for the output.

The exit routine can determine whether the message should be directed to OTMA clients or to IMS Transaction Manager for traditional IMS processing. The exit routine cannot determine the final destination (Tpipe name).

2. The DRU exit routine (for example, the IMS Connect supplied exit HWSYDRU0) is called to determine the final destination (Tpipe name) for the output.

Each OTMA client can specify a separate DRU exit routine. In other words, each OTMA client can specify a single DRU exit for each copy of IMS Connect that is connected to a given datastore (IMS). This means that one IMS Connect can have the same or a different DRU exit for each of the datastore definitions in the IMS Connect configuration file.

9.5.3 HWSYDRU0 sample DRU exit

Figure 9-4 on page 138 shows the function of HWSYDRU0, the IMS Connect-supplied OTMA DRU exit.

You can only use this exit under one of the following conditions:

- ▶ The IMS Connect clientIDs are named CLIENT01 through CLIENT09 and all belong to the same member name.
- ▶ The IMS Connect clientIDs are as follows (in this case, Tpipe names are changed and integrated into another name):
 - TPIPE001 through TPIPE099 and all belong to member MEMBER0
 - TPIPE100 through TPIPE199 and all belong to member MEMBER1
 - TPIPE200 through TPIPE299 and all belong to member MEMBER2
 - TPIPE300 through TPIPE399 and all belong to member MEMBER3
 - TPIPE400 through TPIPE499 and all belong to member MEMBER4

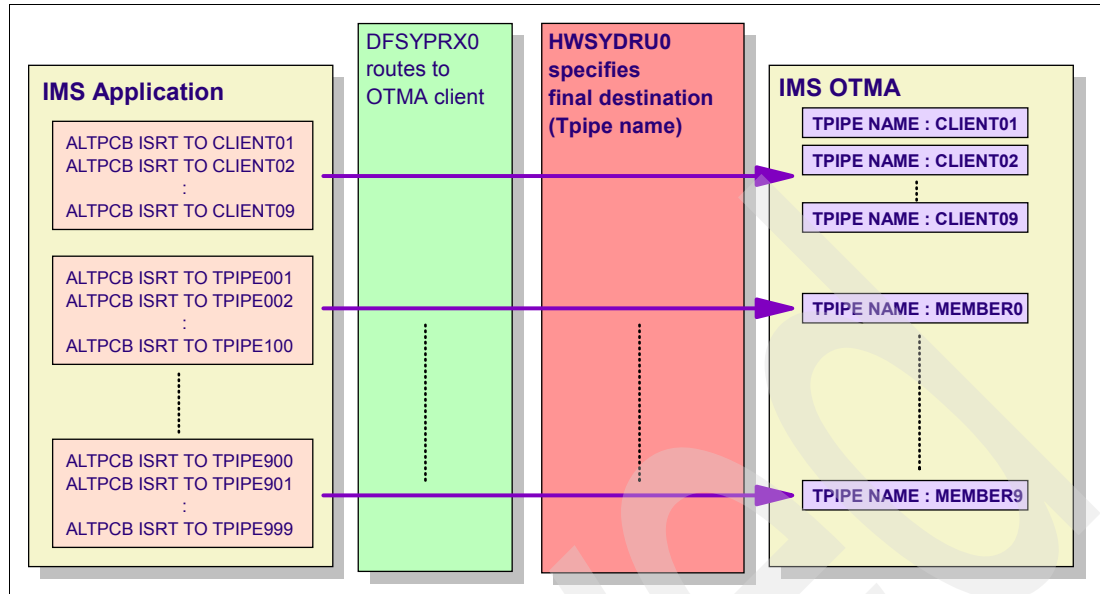


Figure 9-4 The function of the HWSYDRU0 user exit

The HWSYDRU0 exit is only an example, and when you use it, the following sequence of events occurs:

1. The prerouting exit (DFSYPRX0) sets up addressability to the parameters that are passed to the HWSYDRU0 exit.
2. The output member name in the output parameter list is set to blanks.
3. HWSYDRU0 determines the action to take based on whether the name in the input destination parameter (that is, the destination where the message is to be sent) is an IMS LTERM or an IMS Connect destination. After HWSYDRU0 makes this determination, it takes a course of action and sets the contents of register 15 on exit.

9.5.4 Debugging the IMS OTMA exits

When you need to debug an IMS OTMA exit, HWSYPRX0 or HWSYDRU0, keep in mind that they can be called in a cross-memory environment. In addition, those exits must be reentrant. write to operator (WTO) can be used to display debug messages, but you have to be careful:

- You must use the LIST-EXECUTE form of WTO, that is, WTO MF=(E,WTOMSG), or you will find 0C4 abends in the IMS control region.
- You must use LINKAGE=BRANCH if you are running in a cross-memory environment.

Example 9-3 shows how WTOs can be safely issued from an IMS OTMA exit routine.

Example 9-3 Issuing WTOs from an IMS OTMA exit routine

	LA	R7,WTOMSG	
	BAL	R10,DOWTO	
DOWTO	DS	OH	
	L	R5,ADDRWORK	ADDRESS OF 512-BYTE WORK AREA
	LTR	R5,R5	IS THERE A WORK AREA?
	BZR	R10	NO - JUST RETURN
	USING	WTOD,R5	TELL ASSEMBLER
	MVC	WTOMSGE,WTOMSGL	MOVE MESSAGE MASK
	MVC	WTOTEXT,0(R7)	MOVE MESSAGE TEXT

	EPAR	R0	PRIMARY ASID
	ESAR	R1	SECONDARY ASID
	CLR	R0,R1	PRIMARY = SECONDARY?
	BNE	DOWTOX	NO - IN XM MODE
	WTO	MF=(E,WTOMSGE)	WTO TRACE MESSAGE
	BR	R10	RETURN TO CALLER
DOWTOX	DS	0H	
	WTO	MF=(E,WTOMSGE),LINKAGE=BRANCH	
	BR	R10	RETURN TO CALLER
	DROP	R5	TELL ASSEMBLER
	EJECT		
WTOMSGL	WTO	' , X	
		ROUTCDE=(11),DESC=(7),MF=L	
WTOD	DSECT		
WTOMSGE	DS	0CL58	REENTRANT WTO MESSAGE
	DS	CL2	TEXT LENGTH
	DS	CL2	MCSFLAGS
WTOTEXT	DS	CL50	
	DS	CL4	ROUTING AND DESCRIPTOR CODES

Archived

IMS Connect diagnostics

This chapter contains information about the features that IMS Connect provides to help you to diagnose problems. It describes the IMS Connect recorder trace and the IMS Connect traces.

The IMS Connect recorder trace is a function to capture information related to IMS Connect input and output data to a data set. IMS Connect recorder trace is also called IMS Connect line trace.

Through the IMS Connect Base Primitive Environment, IMS Connect enables you to trace diagnostic information about events going on within the address space. These traces are internal incore tables; IMS Connect does not write these trace records to external data sets. To access to this data, you have to obtain a dump and format the trace tables.

This chapter also describes how to use the IMS Connect Dump Formatter to format IMS Connect internal control blocks under the control of the Interactive Problem Control System (IPCS).

10.1 IMS Connect recorder trace

The IMS Connect recorder trace records the messages and the headers received from the client and the messages and headers received from OTMA and sent to the client.

The IMS Connect recorder trace records the messages before and after they are processed by the IMS Connect exits. The trace contains a copy of the first 670 bytes of the data as it is passed to the user message exit and upon return from the user message exit.

There are three types of recorder trace records:

- ▶ Input message from the client. IMS Connect labels them as ITOCRC.
- ▶ Output message sent to the client. IMS Connect labels them as ITOCSN.
- ▶ Read error record. IMS Connect labels them as ITOCER.

10.1.1 Enabling IMS Connect recorder trace

To enable the recorder trace, you have to allocate a data set for the information to be stored and add a HWSRCORD DD card to the IMS Connect startup JCL, as shown in Example 10-1.

Example 10-1 Sample IMS Connect startup JCL with HWSRCORD DD statement

```
//HWS PROC RGN=4096K,SOUT=A,
//      BPECFG=BPECFGHT,
//      HWSCFG=HWSCFG00
//*
//*****
//* BRING UP AN IMS CONNECT *
//*****
//STEP1 EXEC PGM=HWSHWS00,REGION=&RGN,TIME=1440,
//      PARM='BPECFG=&BPECFG,HWSCFG=&HWSCFG'
//STEPLIB DD DSN=SHWSRESL,DISP=SHR
//      DD DSN=SDFSRESL,DISP=SHR
//      DD DSN=CEE.SCEERUN,UNIT=SYSDA,DISP=SHR
//      DD DSN=SYS1.CSSLIB,UNIT=SYSDA,DISP=SHR
//      DD DSN=GSK.SGSKLOAD,UNIT=SYSDA,DISP=SHR
//PROCLIB DD DSN=USER.PROCLIB,DISP=SHR
//SYSPRINT DD SYSOUT=&SOUT
//SYSUDUMP DD SYSOUT=&SOUT
//HWSRCORD DD DSN=HWSRCRD,DISP=SHR
```

Allocate the HWSRCORD data set normal physical sequential data set with record format FB, record length 1,440 and the block size 14,400. The data set can use secondary extents.

IMS Connect stores the trace information into this single data set. There are no spare data sets or any wrap-around processing, so when this data set fills up, the trace is disabled. Starting the trace overwrites trace data recorded in earlier trace.

From an IMS perspective, you can manage this trace data set in a similar manner as to the IMS Monitor Trace (IMSMON) data set.

10.1.2 Starting and stopping the IMS Connect recorder trace

You use the IMS Connect RECORDER command to start and stop the recorder trace:

- ▶ RECORDER OPEN opens the recorder trace data set.
- ▶ RECORDER CLOSE closes the recorder trace data set.

10.1.3 Printing out the recorder trace

Example 10-2 shows the JCL to print the recorder trace data set.

Example 10-2 Sample JCL to print the recorder trace data set

```
//IDCAMS JOB JOB 1,IDCAMS,MSGLEVEL=1,CLASS=K,TIME=1440
//SELECT EXEC PGM=IDCAMS
//DD1 DD DSN=HWSRCR,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
PRINT INFILE(DD1)
```

Example 10-3 shows a sample of the output that you obtain.

Example 10-3 Printed recorder trace output sample

0000000	00000000	C9E3D6C3	D9C30052	00000877	13313877	0105168F	00000000	00000000	*...ITOCRC.....*
0000020	C3D3C9C5	D5E3F0F1	BD2D0A48	11A8A120	BD2D0A48	11AA4C80	00000000	00000000	*CLIENT01.....<.....*
0000040	00000000	00000000	00000000	00000000	00000000	00000000	00000000	5CC9D7C2	*.....*IPB*
0000060	00000093	00500000	2A53414D	504C452A	00000000	00000000	C3D3C9C5	D5E3F0F1	*.....(&<.....CLIENT01*
0000080	00400140	C9E5E3D5	D6404040	C9D4E2C7	40404040	40404040	40404040	40404040	*. . IVTNO IMSG
00000A0	40404040	40404040	40404040	5C5C5C5C	5C5C5C5C	003B0000	C9E5E3D5	D6404040	* *****...IVTNO *
00000C0	4040C4C9	E2D7D3C1	E840D3C1	E2E3F140	40404040	40404040	40404040	40404040	* DISPLAY LAST1 *
00000E0	40404040	40404040	40404040	40404000	04000000	00000000	00000000	00000000	*
000100	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	*

10.1.4 Interpreting the recorder trace printout

As you can see in Example 10-3, the printed recorder trace output has eye-catchers to identify the data that it contains. Figure 10-1 shows the relationship between the messages traced by IMS Connect and the eye-catchers on the recorder log in a simple transaction flow.

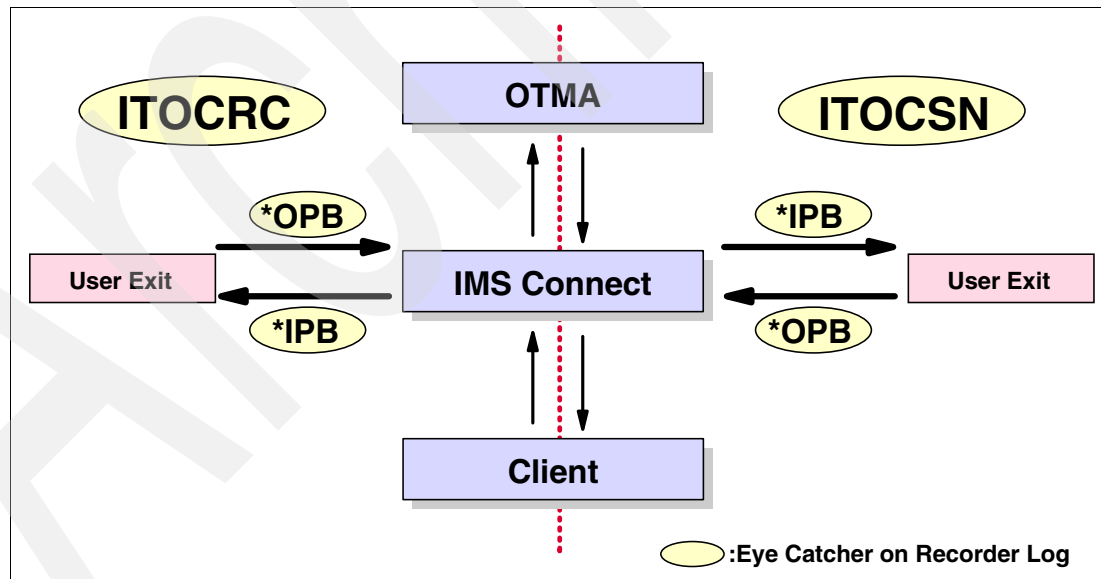


Figure 10-1 Eye-catchers on recorder log

As a first level, the records have an eye-catcher to identify the type of record (if it has data received from the client or sent to the client). As a second level, there are eye-catchers to identify if the data is an input to the exit or an output from the exit.

Table 10-1 on page 144 shows the eye-catchers and the information that you find after each one.

Table 10-1 Recorder log contents

Eye-catcher on recorder log	Contents
ITOCRC: Receive from client	
*IPB	Input to exit
*OPB	Output from exit
ITOCSN: Send to client	
*IPB	Input to exit
*OPB	Output to exit

In this sample flow, you have the following trace records:

► ITOCRC

Contains information about data received from the client. It has the value ITOCRC at the offset x'04'.

The prefix *IPB begins at offset x'5C' and is followed by the actual input message before it is given to the IMS Connect exit at offset X'60'. The ID of the exit (IRM_ID) to which this message will be passed is at offset x'68'.

The prefix *OPB begins at offset x'2FC' and is followed at offset X'300' with the actual output message after it is processed by the IMS Connect exit.

There are two different cases:

- For IMS Connector for Java messages, the format of the exit output message is the same as the format of the input message. The HWSJAVA0 exit does not modify the message.
- For non-IMS Connector for Java messages, the format of the exit output message contains the BPE header, OTMA headers, and data.

► ITOCSN

Contains information about data sent to the client. It has the value ITOCSN at the offset x'04'.

The prefix *IPB is followed by the actual message before it is given to the IMS Connect exit at offset X'60'. Output messages from the OTMA consist of OTMA headers and the data. The ID of the OTMA Tpipe name can be found at offset x'66'.

At offset X'300' after the prefix *OPB begins the actual message after it is processed by the IMS Connect exit.

There are two different cases:

- For IMS Connector for Java messages, the format of the message is the same message format as the output message (except the 12 byte message header added by IMS Connect). The HWSJAVA0 exit does not modify the message.
- For non-IMS Connector for Java messages, the format of the message is Request Mod Message (RMM) if available, data, and the Complete Status Message (CSM) if the input message was processed successfully and output data is available. If IMS Connect or the user message exit determined that an error occurred or notified some information to the client (for example, commit confirmation with synclevel confirm), the only structure contained in the output message is the Request Status Message (RSM). The RSM consists of 20 bytes of information (LL, ZZ, ID *REQSTS*, the return code, and the reason code).

In abnormal situations, you can also find ITOCER records with information about read errors.

For more information about these records, refer to the recorder log record mapping in *IMS Version 9: IMS Connect Guide and Reference*, SC18-9287.

10.1.5 Example of recorder trace output

For this example, we use the standard IMS IVP application, but we convert the COBOL source for transaction IVTNO into a Java application. We use IMS Connector for Java to send the request to IMS Connect. OTMA commit mode is send-then-commit (synclevel none).

We activate the IMS Connect recorder trace for a single iteration of the IVTNO transaction triggered from the Web browser. Figure 10-2 shows the sample transaction flow and the recorder log contents.

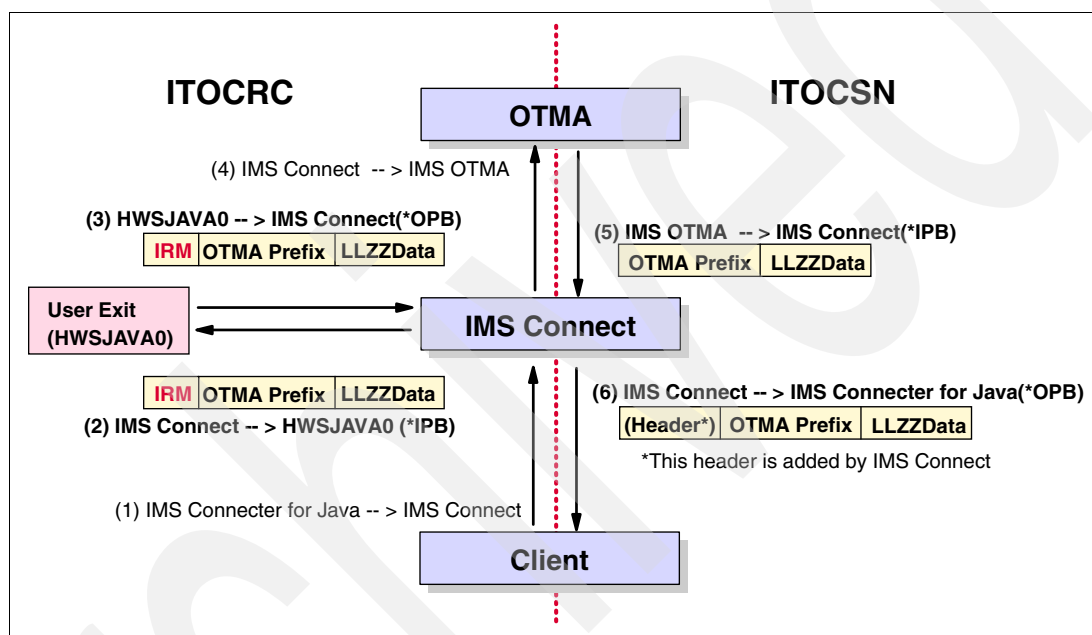


Figure 10-2 Sample transaction flow and recorder log contents

Note: When you use the IMS Connector for Java client, messages are not modified for the HWSJAVA0 exit. The *OPB contents for ITOCSN show the message received from OTMA with the header added by IMS Connect.

Example 10-4 shows the data obtained by printing the recorder trace output.

Example 10-4 Recorder trace output example from IMS Connector for Java

```

ORECORD SEQUENCE NUMBER - 1
000000 00000000 C9E3D6C3 D9C30052 00000877 14413251 0101254F 00000000 00000000 *...ITOCRC.....|.....*
000020 C8E6E2F2 E5E5E3E4 B66C556A BEB2BB00 B66C556A C5F65281 00000000 00000000 *HWS2VVTU.%.....%.E6.....*
000040 00000000 00000000 00000000 00000000 00000000 00000000 5CC9D7C2 *.....*IPB*
000060 0000022D 001C0000 5CC8E6E2 D1C1E55C 00000040 C0000000 C8E6E2F2 E5E5E3E4 *.....*HWSJAV*... {...HWS2VVTU*
000080 01400000 00004040 40404040 4040A0F0 00000000 00000000 00000000 00010000 *.....0.....*
0000A0 00480020 00404040 40404040 40400000 00000000 00000000 00000000 00000000 *.....*
0000C0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00004040 *.....*
0000E0 40404040 40400000 006AC614 09029196 A49296F8 40400903 A2A8A2F1 40404040 *.....F.....8 .....1 *
000100 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
000120 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
000140 00000000 00000000 00000000 00000000 00000100 0000C9D4 E2C24040 4040C8E6 *.....IMSB HW*
000160 E2F2E5E5 E3E40000 00000000 00000000 00000000 00000000 00000000 *S2VVTU.....*
000180 00000000 00009196 A49296F8 40401000 00000000 00000000 00000000 00000000 *.....8 .....*
0001A0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
0001C0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
0001E0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
  
```

```

000200 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
000220 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
000240 00000000 00000000 00000000 00000000 0000003B 0000C9E5 E3D5D640 40404040 *.....IVTNO*
000260 C4C9E2D7 D3C1E840 D1C1D4C5 E2404040 4040C2D6 D5C44040 40404040 F7F7F0F1 *DISPLAY JAMES BOND 7701*
000280 F2F3F4F5 F6F7F1F2 F3F4F540 40000000 00000000 00000000 00000000 *23456712345.....*
0002A0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
0002C0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
0002E0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
000300 0000022D 001C0000 5CC8E6E2 D1C1E55C 00000040 C0000000 C8E6E2F2 E5E5E3E4 *.....*HWSJAV*...{...HWS2VVTU*
000320 01400000 00004040 40404040 4040A0F0 00000000 00000000 00000000 00010000 *......0.....*
000340 00480020 00404040 40404040 40400000 00000000 00000000 00000000 00000000 *.....*
000360 00000000 00000000 00000000 00000000 00000000 00000000 00004040 *.....*
000380 40404040 40400000 006AC614 09029196 A49296F8 40400903 A2A8A2F1 40404040 *.....F.....8.....1*
0003A0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
0003C0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
0003E0 00000000 00000000 00000000 00000000 00000100 0000C9D4 E2C24040 4040C8E6 *.....IMSB HW*
000400 E2F2E5E5 E3E40000 00000000 00000000 00000000 00000000 00000000 *S2VVTU.....*
000420 00000000 00009196 A49296F8 40401000 01000000 00000000 00000000 00000000 *.....8.....*
000440 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
000460 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
000480 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
0004A0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
0004C0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
0004E0 00000000 00000000 00000000 00000000 0000003B 0000C9E5 E3D5D640 40404040 *.....IVTNO*
000500 C4C9E2D7 D3C1E840 D1C1D4C5 E2404040 4040C2D6 D5C44040 40404040 F7F7F0F1 *DISPLAY JAMES BOND 7701*
000520 F2F3F4F5 F6F7F1F2 F3F4F540 40000000 00000000 00000000 00000000 *23456712345.....*
000540 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
000560 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
000580 00000000 00000000 00000000 00000000 00000000 00000000 5CC5D5C4 *.....*END*

ORECORD SEQUENCE NUMBER - 2
00000000 00000000 C9E3D6C3 E2D50052 00000877 14413254 0101254F 00000000 00000000 *...ITOC SN.....|.....*
000020 C8E6E2F2 E5E5E3E4 B66C556A C6980B21 00000000 00000000 B66C556A C6983D61 *HWS2VVTU.%.F.....%.F../*
000040 00000000 00000000 00000000 00000000 00010000 00000000 00000000 5CC9D7C2 *.....*IPB*
000060 01800000 0000F6F0 F0F14040 4040A0F0 0000001E 00000000 00000000 00010000 *.....6001 .0.....*
000080 00480020 00404040 40404040 40400000 00000000 00000000 00000000 00000000 *.....*
0000A0 00000000 00000000 00000000 00000000 00000000 00000000 00004040 *.....*
0000C0 40404040 40400000 006AC614 0902D1D6 E4D2D6F8 40400903 E2E8E2F1 40404040 *.....F...JOUK08 .SYS1*
0000E0 51005001 82555555 15555555 55555555 55555555 55555555 55555555 *..&.....*
000100 55555555 55555555 55555555 55555555 55555555 55555555 55555555 *.....*
000120 55558483 B18783AD 1515B7BD B7A41515 15150100 0000C9D4 E2C24040 4040C8E6 *.....IMSB HW*
000140 E2F2E5E5 E3E4F6F0 F0F14040 4040B66C 556ABEAC B9000000 00000000 00000000 *S2VVTU6001 .%......*
000160 00000CB5 DF180000 00000000 00001000 21000000 0000A5F8 00000000 00000000 *.....8.....*
000180 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
0001A0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
0001C0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
0001E0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
000200 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
000220 00000000 00000000 00000000 00000000 0000005D 0300C5D5 E3D9E840 E6C1E240 *.....).ENTRY WAS*
000240 C4C9E2D7 D3C1E8C5 C4404040 40404040 40404040 40404040 4040C4C9 *DISPLAYED DI*
000260 E2D7D3C1 E840D1C1 D4C5E240 40404040 C2D6D5C4 40404040 4040F7F7 F0F1F2F3 *SPLAY JAMES BOND 770123*
000280 F4F5F6F7 F1F2F3F4 F54040F0 F0F1F100 00000000 00000000 00000000 00000000 *456712345 0011.....*
0002A0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
0002C0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
0002E0 00000000 00000000 00000000 00000000 00000000 00000000 5CD6D7C2 *.....*OPB*
000300 0000023B 5CC8E6E2 D1C1E55C 01800000 0000F6F0 F0F14040 4040A0F0 0000001E *...*HWSJAV*...6001 .0....*
000320 00480000 00000000 00010000 00480020 00404040 40404040 40400000 00000000 *.....*
000340 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
000360 00000000 00000000 00004040 40404040 40400000 006AC614 0902D1D6 E4D2D6F8 *.....F...JOUK08*
000380 40400903 E2E8E2F1 40404040 51005001 82555555 15555555 55555555 55555555 *..SYS1 ..&.....*
0003A0 55555555 55555555 55555555 55555555 55555555 55555555 55555555 *.....*
0003C0 55555555 55555555 55555555 55558483 B18783AD 1515B7BD B7A41515 15150100 *.....*
0003E0 0000C9D4 E2C24040 4040C8E6 E2F2E5E5 E3E4F6F0 F0F14040 4040B66C 556ABEAC *..IMSB HWS2VVTU6001 .%.***
000400 B9000000 00000000 00000000 00000CB5 DF180000 00000000 00001000 21000000 *.....*
000420 0000A5F8 00000000 00000000 00000000 00000000 00000000 00000000 *...8.....*
000440 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
000460 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
000480 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
0004A0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
0004C0 00000000 00000000 00000000 00000000 00000000 00000000 0000005D *.....)*
0004E0 0300C5D5 E3D9E840 E6C1E240 C4C9E2D7 D3C1E8C5 C4404040 40404040 40404040 *..ENTRY WAS DISPLAYED*
000500 40404040 40404040 4040C4C9 E2D7D3C1 E840D1C1 D4C5E240 40404040 C2D6D5C4 * DISPLAY JAMES BOND*
000520 40404040 4040F7F7 F0F1F2F3 F4F5F6F7 F1F2F3F4 F54040F0 F0F1F100 00000000 * 770123456712345 0011.....*
000540 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
000560 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....*
000580 00000000 00000000 00000000 00000000 00000000 00000000 5CC5D5C4 *.....*END*

OIDC00051 NUMBER OF RECORDS PROCESSED WAS 2

```

Table 10-2 on page 147 shows the fields that the ITOCRC record displays in the example.

Table 10-2 IMS Connect receive data

Offset	Description of the fields in the example trace output
x'04'	ITOC prefix.
x'08'	RC prefix, indicating a receive from client.
x'5C'	*IPB prefix, indicating the beginning of the input buffer from the client to IMS Connect.
x'60'	Beginning of input buffer.
x'68'	IRM_ID of the IRM header, that is, the ID of the exit to which this message will be passed. IBM-defined IRM_IDs are: <ul style="list-style-type: none"> ▶ *IRMREQ* for a TCP/IP client ▶ *IRMRE1* for a TCP/IP client/fullword length field prior to message ▶ *SAMPLE* for IMS sample client for Java ▶ *SAMPL1* for IMS sample client/fullword length prior to message ▶ *HWSJAV* for IMS Connector for Java
x'78'	Character value of a unique clientID (HWS2VVTU in this example).
x'80'	OTMA header information. Refer to 2.2, "OTMA message structure" on page 9 for a description.
x'256'	IMS transaction code and user data, beginning with IVTNO.
x'2FC'	*OPB prefix, indicating the beginning of the output buffer from IMS Connect to IMS.
x'300'	Beginning of the output buffer.
x'308'	IRM_ID of the IRM header, that is, the ID of the exit that has processed this message. IBM-defined IRM_IDs are same as in offset x'68' above. This example points to *HWSJAV*.
x'318'	Character value of a unique clientID (HWS2VVTU in this example).
x'320'	OTMA header information.
x'4F6'	IMS transaction code and user data, beginning with IVTNO.

Table 10-3 shows the fields that the ITOCSN record displays in the example.

Table 10-3 IMS Connect send data

Offset	Description of the fields in the example trace output
x'04'	ITOC prefix
x'08'	SN prefix, indicating a receive from client
x'5C'	*IPB prefix, indicating the beginning of the input buffer from IMS to IMS Connect
x'60'	Beginning of OTMA header
x'66'	OTMA Tpipe name, in our case, '6001'
x'232'	Beginning of user data
x'2FC'	*OPB eye-catcher, indicating the beginning of the output buffer from IMS Connect to the client
x'300'	Beginning of the OTMA header
x'4DE'	Beginning of user data

10.2 IMS Connect traces

The IMS Connect traces normally are only used if requested by the IBM support center while analyzing a problem. This section helps you to identify how to process these traces and how to read them in order to provide better information to IBM.

The IMS Connect Base Primitive Environment (IMS Connect BPE) is the component that provides the tracing services.

10.2.1 BPE configuration

You can enable the IMS Connect traces by updating the IMS Connect startup configuration member, as specified by the BPECFG parameter. See Chapter 4, “Configuring IMS Connect” on page 43 for a description of this member.

The following keywords are available for the BPE configuration parameter member:

- ▶ **LANG**

The LANG parameter specifies the language used for IMS Connect BPE and IMS Connect messages. ENU is for U.S. English, which is currently the only supported language.

- ▶ **TRCLEV**

The TRCLEV parameter specifies the trace level for a trace table and, optionally, the number of pages of storage allocated for the trace table. TRCLEV controls the level of tracing (the amount of detail traced) for each specified trace table type. BPE-managed trace tables are areas in storage where IMS Connect BPE, and IMS Connect, can trace diagnostic information about events going on within the address space. Each trace table has a trace table type associated with it. A trace table's type refers to the kind of events that are traced into that table. For example, the BPE DISP trace table contains entries related to events in the BPE dispatcher.

IMS Connect BPE-managed trace tables are internal incore tables only. Trace records are not written to any external data sets. Some trace table types are defined and owned by IMS Connect BPE itself. These are known as system trace tables. IMS Connect also defines its own trace tables. These are known as component trace tables or user-product trace tables.

Refer to Chapter 4, “IMS Connect Definition and Tailoring,” in *IMS Version 9: IMS Connect Guide and Reference*, SC18-9287, for detailed information about the traces available.

10.2.2 Formatting incore trace tables

IMS Connect trace tables are *incore* tables, you can format them from a dump of an IMS Connect address space by using the IMS Connect Dump Formatter of the Interactive Problem Control System (IPCS).

The standard IMS Connect BPE formatting services format the traces. You must link-edit HWSFTRC0 with an alias of HWSFTvrn (where v is the version level, r is the release level, and m is the modification level). For example, IMS Connect V9.1.0 would have the alias of HWSFT910.

The HWSFTRC0 is link-edited as HWSFT910 and must reside in the IMS formatting library to use for formatting IMS Connect. Example 10-5 on page 149 shows the include, alias, and name statement.

Example 10-5 HWSFTRC0 link-edit

INCLUDE LOAD(HWSFTRC0)	HWS FORMATTED TRACE
ALIAS HWSFT910 HWS 9.1.0	FORMATTED TRACE
ALIAS NAME HWSFTRC0(R)	

You must use the parameters shown in Example 10-6 for the link-edit step.

Example 10-6 HWSFTRC0 link-edit parameters

```
//      PARM=( 'SIZE=880K,64K' ,RENT,REFR,  
//      NCAL,LET,XREF,LIST)
```

10.3 IMS Connect Dump Formatter

The IMS Connect Dump Formatter enables you to format various IMS Connect internal control blocks under the ISPF IPCS environment. The IMS Offline Dump Formatter enables you to display the following IMS Connect control blocks:

- ▶ EST
- ▶ SCT
- ▶ DCT
- ▶ INTT
- ▶ SVT
- ▶ DST
- ▶ HCDB
- ▶ FWE
- ▶ TWU
- ▶ CTOKN

You can also format IMS Connect BPE control blocks and trace entries from within the IMS Connect Dump Formatter. IMS Connect executes under the IMS Connect BPE.

10.3.1 IMS Connect Dump Formatter activation

Before you install the IMS Connect portion of the Interactive Dump Formatter, ensure that the Interactive Problem Control System (IPCS) is already working with ISPF and the IMS formatting functions. The actions to activate the IMS Connect Dump Formatter are different if you are using the IMS Connect integrated feature of IMS Version 9 or IMS Connect Version 2.2.

Integrated IMS Connect feature of IMS Version 9

If you have IPCS working with the IMS formatting facilities, you only have to ensure that the IMS Connect Dump Formatter modules are in a data set that is included in the IPCS TASKLIB data set concatenation.

The IMS Connect Dump Formatter modules are:

- ▶ HWSFT910 (alias of HWSFTRC0)
- ▶ HWSOB910

IMS Connect Version 2.2

After you have IPCS working with the IMS formatting facilities, you have to update the DD concatenations in Table 10-4 on page 150.

Table 10-4 DD concatenations

DDNAME	Data set to be added	Contents of data set
SYSPROC	CONN.SHWSCLST	Clists
ISPLLIB	CONN.SHWSPLIB	Panels
ISPTLIB	CONN.SHWSTLIB	Tables
TASKLIB	CONN.SHWSRESL	Formatting modules

After you have updated the concatenations, ensure that the IMS Connect Dump Formatter modules are in a data set that is included in the IPCS TASKLIB data set concatenation.

The IMS Connect Dump Formatter modules are:

- ▶ HWS\$B130
- ▶ HWS\$F130
- ▶ HWS\$S130
- ▶ HWSODF0\$
- ▶ HWSFT220 (alias of HWSFTRC0)
- ▶ HWSOB220

10.3.2 Accessing the IMS Connect Dump Formatter

You can access the IMS Connect Dump Formatter by selecting **DFSAAMPR** from the IPCS Component Analysis panel. You will receive the primary menu options for IMS dump formatting, as shown in Figure 10-3.

```

----- IMS DUMP FORMATTING PRIMARY MENU -----
OPTION  ==> _

  0  INIT          - IMS formatting initialization and content summary
  1  BROWSE        - Browse Dump dataset
  2  HI-LEVEL      - IMS Component level formatting
  3  LOW-LEVEL     - IMS ITASK level formatting
  4  ANALYSIS      - IMS dump analysis
  5  USER          - IMS user formatting routines
  6  OTHER COMP    - Other IMS components (BPE, CQS...)
  7  OTHER PROD    - Other IMS-related products
  E  EDA           - IMS Enhanced Dump Analysis
  T  TUTORIAL      - IMS dump formatting tutorial
  X  EXIT          - Exit IMS dump formatting

Enter END or RETURN command to terminate IMS component formatting.
Use PFKeys to scroll up and down if needed.

*****
*USERID - JOUK04
*DATE   - 05/06/30
*JULIAN - 05.181
*TIME   - 17:34
*PREFIX - JOUK04
*TERMINAL- 3278
*PF KEYS -
*****

```

Figure 10-3 IMS DUMP FORMATTING PRIMARY MENU panel

The way that you access the IMS Connect Dump Formatter depends on the type of IMS Connect that you are using.

IMS Connect integrated feature of IMS Version 9

Select the option 6 **OTHER COMP** in the IMS Dump Formatting Primary Menu panel. You see the IMS Component Selection Dump Formatting Menu, as shown in Figure 10-4 on page 151.

```

----- IMS COMPONENT SELECTION DUMP FORMATTING MENU -----
OPTION  ==> _

    1  BROWSE      - Browse dump dataset                      *****
    2  CQS         - Common Queue Server                     *USERID  - JOUK04
    3  OM          - Operations Manager                       *DATE    - 05/06/30
    4  RM          - Resource Manager                         *JULIAN   - 05.181
    5  SCI         - Structured Call Interface                *TIME     - 17:45
    6  IMS Connect - IMS Connect (9.1 and above only)         *PREFIX   - JOUK04
                                           *TERMINAL- 3278
                                           *PF KEYS  - 24
                                           *****

    B  BPE         - IMS BPE formatting (for any of the
                   above component address spaces)

    X  EXIT        - Exit IMS component menu

Enter END or RETURN command to terminate IMS component formatting.

```

Figure 10-4 IMS COMPONENT SELECTION DUMP FORMATTING MENU panel

From the option 6 **IMS Connect** of that panel, you can access the IMS Connect Dump Formatting Menu panel shown in Figure 10-6 on page 152.

IMS Connect Version 2.2

Select the option 7 **OTHER PROD** in the IMS Dump Formatting Primary Menu panel. You see the IMS-Related Product Selection Dump Formatting Menu, as shown in Figure 10-5.

```

----- IMS-RELATED PRODUCT SELECTION DUMP FORMATTING MENU -----
OPTION  ==> _

    1  BROWSE      - Browse dump dataset (IPCS norm)          *****
    2  IMS Connect - IMS Connect (2.2 and below only)         *USERID  - JOUK04
    3  ORS         - IMS ORS formatting                       *DATE    - 05/06/30
    4  DRF         - IMS Database Recovery Facility            *JULIAN   - 05.181
    5  IMSPM       - IMS Performance Monitor                   *TIME     - 17:54
                                           *PREFIX   - JOUK04
                                           *TERMINAL- 3278
                                           *PF KEYS  - 24
                                           *****

    X  EXIT        - Exit IMS component menu

Enter END or RETURN command to terminate IMS-related product formatting.

The above selections are only valid if the indicated products are installed.

```

Figure 10-5 IMS-RELATED PRODUCT SELECTION DUMP FORMATTING MENU panel

From the option 2 **IMS Connect** of that panel, you can access the IMS Connect Dump Formatting Menu panel, as shown in Figure 10-6 on page 152.

10.3.3 Using the IMS Connect Dump Formatter

Figure 10-6 on page 152 shows the IMS Connect Dump Formatting Menu panel.

```

----- IMS CONNECT DUMP FORMATTING MENU -----
OPTION  ==>  _

    0  INIT      - Show BPE status and initialize dump      *****
    1  BROWSE    - Browse dump dataset (IPCS norm)          *USERID   - JOUK04
    2  HI-LEVEL  - IMS Connect component level formatting    *DATE     - 05/06/30
    3  LOW-LEVEL - IMS Connect detail level formatting       *JULIAN    - 05.181
                                           *TIME     - 18:05
    B  BPE       - BPE formatting                           *PREFIX   - JOUK04
                                           *TERMINAL - 3278
    X  EXIT      - Exit IMS Connect dump formatting menu    *PF KEYS  - 24
                                           *****

Enter END or RETURN command to terminate IMS Connect formatting.

```

Figure 10-6 IMS CONNECT DUMP FORMATTING MENU panel

Before you can format a dump, you need to initialize it. You initialize a dump as follows:

1. Select option 0, **Show BPE status and initialize dump** from the IMS Connect Dump Formatting Menu panel. The IMS Connect initialization panel opens, as shown in Figure 10-7.

```

----- BPE DUMP CONTENT STATUS AND CONTROL -----
COMMAND ==>
SYMBOL BPECSCD NOT FOUND
Enter the BPE jobname to cause the BPE symbols to be set
for this dump.

If your dump formatter is at BPE 1.3 (IMS 7.1) or above, you
can also specify the address space by ASID, and you can get
a list of dumped BPE address spaces by leaving the jobname
and the ASID fields blank.

-----
JOBNAME      ASID      A.S. TYPE      DUMPED?
-----
BPE          -          ???          NO

BPE SDWA Address:      BPE Release:
BPE CSCD Address:      ??? Release:
??? Sys Name:          ??? Product #:

```

Figure 10-7 BPE DUMP CONTENT STATUS AND CONTROL panel

2. A message warning that the symbol HWSCSCD was not found opens. It ask you to enter the IMS Connect jobname or ASID to cause IMS Connect IPCS symbols to be set for the dump.

Specify either the jobname or the ASID of the IMS Connect address you want to format and press Enter. After you have provided a jobname or ASID, the remaining fields are filled out in the initialization panel, as shown in Figure 10-8 on page 153.

```

----- BPE DUMP CONTENT STATUS AND CONTROL -----
COMMAND ==> _

Enter the BPE jobname to cause the BPE symbols to be set
for this dump.

If your dump formatter is at BPE 1.3 (IMS 7.1) or above, you
can also specify the address space by ASID, and you can get
a list of dumped BPE address spaces by leaving the jobname
and the ASID fields blank.

-----
JOBNAME      ASID      A.S. TYPE      DUMPED?
-----
BPE                                     HWS          YES

BPE SDWA Address: 00000000      BPE Release: 010500
BPE CSCD Address: 11D016D0      HWS Release: 090100
HWS Sys Name:      HWS Product #: 5655-J38

```

Figure 10-8 BPE DUMP CONTENT STATUS AND CONTROL panel

- To obtain a list of all dumped IMS Connect address spaces, leave the jobname and ASID fields blank.
3. After initializing the dump, press PF3 to return to the IMS Connect Dump Formatting Primary Menu.

After you have initialized a dump, you can then use the options on the IMS Connect Dump Formatting Primary Menu to browse the dump data set, perform high-level or low-level IMS Connect formatting, or perform IMS Connect BPE formatting.

Figure 10-9 on page 154 shows a sample formatted dump. To obtain, it we choose option 2 **HI-LEVEL** in the IMS Connect Dump Formatting Menu panel. Then, we select option **HWSTRACE** to access the IMS Connect trace data by the high-level (IMS Connect component level) formatting function.

```

IPCS OUTPUT STREAM ----- Line 0 Cols 1 78
Command ==> _                SCROLL ==>
***** TOP OF DATA *****

*****
**** BPE DUMP FORMATTING FOR MSGCONN, ASID=006E, BPE VERSION=1.5.0 ****
*****

*****
* * * TRACE: BPE Trace Tables * * *
*****

-----
--- HWS ENVT Trace Table ---
-----

TTHE: 11D1B940
+0000 ID..... TTHE      LENGTH... 00000090  TYPE..... ENVT
+000C NUMPGS... 0002     ENTLEN... 0020      UDATALEN. 0000
+0012 TRCLEVCT. 0001     FLG1..... A0       LEVEL.... 04
+0016 SP..... 00        IDX..... 01        FLG2..... 80
+0019 RESERVED. 00      EXTMASK.. 0FFF      RESERVED. 00000000
      Trace Table Pointer Section:
+0020 TRTABLE.. 11D51000 FIRSTENT. 11D51000 LASTENT.. 11D52FE0
+002C OLDNEXT.. 11D51000 NEXTENT.. 11D51140 CYCLECT.. 00000000
      TTHE Chaining Section:
+0038 NEXT..... 00000000 FIRST.... 11D1B940 LATCH.... 00000000
+0044          00000000 TOKVAL... 00000000 UDATAPTR. 00000000
+0050 CSCDPTR.. 11D016D0 RESERVED. 00000000 RESERVED. 00000000

```

Figure 10-9 Sample of formatted dump (all IMS Connect trace)

IMS Connect Extensions

IMS Connect Extensions is an IBM DB2 UDB and IMS Tools program product (program number 5655-K48) that extends and enhances the services of IMS Connect. The tool helps you tune IMS and IMS Connect performance, perform problem determination, and better manage workloads, user exits, and security.

This chapter gives an overview of IMS Connect Extensions showing how IMS Connect Extensions can help you manage your IMS Connect system. This chapter includes the following topics:

- ▶ Introduction to IMS Connect Extensions
- ▶ Event collection and reporting
- ▶ Workload management
- ▶ Status Monitor
- ▶ Security
- ▶ User exits management
- ▶ IMS Connect problem determination with IMS Connect Extensions

Note: IMS Connect Extensions Version 1 Release 2 was announced on September 7, 2005 with the general availability date of October 7, 2005. We wrote this chapter and the examples during the summer of 2005 with V1.1. Refer to 11.8, “Highlights of IMS Connect Extensions Version 1 Release 2” on page 216 for a short description of the enhancements provided in V1.2.

11.1 Introduction to IMS Connect Extensions

IMS Connect Extensions provides the following features:

- ▶ Event recording and reporting

IMS Connect Extensions records details of IMS Connect internal events in journal data sets. They give information about these main points:

- Performance and response time for IMS, IMS Connect, and user message exits
- Availability for datastore and ports
- Throughput information for different transaction types
- Resource availability

IMS Connect Extensions provides batch utilities to format and print the recorded events. It also provides interfaces to the IMS Performance Analyzer and IMS Problem Investigator products.

- ▶ Workload management

IMS Connect Extensions enables you to dynamically manage workloads with the following services:

- Transaction routing

IMS Connect Extensions allows for the redirection of transactions from the original datastore destination, as specified by the client, to another datastore.

You define the rules and logical groupings of datastores to manage the transaction routing processing. Using this feature, you can redistribute resources without having to change client applications.

- Workload balancing

IMS Connect Extensions enables the redirection of transactions based on the capabilities of the individual datastores.

- Transaction pacing

IMS Connect Extensions protects datastores from surges in the numbers of transactions by detecting the surge and responding by rejecting the incoming message requests.

Transaction pacing allows IMS Connect Extensions to issue warning messages or to reject transactions if predetermined, user-specified incoming transaction threshold values are exceeded.

- ▶ Status Monitor

The IMS Connect Extensions Status Monitor displays the current activity status information for an active IMS Connect system.

The Status Monitor presents information for the IMS Connect system, datastores, and user message exits for various intervals over the previous hour. It also displays statistical information across the active ports. It enables you to monitor and display IMS Connect activity and utilization in real time.

- ▶ Security

IMS Connect Extensions enhances the security features of IMS Connect by giving an optional verification of user access to IMS Connect.

IMS Connect Extensions performs user ID and password validation. It creates ACEE structures for each user ID and saves them in a cache. On subsequent calls, IMS Connect Extensions does not reissue the security call, saving valuable system resources.

IMS Connect Extensions checks whether the user ID associated with an incoming message request is authorized to use the IMS Connect system.

- User exits management

IMS Connect Extensions supports dynamically reload, addition, deletion, disabling, and enabling of user exits.

IMS Connect Extensions operation

This section shows the components, commands, and activation steps of IMS Connect Extensions.

IMS Connect Extensions components

Figure 11-1 introduces the components of IMS Connect Extensions. Note that IMS Connect Extensions runs in the IMS Connect address space.

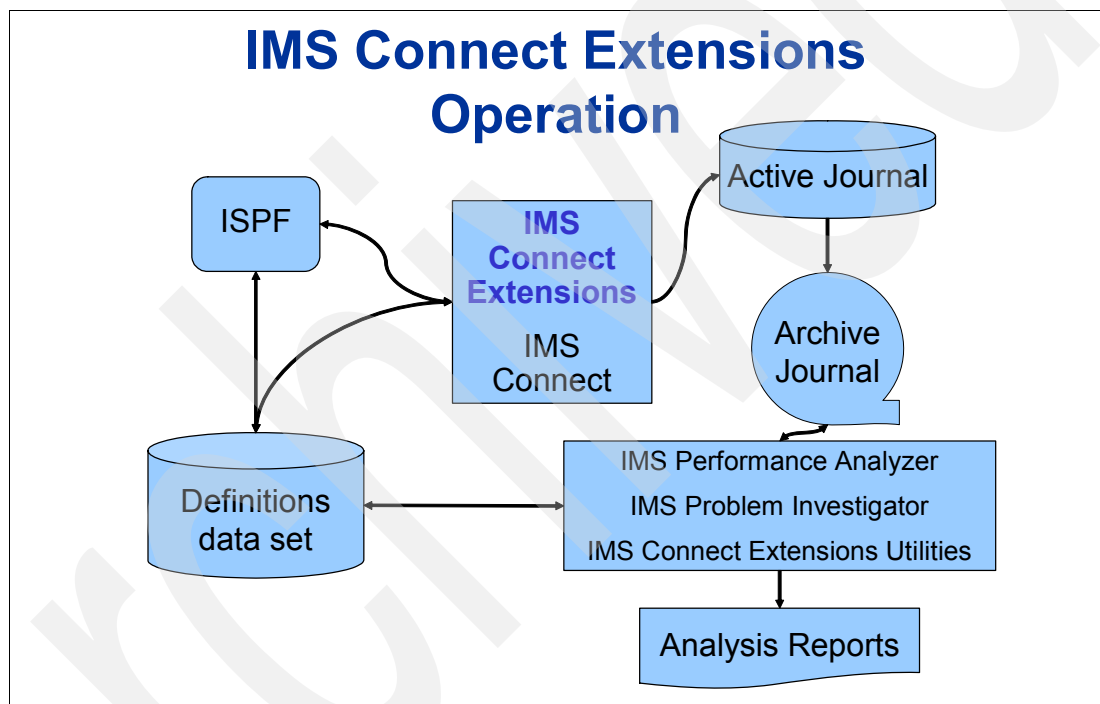


Figure 11-1 IMS Connect Extensions components

The following list describes the IMS Connect Extensions components:

- ISPF interface

IMS Connect Extensions provides an Interactive System Productivity Facility (ISPF) interface to set IMS Connect Extensions options, issue commands, and display IMS Connect activity. Figure 11-2 on page 158 shows the primary menu of the ISPF panel.

File Menu Help		
IMS Connect Extensions 1.1 - Primary Menu		
Option ==> _____		
1	Settings	Settings and user parameters
2	Definitions	Display or maintain IMS Connect Extensions definitions
3	Commands	Send commands to an active IMS Connect Extensions system
4	Status	Display the status of the IMS Connect environment
5	Log	Browse the IMS Connect Extensions Message log
X	Exit	Exit IMS Connect Extensions

Figure 11-2 IMS Connect Extensions Primary Menu

► Definitions data set

The definitions data set is a VSAM KSDS for your IMS Connect system definitions.

Every system definition defines the options and controls for an IMS Connect system. The system definition name *must* be the same HWS ID parameter in the IMS Connect configuration member; otherwise, IMS connections will not be activated.

You create a definition data set using the option 1 **Settings** in the ISPF primary menu. To create system definition, use option 2 **Definitions**.

► Event collection

The event collection monitors and records IMS Connect activity.

Through the option 2 **Definitions** in the ISPF panel, you can edit the your system definition to activate the event collection. You have to define the collection level and the journal management.

You can obtain reports from the archived journals using the IMS Performance Analyzer, IMS Problem Investigator, or the IMS Connect Extensions print utilities.

► Statistics collection

This component provides the data used by the Status Monitor to display current IMS Connect system and port utilization.

The Status Monitor is accessible through option 4 **Status** in the ISPF primary menu.

► Workload management

With IMS Connect Extensions, you can manage and control transaction workloads.

Using the ISPF interface, you are able to edit a system definition to activate transaction routing, transaction pacing, and workload balancing.

IMS Connect Extensions commands

IMS Connect Extensions commands are dynamic and can be executed without stopping IMS Connect.

The interface to issue commands is accessible through the option 3 **Commands** in the primary ISPF menu. Figure 11-3 on page 159 shows it.

_ File Menu Help		
Commands		
Option ==>		
1	User Exit	Alter IMS Connect User Message Exits
2	Refresh	Refresh IMS Connect Definitions from the Definitions data set
3	Security	Reload SAF Class rules or clear User IDs
4	Set	Set temporary override values for IMS Connect Definitions
5	Journal	Switch Journal data set
IMS Connect system . . <u>IMSGCONN</u> +		

Figure 11-3 IMS Connect Extensions Commands menu

The following commands are available:

► User Exit

This command enables you to reload, delete, disable, or enable the user exits defined to IMS Connect Extensions. It also supports dynamically adding new user exits.

► Refresh

This command causes IMS Connect Extensions to be rebuilt from the definitions data set. This affects, for example, pacing thresholds, transaction routing options, datastore capacity weightings, and security settings.

► Security

You use this command to refresh the Security Authorization Facility (SAF) class rules and to delete any cached user ID security profile.

► Set

This command temporarily overrides IMS Connect Extensions definition settings.

► Journal

This command causes the Journal Manager to switch the Active Journal data set and start using the next data set in the Active Journal rotation.

IMS Connect Extensions activation

IMS Connect Extensions installation is managed by normal SMP/E RECEIVE, APPLY, and ACCEPT services, and it is the same as other software products that run under the z/OS environment. Refer to *Program Directory for IBM IMS Connect Extensions for z/OS*, GI10-8504, for more information about the IMS Connect Extensions installation.

After you have IMS Connect Extensions installed, follow these steps to activate it:

1. Create a definitions data set.

Using the ISPF interface, create a definitions data set to store your system definitions. For example, if you have two IMS Connect systems, HWS1 and HWS2, you can define these in a single definitions data set or in separate data sets. If you want to share definitions across IMS Connect systems, keep all definitions in a single data set.

2. Create a system definition.

Create the system definition with the ISPF interface. It contains all the definitions for an IMS Connect system. You must define a system definition for every IMS Connect system, and the system definition *must* have the *same* name as the HWS ID parameter in the IMS Connect configuration member.

3. Modify and submit the IMS Connect JCL.

IMS Connect Extensions runs in the IMS Connect address space. To integrate IMS Connect Extensions with IMS Connect, make the following changes:

- a. Authorize the IMS Connect Extensions product and functional support load libraries (SCEXLINK and SFUNLINK).
- b. Edit your IMS Connect JCL, and in the STEPLIB DD, put the IMS Connect Extensions product and functional support load libraries *in front* of the IMS Connect load library. Add the CEXREPOS DD card for the IMS Connect Extensions definitions data set.
- c. Submit the JCL.

Example 11-1 shows a modified IMS Connect JCL to integrate IMS Connect Extensions.

Example 11-1 IMS Connect JCL with IMS Connect Extensions activated

```
//IMSGCONN PROC RGN=4M,SOUT=S,SYS1=,
//          BPECFG=BPECFG00,
//          HWSCFG=HWSCFG00
//STEP1 EXEC PGM=HWSHWS00,REGION=&RGN,TIME=1440,
//          PARM='BPECFG=&BPECFG,HWSCFG=&HWSCFG'
//STEPLIB DD DSN=CEX.V1R1M0.SCEXLINK,DISP=SHR
//          DD DSN=FUN.V1R1M0.SFUNLINK,DISP=SHR
//          DD DSN=IMS910G.SDFSRESL,DISP=SHR
//PROCLIB DD DSN=IMS910G.&SYS1.PROCLIB,DISP=SHR
//SYSPRINT DD SYSOUT=&SOUT
//SYSUDUMP DD SYSOUT=&SOUT
//HWSRCORD DD SYSOUT=&SOUT
//CEXREPOS DD DSN=CEX.V1R1M0.CEXREPOS,DISP=SHR
```

IMS Connect is now executing with IMS Connect Extensions. Check the CEXPRINT DD in the IMS Connect job to confirm that IMS Connect Extensions has started successfully. You can also see the IMS Connect Extensions Log through the option 5 **Log** in the ISPF primary menu.

Example 11-2 shows an IMS Connect Extensions Log. The log shows the options defined in the IMS Connect system definition.

Example 11-2 IMS Connect Extensions Log

```
FUN1003I Processing started at 2005-06-14 13:07:24
CEX5040I Exit HWSJAVA0 loaded at 1201B410, length 000002D0
CEX5040I Exit HWSMPL0 loaded at 0001F028, length 00001298
CEX5040I Exit HWSCSL00 loaded at 0000E238, length 00000B98
CEX5040I Exit HWSCSL01 loaded at 000204A8, length 00000B50
CEX5020I IMS Connect Extensions initialized and active, console port is 7004
CEX5021I Advanced features active, routing inactive, workload balancing inactive,
statistics collection active, event collection active with collection level 04
FUN2500I Journal Manager initialized with 3 active and 0 overflow journals
FUN2501I Journal full option is reuse, archiving is active
FUN2502I Archive JCL PDS is CEX.V1R1M0.SCEXSAMP, member is CEXARCH
FUN2515I Journal Manager now writing event records to active journal CEX.IMSGCON.ACTIVE.P03
```

Refer to *IBM IMS Connect Extensions for z/OS User's Guide*, SC18-7255, for more information about IMS Connect Extensions activation and configuration.

11.2 Event collection and reporting

This topic covers event collection, describing its use and the reports that you can obtain from the data collected.

Figure 11-4 provides an overview of event collection and reporting showing the flow of a transaction in IMS Connect and the reports that you can obtain with IMS Connect Extensions.

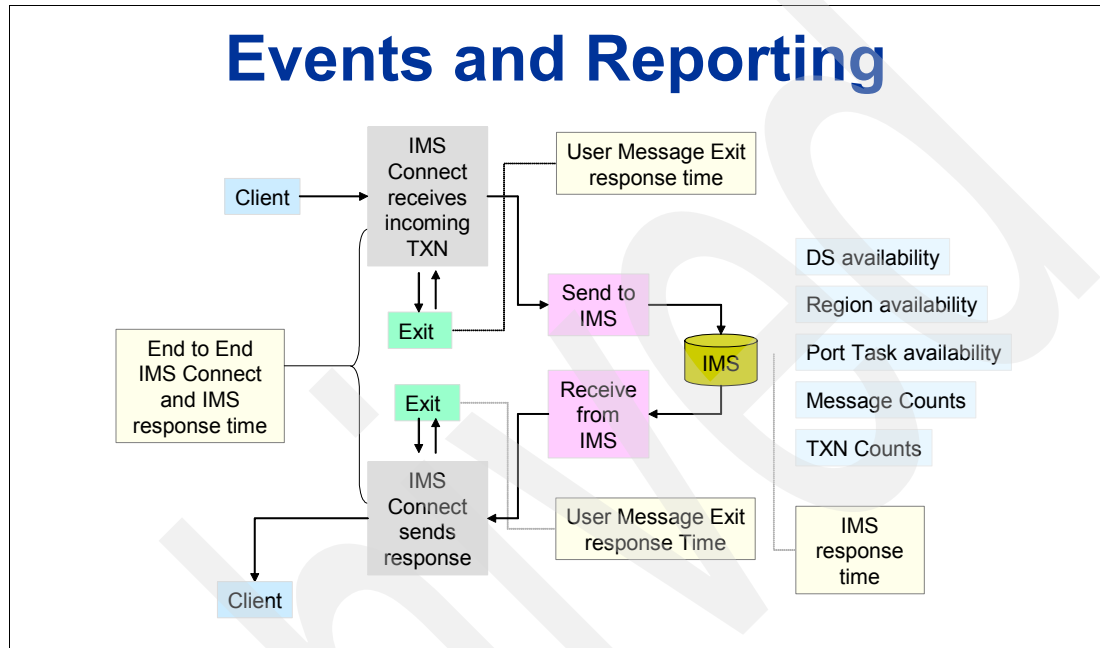


Figure 11-4 IMS Connect Extensions events and reporting

IMS Connect Extensions collects records from IMS Connect. These records provide information about end-to-end IMS Connect and IMS response time, user message exit response time, and IMS response time. They also provide information about resource availability and message and transaction counts.

11.2.1 Activate event collection

If the event collection feature is active, IMS Connect Extensions gets control for all significant IMS Connect events and collects them in an Active Journal. You can activate the event collection using the system definition panel for your IMS Connect system. Figure 11-5 on page 162 shows this panel with the required options selected.

```

. File Menu Settings Help
-----
EDIT                               System Definition
Command ==> _____

Name . . . . : MSGCONN
Description . . : IMS Connect -MSG

Product features:
/ Activate IMS Connect Extensions
/ Activate Event Collection
  Enter '/' to edit Journal Data Set template details
  - Active Data Set 'CEX.&ID..ACTIVE'
  - Archive Data Set 'CEX.&ID..&DATE..&TIME..ARCHIVE'
  Collection Level . . . 4          Log Record number . . A0 (Hex A0 - FF)

- Activate Advanced Features
- Activate Pacing
  Interval count . . . 3
  Warning threshold . . 0          Reject threshold . . 0

- Activate Security
- Activate ACEE Cache             Ageing interval . . . 60 (Minutes)
- Activate IMS Connect validation Security class . . .

- Activate Transaction Routing
- Define Applications for MSGCONN
- Activate Workload Balancing
- Activate Statistics Collection

```

Figure 11-5 IMS Connect Extensions event collection

You have to define the collection level used. The number and type of event records collected varies depending on the collection level specified for the IMS Connect system. These are the options:

- ▶ Level 0
Minimum level. Collects startup and shutdown events along with some error events.
- ▶ Level 1
Accounting level. Collects return from exit events, OTMA timeout, and session error events. This level provides accounting information in terms of the number of messages by transaction, user exit, and so on.
- ▶ Level 2
Transit time reporting. Collects the minimum number of records to run simple transit time reports.
- ▶ Level 3
Comprehensive performance analysis. Collects all TCP/IP read and write events for the analysis of TCP/IP activity.
- ▶ Level 4
Maximum level. Collects *all* event records.

You also define the log record number for IMS Connect Extensions event records through the field Log Record number. Acceptable field values are in the range x'A0' to x'FF'.

Recommendation: Do not use an existing IMS log record number. A unique log record number makes IMS Performance Analyzer reporting easier.

11.2.2 Journal management

You define how IMS Connect Extensions manages the Active and Archive Journals.

Active Journal

IMS Connect Extensions writes events to the Active Journal data sets. The Active Journal is a series of sequential data sets that are used in a rotation. When the first data set is full, the second data set is used. This continues until the last data set is full. At this point, IMS Connect Extensions only reuses Active Journal data sets after the event records are archived or if the **Reuse** option is set.

IMS Connect Extensions automatically allocates the Active Journal at startup based on the template (it also can be preallocated manually). You define the template details in the Active Journal Data Set panel shown in Figure 11-6.

File	Menu	Settings	Reset	Help
EDIT		Active Journal Data Set		End of data
Command ==> _____				
Data Set name	...	'CEX.&ID..ACTIVE'		
Number of data sets	3	(Number of Active Journal data sets)		
Journal full option	REUSE	(WAIT, DISCARD, REUSE, END)		
Archive Manager	...	_ No Archive		
Management class	...	_____ (Blank for default management class)		
Storage class	...	_____ (Blank for default storage class)		
Volume serial	...	IMST00 (Blank for system default volume)		
Device type	...	_____ (Generic unit or device address)		
Data class	...	_____ (Blank for default data class)		
Space units	...	TRKS (TRKS, CYLS)		
Primary quantity	...	50 (In above units)		
Number of Overflows	0	(Number of data sets in the overflow)		
Overflow size	...	4 (Multiple of Primary)		
Job Statement Information:				
==> //JOUK04 JOB (ACCOUNT),'CEX JOURNAL',CLASS=A,MSGCLASS=0,MSGLEVEL=(1,1)				
==> _____				
==> _____				
Archive JCL skeleton	'CEX.V1R1M0.SCEXSAMP(CEXARCH)'			

Figure 11-6 Active Journal template

This list describes the options that you have to set the management of Active Journals:

► Data Set name

This field enables you to specify the data set name prefix used to create the Active Journal data sets.

IMS Connect Extensions allocates the data sets, making them unique by adding a suffix to the data set name.

You can use the &ID parameter as part of the data set name. IMS Connect Extensions substitutes it with the system definition name when the data set is physically allocated. See Figure 11-6 for an example of the use of the &ID parameter.

► Number of data sets

This field enables you to specify the number of data sets in the Active Journal. You can have 1 to 32 data sets (the default value is 3). If you specify one data set, you *must* also specify the **Reuse** option in the Action flag. To select the appropriate number for your installation, you have to consider two points:

- How long you want to wait before accessing the data and running reports
- The amount of processing required by the Archive Manager to copy the files to the Archive Journal data set

► Journal full option

This field enables you to determine the reaction of IMS Connect Extensions if the overflow data set become full, or the active data sets become full and there is no overflow specified. You have these options:

– Wait

Stop reading event records until a data set becomes available. This option causes the IMS Connect system to stop processing incoming message requests. Use this option with care because it might impact transaction response time.

– Discard

Continue to create the event records but discard the records until a data set becomes available.

– Reuse

Reuse/overwrite the data sets.

– End

Shut down the Journal Manager. Event collection stops and cannot be restarted until IMS Connect is restarted.

► Archive Manager

This field enables you to specify whether the Archive Manager has to copy the Active Journals to the Archive Journal data sets.

If you activate the No Archive field, the Active Journal records are not copied to the Archive Journal.

► Allocation parameters

Set of fields to provide IMS Connection Extensions the allocation parameters values for the Active Journal data sets.

► Number of Overflows

This field enables you to specify whether IMS Connect Extensions will use an overflow data set if the Archive Manager falls behind and all Active Journal data sets become full. If the overflow data set also becomes full, the action indicated in the action flag is taken.

The values are 0 (no overflow data set is used, default value) or 1 (use an overflow data set if required). The overflow data set name prefix is the same as the Active Journal data set but with a different suffix.

► Overflow size

This field enables you to specify the size of the overflow data set. This is defined as a multiple of the Primary quantity field. Values are in the range 1 to 99. The default values is four times the primary quantity.

► Job Statement Information

These fields enables you to specify the JOB card to be used when the Archive Manager is submitted.

► Archive JCL skeleton

This field enables you to specify the data set and member name that contain the skeleton JCL to execute the Archive Manager. You can use the sample provided in the CEXARCH member of the IMS Connect Extensions sample library, SCEXSAMP.

For more information about the Archive Manager utility, see *IBM IMS Connect Extensions for z/OS User's Guide*, SC18-7255.

Archive Journal

The IMS Connect Extensions Archive Manager copies event records from the Active Journal data sets to the Archive Journal data sets (DASD or tape).

The Archive data sets on DASD are dynamically allocated by the Archive Manager based on a template that you define in the Archive Journal Data Set menu shown in Figure 11-7.

File Menu Settings Help	
EDIT Archive Journal Data Set Template	
Command ===> _____	
Data Set name	'CEX.&ID...&DATE...&TIME...ARCHIVE'
Management class	(Blank for default management class)
Storage class	(Blank for default storage class)
Volume serial	IMST00 (Blank for system default volume)
Device type	SYSDA (Generic unit or device address)
Data class	(Blank for default data class)
Retention period	(Blank or number of days)
Maximum active	3 (Number of Active data sets in Archive)
Maximum time	60 (Seconds to wait before closing Archive)
Maximum size	(Size of Archive Journal - MB)
Maximum volumes	1 (Number of volumes per Archive)

Figure 11-7 Archive Journal Data Set Template

You have the following options to set the Archive Journal management:

► Data Set name

This field enables you to specify the base data set name used to create the Archive Journal data sets. You can overtype the Archive Journal data set name to specify a new data set name.

The Archive Journal data set name must be defined either as a generation data set group (GDG) or using substitution variables. Use one of the options detailed in this list:

– A GDG base

You can specify a GDG base by entering (+1) at the of the data set name. For example:

'YOUR.ARCHIVE.FILE(+1)'

The GDG base must be preallocated using standard z/OS definition services.

- &DATE (or &JDATE) and &TIME

&DATE, &JDATE, and &TIME will be replaced by the date and time that the Archive Journal was created. (&JDATE is in julian format.) For example:

'YOUR.ARCHIVE.&DATE..&TIME'

- &FIRST or (&JFIRST)

&FIRST and &JFIRST will be replaced by the date and time of the first event on the Archive Journal. (&JFIRST is in julian format.) For example:

'YOUR.ARCHIVE.&FIRST'

- ▶ Allocation parameters

Set of fields to provide IMS Connection Extensions the allocation parameters values for the Archive Journal data sets.

- ▶ Maximum active

This field enables you to specify the maximum number of Active Journals that can be written to an Archive data set. When the number is reached, the Archive data set closes and a new Archive data set starts. Values are in the range of 2 to 32. The default is 3.

- ▶ Maximum time

This field enables you to specify the maximum time, in seconds, that the Archive Manager waits before checking to see if more Active Journals become available for archiving. Acceptable values are in the range of 0 to 120.

- ▶ Maximum size

This field enables you to specify the maximum size, in MB, of an Archive data set. When the size is reached, the Archive data set closes and a new Archive data set starts. Values are in the range of 0 to 32767. The default is no limit.

If the maximum size is reached while an Active Journal data set is being copied, the Active data set copy continues in the new Archive data set.

- ▶ Maximum volumes

This field specifies the number of tape volumes that an Archive data set can span. This value is ignored if Archive data sets are written to DASD devices. When the number is reached, the Archive data set closes and a new Archive data set starts. Values are in the range of 1 to 99. The default is one volume.

For a better understanding of the previous parameters, we give the rules governing the Archive Manager:

1. Whenever an Active Journal data set fills up, the Archive Manager job is submitted.
2. If the Active Journal data set is already being copied or has been copied by another Archive Manager job (because Maximum time or Maximum actives is set), the Archive Manager stops processing.
3. When the Archive Manager has completed copying the Active Journal data set, it processes as follows:
 - a. If the Maximum time is not set, the Archive Manager stops processing.
 - b. If the Maximum time is set, the Archive Manager waits for the specified time period and then checks if another Active Journal data set is full. If no Active data set is full and ready to be copied, the Archive Manager stops processing.
 - c. If the Archive Manager has exceeded the Maximum active value, the Archive Manager ends. The next Archive Manager job copies the new Active Journal data set.

- d. If the Archive Manager has exceeded the Maximum size value, the Archive Manager ends. The next Archive Manager job copies the new Active Journal data set.
 - e. If the Archive Manager is writing to tape and the Maximum volumes value has been exceeded, the Archive Manager ends. The next Archive Manager job copies the new Active Journal data set.
 - f. The Active Journal is copied to the current Archive Journal and processing returns to step 2.
4. When IMS Connect Extensions shuts down and any remaining event records in Active Journal data sets need copying, the Archive Manager job is submitted.
 5. When IMS Connect Extensions restarts and for some reason there are outstanding event records to be archived, the Archive Manager job is submitted.

Using the IMS Connect Extensions utilities and the tools IMS Performance Analyzer and IMS Problem Investigator, you can obtain reports from your Archive Journals.

11.2.3 IMS Connect event records

The IMS Connect Extensions event collection process collects event records continuously, while IMS Connect processes incoming message requests.

An event record consists of an event number and data associated with the event. The event number ranges from X'00' to X'FF' (decimal 0-255) and the associated data varies depending on the event number.

There are two types of event records:

► **Connect status event**

A Connect status event identifies a change in the status of your IMS Connect system, for example, a resource (datastore, TMEMBER) becoming available or unavailable, or a socket becoming accepted for input by a port task. Connect status events are typically not related to the processing of input messages, but can affect their processing.

Connect status event records are identified by a constant event key, 'EVNT'.

► **Message related event**

Message related event records identify an event in the processing of an incoming message request.

Message related event records have a store clock (STCK) token event key. For non-persistent sockets, each incoming message is assigned a unique event key, and every event associated with the processing of the message has the same event key.

For persistent sockets, all incoming messages are assigned the same event key. All events associated with the processing of all messages for the duration of the socket have the same event key.

IMS Connect Extensions can also process two types of trace records:

► **Recorder trace records**

IMS Connect Extensions provides utilities to convert the IMS Connect recorder trace data into variable length records in the format of IMS Connect Extensions event records.

► **IMS Connect trace records**

IMS can optionally collect IMS Connect trace event records. These event records include information about OTMA, Response Status Messages (RSMs), and exits output messages.

For a detailed description of the IMS Connect events records, refer to *IBM IMS Connect Extensions for z/OS User's Guide*, SC18-7255.

Message related events

IMS Connect Extensions associates message related event records with each other so that the sequence of events and event times can be identified and reported. IMS Connect Extensions associates the event records with each other using the event key. This enables IMS Connect Extensions to group together event records in the sequence they occur.

The collection of event records begins and ends with framing event records. IMS Connect Extensions uses the following framing events to clearly identify all events for a transaction or iteration of a conversational transaction:

- Start of frame

The Read Prepare (X'3C') event is used as the start of frame event record. This record starts a collection of event records related by a key token.

- End of frame

IMS Connect generates this trigger (X'48') event at the end of a multiple event process. It can be as a result of a deallocate request or other condition that represents the end of the process. The trigger event contains an indication of why the trigger event was generated.

For a better understanding of IMS Connect Extensions reports, we introduce the records involved in an IMS Connect flow for sync level none and sync level confirm transactions:

- Event flow: Commit mode 1, sync level none

Example 11-3 shows the event flow for a single commit mode 1, sync level none transaction. It presents the hexadecimal code of the events records and a description of them.

Example 11-3 Sample event flow: Commit mode 1, sync level none

3C Prepare READ Socket	→ Incoming message from client
49 READ Socket	
3D Message Exit called for READ	→ User exit processes the message
3E Message Exit returned from READ	
41 Message sent to OTMA	→ Sent to OTMA for processing
42 Message received from OTMA	
3D Message Exit called for XMIT	→ User exit processes the message
3E Message Exit returned from XMIT	
4A WRITE Socket	→ Response sent back to client
0C Begin CLOSE Socket	→ Non persistent socket closed
0D End CLOSE Socket	
48 Trigger Event CLOSE	→ IMS Connect has finished processing message

- Event flow: Commit mode 1, sync level confirm

Example 11-4 shows the event flow for a single commit mode 1, sync level confirm transaction. The difference with the previous example is that the client acknowledgement appears in the flow.

Example 11-4 Sample event flow: Commit mode 1, sync level confirm

3C Prepare READ Socket	→ Incoming message from client
49 READ Socket	
3D Message Exit called for READ	→ User exit processes the message
3E Message Exit returned from READ	
41 Message sent to OTMA	→ Sent to OTMA for processing
42 Message received from OTMA	

3D Message Exit called for XMIT	→ User exit processes the message
3E Message Exit returned from XMIT	
4A WRITE Socket	→ Response sent back to client
49 READ Socket	→ ACK received from client
49 READ Socket	
3D Message Exit called for READ	→ User exit processes the message
3E Message Exit returned from READ	
41 Message sent to OTMA	→ ACK sent to OTMA
42 Message received from OTMA	→ OTMA confirms commit
46 De-allocate Session	
3D Message Exit called for XMIT	→ User exit processes the message
3E Message Exit returned from XMIT	
4A WRITE Socket	→ Commit confirm sent to client
0C Begin CLOSE Socket	→ Non persistent socket closed
0D End CLOSE Socket	
48 Trigger Event	→ IMS Connect has finished processing message

► Event flow: Commit mode 0, sync level confirm

Example 11-5 shows the event flow for a single commit 0, sync level confirm transaction. In this case, IMS (OTMA) does not send a commit confirm to the client.

Example 11-5 Sample event flow: Commit 0, sync level confirm

3C Prepare READ Socket	→ Incoming message from client
49 READ Socket	
3D Message Exit called for READ	→ User exit processes the message
3E Message Exit returned from READ	
41 Message sent to OTMA	→ Sent to OTMA for processing
42 Message received from OTMA	
3D Message Exit called for XMIT	→ User exit processes the message
3E Message Exit returned from XMIT	
4A WRITE Socket	→ Response sent back to client
49 READ Socket	→ ACK received from client
49 READ Socket	
3D Message Exit called for READ	→ User exit processes the message
3E Message Exit returned from READ	
41 Message sent to OTMA	→ ACK sent to OTMA
45 OTMA Time-out	→ IRM_TIMER time-out occurred (No response from OTMA)
3D Message Exit called for XMIT	→ User exit processes the message
3E Message Exit returned from XMIT	
4A WRITE Socket	→ IRM_TIMER time-out notification sent to client
0C Begin CLOSE Socket	→ Non persistent socket closed
0D End CLOSE Socket	
48 Trigger Event	→ IMS Connect has finished processing message

Note: To receive data from the client, IMS Connect issues always two read sockets. The first one reads the first 32 bytes of the message to obtain the architecture fixed data (such as the LLLL value). The second read socket obtains the rest of the message.

IMS Connect trace records

To activate the IMS Connect trace records collection, use the SET command. Choose the option 1 **System Definition** in the Set Command panel to access the Set System Definition panel. Figure 11-8 on page 170 shows this panel.

File Menu Help	
Set System Definition	
Command ==> _____	
System Definition name : IMSGCONN	
Description : IMS Connect -IMSG	
Make changes and press ENTER to process SET SYSTEM_DEFINITION command choice	
Console settings:	
Message recall count . . <u>10</u>	
Product features:	
Collection level <u>4</u>	Tracing Level <u>1</u> _
Advanced features:	
- Activate Pacing	
Interval count <u>3</u>	
Warning threshold . . <u>0</u>	Reject threshold . . <u>0</u>
- Activate Security	
- Activate ACEE Cache	Ageing interval <u>60</u> (Minutes)
- Activate IMS Connect validation	Security class <u>FACILITY</u>
/ Activate Transaction routing	
/ Activate Workload Balancing	
/ Activate Statistics Collection	

Figure 11-8 Set System Definition panel

The Tracing Level field enables you to activate and set the level of tracing for IMS Connect Extensions. It writes the tracing records to the Active Journal data set, from where they are archived to the Archive Journals.

The acceptable field values are:

- ▶ 0: No tracing records are written.
- ▶ 1: Basic tracing records are written (OTMA, IRM, and RSA data).
- ▶ 2: In addition to the basic tracing information, client application data is written to the trace records.

Note: The SET command allows temporary changes. If you recycle IMS Connect, the tracing level is set to 0 again.

11.2.4 Event Collection print utility

The IMS Connect Extensions print utility, CEXEVTPR, prints summary and detailed event records. It produces report output of formatted IMS Connect Extensions event records. You set the level of detail to print.

The IMS Connect Extensions print utility accepts input from one Active Journal or Archive Journal data set.

Example 11-6 shows a sample JCL and the commands needed to request a report from the print utility.

Example 11-6 Sample JCL to run IMS Connect Extensions print utility

```
//userid JOB (ACCOUNT),
//* IMS Connect Extensions Print Utility
//CEXEVTPR EXEC PGM=CEXEVTPR,REGION=8M,
```

```
//          PARM='D'
//STEPLIB DD DISP=SHR,DSN=CEX.V1R1M0.SCEXLINK
// DD DISP=SHR,DSN=FUN.V1R1M0.SFUNLINK
//EVNTIN DD DISP=SHR,DSN=journal.data.set
//SYSUDUMP DD SYSOUT=*
//MSGOUT DD SYSOUT=*
/*
```

IMS Connect Extensions print utility, CEXEVTTPR, has a parameter to specify the type of report to be written. The acceptable values are:

- ▶ T: Transaction summary report
- ▶ D: Transaction detail report

Transaction summary report

Example 11-7 shows output for the transaction summary report for a commit mode 1, sync level none transaction.

The amount of data shown on the report varies depending on the collection level defined. In this case, we use collection level 4 to produce the most output.

Example 11-7 Print utility transaction summary report

```
ID=3C,060, V=01, TSKID=0105, KEY=BD2BAD70AD2B6765, PREPARE READ SOCKET
ID=49,073, V=01, TSKID=0105, KEY=BD2BAD70AD2B6765, READ SOCKET
ID=3D,061, V=01, TSKID=0105, KEY=BD2BAD70AD2B6765, MSG EXIT ENTERED
ID=3E,062, V=01, TSKID=0105, KEY=BD2BAD70AD2B6765, MSG EXIT RETURN
ID=41,065, V=01, TSKID=0103, KEY=BD2BAD70AD2B6765, MSG SENT TO OTMA
ID=42,066, V=01, TSKID=0103, KEY=BD2BAD70AD2B6765, MSG RCV FROM OTMA
ID=42,066, V=01, TSKID=0103, KEY=BD2BAD70AD2B6765, MSG RCV FROM OTMA
ID=3D,061, V=01, TSKID=0105, KEY=BD2BAD70AD2B6765, MSG EXIT ENTERED
ID=3E,062, V=01, TSKID=0105, KEY=BD2BAD70AD2B6765, MSG EXIT RETURN
ID=4A,074, V=01, TSKID=0105, KEY=BD2BAD70AD2B6765, WRITE SOCKET
ID=0C,012, V=01, TSKID=0105, KEY=BD2BAD70AD2B6765, BEGIN CLOSE SOCKET
ID=0D,013, V=01, TSKID=0105, KEY=BD2BAD70AD2B6765, END CLOSE SOCKET
ID=48,072, V=01, TSKID=0105, KEY=BD2BAD70AD2B6765, TRIGGER EVENT
```

This report enables you to follow the message processing flow in IMS Connect, in this case, the same flow introduced in Example 11-3 on page 168.

The transaction summary report displays the following fields:

- ▶ ID

The ID identifies the event record number. This is displayed in hexadecimal and decimal formats.
- ▶ V

The V identifies the version of IMS Connect Extensions.
- ▶ TSKID

The TSKID identifies the task in which the event occurred.
- ▶ KEY

The event key. This is either EVNT for Connect status event records or the token key for message related event records.

You use the event key to identify all the events related with a transaction. In Example 11-7, you can see that every event has the same key.

► Description

A short description of the event record.

Transaction detail report

Example 11-8 shows the transaction detail report for the same transaction of the previous example.

Example 11-8 Print utility transaction detail report

```
ID=3C,060, V=01, TSKID=0105, KEY=BD2BAD70AD2B6765, PREPARE READ SOCKET
TCPIB: LL=0028, BLKID=01, VVRR=0202, APAR=0001, PORT#=7003 , SOCKET#=0002
SOCKET FLAG=40, PORT FLAG=00, RMT REQ LEN=00000020, RMT ACT LEN=00000020
SPECIAL REQUEST DATA=00000000, RC=00000020, RMT RSN CODE=00000000
RECORD TIME=(DATE=2005.06.16, TIME=15.30.57.389904), LSN=0000000000002665

ID=49,073, V=01, TSKID=0105, KEY=BD2BAD70AD2B6765, READ SOCKET
TCPIB: LL=0028, BLKID=01, VVRR=0202, APAR=0001, PORT#=7003 , SOCKET#=0002
SOCKET FLAG=40, PORT FLAG=00, RMT REQ LEN=0000004B, RMT ACT LEN=0000004B
SPECIAL REQUEST DATA=00000000, RC=0000004B, RMT RSN CODE=00000000
RECORD TIME=(DATE=2005.06.16, TIME=15.30.57.389948), LSN=0000000000002666

ID=3D,061, V=01, TSKID=0105, KEY=BD2BAD70AD2B6765, MSG EXIT ENTERED
VAR DATA: APAR=0001, EXITN=HWSSMPL0
LEN=000A, APAR=0001, FUNC=READ, FLAG1=40, CONTENT=00
RECORD TIME=(DATE=2005.06.16, TIME=15.30.57.389961), LSN=0000000000002667

ID=3E,062, V=01, TSKID=0105, KEY=BD2BAD70AD2B6765, MSG EXIT RETURN
VAR DATA: APAR=0001, RC=00000000, RSN=00000000, EXITN=HWSSMPL0
LEN=0028, APAR=0001, FUNCTION=READ, FLAG1=40, FLAG2=00, UFLAG1=00, CONTENT=98
EXPXXX RC=00000000, EXPXX RSN=00000000
IPV4: FAMILY=0002, CLIENT PORT=1077 , CLIENT IP ADDRESS=009.001.059.179
DATA LENGTH=00000227, CLIENT ID=22575236
IRM DATA: SECTION LL=0012, APAR=0001, MSG LL=0050, F5=00, TIMER=00
SOCKET=00, SCHEMA=00, CLIENTID=
MSG DATA: LL=004E, APAR=0001, STATUS=80, ORIG DSN=IMSG , RTED DSN=
TRAN CODE=PART , USERID= , RACF GROUP= , MODNAME=
LTERM= , ORIG ID=22575236, EXIT #=0001, FLAG=14, SYNCLEVEL FLAG=20
SOCKET FLAG=00, AFLAG=00, AFLAG1=40, TFLAG1=04
RECORD TIME=(DATE=2005.06.16, TIME=15.30.57.389996), LSN=0000000000002668

ID=41,065, V=01, TSKID=0103, KEY=BD2BAD70AD2B6765, MSG SENT TO OTMA
DSIB: LL=0038, BLKID=02, FLAG=10, VVRR=0202, APAR=0001
DATASTORE NAME=SCSIM9G , DATA LEN=000001C5, DATA ADDR=133960B0
RC=00000000, RSN=00000000, TPIPE NAME=7003 , TOKEN=0000000000000000
OTMA CTL TYPE=TRAN, RESP=NONE, COMT=NONE, CMD=NONE, PROC= , CHAIN=FL
DUMP OF OTMA CONTROL SECTION FOLLOWS
+0000 01400000 0000F7F0 F0F34040 4040A0F0 *. ....7003 .0*
+0010 0000028F 00000000 00000000 00010000 *.....*
RECORD TIME=(DATE=2005.06.16, TIME=15.30.57.390088), LSN=0000000000002669

ID=42,066, V=01, TSKID=0103, KEY=BD2BAD70AD2B6765, MSG RCV FROM OTMA
DSIB: LL=0038, BLKID=02, FLAG=10, VVRR=0202, APAR=0001
DATASTORE NAME=SCSIM9G , DATA LEN=00000222, DATA ADDR=12F2A1A8
RC=00000000, RSN=00000000, TPIPE NAME=7003 , TOKEN=BD2BAD70AD2B6765
OTMA CTL TYPE=DATA, RESP=NONE, COMT=NONE, CMD=NONE, PROC= , CHAIN=F
DUMP OF OTMA CONTROL SECTION FOLLOWS
+0000 01800000 0000F7F0 F0F34040 404080F0 *.....7003 .0*
+0010 00000255 00000000 00000000 00010000 *.....*
RECORD TIME=(DATE=2005.06.16, TIME=15.30.57.432029), LSN=000000000000266A
```

```

ID=42,066, V=01, TSKID=0103, KEY=BD2BAD70AD2B6765, MSG RCV FROM OTMA
DSIB: LL=0038, BLKID=02, FLAG=10, VVRR=0202, APAR=0001
DATASTORE NAME=SCSIM9G , DATA LEN=000001D2, DATA ADDR=12F7A360
RC=00000000, RSN=00000000, TPIPE NAME=7003 , TOKEN=BD2BAD70AD2B6765
OTMA CTL TYPE=COMT, RESP=NONE, COMT=CMTD, CMD=NONE, PROC= , CHAIN=FL
DUMP OF OTMA CONTROL SECTION FOLLOWS
+0000 01080080 0000F7F0 F0F34040 4040A0E0 *.....7003 .\*
+0010 00000255 00000000 00000000 00010000 *.....*
RECORD TIME=(DATE=2005.06.16, TIME=15.30.57.432318), LSN=000000000000266B

ID=3D,061, V=01, TSKID=0105, KEY=BD2BAD70AD2B6765, MSG EXIT ENTERED
VAR DATA: APAR=0001, EXITN=HWSSMPL0
LEN=000A, APAR=0001, FUNC=XMIT, FLAG1=40, CONTENT=00
RECORD TIME=(DATE=2005.06.16, TIME=15.30.57.432401), LSN=000000000000266C

ID=3E,062, V=01, TSKID=0105, KEY=BD2BAD70AD2B6765, MSG EXIT RETURN
VAR DATA: APAR=0001, RC=00000000, RSN=00000000, EXITN=HWSSMPL0
LEN=0028, APAR=0001, FUNCTION=XMIT, FLAG1=40, FLAG2=00, UFLAG1=00, CONTENT=40
EXPXXX RC=00000000, EXPXX RSN=00000000
DATA LENGTH=00000128
RECORD TIME=(DATE=2005.06.16, TIME=15.30.57.432433), LSN=000000000000266D

ID=4A,074, V=01, TSKID=0105, KEY=BD2BAD70AD2B6765, WRITE SOCKET
TCPIB: LL=0028, BLKID=01, VVRR=0202, APAR=0001, PORT#=7003 , SOCKET#=0002
SOCKET FLAG=40, PORT FLAG=00, RMT REQ LEN=00000128, RMT ACT LEN=00000128
SPECIAL REQUEST DATA=00000000, RC=00000128, RMT RSN CODE=00000000
RECORD TIME=(DATE=2005.06.16, TIME=15.30.57.432579), LSN=000000000000266E

ID=0C,012, V=01, TSKID=0105, KEY=BD2BAD70AD2B6765, BEGIN CLOSE SOCKET
TCPIB: LL=0028, BLKID=01, VVRR=0202, APAR=0001, PORT#=7003 , SOCKET#=0002
SOCKET FLAG=40, PORT FLAG=00, RMT REQ LEN=00000000, RMT ACT LEN=00000000
SPECIAL REQUEST DATA=00000000, RC=00000000, RMT RSN CODE=00000000
RECORD TIME=(DATE=2005.06.16, TIME=15.30.57.432673), LSN=000000000000266F

ID=0D,013, V=01, TSKID=0105, KEY=BD2BAD70AD2B6765, END CLOSE SOCKET
VAR DATA: APAR=0001, RC=00000000, RSN=00000000
RECORD TIME=(DATE=2005.06.16, TIME=15.30.57.527691), LSN=0000000000002670

ID=48,072, V=01, TSKID=0105, KEY=BD2BAD70AD2B6765, TRIGGER EVENT
VAR DATA: APAR=0001, TRIGTYP=CLOSE
RECORD TIME=(DATE=2005.06.16, TIME=15.30.57.527720), LSN=0000000000002671

```

Compared with the transaction summary report, the transaction detail report adds a variable data that is specific for every event record.

An example of the information provided by this variable data is the field TYPE for messages sent to OTMA (event X'41') and messages received back from OTMA (event X'42'). It gives the type of message involved and has the following values:

- ▶ DATA: A data message
- ▶ TRAN: A transaction
- ▶ RESP: A message response
- ▶ CMD: A command
- ▶ COMT: A commit confirmed message

Refer to the event record mapping in the *IBM IMS Connect Extensions for z/OS User's Guide*, SC18-7255, for more information about variable data.

11.2.5 Recorder trace utility

IMS Connect Extensions provides utilities to process the IMS Connect recorder trace output data set. These utilities are:

► CEXRTCNV

Converts the recorder trace data into variable length records in the format of IMS Connect Extensions event records.

Example 11-9 shows a sample JCL for this utility.

Example 11-9 JCL to run recorder trace conversion utility

```
//CEXRTCNV JOB (ACCOUNT),'NAME'
//*
/*****
/* NAME: CEXRTCN
/* DESCRIPTION: IMS CONNECT EXTENSIONS
/* RECORDER TRACE CONVERSION UTILITY
/* FUNCTION: SAMPLE JCL TO RUN THE RECORDER TRACE CONVERSION UTILITY
*****/
/*
//STEP01 EXEC PGM=FUNEXEC,PARM=CEXRTCNV
//STEPLIB DD DISP=SHR,DSN=CEX.V1R1M0.SCEXLINK
// DD DISP=SHR,DSN=FUN.V1R1M0.SFUNLINK
//SYSUDUMP DD SYSOUT=*
//CEXPRINT DD SYSOUT=*
//MSGOUT DD SYSOUT=*
//RCDRIN DD DISP=SHR,DSN=your.recorder.input
//RCDROUT DD DISP=SHR,DSN=your.recorder.output
/*
```

► CEXRTPRT

Formats and prints the recorder trace event records created by CEXRTCNV.

Example 11-10 shows a sample JCL for this utility.

Example 11-10 JCL to run recorder trace print utility

```
//CEXRTPR JOB (ACCOUNT),'NAME'
//*
/*****
/* NAME: CEXRTPR
/* DESCRIPTION: IMS CONNECT EXTENSIONS
/* RECORDER TRACE PRINT UTILITY
/* FUNCTION: SAMPLE JCL TO RUN THE RECORDER TRACE PRINT UTILITY
*****/
/*
//STEP01 EXEC PGM=FUNEXEC,PARM=CEXRTPRT
//STEPLIB DD DISP=SHR,DSN=CEX.V1R1M0.SCEXLINK
// DD DISP=SHR,DSN=FUN.V1R1M0.SFUNLINK
//SYSUDUMP DD SYSOUT=*
//CEXPRINT DD SYSOUT=*
//MSGOUT DD SYSOUT=*
//RCDRIN DD DISP=SHR,DSN=your.recorder.output
/*
```

In Example 11-11 on page 175, you can see the formatted records produced by the recorder trace print utility.

Example 11-11 Recorder trace print utility output

```
ID=A0,160, TOKEN=00000001
ITOC RECORD: LEN=0056 ,APAR=0001 ,ID=IT ,CONTENT=C0
SMF: LEN=0052 ,RECORD TYPE=77 ,TIME=13313877 ,SEQ#=0105168F
UOW: CLNT=CLIENT01 ,RCVT=BD2D0A4811A8A120 ,ENQT=BD2D0A4811AA4C80
DS 1ST MSG DQT=0000000000000000, DS CLR DQT=0000000000000000
ERR TIME=0000000000000000, # MSG XMIT=0000, # MSG RCVD=0000
TIME=BD2D163D8115DF05, LSN=0000000000000001

ID=A1,161, TOKEN=00000001
ITOC REC: LEN=0010 ,APAR=0001 ,CON=80 ,CON1=02 ,TYPE=ITOC / RC / IPB
REMOTE CLIENT IRM: LLLL=X'00000093'
DUMP OF IRM HEADER FOR LENGTH=X'0050'
+0000 00500000 2A53414D 504C452A 00000000 *.&.....(&<.....*
+0010 00000000 C3D3C9C5 D5E3F0F1 00400140 *....CLIENT01. . *
+0020 C9E5E3D5 D6404040 C9D4E2C7 40404040 *IVTNO      MSG      *
+0030 40404040 40404040 40404040 40404040 *                  *
+0040 40404040 40404040 5C5C5C5C 5C5C5C5C *          *****
DUMP OF CLIENT MSG SEGMENT FOR LENGTH=X'003B'
+0000 003B0000 C9E5E3D5 D6404040 4040C4C9 *....IVTNO      DI*
+0010 E2D7D3C1 E840D3C1 E2E3F140 40404040 *SPLAY LAST1      *
+0020 40404040 40404040 40404040 40404040 *                  *
+0030 40404040 40404040 40404040 40404040 *                  *
DUMP OF CLIENT MSG SEGMENT FOR LENGTH=X'0004'
+0000 00040000 *....      *
```

The recorder trace print utility formats the IMS Connect trace, providing records to simplify the analysis.

11.2.6 Active session utility

IMS Connect Extensions provides the active session utility by APAR PQ97669. IMS Connect Extensions V1R2 further enhances this by providing an online ISPF-based viewer of active sessions. Refer to 11.8, “Highlights of IMS Connect Extensions Version 1 Release 2” on page 216.

The IMS Connect Extensions active session utility analyzes an IMS Connect Extensions Journal data set to determine the status of all active sessions. This is useful during problem determination of sessions that have been active for a long period of time.

A session is active after a Read Prepare event for a port/socket has occurred and before the trigger event following a close socket event has occurred. Sessions that are using persistent sockets will be active following the trigger event.

The IMS Connect Extensions active session utility accepts input from one Active Journal or Archive Journal data set. You have to provide to the utility the journals produced since the beginning of the session.

It produces a report output of formatted IMS Connect Extensions event records useful in identifying active sessions as part of problem determination. The report helps to identify the socket and the event key of the active transaction.

Example 11-12 on page 176 shows a sample JCL to use the active session utility.

Example 11-12 JCL to run active session utility

```
//userid JOB (ACCOUNT),'NAME'
//*
/* DESCRIPTION: IMS CONNECT EXTENSIONS ACTIVE SESSION UTILITY
/*
//STEP01 EXEC PGM=FUNEXEC,PARM='CEXJASKT,Y,U',REGION=0M
//STEPLIB DD DSN=FUN.V1R1M0.SFUNLINK,DISP=SHR
//          DD DSN=CEX.V1R1M0.SCEXLINK,DISP=SHR
//SYSUDUMP DD SYSOUT=*
//MSGOUT   DD SYSOUT=*
//EVTIN    DD DISP=SHR,DSN=journal.data.set
```

CEXJASKT has the following parameters:

► Help indicator

Specifies whether help description lines are to be printed. The values are:

- Y: Help description lines are printed in the report.
- N: Help description lines are suppressed in the report.

► Case indicator

Specifies whether the report lines are printed in uppercase or lowercase. The values are:

- L: Report is printed in lowercase.
- U: Report is printed in mixed case.

For an example, we send a commit mode 1 transaction that is not sending the ACK after the IMS response. Example 11-13 shows the report obtained from the active session utility.

Example 11-13 Active session utility report

```
IMS CONNECT EXTENSION ACTIVE SESSION REPORT VERSION 01:01:001

FIRST RECORD DATE=2005.06.17, TIME=23.49.41.713152

EOF ON INPUT DATASET
LIST OF ACTIVE SESSIONS

PORT N= 7003, SOCKET N= 9, KEY=BD2D5EEFF42C9200
EVENT HISTORY - KEY EVENTS PROCESSED FOR THIS MESSAGE
  READ PREPARE SOCKET
  READ EXIT RETURNED
  MESSAGE SENT TO OTMA
  MESSAGE RECEIVED FROM OTMA
  WRITE EXIT RETURNED
  WRITE SOCKET

STATE FLAGS AND STATUS INFORMATION AFTER RETURN FROM READ EXIT
  SVT FLAG=14
    ORIGINAL DESTID IS VALID
    UNSUPPORTED EXIT
  SVT SYNC FLAG=21
    COMMIT MODE 1
    SYNC LEVEL=1 (CONFIRM)
  SVT SOCKET FLAG=00
    TRANSACTION SOCKET

EVENT RECORD 62 READ EXIT RETURN DATA
  EXIT RC=00000000, EXIT REASON=00000000, TXNAME=PART , UID=
  EXIT NAME=HWSSMPLO, OLTERM= , CLIENT ID=22653060
  ORIGINAL DSID=IMSG , TARGET DISID=
```

```

IPV4: FAMILY=0002, PORT=0FE5, IP ADDRESS=009.001.039.117

EVENT RECORD 62 WRITE OR EXER EXIT RETURN DATA
EXIT RC=00000000, EXIT REASON=0000005E

EVENT RECORD TRACE - OLDEST TO NEWEST
EVENT=3C, PREPARE FOR SOCKET READ
EVENT=49, READ SOCKET
EVENT=49, READ SOCKET
EVENT=3D, USER MESSAGE EXIT ENTERED FOR READ, XMIT OR EXER
EVENT=3E, USER MESSAGE EXIT RETURN FOR READ, XMIT OR EXER
EVENT=41, MESSAGE SENT TO OTMA
EVENT=42, MESSAGE RECEIVED FROM OTMA
EVENT=3D, USER MESSAGE EXIT ENTERED FOR READ, XMIT OR EXER
EVENT=3E, USER MESSAGE EXIT RETURN FOR READ, XMIT OR EXER
EVENT=4A, WRITE SOCKET
LAST TRACE EVENT DATE=2005.06.17, TIME=23.50.23.180662

PREDICTED SESSION STATUS BASED UPON LAST TRACE ENTRY
P003 - WAITING FOR ACK/NAK FROM REMOTE CLIENT

```

The active session utility detects the active session and provides information such as the event key, port, the socket number within the port and the events related with the session. The active session utility also provides a predicted session status based on the sequence of event records processed for the session. In Example 11-13 on page 176, there is the following prediction, which is the situation that we forced:

WAITING FOR ACK/NAK FROM REMOTE CLIENT

After you have the event key provided by the utility, you can use IMS Problem Investigator or the IMS Connect Extensions print utility to obtain details of the events for this session.

For more information about the records displayed in the active session utility, refer to the updates to *IMS Connect Extensions for z/OS V1.1 User's Guide*, SC18-7255, available at:

http://www.ibm.com/support/docview.wss?rs=434&context=SSZJXP&dc=DA400&uid=swg27005885&loc=en_US&cs=utf-8&lang=en

11.2.7 IMS Performance Analyzer IMS Connect reports

IMS Performance Analyzer Version 3.3, program number 5655-E15, provides more in-depth analysis and reporting of IMS Connect event records.

IMS Performance Analyzer provides a comprehensive set of reports from the IMS Connect performance and accounting data collected by IMS Connect Extensions. The reports provide a summary and detailed analysis of IMS Connect transaction transit time, resource usage, and resource availability.

Selection criteria enable you to filter your reporting, for example, to include data only for a particular transaction code, user ID, and datastore and only for a specific period of time. IMS Performance Analyzer allows automatic archive selection based on date and time.

These are the IMS Performance Analyzer groups of reports by functional category:

- Transaction transit reports

These reports provide performance statistics to measure the performance of your IMS Connect transactions. Transaction transit (response) time is broken down into its components; input, processing (by OTMA), acknowledgement from the client, and output.

They can help identify any bottlenecks in transaction flow and are used for monitoring system performance, gathering diagnostic information, and tuning IMS.

This group contains the following reports:

- Transit analysis
- Transit log
- Transit extract
- Resource usage reports

These reports contain detailed and summary information about the use and availability of various IMS Connect resources including TCP/IP ports and Tpipes.

This group contains the following reports:

- Port usage
- RESUME TPIPE
- ACK/NAK
- Exceptions events
- Trace reports

These reports provide chronological listings of selected log records.

The following topics present every type of report. For information about how to create these reports with IMS Performance Analyzer, refer to *IBM IMS Performance Analyzer for z/OS User's Guide*, SC27-0912.

Transit Analysis report

The IMS Connect Transit Analysis report provides a summary of IMS Connect transaction performance. Performance data can be summarized by one or two sort keys, including time of day, transaction code, user ID, datastore (original and target), and port number.

Performance statistics are provided as averages, and optionally, peak percentiles. For example, you can specify 90 to report the elapsed time within which 90% of transactions completed. To be complete, this report requires IMS Connect Extensions to collect event data at collection level 3 or 4.

Example 11-14 shows a Transit Analysis report for sync level none transactions with no RACF security activated in IMS Connect. The transaction peak is set to 80%, and the data is summarized by transaction code.

Example 11-14 Transit Analysis report for sync level none transactions

IMS Performance Analyzer 3.3														
IMS Connect Transit Analysis - IMSGCONN														
From 13Jun2005 18.16.15.36 To 20Jun2005 14.47.43.06														
Transact Code	Message Count	Response Time	Input				-Process- OTMA	Confirm	Output			Rate /Sec	Time Outs	NAK
			Pre-OTMA	READ	Sock	READ Ex			SAF	Post-OTMA	XMIT			
IVTNO	234	Avg	9.284	0.229	0.052	0.025	0.000	8.794	0.000	0.261	0.027	0	0	0
		80%	23.329	0.396	0.077	0.033	0.000	22.826	0.000	0.371	0.040			
PART	50	Avg	103.958	0.332	0.057	0.129	0.000	37.566	0.000	66.058	0.041	0	0	0
		80%	154.813	0.934	0.086	0.655	0.000	86.206	0.000	96.568	0.068			
Total	284	Avg	25.952	0.247	0.053	0.043	0.000	13.859	0.000	11.845	0.029	0	0	0
		80%	65.134	0.541	0.078	0.265	0.000	39.495	0.000	36.503	0.046			

The report gives the total response time and the intermediate times divided by groups: input, process, and output times. It helps you to identify where is the cause of bad response times.

The IMS Performance Analyzer uses the connect events record time stamps to give the average values of the IMS Connect transaction performance. Figure 11-9 on page 179 uses

the event flow for sync level none transactions to show graphically the meaning of the most relevant fields of the Transit Analysis report.

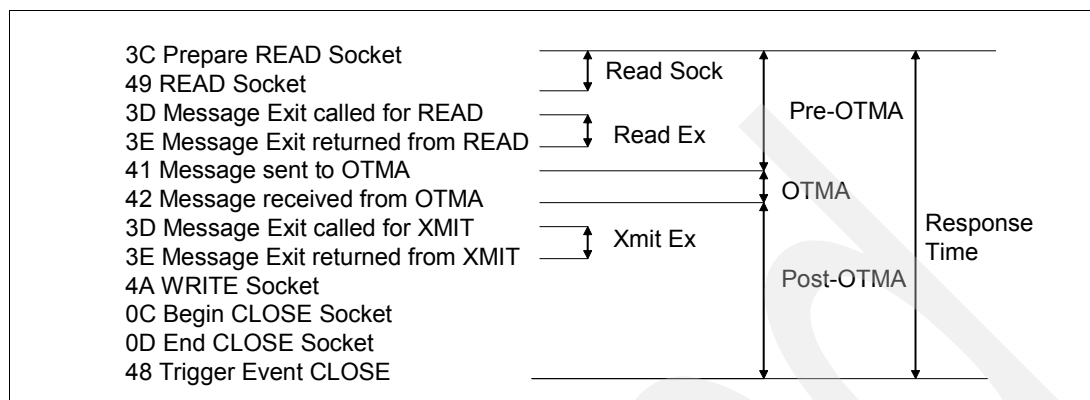


Figure 11-9 Transit Analysis report fields for sync level none transactions

The Transit Analysis report also gives timeout and NAK counters. In Example 11-15, you can see the Transit Analysis report in the same environment as the previous example but with sync level confirm transactions.

Example 11-15 Transit Analysis report for sync level confirm transactions

IMS Performance Analyzer 3.3															
IMS Connect Transit Analysis - IMSGCONN															
From 13Jun2005 19.23.57.91 To 20Jun2005 16.36.47.51															
Transact Code	Message Count	Response Time	Input				-Process- OTMA	Output			Page 1		Rate /Sec	Time Outs	NAK
			Pre-OTMA	READ Sock	READ Ex	SAF		Confirm	Post-OTMA	XMIT	Ex				
IVTNO	199	Avg 423.346	0.244	0.050	0.059	0.000	6.430	91.360	215.038	0.056	0	199	0		
PART	42	80% 509.536	0.425	0.071	0.083	0.000	16.250	118.189	280.791	0.093					
		Avg 223.626	0.210	0.048	0.094	0.000	32.199	111.543	79.672	0.055	0	0	0		
		80% 246.098	0.276	0.060	0.140	0.000	41.315	127.335	80.381	0.060					
Total	241	Avg 388.540	0.238	0.049	0.065	0.000	10.921	94.877	191.447	0.056	0	199	0		
		80% 490.035	0.405	0.069	0.096	0.000	23.640	120.919	265.232	0.090					

The report changes for the sync level confirm transaction. A new field, Confirm, appears and fields such as OTMA and the fields related with exits now have a different meaning.

Figure 11-10 on page 180 uses the event flow for sync level confirm transactions to show graphically the meaning of the most relevant fields of the Transit Analysis report. Notice that now the OTMA and exits average time is calculated as the addition of two different times.

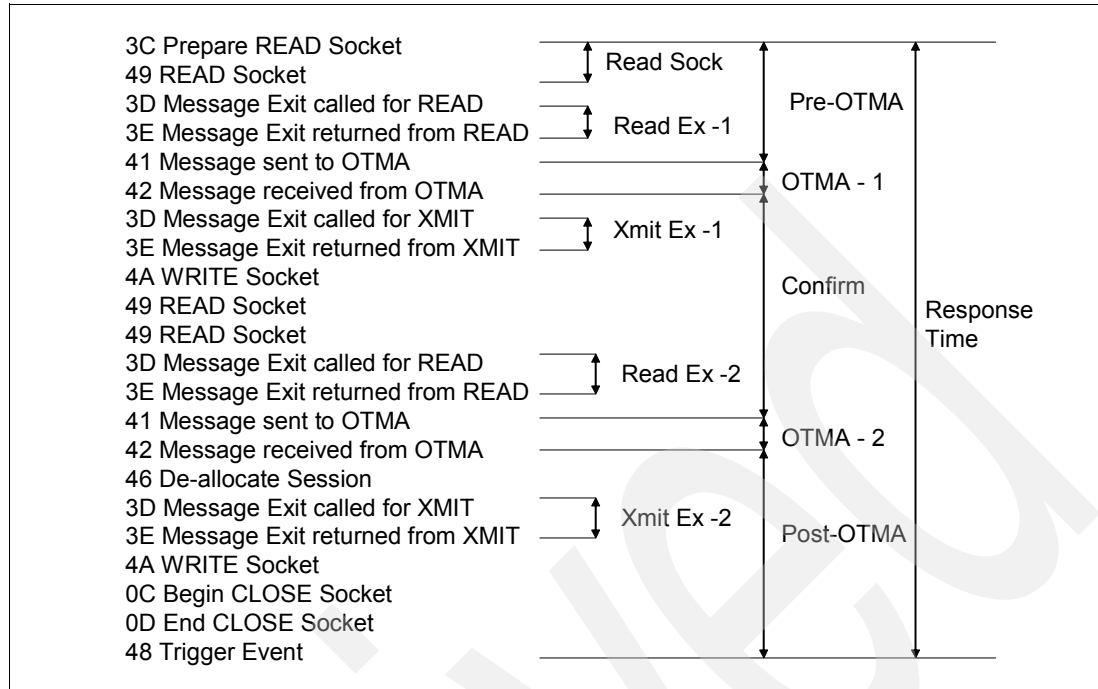


Figure 11-10 Transit Analysis report fields for sync level confirm transactions

To focus on the meanings of the time fields, the previous examples shows the total average times for every transaction, but you are able to obtain the same results by time intervals with total and subtotal times. Example 11-16 shows this Transit Analysis report option using one minute time intervals.

Example 11-16 Transit Analysis report by transaction and one minute time with subtotals

IMS Performance Analyzer 3.3																
IMS Connect Transit Analysis - IMSGCONN																
From 20Jun2005 17.17.22.77 To 20Jun2005 17.22.09.84																
Transact Code	Time	Message Count	Response Time	Input				-Process- OTMA	Output			Page 1		Rate /Sec	Time Outs	NAK
				Pre-OTMA	READ Sock	READ Ex	SAF		Confirm	Post-OTMA	XMIT Ex					
DSPINV	17.20.00	1	Avg	222.965	0.220	0.043	0.075	0.000	41.166	101.635	79.943	0.070	0	0	0	
		80%	222.965	0.220	0.043	0.075	0.000	41.166	101.635	79.943	0.070					
	17.21.00	4	Avg	213.761	0.244	0.070	0.171	0.000	25.613	101.011	86.892	0.052	0	0	0	
		80%	229.149	0.294	0.087	0.321	0.000	26.738	112.305	94.814	0.058					
DSPINV	17.22.00	1	Avg	439.132	0.189	0.040	0.084	0.000	38.251	105.004	295.687	0.054	0	0	0	
		80%	439.132	0.189	0.040	0.084	0.000	38.251	105.004	295.687	0.054					
	Subtotal	6	Avg	252.857	0.231	0.061	0.141	0.000	30.312	101.781	120.533	0.055	0	0	0	
		80%	330.675	0.274	0.079	0.263	0.000	36.550	110.632	193.081	0.063					
PART	17.17.00	8	Avg	232.005	0.239	0.044	0.112	0.000	27.809	117.323	86.633	0.055	0	0	0	
		80%	247.871	0.314	0.048	0.178	0.000	30.585	132.627	94.044	0.060					
	17.18.00	3	Avg	265.415	0.243	0.083	0.079	0.000	68.941	106.098	90.131	0.094	0	0	0	
		80%	317.124	0.263	0.113	0.082	0.000	127.488	118.752	101.920	0.124					
PART	17.21.00	4	Avg	198.215	0.198	0.072	0.066	0.000	28.143	84.567	85.306	0.051	0	0	0	
		80%	245.768	0.228	0.103	0.083	0.000	32.060	117.586	92.001	0.065					
	17.22.00	1	Avg	229.447	0.178	0.041	0.079	0.000	25.839	111.343	92.085	0.053	0	0	0	
		80%	229.447	0.178	0.041	0.079	0.000	25.839	111.343	92.085	0.053					
PART	Subtotal	16	Avg	229.662	0.225	0.058	0.092	0.000	35.482	106.655	87.298	0.061	0	0	0	
		80%	266.838	0.282	0.081	0.141	0.000	61.158	133.529	94.799	0.080					
-----			-----													
Total		22	Avg	235.988	0.227	0.059	0.105	0.000	34.072	105.326	96.362	0.060	0	0	0	
			80%	286.070	0.279	0.080	0.180	0.000	56.074	128.521	134.520	0.076				

Transit Log report

The IMS Connect Transit Log report provides performance details about every transaction processed by IMS Connect, providing a complete picture of transaction processing. To be complete, this report requires IMS Connect Extensions to collect event data at collection level 3 or 4. The order of transactions in the report is based on when they end, not when they start.

Example 11-17 shows a Transit Log report. The meaning of the fields are the same as in the Transit Analysis report with the addition of the port used and the datastore that processes the transaction.

Example 11-17 Transit Log report

IMS Performance Analyzer 3.3 IMS Connect Transit Log - IMSGCONN												
Log from 20Jun2005 17.17.22.77												
Start Time	Transact	Target	Port	Response	Input				-Process-	Page		
HH.MM.SS.Thmju	Code	DataStor	Number	Time	Pre-OTMA	READ	Sock	READ	Ex	OTMA	Confirm	Output
								SAF				Post-OTMA
												XMIT
												Ex
												E
17.17.22.778483	PART	IMSG	7003	214.073	0.308	0.042	0.133	0.000	29.777	104.885	79.101	0.055
17.17.26.045348	PART	IMSG	7003	224.563	0.257	0.057	0.073	0.000	26.696	117.480	80.129	0.067
17.17.32.586330	PART	IMSG	7003	250.720	0.177	0.042	0.075	0.000	25.707	131.270	93.565	0.049
17.17.43.082958	PART	IMSG	7003	208.069	0.427	0.041	0.300	0.000	23.704	105.741	78.196	0.048
17.17.45.298137	PART	IMSG	7003	219.878	0.176	0.040	0.090	0.000	31.792	103.990	83.918	0.062
17.17.48.061337	PART	IMSG	7003	232.730	0.179	0.042	0.074	0.000	31.820	105.939	94.790	0.055
17.17.50.724001	PART	IMSG	7003	261.835	0.199	0.041	0.075	0.000	23.820	156.111	81.704	0.053
17.17.58.487076	PART	IMSG	7003	244.168	0.186	0.043	0.073	0.000	29.156	113.164	101.660	0.050
17.18.02.030706	PART	IMSG	7003	232.823	0.220	0.068	0.083	0.000	29.529	122.312	80.760	0.061
17.18.14.209534	PART	IMSG	7003	336.252	0.268	0.123	0.077	0.000	149.227	103.346	83.409	0.087
17.18.35.630495	PART	IMSG	7003	227.170	0.240	0.059	0.076	0.000	28.067	92.637	106.225	0.133
17.20.46.704486	DSPINV	IMSG	7003	222.965	0.220	0.043	0.075	0.000	41.166	101.635	79.943	0.070
17.21.01.700277	DSPINV	IMSG	7003	225.803	0.291	0.095	0.101	0.000	27.305	97.813	100.392	0.062
17.21.05.265378	PART	IMSG	7003	245.768	0.183	0.045	0.083	0.000	32.664	115.857	97.062	0.053
17.21.08.812926	PART	IMSG	7003	108.873	0.183	0.049	0.035	0.000	25.623	0.000	83.065	0.031
17.21.18.622821	DSPINV	IMSG	7003	196.378	0.197	0.061	0.071	0.000	24.042	91.923	80.214	0.047
17.21.27.719390	PART	IMSG	7003	230.551	0.251	0.125	0.081	0.000	31.393	117.586	81.319	0.050
17.21.41.557448	DSPINV	IMSG	7003	232.871	0.189	0.048	0.075	0.000	25.639	120.782	86.260	0.050
17.21.49.355531	PART	IMSG	7003	207.667	0.174	0.068	0.065	0.000	22.890	104.825	79.776	0.071
17.21.57.135709	DSPINV	IMSG	7003	199.993	0.299	0.076	0.437	0.000	25.467	93.526	80.700	0.049
17.22.05.859506	DSPINV	IMSG	7003	439.132	0.189	0.040	0.084	0.000	38.251	105.004	295.687	0.054
17.22.09.842509	PART	IMSG	7003	229.447	0.178	0.041	0.079	0.000	25.839	111.343	92.085	0.053

Transit Extract report

The IMS Connect Transit Extract report gathers performance details about every transaction processed by IMS Connect. You can request a list or summary extract, or both. The List Extract report provides similar details as the Transit Log report, while the Summary Extract summarizes these details over a specified time interval, typically 15 minute intervals.

The extract data is suitable for importing into DB2 or PC tools from where you can run queries or produce reports and graphs. To be complete, this report requires IMS Connect Extensions to collect event data at collection level 3 or 4.

Port Usage report

The IMS Connect Port Usage report provides a summary of the TCP/IP ports used by the IMS Connect system. For each port, it provides average statistics for port depth, message processed count, and ACCEPT, READ, and WRITE socket counts. It also provides peak percentile statistics for input READ and ACK/NAK READ socket counts. To be complete, this report requires IMS Connect Extensions to collect event data at collection level 3 or 4.

Example 11-18 on page 182 shows a sample Port Usage report. In addition, the counters it also provides average times for READ socket commands (including ACK/NAK READ socket commands). The report can optionally be summarized by time interval.

Example 11-18 Port Usage report

IMS Performance Analyzer 3.3														
IMS Connect Port Utilization - DVPCFGDA														
From 08Mar2004 08.47.44.11 To 25Mar2004 12.20.02.83														
Port	-- Depth --		Message Count	ACCEPT Count	---- READ ----		--- Input ---		-- ACK/NAK --		READ ---		Page 1	
	Ave	Max			Count	Len	Average	90% Peak	Average	90% Peak	Count	Len		
8801	11	26	1010	443	3681	32	668.505	1.888.699	773.304	4.310.637	910	128		
8802	33	83	2500	2514	7412	6	1.072.717	2.006.237	0.000	0.000	2500	91		
8803	0	0	0	14	0	0	0.000	0.000	0.000	0.000	0	0		
8804	0	0	0	14	0	0	0.000	0.000	0.000	0.000	0	0		
8805	0	0	0	14	0	0	0.000	0.000	0.000	0.000	0	0		
8806	0	0	0	14	0	0	0.000	0.000	0.000	0.000	0	0		
8807	0	0	0	14	0	0	0.000	0.000	0.000	0.000	0	0		
8808	0	0	0	14	0	0	0.000	0.000	0.000	0.000	0	0		
8809	0	0	0	14	0	0	0.000	0.000	0.000	0.000	0	0		

Resume Tpipe report

The IMS Connect Resume Tpipe report provides a summary of Resume Tpipe command activity.

The report provides command statistics, including a command count and a breakdown by command type: Auto (with timeout), No Auto, and Single. Command statistics include the count of commands issued, IMS messages received, negative responses (Tpipe queue empty), NAK, and timeout interval. It can contain generated Tpipe names. This report requires IMS Connect Extensions to collect event data at collection level 2, 3, or 4.

Example 11-19 shows a sample Resume Tpipe report. The report can optionally be summarized by time interval.

Example 11-19 Resume Tpipe report

IMS Performance Analyzer 3.3																		
IMS Connect Resume Tpipe - DVPCFGDA																		
From 15Mar2004 09.25.43.79 To 15Mar2004 12.20.02.83																		
Page 1																		
----- Noauto ----- Auto ----- Single -----																		
Time		--- Msg ---						--- Msg ---						-----				
15Mar	Tpipe	Count	NResp	Fail	Avg	Max	Avg	Count	NResp	Fail	Avg	Max	Avg	Count	NResp	Fail	Avg	
							Timeout						Timeout				Timeout	
9.25.00	TRRBS001	1	0	0	2	2	0.25	0	0	0	0	0	0.00	0	0	0	0.00	
9.26.00	TRRBS001	0	0	0	0	0	0.00	0	0	0	0	0	0.00	2	0	0	0.25	
	TRRBS002	2	0	0	2	2	0.25	0	0	0	0	0	0.00	0	0	0	0.00	
9.27.00	CEX30001	1	0	0	2	2	0.25	0	0	0	0	0	0.00	0	0	0	0.00	
	TRRBS001	1	0	0	2	2	0.25	0	0	0	0	0	0.00	0	0	0	0.00	
	16034180	0	0	0	0	0	0.00	1	0	0	2	2	0.25	0	0	0	0.00	
9.28.00	CEX40002	0	0	0	0	0	0.00	0	0	0	0	0	0.00	1	1	0	7.00	
	TRRBS001	0	0	0	0	0	0.00	2	1	0	1	2	5.00	0	0	0	0.00	
0.14.00	TRRBS001	1	0	0	2	2	0.25	0	0	0	0	0	0.00	0	0	0	0.00	
0.15.00	TRRBS001	1	0	0	2	2	0.25	0	0	0	0	0	0.00	0	0	0	0.00	
	TRRBS002	0	0	0	0	0	0.00	0	0	0	0	0	0.00	2	0	0	0.25	
0.16.00	CEX40001	1	0	0	2	2	0.25	0	0	0	0	0	0.00	0	0	0	0.00	
0.24.00	CEX40001	1	0	0	2	2	0.25	0	0	0	0	0	0.00	0	0	0	0.00	

ACK/NAK report

The IMS Connect ACK/NAK report provides a summary of acknowledgement activity for transactions that use sync level confirm.

Positive acknowledgement (ACK) and negative acknowledgement (NAK) statistics are reported for each transaction code. NAK is further broken down as negative acknowledgement from either OTMA (NAK sense code) or from the client. It also includes NAK code explanations for OTMA NAKs.

Example 11-20 shows an ACK/NAK report summarized by time interval. In all these reports, when the input message is a command and not a transaction, a mnemonic prefixed by *CMD is reported in the Transact Code field.

Example 11-20 ACK/NAK report

IMS Performance Analyzer 3.3									
IMS Connect ACK/NAK - DVPCFGDA									
From 01Apr2004 11.57.06.98 To 01Apr2004 15.17.58.84									
Page 1									
Time	Transact	Target	Confirm	Count	Avg	Elaps	Count	Avg	Elaps
19Apr	Code	Datstore	Count	Count			Count		
12.00.00	PART	IMD3	1	0	0.000	0	0.000	1	1A Message cancelled due to IMS error
12.30.00	PART	IMD3	3	3	2.641.491	0	0.000	0	
13.00.00	*CMD_RHQ	IMD3	10	10	0.491	0	0.000	0	
	*CMD_RHQ	IMD4	10	10	0.323	0	0.000	0	
	*CMD_RTP	IMD3	8	8	0.293	0	0.000	0	
	*CMD_RTP	IMD4	8	8	0.361	0	0.000	0	
	DSPALLI	IMD3	1	1	271.069	0	0.000	0	
	DSPALLI	IMD4	1	1	217.887	0	0.000	0	
	DVPTRAN5	IMD3	2	0	0.000	2	255.411	0	
15.00.00	*CMD_RHQ	IMD3	4	4	0.395	0	0.000	0	
	*CMD_RHQ	IMD4	4	4	45.767	0	0.000	0	
	*CMD_RTP	IMD3	6	6	0.358	0	0.000	0	
	*CMD_RTP	IMD4	6	6	0.337	0	0.000	0	
	DVPTRAN5	IMD3	8	6	2.568.341	2	1.631.982	0	
	DVPTRAN5	IMD4	9	7	895.849	2	1.152.072	0	
	ID=EXAMP	IMD3	3	0	0.000	0	0.000	3	24 Previous conversation still in progress
	ID=EXAMP	IMD4	3	0	0.000	0	0.000	3	1A Message cancelled due to IMS error

Exception Events report

The IMS Connect Exception Events report provides details about events that cause transactions to fail or that signal critical resources are no longer available. The List report provides a list (in chronological order) of all exception events. To be complete, this report requires IMS Connect Extensions to collect event data at collection level 3 or 4.

Example 11-21 shows a sample Exception Events report. A summary report is also available, providing a recap of each exception event encountered and a count of the number of times it occurred.

Example 11-21 Exception Events report

IMS Performance Analyzer 3.3		
IMS Connect Exception Report - DVPCFGDA		
Report from 04Mar2004 10.19.36.22		
Information		
Event Time	ID	Description
10.19.36.229174	11	Datstore Un-Available
10.19.36.234938	11	Datstore Un-Available
10.19.36.376444	13	TMember leaves XCF group
10.19.36.380776	13	TMember leaves XCF group
10.27.50.682345	11	Datstore Un-Available
10.27.50.684494	11	Datstore Un-Available
10.27.50.940388	13	TMember leaves XCF group
10.27.50.944613	13	TMember leaves XCF group
08.49.02.395660	41	Msg to OTMA response is NAK
08.50.20.882001	11	Datstore Un-Available
08.50.20.882091	47	Session error
08.50.20.889827	11	Datstore Un-Available
08.50.20.899802	47	Session error
11.31.54.190824	13	TMember leaves XCF group
11.31.54.194165	13	TMember leaves XCF group
10.25.53.984955	45	OTMA time-out

IMS Connect Trace report

The IMS Connect Trace report provides detailed analysis of individual IMS Connect event records. Trace reports are typically used to investigate point-in-time performance problems

because they provide all available information. To focus on the desired problem area or to minimize the size of the report, specify the date and time range or selection criteria, or both.

The IMS Connect Trace report provides a list of transactions, each with detailed information about every event in the life of that transaction. You can see when a transaction starts, followed by all the events associated with the transaction in the order they occurred.

Example 11-22 shows a sample trace report for a commit mode 1 sync level confirm transaction.

Example 11-22 Trace report for a commit 1 sync level confirm transaction

IMS Performance Analyzer 3.3 IMS Connect Trace - IMSGCONN				Page 1	
Trace from 20Jun2005 19.40.39.08 Information					
Start/+Relative	Elapsed	ID	Description		
20.19.44.295864	*Start*	3C	Prepare Read Socket	Key=BD312B17A81F0660 Depth=1 Port=7003 Sock=2	
+0.000060	0.060	49	Read Socket	Port=7003 Sock=2	
+0.000127	0.066	3D	Message Exit called for READ	Exit=HWSSMPL0	
+0.000168	0.040	3E	Message Exit return for READ	RC=00 RSN=00 IP=9.1.59.179 DSOrig=IMSG TC=PART	
				Commit Mode=1 Synch Level=CONFIRM Socket Type=Transaction	
+0.000264	0.096	41	Message sent to OTMA	TPipe=7003 MSG=Transaction	
+0.025716	25.452	42	Message received from OTMA	TPipe=7003 MSG=Data Sense=0000	
+0.025902	0.185	3D	Message Exit called for XMIT	Exit=HWSSMPL0	
+0.025941	0.039	3E	Message Exit return for XMIT	RC=00 RSN=00	
+0.026179	0.237	4A	Write Socket	Port=7003 Sock=2	
+0.144417	118.238	49	Read Socket	Port=7003 Sock=2	
+0.144585	0.167	49	Read Socket	Port=7003 Sock=2	
+0.144610	0.025	3D	Message Exit called for READ	Exit=HWSSMPL0	
+0.144654	0.043	3E	Message Exit return for READ	RC=00 RSN=00 IP=9.1.59.179 DSOrig=IMSG TC=PART	
				Commit Mode=1 Synch Level=CONFIRM Socket Type=Transaction	
+0.144743	0.089	41	Message sent to OTMA	TPipe=7003 MSG=Response RESP=ACK	
+0.145384	0.640	42	Message received from OTMA	TPipe=7003 MSG=Commit Conf CONFIRM=Committed Sense=0000	
+0.145453	0.069	46	De-allocate Session	RSN=COMT	
+0.145474	0.020	3D	Message Exit called for XMIT	Exit=HWSSMPL0	
+0.145497	0.022	3E	Message Exit return for XMIT	RC=00 RSN=00	
+0.145689	0.191	4A	Write Socket	Port=7003 Sock=2	
+0.145720	0.030	0C	Begin Close Socket	Port=7003 Sock=2	
+0.225328	79.608	0D	End Close Socket	Port=7003 Sock=2	
+0.225356	0.028	48	Trigger event	Trigger=CLOSE	

For more information about the IMS Performance Analyzer reports, refer to *IBM IMS Performance Analyzer for z/OS Report Analysis*, SC27-0913.

11.2.8 IMS Problem Investigator

APAR PQ92211 in IMS Problem Investigator Version 1.1 introduces support for IMS Connect Extensions event record analysis.

You can use IMS Problem Investigator for z/OS to analyze IMS Connect event data. IMS Problem Investigator offers its full reporting and viewing capabilities to assist in the analysis of event data in Archive Journal data sets. Archive File Selection is available for automatic selection of archive data sets for batch analysis of an IMS Connect system.

IMS Connect Extensions settings

Before using the IMS Connect Extensions events in IMS Problem Investigator, you have to define the IMS Connect Extensions settings. Follow these steps:

1. From the Primary Option Menu, select option 0 **Profile**, and then from the submenu select option 4 **IMS Connect Extensions Settings**. Figure 11-11 on page 185 shows the menu.

```

File  Help
----- IMS PI Profile Options -----
File  Help
-----
CMD                      IMS Connect Extensions Settings
Command ==> _____

Specify IMS Connect Extensions settings.

Log Record Code . . . . . A0 (Hex A0-FF)

Definitions Data Set . . 'CEX.V1R1M0.CEXREPOS'

```

Figure 11-11 IMS Connect Extensions Settings menu

2. Specify:

- The Log Record Code that IMS Connect Extensions assigns to the event records. The default is A0.
- The name of the IMS Connect Extensions Definitions Data Set that defines the IMS Connect systems and archive data sets on which you want to report.

IMS Connect Extensions event analysis

To access to the IMS Connect Extensions Archive Journals, follow these steps:

1. Select option 6 **Connect** from the Primary Option Menu to view the list of IMS Connect systems and archive data sets. Figure 11-12 shows the IMS Connect System Definitions panel.

```

File  Menu  Help
-----
CMD                      IMS Connect System Definitions          Row 1 to 1 of 1
Command ==> _____          Scroll ==> PAGE

Definitions Data Set . . : CEX.V1R1M0.CEXREPOS

Enter "/" to select action.

      Name              Description              Changed              ID
____ IMSGCONN  IMS Connect -IMSG          2005/06/06 20:22:46  JOUK04
***** Bottom of data *****

```

Figure 11-12 IMS Connect System Definitions panel

2. You can specify **S** to display the list of Archive Data Sets, as shown in Figure 11-13.

File Menu Help			
CMD		Archive Data Sets for IMSGCONN	Row 1 to 24 of 37
Command ==>			Scroll ==> PAGE
Select to browse Archive data set.			
/	Data Set Name	----- From -----	To
___	CEX.IMSGCONN.D050610.T223841.ACTIVE	2005-06-10 18.36	18.38
___	CEX.IMSGCONN.D050613.T165911.ARCHIVE	2005-06-10 18.38	12.59
___	CEX.IMSGCONN.D050613.T173310.ARCHIVE	2005-06-13 13.29	13.33
___	CEX.IMSGCONN.D050613.T175428.ARCHIVE	2005-06-13 13.33	13.54
___	CEX.IMSGCONN.D050613.T181731.ARCHIVE	2005-06-13 13.54	14.17
___	CEX.IMSGCONN.D050613.T205908.ARCHIVE	2005-06-13 14.17	16.59
___	CEX.IMSGCONN.D050613.T220100.ARCHIVE	2005-06-13 16.59	18.00
___	CEX.IMSGCONN.D050613.T220808.ARCHIVE	2005-06-13 18.00	18.08
___	CEX.IMSGCONN.D050613.T221831.ARCHIVE	2005-06-13 18.08	18.18
___	CEX.IMSGCONN.D050613.T225058.ARCHIVE	2005-06-13 18.18	18.51
___	CEX.IMSGCONN.D050613.T225840.ARCHIVE	2005-06-13 18.51	18.58
___	CEX.IMSGCONN.D050613.T230616.ARCHIVE	2005-06-13 18.58	19.06
___	CEX.IMSGCONN.D050613.T232207.ARCHIVE	2005-06-13 19.06	19.22
___	CEX.IMSGCONN.D050613.T232340.ARCHIVE	2005-06-13 19.22	19.23
___	CEX.IMSGCONN.D050613.T232509.ARCHIVE	2005-06-13 19.23	19.25
___	CEX.IMSGCONN.D050614.T120708.ARCHIVE	2005-06-13 19.25	08.07
___	CEX.IMSGCONN.D050614.T170654.ARCHIVE	2005-06-14 12.52	13.06
___	CEX.IMSGCONN.D050616.T153028.ARCHIVE	2005-06-14 13.07	11.30
___	CEX.IMSGCONN.D050616.T153214.ARCHIVE	2005-06-16 11.30	11.32
___	CEX.IMSGCONN.D050616.T172526.ARCHIVE	2005-06-16 11.32	13.25

Figure 11-13 Archive Data Sets for IMS Connect Extensions list

3. You have three valid line actions:

- **SUB** to submit a report or extract request for the system.
- **S** to select this data set for processing (IMS Problem Investigator Formatted Browse).
- **P** to add file to the Process Log Files list. The data set name is placed at the top of the list from where you can then select it for dialog or batch processing.

Submit a report or extract a request for the system

When you choose the **SUB** option, the runtime options are displayed, as shown in Figure 11-14.

```
File Menu Help      Submit Report/Extract Request
File Menu Help
CMD IT      CEX.IMGCONN.D050610.T223841.ACTIVE
Command ==>

Specify submission options.      Report Interval
                                  YYYY-MM-DD  HH.MM.SS.TH
Filtering Criteria:              From 2005-06-13 14.00.00.00
Filter . . .      +              To  2005-06-13 14.30.00.00

Extract Output File:
Data Set Name . . .

Request Options                  Enter "/" to select option
/ Create Report                  / Edit JCL before submit
_ Create Extract Data Set

Formatting Options
_ Form _ STD _ Dump _ HEX0 _ HEX1
```

Figure 11-14 Submit Report/Extract Request

The submit options are the same as for IMS log reporting. The difference is that when **Report Interval** is specified, IMS Connect Archive File Selection is used to locate the archive data sets.

IMS Problem Investigator Formatted Browse

When you choose **S**, you access the IMS Problem Investigator Formatted Browse, as shown in Figure 11-15.

File Menu Mode Navigate Track Filter UTC Help			
CMD SE CEX.IMGCONN.D050613.T221831.ARCHIVE		Record 00000030 More: < >	
Command ==>		Scroll ==> CSR	
Forwards / Backwards . . HH.MM.SS.THMIJU		Time of Day . . HH.MM.SS.THMIJU	
Code Description		Date 2005-06-13 Monday Time (LOCAL)	
/ -----			
— A049	READ Socket		18.16.15.367667
— A03D	Message Exit called for READ, XMIT, EXER		18.16.15.367710
— A03E	Message Exit returned from READ, XMIT, EXER		18.16.15.367751
— A041	Message sent to OTMA		18.16.15.367793
— A03C	Prepare READ Socket		18.16.15.367891
— A049	READ Socket		18.16.15.367930
— A03D	Message Exit called for READ, XMIT, EXER		18.16.15.367945
— A03E	Message Exit returned from READ, XMIT, EXER		18.16.15.367967
— A041	Message sent to OTMA		18.16.15.368039
— A03C	Prepare READ Socket		18.16.15.368856
— A049	READ Socket		18.16.15.368895
— A03D	Message Exit called for READ, XMIT, EXER		18.16.15.368910
— A03E	Message Exit returned from READ, XMIT, EXER		18.16.15.368933
— A041	Message sent to OTMA		18.16.15.369003
— A03C	Prepare READ Socket		18.16.15.370056
— A049	READ Socket		18.16.15.370097
— A03D	Message Exit called for READ, XMIT, EXER		18.16.15.370113
— A03E	Message Exit returned from READ, XMIT, EXER		18.16.15.370134
— A041	Message sent to OTMA		18.16.15.370202
— A03C	Prepare READ Socket		18.16.15.371286
— A049	READ Socket		18.16.15.371376
— A03D	Message Exit called for READ, XMIT, EXER		18.16.15.371393
— A03E	Message Exit returned from READ, XMIT, EXER		18.16.15.371414
— A041	Message sent to OTMA		18.16.15.371488
— A03C	Prepare READ Socket		18.16.15.372157

Figure 11-15 IMS Problem Investigator Formatted Browse

You can select any of the events to view the details in formatted records. Figure 11-16 on page 189 shows an example for an x'3D' record. In the formatted records, you can see information such as the name of the exit called (HWSSMPL0) and the functions for which it is called (READ).

```

File Menu Format Help
-----
CMD SE      CEX.IMGCONN.D050623.T180603.ARCHIVE  Record 00000033 Line 00000000
Command ==>  _
Form ==>      + Use Form in Filter                Scroll ==> PAGE
Format ==> STD
***** Top of data *****
+0004 Code... A03D Message Exit called for READ, XMIT, EXER
+002A STCK... BD349A84E3872B84 LSN.... 0000000000000021
      Date... 2005-06-23 Thursday Time... 13.54.13.589106

+0000 CERE_3D_LL..... 003A
+0002 CERE_3D_ZZ..... 0000
+0004 CERE_3D_RECID..... A0 CERE_3D_EVTID..... 3D
+0006 CERE_3D_PFXLL..... 0014
+0008 CERE_3D_EFLAG..... 00 CERE_3D_VER#..... 01
+000A CERE_3D_TASKID..... ID of task recording event
+000A CERE_3D_COL#..... 01 CERE_3D_TKS#..... 05
+000C CERE_3D_EVKEY..... BD349A84E2782A84
+0014 CERE_3D_VAR_LL..... 000C
+0016 CERE_3D_VAR_APAR... 0001
+0018 CERE_3D_VAR_EXITN..... 'HWSSMPL0'

+0020 CERE_3D_VDR_1_SLL..... VDR section
+0020 CERE_3D_VDR_1_SLL..... VDR section length
+0020 CERE_3D_CX1_LL..... 000A
+0022 CERE_3D_CX1_APAR... 0001
+0024 CERE_3D_CX1_FUNC... 'READ'
+0028 CERE_3D_CX1_FLAG1..... 40
+0029 CERE_3D_CX1_CONT... 00
***** End of data *****

```

Figure 11-16 IMS Problem Investigator event detail formatted records

11.3 Workload management

As we introduced in 11.1, “Introduction to IMS Connect Extensions” on page 156, IMS Connect Extensions enables you to manage dynamic workloads with the following services:

- ▶ Transaction routing
- ▶ Workload balancing
- ▶ Transaction pacing

To manage the workload management, IMS Connect Extensions uses the definitions made in the options 4, 5, and 6 of the Definitions menu, as shown in Figure 11-17.

```

File Menu Help
-----
CEX
Option ==>  _ Definitions
-----
1 System Definitions Specify definitions for IMS Connect systems
2 User Exits         Specify User Message Exits
3 Datastores         Define datastore processing options
4 Datastore Groups   Group datastores for collective controls
5 Affinity Lists     Associate datastores for Affinity processing
6 Applications       Group transactions for Application controls
7 Transactions       Define transaction processing options

```

Figure 11-17 Definitions menu

The following list describes these options:

► **Datastore Groups**

A datastore group is a logical collection of datastores. Typically, datastore groups are constructed to reflect the development life cycle: development, testing, and production.

A datastore can only be a member of one datastore group. Datastore groups are useful during the construction of transaction routing rules for either transactions or datastores and for the setting of pacing thresholds.

► **Affinity Lists**

An affinity list is another logical group of datastores. Typically, affinity lists are constructed on the basis of workload (transactions). A datastore can be included in more than one affinity list. It is useful to define affinity lists when constructing transaction routing rules for either transactions or datastores.

► **Applications**

An application is a logical collection of transactions. A transaction can only belong to a single application.

Applications are useful in the construction of transaction routing rules. They simplify the definition of transaction routing rules by providing default options for all transactions belonging to the application. If required, transaction routing options can override the default application routing options.

11.3.1 Transaction routing

Transaction routing allows IMS Connect Extensions to alter the target IMS datastores that process incoming transaction requests by dynamically changing the target datastore used for IMS OTMA communication. This improves availability and performance.

An identifier within the incoming transaction request header sent by the client application normally determines the target IMS datastore for processing. However, IMS Connect Extensions provides routing by transaction that allows the target IMS datastore to be substituted with an alternative. IMS Connect Extensions does not route all incoming message requests. Table 11-1 shows which message types can be routed.

Table 11-1 Routing options for message types

Message type	Routing option
Conversational	The first message of a conversational transaction can be routed. All subsequent messages of a conversational transaction will be routed to the datastore that processed the first message in the conversation.
Non-conversational	A non-conversational transaction can be routed.
Send Only	A Send Only message can be routed. Note: If the output from a Send Only is retrieved using RTPIPE then transaction routing should not be active for these Send Only transactions.
RESUME TPIPE	A RTPIPE message <i>cannot</i> be routed. It will always be directed to the datastore as defined in the incoming message request. Ensure transaction routing is inactive for all transactions that create the asynchronous output or change RTPIPE processing and send RTPIPEs to all datastores that may have output.
ACK/NAK/DEALLOC	All ACKs, NAKs and DEALLOCs will be routed to the datastore that processed the first message.

This dynamic routing is performed using a combination of transaction affinity and datastore affinity:

► Transaction affinity

Transaction affinity enables you to define on which datastores a transaction can execute. If the transaction can execute on other datastores, you need to define the transaction and its affinity processing to IMS Connect Extensions.

Transaction affinity is the list of datastores that are candidates for processing the incoming transaction. This candidate list can be a single datastore or a list of datastores.

► Datastore affinity

Datastore affinity enables you to create groups of datastores with similar processing characteristics. For example, datastore affinity can be created for test or production datastores. By using datastore affinity, a test transaction might benefit from transaction routing but still be assured it will process only on a test system.

You set these affinities when you define a transaction to IMS Connect Extensions using the option 7 **Transactions** in the Definitions menu. Figure 11-18 shows the Transaction panel.

```

File  Menu  Settings  Help
-----
CEX      Transaction      Er
Command ==> _____

Name . . . . : PART
Description . . _____

Application . . . _____ +

Transaction processing options:
- Activate Transaction routing

- Override Application options

Route transactions to:
- 1. All Datastores
  2. Datastore . . . . . +
  3. Datastore Group . . . +
  4. Affinity List . . . . +

Routing Error processing:
- 1. Use the original datastore in the message request
  2. Reject the transaction
  
```

Figure 11-18 Transaction definition panel

This panel enables you to define the datastores where the transaction can be routed using datastores groups or affinity lists. From this panel, you activate or deactivate the routing option for a transaction. You also define the response of IMS Connect Extensions when the transaction routing logic cannot find a datastore to route the message.

You also define affinity when you define a datastore to IMS Connect Extensions using the option 3 **Datastores** in the definitions menu. Figure 11-19 on page 192 shows the panel.

_ File Menu Settings Help	
CEX	Datastore
Command ===> _____	
Name	: IMSG
Description	_____
Datastore Group	_____ +
Datastore Options:	
- Activate pacing	
Warning threshold	0 _____ Reject threshold 0 _____
- Activate Transaction routing	
Route transactions to:	
- 1. All Datastores	
2. Datastore	_____ +
3. Datastore Group	_____ +
4. Affinity List	_____ +
Datastore Controls:	
Capacity weight rating	1 _____

Figure 11-19 Datastore definition panel

This panel enables you to define the datastore affinity for a particular datastore. From this panel, you activate or deactivate the routing option for a datastore. You can also activate the transaction routing for an application when you define it.

Candidate list

The combination of transaction affinity and datastore affinity determines the candidate list of datastores that can process the incoming transaction. The candidate list for a transaction can contain multiple datastores.

Figure 11-20 on page 193 shows a graphical example of the rules used by IMS Connect Extensions to find a candidate list of datastores. The candidate list is built using the intersection of transaction affinity and datastore affinity. The diagram shows an IMS Connect system containing eight datastores and assumes an incoming message request running TX1 on datastore IMD2 and that the definitions for both TX1 and IMD2 have transaction routing set to Active.

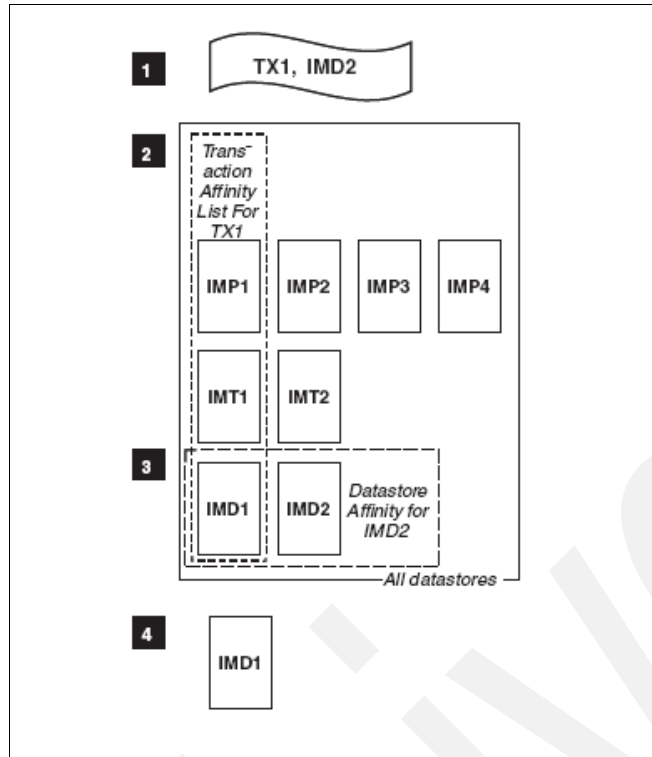


Figure 11-20 Candidate list for a single transaction

Step 1 in the diagram shows the incoming message request. Step 2 shows that TX1 has a transaction affinity of datastores IMP1, IMT1, and IMD1. Step 3 shows that IMD2 has a datastore affinity of datastores IMD1 and IMD2. Step 4 shows the intersection of transaction affinity and datastore affinity as IMD1. The message is routed from IMD2 to the new target datastore of IMD1.

11.3.2 Workload balancing

Workload balancing allows IMS Connect Extensions to redirect incoming transaction requests to one of multiple IMS datastores, thereby balancing the transaction workload across multiple IMS systems, ensuring both availability and responsiveness for the remote client.

Workload balancing uses a weighted rotate algorithm that considers the processing capacity of a datastore. Datastores with greater capacity receive more transactions. Basic rotation is achieved by assigning each datastore an equal weighting value. You specify the relative weighting of each datastore during the datastore definition process. See Figure 11-19 on page 192. The Capacity weight rating field accepts values in the range of 1 to 100.

If the workload balancing option on the system definition is set to Active and IMS Connect Extensions identifies that a datastore is no longer available, the datastore is not used to process incoming message requests.

Figure 11-21 on page 194 illustrates workload balancing. The figure shows an IMS system containing eight datastores and assumes an incoming message request running TX1 on datastore IMD1 and that the definitions for both TX1 and IMD1 have transaction routing set to Active.

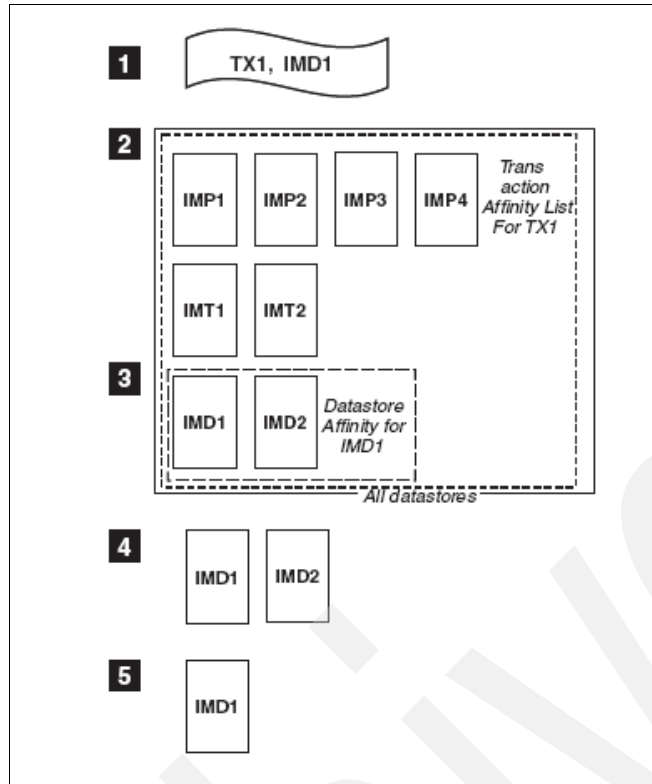


Figure 11-21 Workload balancing example

Step 1 in this diagram shows the incoming message request. Step 2 shows that TX1 has a transaction affinity of all datastores. Step 3 shows that IMD1 has a datastore affinity of datastores IMD1 and IMD2. Step 4 shows the intersection of transaction affinity and datastore affinity as IMD1 and IMD2. Step 5 shows workload balancing using the relative datastore capacity weightings of IMD1 and IMD2 to select the new destination. In this example, assume that IMD1 has twice the capacity of IMD2 and is the new target destination. The message is routed from IMD2 to the new target datastore of IMD1.

11.3.3 Transaction pacing

Transaction pacing enables IMS Connect Extensions to reject transactions if predetermined incoming message threshold values are exceeded. Transaction pacing is performed after transaction routing and workload balancing when the target datastore is determined.

You can define pacing thresholds at three levels of granularity:

- ▶ Datastores: The most granular. See Figure 11-19 on page 192.
- ▶ Datastore groups.
- ▶ IMS Connect system: The least granular. See Figure 11-5 on page 162.

At each of these levels, you can activate transaction pacing and set warning and reject threshold values. IMS Connect Extensions continually monitors incoming message arrival rates against these user-defined thresholds for identifying unusually high incoming message arrival rates.

At the IMS Connect system, IMS Connect Extensions defines transaction pacing using a number of controls:

- Interval count

This defines the number of consecutive 20 second intervals that must be exceeded before IMS Connect Extensions issues warning messages or rejects transactions. This value is set for the whole of the IMS Connect system and applies to all levels of transaction pacing.

- Warning threshold

At this threshold, IMS Connect Extensions issues a warning message to the IMS Connect Extensions Log and system console indicating that this threshold level has been exceeded.

- Reject threshold

At this threshold, IMS Connect Extensions rejects incoming message requests. IMS Connect Extensions issues a message to the IMS Connect Extensions Log and system console indicating that maximum pacing rates have been reached.

The SET and REFRESH commands allow the transaction pacing definitions to be altered dynamically.

Recommendation: Set pacing thresholds at the most granular level of datastores and datastore groups rather than for the IMS Connect system. This minimizes the overall number of incoming message requests that will be rejected.

11.4 Status Monitor

IMS Connect Extensions is constantly recording statistics about current activity and processing rates. It collects statistics at the end of every 20 second interval and displays on the Status Monitor for each IMS Connect system in an overview and detailed format. Statistics displayed on the screen are delayed by as much as 20 seconds.

The Status Monitor is displayed when you invoke option 4 **Status** of the IMS Connect Extensions Primary Options menu. Figure 11-22 shows the Status Monitor panel.

```
File  Menu  Help
-----
Status Monitor
CEX
Command ==>
Enter the name of the IMS Connect system.
Name . . . IMSGCONN +
View . . . 1  1. System
              2. Ports
```

Figure 11-22 Status Monitor panel

The Status Monitor provides an overview of activity for a single IMS Connect system. You have to specify its name. The Status Monitor has two views:

- System view: To select system information for IMS Connect systems, datastores, datastore groups, and user exits.

- The port view: To select port information for IMS Connect systems, datastores, and user exits.

To work with the Status Monitor, set the Activate Statistics collection option on the System Definitions panel to **Active**. See Figure 11-5 on page 162.

11.4.1 System view

With this view, you can access the System Overview and System Detail panels.

System Overview

The System Overview panel provides a global view of activity across the IMS Connect system. The panel displays information over various intervals for the following definition types:

- HWS: The IMS Connect system
- DG: Datastore groups
- DS: Datastores
- EXIT: User exits

Figure 11-23 shows the Status Monitor - System Overview panel.

File Menu Form Help

CEX

Status Monitor - System Overview

Row 1 of 8

Command ==>

Scroll ==> PAGE

Interval . . . 2 1. 20 Seconds 3. 01-15 Minutes 5. 31-45 Minutes 7. Hour
2. Minute 4. 16-30 Minutes 6. 46-60 Minutes

21.57.06

Name	Type	Input Message Total	Input Messages Per Sec	ACK Message Total	ACK Messages Per Sec	Routed Message Total
— IMSCONN	HWS	14	0.02	7	0.01	0
— IMSCG	DS	14	0.02	7	0.01	0
— Unknown	DS	0	0.00	0	0.00	0
— HWSJAVA0	EXIT	0	0.00	0	0.00	0
— HWSMPL0	EXIT	14	0.02	7	0.01	0
— HWSCSLO0	EXIT	0	0.00	0	0.00	0
— HWSCSLO1	EXIT	0	0.00	0	0.00	0
— Unknown	EXIT	0	0.00	0	0.00	0

Figure 11-23 Status Monitor - System Overview panel

The Status Monitor - System Overview panel recognizes the following commands:

- GO

The GO command activates automatic screen refresh. The screen is refreshed every 20 seconds. To stop the GO command, press ATTN/PA1.

- FORM

The FORM command displays the Form Definition screen. This enables you to customize the information displayed on this screen.

The Status Monitor displays information for the various definitions based on the current form definition. In the example, the default is modified to show data related to ACKs and messages routed by IMS Connect Extensions.

In the Interval field, you specify the time interval for which statistics are displayed. The line action enables you to access to the System Detail panel. Use \$ to select the definition.

System Detail

The Status Monitor System Detail panel displays statistics for a single definition over various time periods in the last hour. You can choose any of the components shown in the System Overview panel.

Figure 11-24 shows a System Detail panel for an IMS Connect system. The default data displayed is modified with the FORM command to show the incoming message processed by the read exit with RC=0 and accepted by IMS Connect Extensions (not rejected by security, pacing, and so on).

File Menu Form Help						
CEX Command ==>		Status Monitor - HWS MSGCONN				Row 1 of
						Scroll ==> PAGE
Interval . . . I	1. 20 Seconds	3. 01-15 Minutes	5. 31-45 Minutes	7. Hour		
	2. Minute	4. 16-30 Minutes	6. 46-60 Minutes			
Interval	Input Message Total	Input Length Total	Input Length Min	Input Length Max	Accepted Message Total	Accepted Messages Per Sec
22.52.26	32	3424	107	107	32	0.04
22.37.26	0	0	0	0	0	0.00
22.22.26	0	0	0	0	0	0.00
22.07.26	20	2140	107	107	20	0.02
Total	52	5564	107	107	52	0.01

Figure 11-24 System Detail panel for an IMS Connect system

As in the System Overview panel, you have the GO and FORM commands and the interval option.

11.4.2 Port view

With this view, you can access the Port Overview and Port Detail panels.

Port Overview

The Port Overview panel provides a global view of activity across all ports in the IMS Connect system. Figure 11-25 shows the Port Overview panel for an IMS Connect System with only one port.

File Menu Form Help

CEX Command ==> _____

Status Monitor - Port Overview

Row 1 of :
Scroll ==> PAGE

Interval . . . 1 1. 20 Seconds 2. Minute

Port	Input Message Total	Input Length Total	Input Length Min	Input Length Max	Routed Message Total	SendOnly Bytes Per Sec	RTPIPE Bytes Per Sec
_ 7003	8	856	107	107	0	0.00	0.00

Figure 11-25 Port Overview panel

As in the System Overview panel, you have the GO and FORM commands and the interval option. The interval option in this case shows the activity in the last 20 seconds or minute intervals.

In this case, we add information related to send only messages, routed messages, and RTIPIE with the FORM command. The line action enables you to access to the Port Detail panel. Use \$ to select the desired port.

Port Detail

The Port Detail panel provides detailed activity for a single port in the IMS Connect system. For this panel, you have two view options, Historical and System. Figure 11-26 shows the Port Detail panel for the System view, which shows the activity in the interval selected on the Port Overview panel.

File Menu Form Help							
CEX Command ==>			Status Monitor - Port Detail				Row 1 of
							Scroll ==> PAGE
Port . . . : 7003			Interval . . : 20 Seconds				
View . . . : 2			1. Historical 2. System				
23.27.06		Input	Input	SendOnly	ACK	NAK	SentErr
		Message	Length	Message	Message	Message	Length
Name	Type	Total	Total	Total	Total	Total	Ave
-----		-----	-----	-----	-----	-----	-----
IMSGCONN	HWS	6	642	0	3	0	20
IMSG	DS	6	642	0	3	0	20
Unknown	DS	0	0	0	0	0	0
HWSJAVA0	EXIT	0	0	0	0	0	0
HWSSMPL0	EXIT	6	642	0	3	0	20
HWSCSL00	EXIT	0	0	0	0	0	0
HWSCSL01	EXIT	0	0	0	0	0	0

Figure 11-26 Port Detail: System view

In this case, we add information related to send only messages, ACK, NAKs, and sent error messages sent to the client with the FORM command. Figure 11-27 shows the Port Detail panel for the Historical view, which shows the port activity in different points of time.

File Menu Form Help						
CEX		Status Monitor - Port Detail				Row 1 of 8
Command ==>						Scroll ==> PAGE
Port . . . : 7003		Interval . . . : 20 Seconds				
View . . . : 1		1. Historical 2. System				
Interval	Input Message Total	Input Length Total	SendOnly Message Total	ACK Message Total	NAK Message Total	SentErr Length Ave
23.36.06	7	749	0	3	0	20
23.35.46	4	428	0	2	0	20
23.35.46	4	428	0	2	0	20
23.35.26	8	856	0	4	0	20
23.35.26	8	856	0	4	0	20

Figure 11-27 Port Detail: Historical view

As in the Port Overview panel, you have the GO and FORM commands. The interval option is the one selected in the Port Overview panel.

11.4.3 Form definition

Forms control the presentation of the Status Monitor screen output. Forms enable you to tailor the formatting of the data displayed by the Status Monitor screens. You can modify the Status Monitor to present only information relevant to you, making it easier and quicker to interpret the screens.

Forms are defined using the FORM command or from Form in the action bar of most Status Monitor panels. You can define a form for each Status Monitor panel. Figure 11-28 shows the panel that opens when you enter **form** at the command line. It displays the current form definition.

```

File  Menu  Form  Help
-----
CEX                                     Status Monitor
Command ==> _____ Form Definition                                     Row 1 to 6 of 6
                                                                    Scroll ==> PAGE

/   Name +      Count/   Func +   Description
-   INPUT      COUNT   TOTAL   Input Messages
-   INPUT      LENGTH   TOTAL   Input Messages
-   SENDONLY    COUNT   TOTAL   Send Only Messages
-   ACK         COUNT   TOTAL   ACK Count
-   NAK         COUNT   TOTAL   NAK Count
-   SENTERR     LENGTH   AVE     Sent Unsuccessful
***** Bottom of data *****

```

Figure 11-28 Form Definition panel

Use the Form Definition panel to tailor the information displayed on the Status Monitor to meet your individual requirements:

- ▶ Insert new columns in the Status Monitor display.
- ▶ Delete existing columns from the display.
- ▶ Alter the order in which columns are displayed.

You can choose the fields and formats that you need. Using the **PROMPT** (F4) in the Name field, you obtain a list of the allowed values, as shown in Figure 11-29.

```

----- Status Monitor -----
CEX                                     Field Names   Row 1 to 18 of 19
Command ==> _____ Scroll ==> PAGE

Name      Description
. INPUT    INPUT MESSAGES
. RTPIPE   RESUME TPIPES
. SENDONLY SEND ONLY MESSAGES
. SENTERR  SENT UNSUCCESSFUL
. SENTOK   SENT SUCCESSFUL
. ACCEPTED ACCEPTED MESSAGES
. REJECTED REJECTED MESSAGES
. ROUTED   ROUTED MESSAGES
. ACK      ACK COUNT
. NAK      NAK COUNT
. DEALLOC  DEALLOC COUNT
. RETURNED RETURNED MESSAGES
. REJEXER  MESSAGES REJECTED BY EXER
. REJSEC   MESSAGES REJECTED BY SECURITY
. REJPACE  MESSAGES REJECTED BY PACING
. REJROUTE MESSAGES REJECTED BY ROUTING
. REQUEST  MESSAGE REQUESTS
. IGNORE   MESSAGES IGNORED

```

Figure 11-29 Field values for form definitions

11.5 Security

IMS Connect Extensions enhances the security features of IMS Connect.

User ID and password validation

IMS Connect Extensions performs user ID and password validation by making a call to the installation security system using the standard SAF interface.

It creates ACEE structures for each user ID and saves this ACEE control block in a cache. On subsequent calls for the same user ID, password, or PassTicket, group, and application, IMS Connect Extensions does not reissue the security call, instead it uses the ACEE from the cache, which is more efficient.

You activate the security using the System Definitions panel. Figure 11-30 shows the fields involved.

/	Activate Security	
/	Activate ACEE Cache	Ageing interval . . . 60 (Minutes)
/	Activate IMS Connect validation	Security class . . . FACILITY

Figure 11-30 Security fields in System Definitions panel

User ID ACEE ageing interval

IMS Connect Extensions keeps the ACEEs in the cache for a specific period defined by the Ageing interval on the System Definition panel. The ACEE structure is cleared when this time interval is reached. An ageing interval of 0 minutes indicates that the ACEEs are not deleted after a certain time period.

Any ACEE structure that is in the cache for a period greater than the ageing interval is not used. IMS Connect Extensions makes a new security call to perform user ID and password verification whenever the:

- ▶ Password or PassTicket has changed
- ▶ Group or application has changed
- ▶ ACEE has been in the cache for longer than the ageing interval

IMS Connect validation

This feature of IMS Connect Extensions checks whether the user ID associated with an incoming message request is authorized to use the IMS Connect system. IMS Connect Extensions rejects the message request if security returns an invalid status.

IMS Connect Extensions performs this validation against a predefined security resource class. This resource class can be an existing resource class, but it must be defined in the security class table with a length of 56 bytes or less.

If IMS Connect Extensions is performing IMS Connect validation, it preloads the security resource class in storage to improve performance.

Online security commands

Through the Commands menu shown in Figure 11-2 on page 158, you can access the security commands.

Figure 11-31 shows the Security Commands panel. Use this panel to:

- ▶ Refresh SAF class rules.
- ▶ Delete any cached user ID security profiles.

File	Menu	Help
CEX Security Commands		
Command ==> _____		
IMS Connect system . : IMSGCONN		
Enter "/" to select option		
_ Refresh SAF Class rules		
_ Clear ACEE for User Id (or prefix) : _____		

Figure 11-31 Security Commands panel

Not: Activate either IMS Connect security or IMS Connect Extensions security. Do not activate both, because this doubles the amount of security processing.

11.6 User exits management

IMS Connect Extensions enables you to manage user exits without interrupting IMS Connect execution.

11.6.1 User exits definition

To manage exits with IMS Connect Extensions, you have to define them through the option 2 **User Exits** in the Definitions panel. Figure 11-32 shows the User Exit panel.

File	Menu	Settings	Help
CEX User Exit			
Command ==> _____			
Name : HWSJAVA0			
Description . . _____			
/ Activate Exit Definition			
Choose how IRM offsets are determined			
1 1. Use default IRM offsets as referenced in the IMS Connect sample exit			
2. Use IRM offsets as defined below:			
User ID Offset (0 or 29-65527)			
Password Offset (0 or 29-65527)			
Group Offset (0 or 29-65527)			
APPLname offset (0 or 29-65527)			
3. Use supplementary exit named below:			
Pre-Processing Exit . . . _____			
Identify whether Exit supports 4 byte length prefix			
/ Use a 4 byte length prefix for Client			
Message ID support			
MSG ID1 . . . 1 1. ASCII 2. EBCDIC			
MSG ID2 . . . 2 1. ASCII 2. EBCDIC			

Figure 11-32 User Exit definition panel

The following list explains the most relevant fields:

► **Activate Exit Definition**

This field enables you to activate this user exit. If the user exit is not active (or not defined), no IMS Connect Extensions features can execute for incoming message requests processed by this user exit.

► **IRM offset definition**

IMS Connect Extensions requires some information defined in the user portion of the IRM header. This field enables you to define the IRM offsets used by this user exit. The acceptable values are:

- 1. Use default offsets as referenced in the IMS Connect sample exits supplied by IBM
- 2. Use IRM offsets as defined below

The security fields are in fixed offsets within the IRM as defined in this list:

- **User ID Offset**

The IRM offset containing the security user ID. Acceptable values are 0 or in the range of 29 to 65527. A value of zero (0) or blank indicates that the field is not present.

- **Password Offset**

The IRM offset containing the security password. Acceptable values are 0 or in the range of 29 to 65527. A value of zero (0) or blank indicates that the field is not present.

- **Group Offset**

The IRM offset containing the security group name. Acceptable values are 0 or in the range of 29 to 65527. A value of zero (0) or blank indicates that the field is not present.

- **APPLname offset**

The IRM offset containing the security APPL name. Acceptable values are 0 or in the range of 29 to 65527. A value of zero (0) or blank indicates that the field is not present.

- 3. Use supplementary exit named below

The security fields are not located at fixed offsets within the IRM. A supplementary user exit is be called to provide the fields as each message is received and processed by IMS Connect Extensions. Enter the name of a user exit that defines the IRM format used.

► **Length prefix**

This field enables you to specify whether this user exit uses a 4 byte length prefix for messages returned to the client. The acceptable values are:

- /
User exit uses a 4 byte length prefix.
- Blank
User exit uses a 2 byte length prefix.

► **Message ID support**

These fields enables you to specify whether MSG ID1 and MSG ID2 for this user exit support ASCII or EBCDIC.

The acceptable values are:

- 1. ASCII

- 2. EBCDIC

The default values for the IBM-supplied exits are ASCII (MSG ID1) and EBCDIC (MSG ID2), excluding HWSJAVA0 where the values are reversed.

Defining IBM sample exits to IMS Connect Extensions

You can define the standard IBM-supplied exits to IMS Connect Extensions using a **load** command. You can invoke this command from the action bar or the **load** command on the User Exit List panel. It shows you the list of IBM sample exits not defined in IMS Connect Extensions. You can load them by using **S** in the action field. Figure 11-33 shows the load IBM sample User Exits panel.

```

Definitions - User Exits
CEX                               Row 1 to 3 of 3
Command ==>

The following IBM Sample User Exits are not
defined. Select those you wish to add.

Name      Description
- HWSIMSO0  IBM Sample Exit
- HWSIMSO1  IBM Sample Exit, 4 byte prefix
- HWSMPL1   IBM Sample Exit, 4 byte prefix
***** Bottom of data *****

```

Figure 11-33 IBM sample User Exits panel

11.6.2 User exits commands

Through the Commands menu shown in Figure 11-2 on page 158, you can access the user exits commands. Figure 11-34 shows the User Exit Commands panel.

```

File  Menu  Help
CEX                               User Exit Commands          Row 1 to 4 of 4
Command ==>                               Scroll ==> PAGE

IMS Connect system . . : IMSGCONN

Enter "/" to select action

Name      Description      Status
- HWSJAVA0
- HWSSMPL0
- HWSCSL00
- HWSCSL01
Active
Active
Active
Active
***** Bottom of data *****

```

Figure 11-34 User Exit Commands panel

You can reload, add, delete, or make active/inactive for the next message requesting processing your user exits. IMS Connect Extensions performs it without interrupting the execution of IMS Connect.

The User Exit Commands panel recognizes the following command:

► ADD

The ADD command loads a new user exit executable. The user exit must exist in one of the STEPLIB data sets associated with the IMS Connect system and be defined to IMS Connect Extensions.

IMS Connect Extensions initializes the new user exit executable and after successful initialization is ready to process incoming messages for the MSG IDs specified.

Note the following considerations:

- The ADD request is rejected if the new user exit uses message IDs that are used by existing user exits.
- IMS Connect Extensions does not automatically update the IMS Connect configuration member with the name of the user exit. You must do this manually.
- After a user exit has been added, it counts against the total number of message exits allowed by IMS Connect. This is true even if the user exit is later deleted or disabled.

The User Exit Commands panel enables the following actions through the action field:

► **RELOAD**

The RELOAD option reloads a new copy of the user exit executable.

The RELOAD command cannot be used to change or alter the message ID strings supported by the user exit. If you need to change the message ID string for a user exit, use the ADD command to add a new user exit supporting the message IDs.

After the RELOAD command is issued for a given user exit, the exit must process at least one input message before the RELOAD command can be used again for the same user exit.

► **DELETE**

The DELETE option deletes the association between the MSG IDs and the user exit. After a user exit has been deleted, all messages for those MSG IDs are rejected by IMS Connect Extensions.

Note the following considerations:

- The user exit is logically, not physically, deleted from the IMS Connect system.
- The user exit executable is not physically deleted from the load library.
- If required, the ADD command can be used to reinstate a deleted user exit.

► **DISABLE**

The DISABLE option suspends processing for the user exit. The exit is not physically removed from the IMS Connect system.

IMS Connect Extensions rejects incoming messages for those MSG IDs supported by the user exit. Messages for XMIT or EXER are supported.

► **ENABLE**

The ENABLE option reenables a previously disabled user exit and reassociates MSG IDs with the user exit. If the user exit is not in a disabled state, the command is rejected.

User exit considerations

The following restrictions apply to all user exit commands:

- If the remote client is using IMS conversational processing, the user exits and the remote client have to be able to tolerate an environment in which some iterations of the conversation are processed by one version of the user exit and subsequent iterations by a different version.
- If the remote client is using persistent sessions, the user exits and the remote client have to be able to tolerate an environment where some messages for the session are processed by one version of the user exit and subsequent messages for the session are processed by a different version.

- ▶ If a user exit has been disabled or deleted and IMS Connect Extensions advanced features is turned off, the user exits might begin to receive messages.

Recommendation: If possible, RELOAD user exits during a period of low activity.

11.7 IMS Connect problem determination

This topic provides several examples of IMS Connect problem determination using the IMS Connect Extensions features. It shows how to use of the reports described in 11.2, “Event collection and reporting” on page 161 to analyze problems related with the following issues:

- ▶ NODELAYACK issues
- ▶ Incorrect message length
- ▶ Client fails to ACK message
- ▶ Timeout issues
- ▶ Duplicate clients

11.7.1 NODELAYACK issues

NODELAYACK is a TCP/IP parameter that you can set in the TCP/IP profile in the port statement or in the gateway statement.

It allows non-data transmissions from the host to flow without data. If NODELAYACK is used, the z/OS TCP/IP immediately sends an ACK to the remote server TCP/IP. The ACK is not appended to the data being sent from IMS Connect.

It works in the following way:

- ▶ If the client code sends one SEND followed by a READ to the host with a NODELAYACK setting, an ACK is sent separately.
- ▶ If the client code sends two or more SENDs followed by a READ to the host, the host TCP/IP sends an ACK immediately to the data received. This allows the next SEND of data from the client to flow.

The other option is to use DELAYACK to add a delay before sending an ACK to the remote server TCP/IP. The ACK is appended to the data being sent from IMS Connect, so you do not have to suffer a delay. It is useful to minimize non-data transmissions from the host.

Note: If your client application performs a single SEND followed by a READ, we recommend using DELAYACK. We recommend using NODELAYACK if your client application sends more than one SEND followed by a READ.

Delays receipt of data

The ACK delay value can produce delays in data receipt. For sync level confirm, the delay also impacts the receipt of the ACK/NAK sequence. Without IMS Connect Extensions to calculate these delays and evaluate their impact, you have to use internal trace entries.

Using the collected data from IMS Connect Extensions and IMS Performance Analyzer, you can obtain detailed, summary, and trace reports about read socket timings. In addition, you can use the IMS Problem Investigator or IMS Connect Extensions print utility to format individual read socket event records.

Delay example

In this example, we have two ports, 7003, which has defined the standard default ACK value, and 7005, which has defined the NODELAYACK parameter.

Figure 11-35 shows the IMS Performance Analyzer Transit Log report including transactions for both ports. The report shows a worse response time in transactions in port 7003 due to high Pre-OTMA times.

Display Filter View Print Options Help							
SDSF OUTPUT DISPLAY JOUK04A JOB14555 DSID 106 LINE 0				COLUMNS 02- 81			
COMMAND INPUT ==> _				SCROLL ==>			
***** TOP OF DATA *****							
IMS Performance Analyzer 3.3							
IMS Connect Transit Log - IMSGCO							
Log from 30Jun2005 12.53.30.42							
Start Time	Transact	Target	Port	Response	Input		
HH.MM.SS.THmiju	Code	DataStor	Number	Time	Pre-OTMA	READ Sock	READ
12.53.30.428084	PART	IMSG	7003	840.480	353.373	353.197	0.1
12.57.24.838479	PART	IMSG	7003	811.647	327.409	327.230	0.1
12.57.26.965121	PART	IMSG	7003	796.763	311.163	310.899	0.1
12.57.28.924203	PART	IMSG	7003	847.858	364.465	364.268	0.0
12.57.30.826973	PART	IMSG	7003	855.677	371.455	371.248	0.0
12.57.40.395359	PART	IMSG	7005	350.986	80.305	80.131	0.1
12.57.43.137376	PART	IMSG	7005	368.739	80.704	80.200	0.0
12.57.44.786505	PART	IMSG	7005	351.157	79.449	79.212	0.0
12.57.46.291199	PART	IMSG	7005	347.709	79.980	79.768	0.0
***** BOTTOM OF DATA *****							

Figure 11-35 NODELAYACK example: Transit Log report

With this report, you can easily identify the READ Sock time as the element that is producing the bad response time.

Figure 11-36 on page 207 shows the IMS Performance Analyzer Trace report, which provides the detailed events information for these transactions.

SDSF OUTPUT DISPLAY JOUK04A JOB14555 DSID 107 LINE 0				COLUMNS 02- 81
COMMAND INPUT ==> _				SCROLL ==>
***** TOP OF DATA *****				
				IMS Performance Analyzer 3.3
				IMS Connect Trace - IMSGCONN
				Trace from 30Jun2005 12.53.30.
Start/+Relative	Elapsed	ID	Description	Information
12.53.30.428084	*Start*	3C	Prepare Read Socket	Key=BD3D5A00B92C
+0.353112	353.112	49	Read Socket	Port=7003 Sock=3
+0.353197	0.085	49	Read Socket	Port=7003 Sock=3
+0.353212	0.014	3D	Message Exit called for READ	Exit=HWSSMPL0
+0.353268	0.056	3E	Message Exit return for READ	RC=00 RSN=00 IP=
				Commit Mode=1 Sy
+0.353373	0.104	41	Message sent to OTMA	TPipe=7003 MSG=1
+0.375946	22.572	42	Message received from OTMA	TPipe=7003 MSG=1
+0.376035	0.089	3D	Message Exit called for XMIT	Exit=HWSSMPL0
+0.376081	0.045	3E	Message Exit return for XMIT	RC=00 RSN=00
+0.376432	0.351	4A	Write Socket	Port=7003 Sock=3
+0.460720	84.287	49	Read Socket	Port=7003 Sock=3
+0.758761	298.040	49	Read Socket	Port=7003 Sock=3
+0.758831	0.070	49	Read Socket	Port=7003 Sock=3
+0.758849	0.017	3D	Message Exit called for READ	Exit=HWSSMPL0
+0.758896	0.046	3E	Message Exit return for READ	RC=00 RSN=00 IP=
				Commit Mode=1 Sy

Figure 11-36 DELAYACK example: Trace report

You can see that the time between the prepare read socket and the first read socket finishes is about 350 milliseconds. This clearly shows the impact of the ACK delay parameter. You can also see in the second read socket the impact of the ACK delay value in the receipt of the ACK/NAK (sync level confirm transactions).

Figure 11-37 shows the same report for port 7005, which has NOACKDELAY activated. In this case, the time of the first read socket is 80 milliseconds and the ACK/NAK read socket delay is 79 milliseconds.

SDSF OUTPUT DISPLAY JOUK04A JOB14555 DSID 107 LINE 191				COLUMNS 02- 81
COMMAND INPUT ==>				SCROLL ==>
12.57.43.137376	*Start*	3C	Prepare Read Socket	Key=BD3D5AF1B97B
+0.080141	80.141	49	Read Socket	Port=7005 Sock=3
+0.080200	0.059	49	Read Socket	Port=7005 Sock=3
+0.080212	0.012	3D	Message Exit called for READ	Exit=HWSSMPL0
+0.080264	0.051	3E	Message Exit return for READ	RC=00 RSN=00 IP=
				Commit Mode=1 Sy
+0.080704	0.439	41	Message sent to OTMA	TPipe=7005 MSG=T
+0.116022	35.317	42	Message received from OTMA	TPipe=7005 MSG=D
+0.116293	0.271	3D	Message Exit called for XMIT	Exit=HWSSMPL0
+0.116338	0.044	3E	Message Exit return for XMIT	RC=00 RSN=00
+0.116554	0.215	4A	Write Socket	Port=7005 Sock=3
+0.201014	84.460	49	Read Socket	Port=7005 Sock=3
+0.280215	79.201	49	Read Socket	Port=7005 Sock=3
+0.280279	0.063	49	Read Socket	Port=7005 Sock=3
+0.280293	0.014	3D	Message Exit called for READ	Exit=HWSSMPL0
+0.280334	0.041	3E	Message Exit return for READ	RC=00 RSN=00 IP=
				Commit Mode=1 Sy

Figure 11-37 NODELAYACK example: Trace report

IMS Connect Extensions enables you to detect ACK delays problems without making calculations based on IMS Connect traces.

11.7.2 Incorrect message length

IMS Connect does not process an input message until the entire the message is read. The initial LLLL value of the message controls the total length of the message. If the message contains an erroneous LLLL value, you can find problems in IMS Connect. For example, if the client specifies a larger LLLL value, the connection hangs because IMS Connect is expecting more data.

The determination of this problem is difficult, even with IMS Connect Extensions, because there is no information because IMS Connect does not start to process the message. Because the problem is related to a hanged session, the procedure must be to use the active session utility to identify the session and then go to IMS Problem Investigator or print utility to view the read socket events.

Incorrect message length example

In this example, a session hangs because it provides in the LLLL field a value larger than the length of the message. The length of the message is 143 bytes, but the LLLL field specifies 243.

The active session report in Figure 11-38 shows only two events, read prepare and read socket. The predicted session status indicates that IMS Connect is reading the client input. It also provides the session key.

```
IMS CONNECT EXTENSION ACTIVE SESSION REPORT VERSION 01:01:001

FIRST RECORD DATE=2005.06.28, TIME=01.49.41.321162

EOF ON INPUT DATASET
LIST OF ACTIVE SESSIONS

PORT #= 7003, SOCKET #= 2, KEY=BD3C0FB7C5934D04
EVENT HISTORY - KEY EVENTS PROCESSED FOR THIS MESSAGE
READ PREPARE SOCKET

EVENT RECORD TRACE - OLDEST TO NEWEST
EVENT=3C, PREPARE FOR SOCKET READ
EVENT=49, READ SOCKET
LAST TRACE EVENT DATE=2005.06.29, TIME=16.15.50.336812

PREDICTED SESSION STATUS BASED UPON LAST TRACE ENTRY
P014 - READING REMOTE CLIENT INPUT
```

Figure 11-38 Incorrect message length example: Active Session report

You can identify the events related to that session in IMS Problem Investigator using the session key obtained in the Active Session report. Figure 11-39 on page 209 shows the IMS Problem Investigator interface with the two socket events.

BROWSE	CEX.IMGCONN.D050629.T164106.ARCHIVE	Record 00000237	More: < >
Command ==>		Scroll ==>	CSR
Forwards / Backwards . .	HH.MM.SS.THHIJU	Time of Day . .	HH.MM.SS.THHIJU
Code Description	Date 2005-06-28 Tuesday	LSN	
/	-----		
— A048 Trigger Event		00000000000000ED	
OrgUOWID=CEX/BD3B3298ACF9CE41			
— A00B End ACCEPT Socket		00000000000000EE	
Port=7003			
— A00A Begin ACCEPT Socket		00000000000000EF	
Port=7003			
— A03C Prepare READ Socket		00000000000000F0	
Port=7003 OrgUOWID=CEX/BD3C0FB7C5934D04			
— A049 READ Socket		00000000000000F1	
Port=7003 OrgUOWID=CEX/BD3C0FB7C5934D04			
— A0A7 Internal Command Event		00000000000000F2	

Figure 11-39 Incorrect message length example: IMS Problem Investigator events

By selecting the events, you can see the details. Figure 11-40 shows the formatted records for the read prepare event.

BROWSE	CEX.IMGCONN.D050629.T164106.ARCHIVE	Record 00000240	Line 00000000
Command ==>		Scroll ==>	PAGE
Form ==>	+ Use Form in Filter	Format ==>	STD
***** Top of data *****			
+0004	Code... A03C Prepare READ Socket		
+003C	STCK... BD3C0FB7C69D7A40	LSN... 00000000000000F0	
	Date... 2005-06-29 Wednesday	Time... 12.15.50.233559	
+0000	CERE_3C_LL.....	004C	
+0002	CERE_3C_ZZ.....	0000	
+0004	CERE_3C_RECID.....	A0 CERE_3C_EVTID.....	3C
+0006	CERE_3C_PFXLL.....	0014	
+0008	CERE_3C_EFLAG.....	00 CERE_3C_VER#.....	01
+000A	CERE_3C_TASKID.....	ID of task recording event	
+000A	CERE_3C_COL#.....	01 CERE_3C_TKS#.....	05
+000C	CERE_3C_EVKEY.....	BD3C0FB7C5934D04	
+0014	CERE_3C_TLL.....	CEX TCPIP Block	
+0014	CERE_3C_TLL.....	0028	
+0016	CERE_3C_BLKID.....	+1	
+0018	CERE_3C_VVRR.....	0202	
+001A	CERE_3C_APAR_SEQ...	0001	
+001C	CERE_3C_PORT_#.....	+7003	
+001E	CERE_3C SOCK_#.....	+2	
+0022	CERE_3C_SKFLAG.....	40 CERE_3C_PTFLAG.....	00
+0024	CERE_3C_REQ_LEN....	+32	
+0028	CERE_3C_ACT_LEN....	+32	
+002C	CERE_3C_REQ_DATA...	00000000	
+0030	CERE_3C_RC_CD.....	00000020	
+0034	CERE_3C_RMT_RSN...	00000000	

Figure 11-40 Incorrect message length example: Prepare READ Socket formatted records

Marked in the figure, you see the requested read length followed by the actual read length. Before reading all the data of an input message, IMS Connect always reads the first 32 bytes. These bytes are fixed by the architecture and cannot be modified by the user.

Reading these bytes, IMS Connect obtains the LLLL value and sends another read socket to read the rest of the message. Figure 11-41 shows the event details for the second read socket.

```

BROWSE      CEX.IMGCONN.D050629.T164106.ARCHIVE  Record 00000241 Line 00000000
Command ==> -                                     Scroll ==> PAGE
Form      ==> +       Use Form in Filter          Format ==> STD
***** Top of data *****
+0004 Code... A049 READ Socket
+003C STCK... BD3C0FB7C6A51006      LSN... 00000000000000F1
      Date... 2005-06-29 Wednesday   Time... 12.15.50.233681

+0000 CERE_49_LL..... 004C
+0002 CERE_49_ZZ..... 0000
+0004 CERE_49_RECID..... A0 CERE_49_EVTID..... 49
+0006 CERE_49_PFXLL..... 0014
+0008 CERE_49_EFLAG..... 00 CERE_49_VER#..... 01
+000A CERE_49_TASKID..... ID of task recording event
+000A CERE_49_COL#..... 01 CERE_49_TKS#..... 05
+000C CERE_49_EVKEY..... BD3C0FB7C5934D04

+0014 CERE_49_TLL..... CEX TCP/IP Block
+0014 CERE_49_TLL..... 0028
+0016 CERE_49_BLKID..... +1
+0018 CERE_49_VVRR..... 0202
+001A CERE_49_APAR_SEQ... 0001
+001C CERE_49_PORT_#..... +7003
+001E CERE_49 SOCK_#..... +2
+0022 CERE_49_SKFLAG..... 40 CERE_49_PTFLAG..... 00
+0024 CERE_49_REQ_LEN.... +211
+0028 CERE_49_ACT_LEN.... +111
+002C CERE_49_REQ_DATA... 00000000
+0030 CERE_49_RC_CD..... 0000006F
+0034 CERE_49_RMT_RSN... 00000000

```

Figure 11-41 Incorrect message length example: Second READ Socket formatted records

IMS Connect calculates the data requested in this second read socket as the difference between the input LLLL value and the data already read (32 bytes). As you can see in the IMS Problem Investigator formatted records, IMS Connect requested 211 bytes, but the data read is 111 bytes, which is the real length of the message without the first 32 bytes. As a consequence, IMS Connect waits for more input and the session becomes hanged.

11.7.3 Client fails to ACK message

When a client fails to ACK/NAK a message, the symptoms can vary depending on the client actions. IMS Connect Extensions events help you to identify the problem by following the event flow. By activating the traces, it also provides trace event formatted records to show the output write data, including *CSMOKY* and the RSM returned if the client does not send the required ACK.

Client fails to ACK message example

In this example, we have a transaction with commit mode 1, sync level confirm, and transaction socket type. Instead of sending an acknowledgement of the message, the client closes the socket.

Figure 11-42 on page 211 shows the sequence of events obtained through the IMS Problem Investigator ISPF interface. It shows the output message sending and the client response. You can also see a session error event. IMS Connect issues this kind of event every time that an unexpected circumstance occurs.

BROWSE	CEX.IMGCONN.D050629.T174920.ARCHIVE	Record 00000038	More: < >
Command ==>		Scroll ==>	CSR
Forwards / Backwards . .	HH.MM.SS.THMIJU	Time of Day . .	HH.MM.SS.THMIJU
Code Description	Date 2005-06-29	Wednesday	LSN

— A0A3	Event Collection OTMA Trace		0000000000000443
	OrgUOWID=CEX/BD3C248E2D055461		

— A0A3	Event Collection OTMA Trace		0000000000000444
	OrgUOWID=CEX/BD3C248E2D055461		

— A03D	Message Exit called for READ, XMIT, EXER		0000000000000445
	OrgUOWID=CEX/BD3C248E2D055461		

— A0A6	Event Recording EXIT Output Message Trace		0000000000000446
	OrgUOWID=CEX/BD3C248E2D055461		

— A03E	Message Exit returned from READ, XMIT, EXER		0000000000000447
	OrgUOWID=CEX/BD3C248E2D055461		

— A04A	WRITE Socket		0000000000000448
	Port=7003 OrgUOWID=CEX/BD3C248E2D055461		

— A049	READ Socket		0000000000000449
	Port=7003 OrgUOWID=CEX/BD3C248E2D055461		

— A047	Session Error		000000000000044A
	OrgUOWID=CEX/BD3C248E2D055461		

— A00C	Begin CLOSE Socket		000000000000044B

Figure 11-42 ACK failed: Event flow

The message exit receives the message from OTMA and constructs the output to send to the client. Therefore, in the event trace for the exit, you find relevant information. Figure 11-43 shows the formatted records for the event recording exit output message trace.

BROWSE	CEX.IMGCONN.D050629.T174920.ARCHIVE	Record 00000041	Line 00000037
Command ==>		Scroll ==>	CSR
Form	+	Use Form in Filter	Format ==> STD
+00B6	CERE_A6_MSG.....	Output Message Section	
+00B6	CERE_A6_MSG_LL.....	0044	
+00B6	CERE_A6_MSG_DTA....		
+0000	00440300	20202020 20202020 20204D61	*.....(/*
+0010	6B652044	6570742E 2E2E2E2E 2E202020	*,.....*
+0020	20202020	2031322D 30303B20 506C616E	*.....&%/>*
+0030	20526576	204E756D 2E2E2E20 20202020	*.....+.....*
+0040	20202020		*....*
+00FA	CERE_A6_MSG.....	Output Message Section	
+00FA	CERE_A6_MSG_LL.....	0044	
+00FA	CERE_A6_MSG_DTA....		
+0000	00440300	20202020 20202020 20204D61	*.....(/*
+0010	6B652054	696D652E 2E2E2E2E 2E202020	*,.....*
+0020	20202020	20202020 36333B20 436F6D6D	*.....?___*
+0030	20436F64	652E2E2E 2E2E2E20 20202020	*..?.....*
+0040	20203134		*....*
+013E	CERE_A6_MSG.....	Output Message Section	
+013E	CERE_A6_MSG_LL.....	000C	
+013E	CERE_A6_MSG_DTA....	000C20002A43534D4F4B592A	
***** End of data *****			

Figure 11-43 ACK failed: Trace event

Marked in the figure, you can see the CSMOKY translated to ASCII and the flag that indicates that an ACK/NAK is requested for this IMS Connect output.

Important: Remember that you can only obtain this event if the tracing level is activated. In this case, we use tracing level 2 (see Figure 11-8 on page 170).

After this event, you can see a write socket event indicating that the IMS Connect output is sent to the client followed by a read socket issued to get the requested ACK. Figure 11-44 shows the formatted records for the read socket.

```

BROWSE      CEX.IMSGCONN.D050629.T174920.ARCHIVE  Record 00000044 Line 00000000
Command ==> _____ Scroll ==> CSR
Form      ==>      +      Use Form in Filter      Format ==> STD
***** Top of data *****
+0004 Code... A049 READ Socket
+003C STCK... BD3C248E4AD05C66      LSN.... 0000000000000449
      Date... 2005-06-29 Wednesday Time... 13.49.03.879429

+0000 CERE_49_LL..... 004C
+0002 CERE_49_ZZ..... 0000
+0004 CERE_49_RECID..... A0 CERE_49_EVTID..... 49
+0006 CERE_49_PFXLL..... 0014
+0008 CERE_49_EFLAG..... 00 CERE_49_VER#..... 01
+000A CERE_49_TASKID..... ID of task recording event
+000A CERE_49_COL#..... 01 CERE_49_TKS#..... 05
+000C CERE_49_EVKEY..... BD3C248E2D055461

+0014 CERE_49_TLL..... CEX TCP/IP Block
+0014 CERE_49_TLL..... 0028
+0016 CERE_49_BLKID..... +1
+0018 CERE_49_VVRR..... 0202
+001A CERE_49_APAR_SEQ... 0001
+001C CERE_49_PORT_#..... +7003
+001E CERE_49 SOCK_#..... +3
+0022 CERE_49_SKFLAG.... 40 CERE_49_PTFLAG.... 80
+0024 CERE_49 REQ_LEN.... +32
+0028 CERE_49 ACT_LEN.... +0
+002C CERE_49_REQ_DATA... 00000000

```

Figure 11-44 ACK failed: READ Socket event

This event shows that IMS Connect does not receive an ACK/NAK. It requests 32 bytes of data to receive, but no data is received, because the client closes the connection without sending the ACK.

The last event is the session error that IMS Connect issues. Figure 11-45 on page 213 shows its details. Note that IMS Connect attempts to identify the type of event that caused the error.

```

BROWSE      CEX.IMGCONN.D050629.T174920.ARCHIVE  Record 00000045 Line 00000002
Command ==> _____ Scroll ==> CSR
Form      ==> - + Use Form in Filter Format ==> STD
+00B0 STCK... BD3C248E4B50ED06 LSN.... 000000000000044A
      Date... 2005-06-29 Wednesday Time... 13.49.03.881486

+0000 CERE_47_LL..... 00C0
+0002 CERE_47_ZZ..... 0000
+0004 CERE_47_RECID..... A0 CERE_47_EVTID..... 47
+0006 CERE_47_PFXLL..... 0014
+0008 CERE_47_EFLAG..... 00 CERE_47_VER#..... 01
+000A CERE_47_TASKID..... ID of task recording event
+000C CERE_47_COL#..... 01 CERE_47_TKS#..... 05
+000C CERE_47_EVKEY..... BD3C248E2D055461
+0014 CERE_47_VAR_LL..... 009C
+0016 CERE_47_VAR_APAR... 0001
+0018 CERE_47_VAR_FLAG3..... 80
+001A CERE_47_VAR_MSG....
+0000 00520000 C8E6E2D7 F1F4F1F5 C540E3C3 *...HWSP1415E TC*
+0010 D761C9D7 40E2D6C3 D2C5E340 C6E4D5C3 *P/IP SOCKET FUNC*
+0020 E3C9D6D5 40C3C1D3 D340C6C1 C9D3C5C4 *TION CALL FAILED*
+0030 5E40C67E D9C5C1C4 40404040 6B40D97E *; F=READ , R=*
+0040 60F16B40 C57EF1F1 F2F16B40 D47EE2C4 *-1, E=1121, M=SD*
+0050 D9C30000 00000000 00000000 00000000 *RC.....*
+0060 00000000 00000000 00000000 00000000 *.....*
+0070 00000000 00000000 00000000 00000000 *.....*
+0080 00000000 0000 *.....*
+00A0 CERE_47_VAR_SESRN..... 'READFAIL'
+00AB CERE_47_VAR_TOKEN..... 0000000000000000

```

Figure 11-45 ACK failed: Session error event

11.7.4 Timeout issues

IMS Connect Extensions helps you handle timeout issues, providing reports about all client timeouts events. The events include the original timer values. IMS Connect Extensions collects a session error event for any message issued by IMS Connect.

Timeout example

Figure 11-46 on page 214 shows the events that identify a timeout. The message is sent to IMS and the response is not received within the timeout value.

BROWSE	CEX.IMGCONN.D050629.T184907.ARCHIVE	Record 00000007 More: < >
Command ==>		Scroll ==> CSR
Forwards / Backwards . .	HH.MM.SS.THMIJU	Time of Day . . HH.MM.SS.THMIJU
Code Description	Date 2005-06-29 Wednesday	Time (LOCAL)

— A049	READ Socket Port=7003 OrgUOWID=CEX/BD3C31FF631D8A64	14.49.12.131851

— A03D	Message Exit called for READ, XMIT, EXER OrgUOWID=CEX/BD3C31FF631D8A64	14.49.12.131910

— A03E	Message Exit returned from READ, XMIT, EXER TranCode=PART OrgUOWID=CEX/BD3C31FF631D8A64 IMSID=IMSG	14.49.12.131956

— A041	Message sent to OTMA OrgUOWID=CEX/BD3C31FF631D8A64	14.49.12.132047

— A045	OTMA Time-out OrgUOWID=CEX/BD3C31FF631D8A64	14.49.12.182927

— A03D	Message Exit called for READ, XMIT, EXER OrgUOWID=CEX/BD3C31FF631D8A64	14.49.12.182959

— A03E	Message Exit returned from READ, XMIT, EXER OrgUOWID=CEX/BD3C31FF631D8A64	14.49.12.182985

— A04A	WRITE Socket Port=7003 OrgUOWID=CEX/BD3C31FF631D8A64	14.49.12.183128

Figure 11-46 Timeouts: Event flow

You can easily identify the timeout in the event flow. You also can see in the formatted event records the timeout value that expired, as shown in Figure 11-47.

BROWSE	CEX.IMGCONN.D050629.T184907.ARCHIVE	Record 00000011 Line 00000000
Command ==>		Scroll ==> CSR
Form ==>	+ Use Form in Filter	Format ==> STD
***** Top of data *****		
+0004	Code... A045 OTMA Time-out	
+001C	STCK... BD3C31FF7048F645	LSN... 0000000000002E21
	Date... 2005-06-29 Wednesday	Time... 14.49.12.182927
+0000	CERE_45_LL.....	002C
+0002	CERE_45_ZZ.....	0000
+0004	CERE_45_RECID.....	A0 CERE_45_EVTID..... 45
+0006	CERE_45_PFXLL.....	0014
+0008	CERE_45_EFLAG.....	00 CERE_45_VER#..... 01
+000A	CERE_45_TASKID.....	ID of task recording event
+000A	CERE_45_COL#.....	01 CERE_45_TKS#..... 05
+000C	CERE_45_EVKEY.....	BD3C31FF631D8A64
+0014	CERE_45_VAR_LL.....	0008
+0016	CERE_45_VAR_APAR...	0001
+001B	CERE_45_VAR_TOV...	05
***** End of data *****		

Figure 11-47 Timeout value

In this case, the timeout values is too short by only a few milliseconds (just to illustrate).

11.7.5 Duplicate clients

IMS Connect does not know the client name until the message returns from the message exit. If the same client is already active for the port, it sends a duplicate client error message.

This problem is often caused by an incorrect client timeout recovery. A common cause is a client timeout shorter than the IMS Connect timeout value. When the client receives a timeout and resends the message through another socket, the client can be active in IMS Connect because its timeout is not expired.

The IMS Connect Extensions events show the timing and sequence of incorrect recovery and client name usage overlap.

Duplicate client examples

Figure 11-48 shows the flow for a duplicate client error. After receiving the message from the read message exit, IMS Connect issues a session error event.

BROWSE CEX.IMGCONN.D050629.T203629.ARCHIVE		Record 00000110 More: < >
Command ==>		Scroll ==> CSR
Forwards / Backwards . . HH.MM.SS.THMIJU		Time of Day . . HH.MM.SS.THMIJU
Code Description		Date 2005-06-29 Wednesday Time (LOCAL)
-----		-----
A049	READ Socket Port=7003 OrgUOWID=CEX/BD3C4A1430513346	16.36.56.394668
-----		-----
A03D	Message Exit called for READ, XMIT, EXER OrgUOWID=CEX/BD3C4A1430513346	16.36.56.394681
-----		-----
A03E	Message Exit returned from READ, XMIT, EXER TranCode=PART OrgUOWID=CEX/BD3C4A1430513346 IMSID=IMSG	16.36.56.394727
-----		-----
A047	Session Error OrgUOWID=CEX/BD3C4A1430513346	16.36.56.396564
-----		-----
A03D	Message Exit called for READ, XMIT, EXER OrgUOWID=CEX/BD3C4A1430513346	16.36.56.396588
-----		-----
A03E	Message Exit returned from READ, XMIT, EXER OrgUOWID=CEX/BD3C4A1430513346	16.36.56.396610
-----		-----
A04A	WRITE Socket Port=7003 OrgUOWID=CEX/BD3C4A1430513346	16.36.56.396964
-----		-----
A00C	Begin CLOSE Socket Port=7003 OrgUOWID=CEX/BD3C4A1430513346	16.36.56.396994
-----		-----
A00D	End CLOSE Socket	16.36.56.479746

Figure 11-48 Duplicate client: Event flow

Select the session error event and you can see that is caused by a duplicate client error. See Figure 11-49 on page 216.

```

BROWSE      CEX.IMSGCONN.D050629.T203629.ARCHIVE  Record 00000113 Line 00000000
Command ==> _____ Scroll ==> CSR
Form ==> _____ + Use Form in Filter Format ==> STD
***** Top of data *****
+0004 Code... A047 Session Error
+00B0 STCK... BD3C4A1431714C84 LSN.... 0000000000002E99
      Date... 2005-06-29 Wednesday Time... 16.36.56.396564

+0000 CERE_47_LL..... 00C0
+0002 CERE_47_ZZ..... 0000
+0004 CERE_47_RECID..... A0 CERE_47_EVTID..... 47
+0006 CERE_47_PFXLL..... 0014
+0008 CERE_47_EFLAG..... 00 CERE_47_VER#..... 01
+000A CERE_47_TASKID..... ID of task recording event
+000A CERE_47_COL#..... 01 CERE_47_TKS#..... 05
+000C CERE_47_EVKEY..... BD3C4A1430513346
+0014 CERE_47_VAR_LL..... 009C
+0016 CERE_47_VAR_APAR... 0001
+0018 CERE_47_VAR_FLAG3..... 80
+001A CERE_47_VAR_MSG....
+0000 E6E3D6F6 40404040 00000000 00000000 *WT06 .....*
+0010 C8E6E2E2 F0F7F4F2 F7F0F0F3 40404040 *HWSS07427003 *
+0020 C3D3C9C5 D5E3F0F1 C9D4E2C7 40404040 *CLIENT01IMSG *
+0030 00000008 00000000 C4E4D7C5 C3D3D5E3 *.....DUPECLNT*
+0040 E2D9C5F4 40404040 00000000 00000000 *SRE4 .....*
+0050 00000000 00000000 00000000 00000000 *.....*
+0060 00000000 00000000 00000000 00000000 *.....*
+0070 00000000 00000000 00000000 00000000 *.....*
+0080 00000000 0000 *.....*
+00A0 CERE_47_VAR_SESRN..... C4E4D7C500000000

```

Figure 11-49 Duplicate client: Session error event

The message exit returned event in Figure 11-48 on page 215 provides the client name. By searching in the journal for the previous events related with that client name, you determine the problem. If the problem is related to a timeout, you will probably find the same client sending a previous message sent to OTMA without receiving any response.

11.8 Highlights of IMS Connect Extensions Version 1 Release 2

Enhancements in IMS Connect Extensions Version 1.2 include:

- ▶ Active session display, providing detailed information about the state and wait times of active sessions
- ▶ Programming interface for user applications, allowing external applications to access IMS Connect data
- ▶ Alternate routing when a primary datastore is unavailable
- ▶ Journal and journal print enhancements, including the ability to change many journal options dynamically
- ▶ New facilities to provide IMS Connect clients with password change services and identification information
- ▶ Improved resource tracing, providing selective tracing using criteria such as port, transaction, or client
- ▶ Support for Internet Protocol Version 6 (IPv6)
- ▶ Support for a user version of HWSTECL0
- ▶ Definitions migration utility to import and export definitions data sets from one IMS Connect Extensions system to another

IMS Connect Extensions V1.2 supports all the currently supported IMS versions. If either IMS Version 7 or IMS Version 8 is used, IMS Connect Extensions for z/OS requires IMS Connect for z/OS V2.2. If IMS Version 9 is used, IMS Connect for z/OS, V2.2 is not needed because IMS Version 9 provides an integrated IMS Connect function that replaces IMS Connect V2.2.

11.8.1 Status Monitor: Active sessions

Active session display is a real-time exception report on the state of all active socket sessions processed by IMS Connect. It is an extension of the batch utility.

The utility lets you:

- ▶ View a summary of all in-flight transactions (see Figure 11-50).
- ▶ Select individual session states and get more information about the transactions currently being processed.
- ▶ Identify socket transactions with long wait times.
- ▶ Use filters to select sessions based on conditions.
- ▶ Create forms to completely customize the display.

File Menu Form Filter Help						
EDIT			Active Sessions		Row 1 of 4	
Command ==>					Scroll ==> CSR	
Session Wait Time . . . (Seconds) / Include persistent sockets						
Port	Socket	Event	Session Start Time	Session Wait Time		
— 3702	2	Message Exit entered for	15.24.12.859832	00.09.38.618073		
— 3702	2	Message sent to OTMA	15.24.12.859832	00.09.38.609528		
— 3702	2	Message Exit entered for	15.24.12.859832	00.09.38.592438		
— 3702	2	Trigger Event	15.24.12.859832	00.09.38.583893		

Figure 11-50 View a real-time summary of active socket sessions

In addition to the summary view, you can select individual session states and get more information about the transactions currently being processed (see Figure 11-51 on page 218).

```

File Menu Help
-----
EDIT                               Active Sessions - Session Status           Row 1 of 35
Command ==> _____ Scroll ==> CSR

IMS Connect:
Port . . . . . : 3702                Socket . . . . . : 2
Event Key . . . . . : BDE0B88145DB8C41
Session Start Time . . : 2005-11-07 15.24.12.859832
Last Trace Time . . . : 2005-11-07 15.24.12.894012
Session Wait Time . . : 00.09.38.592438

READ exit:
Return Code . . . . . : 0              Reason Code . . . . : 00
Transaction . . . . . : PART           User ID . . . . . : CMW
Override LTERM name . :                Exit name . . . . . : HWSSMPL0
Original Datastore . . : IMD3          Target Datastore . . : IMD3
Client ID . . . . . : TRNBS001
Family . . . . . : 2                  Port . . . . . : 1447
IP address . . . . . : 172.17.69.87

Predicted session status:
P004 - Processing in message exit=HWSSMPL0

Event Record Trace:
Event=3D, Message Exit entered for READ, XMIT or EXER
Event=42, Message received from OTMA
Event=41, Message sent to OTMA
Event=3E, Message Exit returned from READ, XMIT or EXER
Event=3D, Message Exit entered for READ, XMIT or EXER
Event=49, READ socket
Event=3C, Prepare READ socket

```

Figure 11-51 View details of socket sessions

11.8.2 Programming interface for user applications

The programming interface for user applications is a High Level Assembler (HLASM) API that:

- ▶ Provides IMS Connect event data
- ▶ Lists IMS Connect systems
- ▶ Lists tasks in an IMS Connect region
- ▶ Lists active sessions

11.8.3 Primary datastore routing

The primary datastore routing feature enhances transaction routing by enabling you to use a virtual datastore as the message-determined target datastore. IMS Connect Extensions translates the virtual datastore to a primary datastore based on the IMS Connect system that is processing the message.

If the primary is not available, you can automatically use dynamic routing and workload balancing to route the message.

11.8.4 Journal and journal print enhancements

The journal print utility includes the following enhancements:

- ▶ Improved event record selection:
 - By event ID.
 - By event key.
 - By event data contained in event type 62 (return from read exit).

- Find the event key based on client ID, transaction code, LTERM, and more.
- ▶ Reduce the amount of print output by using start-after and stop-after values

Archive Journaling is now more flexible and easier to use, enabling you to select different levels of archiving control:

- ▶ Custom
Enter a completely customized archive job.
- ▶ Partial
Use substitution variables to generate portions of the archive statement dynamically.
- ▶ Automated
Archive options are generated automatically.

11.8.5 Client services exit

The client services exit is an IMS Connect message exit that provides clients with:

- ▶ IMS Connect password change facilities
- ▶ Who-am-I services to obtain information such as:
 - Client name
 - IP address
 - Port number
 - Event key

The client application only needs to send a message with CEXSVC01 in the message header. The message is intercepted by the exit; it is not forwarded to IMS.

11.8.6 Enhanced tracing

Tracing is enhanced by letting you selectively trace messages based on:

- ▶ Port number or all ports
- ▶ Client name
- ▶ Transaction name
- ▶ Exit name
- ▶ User ID
- ▶ LTERM
- ▶ IP address

Archived

IMS Connector for Java

Although access to IMS transactions using the IMS Connect software can be used by non-Java clients, it is most commonly used by Java clients on a local host or a distributed platform. In this chapter, we outline the different methods that can be used to build a Java IMS client. Some of these are based on an architectural design and resources provided by IMS Connector for Java, others are based on the socket interface without assistance from any architected resource manager code.

First, we describe the J2EE Connector architecture (JCA) as it has been defined within Java 2 Platform, Enterprise Edition (J2EE). Then, we explain how a Java client uses JCA by coding to the Common Client Interface (CCI) provided by IMS Connector for Java. We also explain which tools and wizards are available to quickly build applications aimed at calling existing IMS transactions, for which Cobol, C, or MFS descriptions of the transaction input and output messages are available.

The Java clients can be stand-alone programs or can be modules scheduled in WebSphere Application Server or similar J2EE application server, or even in stored procedures.

This chapter also addresses other important areas such as two-phase commit (2PC), security, interaction verbs, message rerouting, and coding guidelines.

12.1 J2EE Connector architecture (JCA)

The current JCA specification is v1.5. It is part of the J2EE 1.4 specification. Although it has been implemented in the latest *resource adapter* for IMS Connector for Java, the new extensions are not necessarily used by the connector implementations created by the tool wizards. A good start to understanding JCA is provided by the article “Introduction to the J2EE Connector Architecture,” available at:

<http://www.ibm.com/developerworks/java/edu/j-dw-javajca-i.html>

In JCA, we distinguish the following parts:

- ▶ System contracts
- ▶ Common Client Interface (CCI)
- ▶ Resource adapter module

12.1.1 System contracts

We expect the JCA to fulfill some requirements. Those requirements are not unique to IMS but common to all enterprise information system (EIS) connections. IMS is the case that we outline in this book. In Figure 12-1, we show the relationship between the CCI components and the elements of the system contracts.

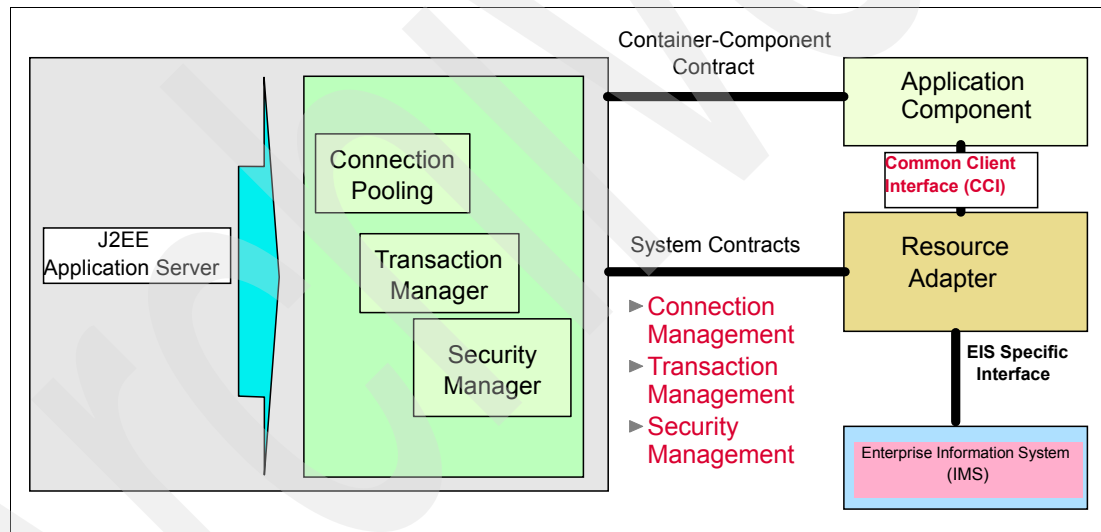


Figure 12-1 CCI system contracts

The system contracts can be considered as a group of agreements that have to be fulfilled by the resource adapter and application server implementations and accessible for a client through the CCI.

We distinguish the following elements within the contracts, shown in Figure 12-1:

- ▶ The *connection management* contract enables physical connections to the EIS and provides a mechanism for the application server to pool those connections. Connections can be local through internal memory mechanisms (program call) or remote (TCP/IP) connections. The pooling implementation will be different and can change over time, but the change should be invisible for the client API.

- ▶ The *transaction management* contract supports access to an EIS in a transactional context. Transactions can be managed by an external application server, for example, WebSphere Application Server, providing transactions that incorporate other resources besides the EIS (global transactions), or they can be internal to the EIS resource manager (local transactions), in which case, no external Transaction Manager is required. A global transaction is under the control of RRS, and it can, in addition to IMS transactions, also include access to DB2, WebSphere MQ, or other resources directly from the program that invoked the IMS transaction. IMS is a participant in the global transaction case. For a local transaction, IMS is the coordinator, and it controls the synchronization for all resources accessed from the IMS processing program in this case.
- ▶ The *security contract* supports secure access to the EIS. It has the responsibility to pass a credential to the EIS representing a user name and password or an already authenticated user so that the EIS can verify that the users are who they claim to be and are allowed to access the resources of an EIS. The user ID that is passed is determined based on the security configuration used for the application server.

How system contracts are implemented on each side (application server and resource adapter) is not specified by JCA; they can be implemented as each vendor sees appropriate.

12.1.2 Common Client Interface

The second element of JCA is the *client API*. This API as defined in JCA is the *Common Client Interface (CCI)*. The CCI is a standard client API for use by application components. It is designed to provide a base-level API for EIS access on which higher-level functionality will be built by EAI and tools vendors. The CCI is divided into five parts:

- ▶ Connection-related interfaces
- ▶ Interaction-related interfaces
- ▶ Data representation-related interfaces
- ▶ Metadata-related interfaces
- ▶ Exceptions and warnings

The client API implementation for a particular EIS is distributed as a set of Java classes. This set contains the implementation of the interfaces specified by the JCA, mostly contained in the JAR file `connector.jar`. For IMS, this set of classes is distributed in the JAR file `imsico.jar`.

12.1.3 Resource adapter module

The *resource adapter* module contains all the elements necessary to provide EIS connectivity to J2EE applications. All of the resource adapter module's files are packaged into a single resource adapter archive (RAR) file. Specifically, the RAR file includes the following components:

- ▶ The Java classes and interfaces that implement the resource adapter
- ▶ Any utility Java classes required by the resource adapter
- ▶ Any EIS-specific, platform-dependent native libraries
- ▶ A deployment descriptor

The deployment descriptor is a meta-file (XML) that describes the resource adapter and provides information that is used in its deployment. It is named `ra.xml` and is located in the `META-INF` folder of the RAR file. Application servers, such as WebSphere Application Server, make use of the deployment descriptor supplied with a resource adapter to configure it to a specific operational environment.

All Java classes and interfaces are packaged in JAR files, which are then contained by the RAR file. The native files, Java Native Interface (JNI) files, are also packaged in the RAR file.

JNI files are, in general, DLL files, in Microsoft Windows they have the extension *dll*, while in a UNIX® environment (z/OS included) the extension is *so* (shared object). JNI files that are not written in Java are often also called drivers and are required if the access to the resource cannot be fulfilled in Java. For example, the local option provided by IMS Connector for Java and IMS Connect uses JNI, as does JDBC™ type 2 access and DLI database access. The TCP/IP protocol is entirely supported in Java and does not require JNI.

12.2 JCA infrastructure and API

Before any message can be sent to or received from an EIS system, a connection has to be established. The duration of this connection and the way it is obtained are essential for the performance and the quality of the interchange with the EIS.

After a connection has been obtained, an interaction with the EIS can be performed. The same connection can be used by several consecutive interactions. Basically, an interaction is a communication with the EIS that follows one of the possible patterns: send, send-receive, receive, and so on. For example, in the case of the IMS resource adapter, a typical interaction is a send-receive communication that runs a transaction in IMS.

In the past, the EIS was the coordinator of the transactional context, because all resource changes were done in its transactional environment. This is what we now call a *local transaction*. The transactional context can also be coordinated on a higher level by either an XA coordinator or Resource Recovery Services (RRS) on z/OS, in which case, several EIS interactions and direct resource accesses (DB2, DLI, WebSphere MQ) can be part of the same *global* transaction.

In the following section, we explain more in detail how this JCA infrastructure has been implemented in the *resource adapter* and how it can be used by the Common Client Interface. We also outline the enhancements in JCA v1.5 as compared to JCA v1.0.

12.2.1 Connection management

An application component accesses the EIS through `ConnectionFactory` and `Connection` interfaces, which are provided by the resource adapter. These CCI interfaces (`javax.resource.cci.ConnectionFactory` and `javax.resource.cci.Connection`) are specific to the resource adapter. In the case of the IMS resource adapter, they are implemented by the classes `com.ibm.connector2.ims.ico.IMSConnectionFactory` and `com.ibm.connector2.ims.ico.IMSConnection`. The classes that implement these interfaces are provided in JAR files that are packaged in the RAR file.

In JCA v1.0 and JCA v1.5, an application can obtain a `ConnectionFactory` in two ways. Figure 12-2 on page 225 shows this. The common point is the availability of the factory from which a `Connection` object can be created. An application uses a `Connection` object as a handle to the underlying physical connection to the EIS. Be aware that in both cases, a `ConnectionFactory` object is instantiated. However, the `ConnectionFactory` object obtained by a lookup in a managed environment, such as WebSphere Application Server, provides better connection management through pooling and reuse of connections. The other `ConnectionFactory` object, obtained in the non-managed way, is associated with a default connection manager. The default connection manager does not provide pooling and reuse of connections.

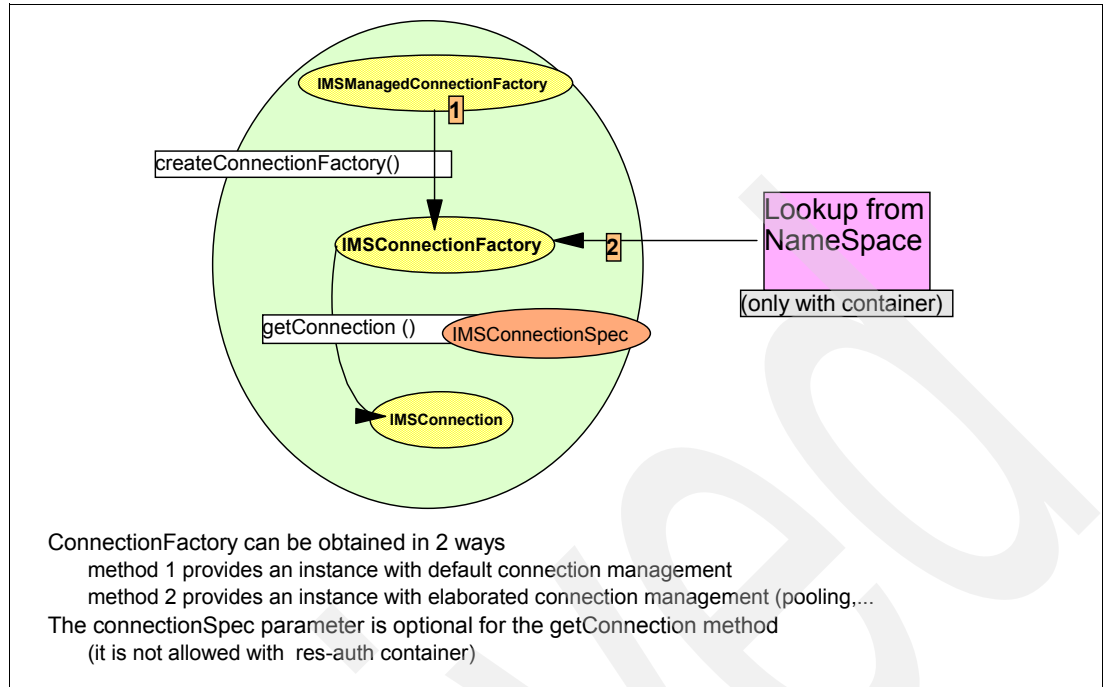


Figure 12-2 Instantiating the IMS ConnectionFactory

An application obtains an IMSConnectionFactory in the following ways:

- IMSConnectionFactory by lookup

The IMSConnectionFactory object is obtained through a lookup through the Java Naming and Directory Interface™ (JNDI). Because the application only needs to know the JNDI name of the connection factory, it does not need to have any knowledge of the actual configuration of the connection factory. On different platforms and environments, we can use the same JNDI name, with different properties, which makes a JCA program highly independent from the platform on which it runs. This supposes that the associated factory information has been registered in a namespace. A connection factory obtained in this way is said to produce managed connections.

Note: In WebSphere, a JCA connection factory is defined and configured using the administration functions of the application server (administrative console or wsadmin scripting).

- IMSConnectionFactory from ManagedConnectionFactory

The connection factory can also be obtained using an IMSManagedConnectionFactory object, which is instantiated at run time, and then populated with the required properties. An IMSConnectionFactory object can then be instantiated through the createConnectionFactory method of the IMSManagedConnectionFactory object. A connection factory obtained in this way is said to handle non-managed connections.

After the IMSConnectionFactory object is instantiated, an IMSConnection object is obtained from it using the getConnection() method. IMS Connector for Java provides two variants of the getConnection method; one with and one without a connectionSpec parameter. An IMSConnectionSpec object is passed on the getConnection method if connection-related information needs to be provided, or if component-managed EIS sign-on is being used by the application.

It is important to note that an `IMSCConnection` object is an application-level handle to an underlying physical connection to the EIS (IMS). The physical connection to the EIS is represented by a `ManagedConnection` object, which is internal to the resource adapter. By using an application-level handle to an underlying physical connection, an application is isolated from the creation and maintenance of physical connections. Functions, such as creation, pooling, and reuse of physical connections to the EIS, are the combined responsibility of the resource adapter and application server.

JCA v1.0 implements the previously discussed connection management and provides an application with the following two models for using the connection handle:

- **Get-use-close**

In the get-use-close model, an application always obtains a new connection when it needs one, uses it, and then closes it again. The get-use-close model might sound inefficient, but the connection pooling that the application server implements should make the get operation inexpensive. And because the application holds on to the connection for only as long as it is needed, different instances or parts of the application can reuse the connection, thereby reducing the total resource usage. Each time the application invokes the `getConnection` method to create a new connection handle, the connection manager reuses a managed connection from the pool and only creates a new managed connection when none are available. When the connection handle notifies the connection manager that it has been closed, the managed connection is cleaned up and returned to the pool.

- **Cached-handle**

In the cached-handle model, the application obtains the connection once up-front and caches a reference to it in an instance field. Application writers typically use the cached-handle approach because they believe the application will perform better or when no pooling is available. But because of the benefits of connection pooling under the get-use-close model, there is generally little performance difference between the two usage models. Although the cached-handle model makes the logic in the business method simpler, additional logic is required to close the connection on passivation and re-create it on activation. The possibility also exists that, if the bean gets destroyed by the container (for example, because a method has thrown a runtime exception), an open connection might be left dangling. The biggest problem with the cached-handle model is that while one instance of the bean or servlet is holding on to the connection, another instance cannot use it, so you end up with at least as many connections as instances.

Because a cached-handle is held across method and transaction boundaries by an application, for JCA v1.0, you are encouraged not to cache the connection across the transaction boundary for shareable connections. The get-use-close pattern is preferred.

Connection management in JCA v1.5

The JCA v1.5 specification has addressed the cached-handle issue by introducing an optimization called *lazy connection association*. This optimization has two components, disassociation and lazy association, as described here:

- **Dissociation**

A resource adapter indicates to the connection manager that the adapter supports this optimization. When a connection goes temporarily out of scope (that is, when the bean or servlet method exits), the connection manager can, if it also supports the optimization, dissociate the connection handle held by the application from the underlying managed connection that represents the physical connection (for example, a TCP/IP socket). This lets the managed connection be returned to the pool for use by other parts of the application.

- Lazy association

Rather than reassociating the connection handle with the managed connection the next time a method is called, the optimization uses *lazy association*. If the method does not use the connection, or it only calls simple methods on the connection handle that do not require access to the back end, a managed connection is not removed from the pool unnecessarily. Only when the connection handle is actually used does the connection manager locate a suitable managed connection from the pool and reassociate it with the connection handle.

For more information about JCA 1.5, see article “JCA 1.5, Part 1: Optimizations and life-cycle management,” available at:

<http://www.ibm.com/developerworks/java/library/j-jca1/>

12.2.2 Transaction management

Two types of transaction management are available in JCA:

- The first type involves a Transaction Manager coordinating the activity of multiple resource managers across a single transaction.
- The second type involves a transaction with only a single resource manager, called a local transaction. This second type is not always implemented by the external coordinator, for example, this is not done for IMS Connect.

A resource adapter can also indicate, through its deployment descriptor, that it does not support transactions. Multiple resource managers participating in the same transaction are supported by the Java Transaction API (JTA) `XAResource` interface. This interface allows a Transaction Manager to manage transactions among multiple resources that support the interface. The `ManagedConnection` interface contains a method, `getXAResource()`, that returns an `XAResource` object. The application server's Transaction Manager uses this object to manage the transaction.

Note: When we use the word transaction in the previous text, we mean a unit of work (UOW) in which one or many IMS transactions can be participants, together with other transactions, directly updating resources such as DB2 and DL/I.

In this case, JCA v1.5 also tries to implement more efficiency.

To enlist, or not to enlist

Everyone knows that transactions can be expensive, particularly XA (global) transactions. This makes it all the more important for a transaction not to do more work than necessary. Suppose you have Enterprise JavaBeans (EJB) deployed using container-managed transactions and a transactional attribute of `RequiresNew` for the business method. A new global transaction begins when the method is invoked. When the connection to the EIS is created, the connection manager has no idea how it will be used, so it must obtain an `XAResource` from the associated managed connection and enlist it in the external transaction. The connection might be used only to query the database, or might not be used at all, but the connection manager must enlist the connection anyway, in case, for example, an `IMSTransaction` with update intent is performed. This means that, at a minimum, the resource must make *start*, *commit or rollback*, and *end* flows to the back end.

Completion needs to flow to the resource when the transactional method ends, even if the method did not use the connection transactionally. If another resource becomes involved in the transaction, you have forced an unnecessary two-phase commit, causing an additional prepare flow. The only good thing here is that the resource manager can still return

XA_RDONLY (for *read only*) from the prepare call to indicate that it has not actually done any work. The Transaction Manager does not then need to flow the completion call to that resource manager and, if only one resource manager actually did any work in the transaction, the Transaction Manager might be able to avoid a slow write to its log file.

Avoiding enlisting with JCA v1.5

By now, you should have the idea that you do not want to enlist in the transaction unless absolutely necessary. The JCA V1.5 specification has a solution: *lazy enlistment*. The Java specification, expressed by *Interfaces* and *Abstract* classes, force the vendors to implement the lazy enlistment extension.

The LazyEnlistableManagedConnection interface is a marker interface implemented by the managed connection to indicate to the connection manager that it does not need to eagerly enlist the managed connection in an existing transaction when a new connection is created in a transaction or in a new transaction started when a connection already exists. If a connection handle is about to perform some work that should be part of any transaction, and its managed connection has not already been enlisted, it determines whether the connection manager implements the LazyEnlistableConnectionManager interface. If it does, it calls the lazyEnlist method, passing the managed connection. If a transaction is associated with the calling thread, the XAResource from the managed connection is enlisted at that point.

12.2.3 Other JCA v1.5 items

In addition to the contracts already mentioned, other contracts are part of the new JCA v1.5 specification. For a more detailed explanation of the specification, see the articles written by David Currie, available at:

- ▶ <http://www.ibm.com/developerworks/library/j-jca1>
- ▶ <http://www.ibm.com/developerworks/library/j-jca2>
- ▶ <http://www.ibm.com/developerworks/library/j-jca3>

12.2.4 Interaction with EIS

After a Java application has an IMSConnection object, an IMSInteraction object can be obtained through a createInteraction method on the connection. Figure 12-3 on page 229 shows the basic steps for connecting and interacting with an EIS. Programming this is indeed very simple for a Java programmer; nevertheless, an additional complexity would need to cope with exceptions within try/catch groups.

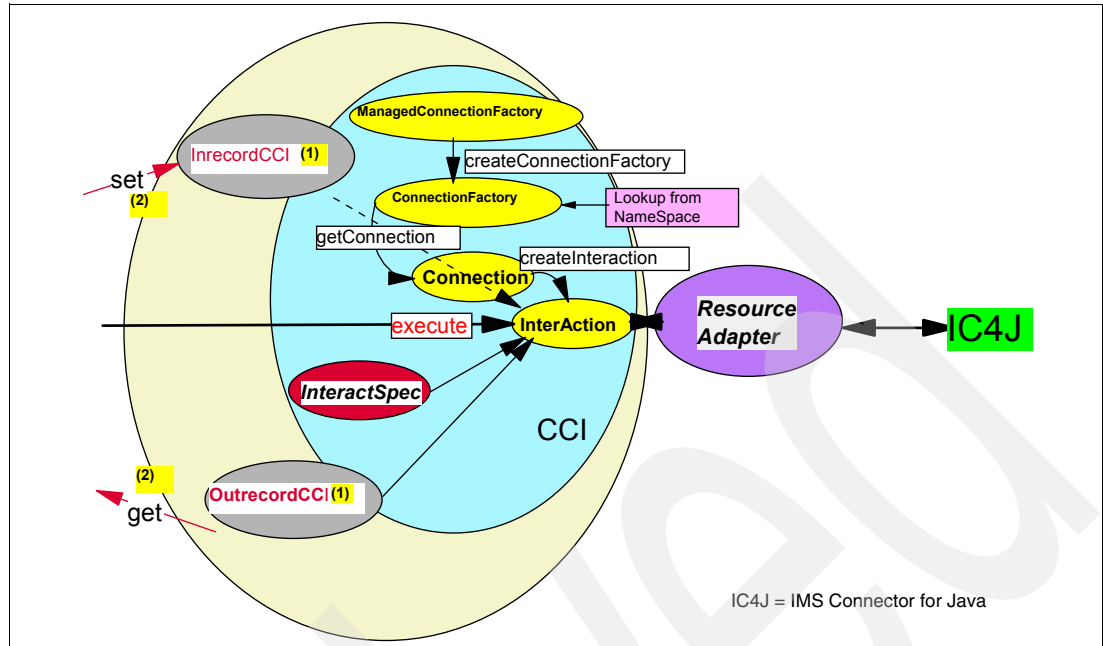


Figure 12-3 Connection and interaction flow with EIS

In the upper part of Figure 12-3, you find again the acquisition of the connection factory as outlined before, followed by `getConnection`. From the connection, you can follow the creation of the `IMSInteraction` object. At first glance, the use of the `IMSInteraction` object is very simple. It has an `execute` method that can be invoked with three parameters:

- ▶ `IMSInteractionSpec`
- ▶ Input record
- ▶ Output record

InteractionSpec

The `InteractionSpec` parameter defines the characteristics of the interaction with IMS. In the case of IMS Connector for Java, the interaction with IMS is through IMS Connect. When IMS Connector for Java communicates with IMS Connect using TCP/IP, the socket connection can be either a dedicated persistent socket connection or a shareable persistent socket connection. The type of socket connection used for an interaction is determined prior to the interaction, when the connection is established. Depending on the EIS that is addressed, specialized implementations exist. For example, the implementation for IMS Connector for Java is called `IMSInteractionSpec`, which contains specific properties regarding the exchange with IMS. Example 12-1 lists the properties in the `IMSInteractionSpec` object. The properties `convEnded` and `asyncOutputAvailable` are output only properties, property `mapName` is an input/output property, and all other properties are input only.

Example 12-1 InteractionSpec properties

```

int commitMode
int executionTimeout
int imsRequestType
int interactionVerb
String ltermName
String mapName,
boolean purgeAsyncOutput
boolean reRoute
String reRouteName
int socketTimeout

```

```
boolean convEnded; (Output)
boolean asyncOutputAvailable; (Output)
```

New properties and property values can be added to support additional function in later releases of IMS Connector for Java. The `IMSInteractionSpec` object has getter and setter methods for all properties. We explain these properties later.

Input record

IMS Connector for Java views the JCA input record as a Java byte array, in a well-defined encoding and containing the fields of the IMS transaction input message in the right order, at the correct offset, with the required length, and aligned as expected by the IMS application program that is invoked. In JCA, this byte array has to be a Java object that implements two CCI interfaces:

- ▶ `javax.resource.cci.Record`
- ▶ `javax.resource.cci.Streamable`

This record can be built using the tooling wizards of an integrated development environment such as WebSphere Studio Application Developer Integration Edition or Rational Application Developer, or your Java application can build the input record without using tooling.

Output record

The output record implements the same CCI interfaces as the input record. For some IMS transactions, there are additional considerations for the output record. An IMS application program can return one of a number of possible output messages. An IMS application program can also return a variable length output message. Variable length output messages include multiple segment output messages and output messages containing arrays with a variable number of elements. In these cases, the Java application must include special logic to process the output message. This logic is illustrated in samples provided by IMS Connector for Java. You can find the samples at the following Web page:

<http://www.ibm.com/software/data/ims/examples/exHome.html>

12.2.5 Security

The JCA security contract uses the Java Authentication and Authorization Service (JAAS) `Subject` class to provide security information. When a new `ManagedConnection` is created, the `createManagedConnection()` method is passed a `Subject` instance. The connection uses the `Subject` when it attempts to sign on to the EIS. A `Subject` contains information about the principal, or name, of the `Subject`, along with information about the security credentials held by the principal.

JCA v1.0 defines two types of credentials:

- ▶ A `GenericCredential` is a Java wrapper, representing a specific security mechanism, such as a Kerberos credential.
- ▶ A `PasswordCredential` holds user name and password information.

An application server supporting JCA v1.0 must provide implementations of both of these interfaces. In JCA v1.5, the `GenericCredential` interface has been deprecated but is still usable. The recommended replacement interface, the `org.ietf.jgss.GSSCredentialInterface` in J2SE™ v1.4, is not yet fully supported by WebSphere Application Server for z/OS nor by IMS Connector for Java. As a result, the `GenericCredential` interface continues to be used by WebSphere Application Server for z/OS and IMS Connector for Java.

The resource adapter's deployment descriptor lists the type of security supported, the authentication mechanism used, the interface of the credentials supported, and whether reauthentication is supported. The `getConnection()` method of a `ManagedConnection` takes a `Subject` as a parameter to support reauthentication.

12.2.6 Summary

JCA describes and enforces with interfaces the way the connection and interaction specifications must be implemented by the vendors, who want to have connection support for their EIS. The specification is EIS neutral. The practical distribution for a particular EIS consists of a framework of classes, extending and implementing the interfaces outlined in the specification. JCA v1.5 brings extensions to the JCA v1.0 specification. Compliance with the new specification requires the implementation of those additional interfaces.

IMS currently provides the following distributions of the IMS resource adapter, where x is a maintenance level of a particular distribution:

- ▶ IMS Connector for Java Version 2.1.0.4
 - Based on JCA v1.0
 - To be used with WebSphere Application Server V5
 - New features only supported with IMS Version 8.1
 - Earlier release features supported:
 - IMS Connect Versions 1.2 and 2.1
 - IMS Versions 7.1 and 8.1
- ▶ IMS Connector for Java Version 2.2.x
 - Based on JCA v1.0
 - To be used with WebSphere Application Server V5 and V6
 - Prerequisites:
 - IMS Connect Version 2.2 and IMS Version 8.1 or IMS Version 9.1
 - Or:
 - Integrated IMS Connect in IMS Version 9.1
- ▶ IMS Connector for Java Version 9.1.0.1.x (requires a license for IMS V9.1)
 - Based on JCA v1.0
 - To be used with WebSphere Application Server V5 and V6
 - Prerequisites:
 - IMS Connect Version 2.2 and IMS Version 8.1 or IMS Version 9.1
 - Or:
 - Integrated IMS Connect in IMS Version 9.1
- ▶ IMS Connector for Java Version 9.1.0.2.x
 - Based on JCA v1.5
 - To be used with WebSphere Application V6
 - Prerequisites:
 - IMS Connect Version 2.2 and IMS Version 8.1 or IMS Version 9.1
 - Or:
 - Integrated IMS Connect in IMS Version 9.1

12.3 Building applications that use IMS Connector for Java

Applications that use IMS Connector for Java can be built in different ways:

- ▶ Using the tools provided with IBM WebSphere Studio Application Developer Integration Edition Version 5.1

These tools generate applications that are based on an enterprise service that represents the IMS transaction. The generated application uses Web Services Invocation Framework (WSIF). The tools also use WSIF. The tools also generate Format Handler classes to aid in the conversion of IMS transaction input and output messages.

- ▶ Using the tools provided with IBM Rational Application Developer V6

Rational Application Developer tools generate applications that are based on a JCA Java bean that represents the IMS transaction. Java data bindings for the input and output message of the IMS transaction are generated separately.

- ▶ Coding directly to the CCI provided by the IMS resource adapter

For this option, the user has the option of using WebSphere Studio Application Developer Integration Edition or Rational Application Developer to generate classes to process the input and output messages of the IMS transaction.

We give examples of all three types of applications and explain how to build them. All three solutions use the CCI. In the case of WebSphere Studio Application Developer Integration Edition and Rational Application Developer, the CCI is used by the generated code.

12.3.1 Introduction

As already outlined, each execution of a transaction in IMS requires two basic steps:

- ▶ Connection: To IMS through IMS Connect

Connections can be reused by several connector clients if they are managed with connection pooling or a connection handler obtained in one particular client is reused for several Interactions.

Connection characteristics are determined by the properties of the ConnectionFactory and the ConnectionSpec instances used during the getConnection method.

- ▶ Interaction: With an IMS transaction (conversational or non-conversational)

Interaction characteristics are determined by the properties of the InteractionSpec instance provided when the execute method of the Interaction instance is invoked. In addition, Input and Output objects are specified during the invocation of the call.

IMS Connector for Java allows for many options in its connection and interaction. These options are specified as properties of objects provided at run time.

In the next section, we describe the different options that influence the behavior of your application.

12.3.2 Connection properties

The characteristics or properties of a connection are determined when the connection is acquired. They are obtained mainly from two sources:

- ▶ IMSConnectionFactory

You should remember that this item can be described as a JNDI reachable object from a namespace from which it can be looked up or through the instantiation of an IMSManagedConnectionFactory from which an IMSConnectionFactory can be created.

Connections that were obtained from a looked-up connection factory are *managed connections*. In this case, the connection pooling mechanism implemented in the environment (container) will provide connection optimizations. This is the case for both shareable and dedicated (with clientID) sockets. The socket connections (shareable or dedicated) are serially reusable, so they cannot be used for two requests at the same time. When one request uses the connection and closes it, the connection is returned to the pool for reuse by another request, then the other request uses it, and so on.

Figure 12-4 shows the difference between managed connection pooling for shareable and dedicated sockets. Connection management provides the efficient use of connections by:

- Avoiding the physical open and close of connections
- Reusing free connections

As shown in Figure 12-4, dedicated sockets are only supported with commit mode 0 protocol.

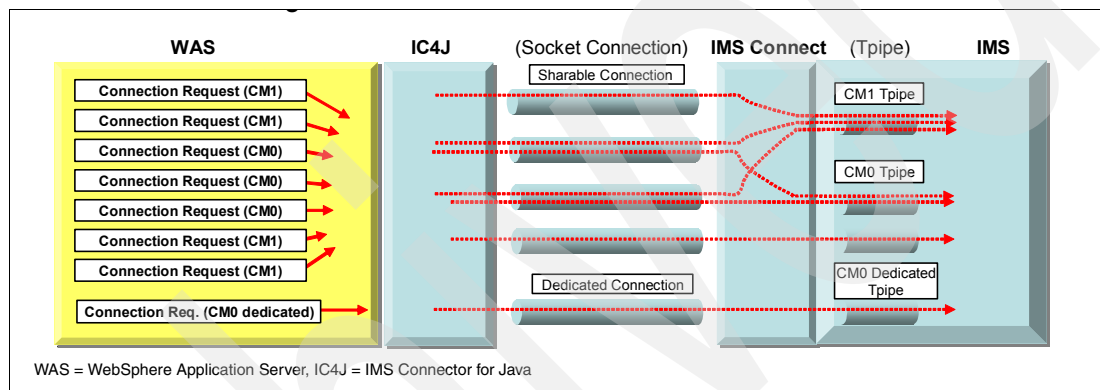


Figure 12-4 Connection pooling

If your client program configures a managed connection factory at runtime and uses it to obtain a connection factory, rather than using a JNDI lookup to obtain a connection factory, it will be using non-managed connections. If your client program uses non-managed connections, it does not have the benefit of an efficient pooling mechanism. In addition, if your client program is using non-managed connections defined as shareable, be aware that a lot of Tpipes can be generated. To avoid this, the client program should use dedicated socket connections.

Example 12-2 shows the properties of the `IMSManagedConnectionFactory` object. This object is populated at execution time. For each of the properties, we have Java accessors. A `ConnectionFactory` for Nonmanaged Connections is obtained by a `createConnectionFactory()` method and carries the properties from the creation class from which it was derived.

Example 12-2 `IMSConnectionFactory` properties

```
Boolean CM0Dedicated
String dataStore
String groupName
String hostName
String IMSConnectName
PrintWriter logWriter
String MFSXMIRepositoryID
String MFSXMIRepositoryURI
String password
Integer portNumber
Boolean SSLEnabled
String SSLEncryption
```

```
String SSLKeystoreName
String SSLKeystorePassword
String SSLTruststoreName
String SSLTruststorePassword
Integer traceLevel
String transactionResourceRegistration
String userName
```

A connection factory obtained by lookup in the name space receives the characteristics from the definition in the namespace and supports *managed connection* handling. This can be realized in a WebSphere Application Server environment.

► **IMSConnectionSpec**

From the connection factory, a handle to an object representing a physical connection is instantiated through the `getConnection` method. An `IMSConnectionSpec` object can be passed as a parameter of the `getConnection` method. This object has two purposes:

- In the case of component-managed EIS sign-on (also referred to as application authentication), it defines the security information that can override the security information in the connection factory.
- In the case of a connection factory configured to create dedicated persistent socket connections (`CM0Dedicated = TRUE`), the `clientId` is specified in this object. Dedicated persistent socket connections are used for interactions that use commit mode 0 synchronization protocol.

Example 12-3 shows the fields of the object.

Example 12-3 IMSConnectionSpec properties

```
String groupName
String password
String userName
String clientId
```

Connection pooling

Connection pooling, as explained earlier, is mostly dictated by the way the connection factory has been obtained. If the container is not managing the connections, the program can preserve the connection handlers. The use of dedicated or shareable persistent sockets also influence the behavior. For a persistent dedicated socket, obviously there is a one-to-one relationship with the `clientId`.

Application security

Application security is based on the following properties:

► **User name**

The Security Authorization Facility (SAF) user ID that will be used for all connections created by this connection factory. This property can be provided by different sources depending on whether you use Component/Application or Container security binding. For the overwrite rules, refer to Figure 12-5 on page 235.

Note: Component/Application or Container security binding can be set in the deployment descriptor of the J2EE artifacts.

- Password

The password that will be used by IMS Connect to verify the SAF user ID for all connections created by this connection factory. The password is optional; it will always be sent to IMS Connect but will only be used if IMS Connect has RACF turned on (RACF=Y in the IMS Connect configuration member or IMS Connect SETRACF ON command has been issued).

- Group name (optional)

The IMS group name that will be used for all connections created by this connection factory.

Note: The IMS group name can only be provided in a Component/Application managed environment.

In a J2EE environment, for example, WebSphere Application Server, the security user ID that is passed to IMS Connector for Java and on to IMS Connect is a user ID that is derived from different sources. We define two terms from Figure 12-5:

- Res-Auth: Specified in a Deployment Descriptor
- ServantID: User ID of the server started task in J2EE

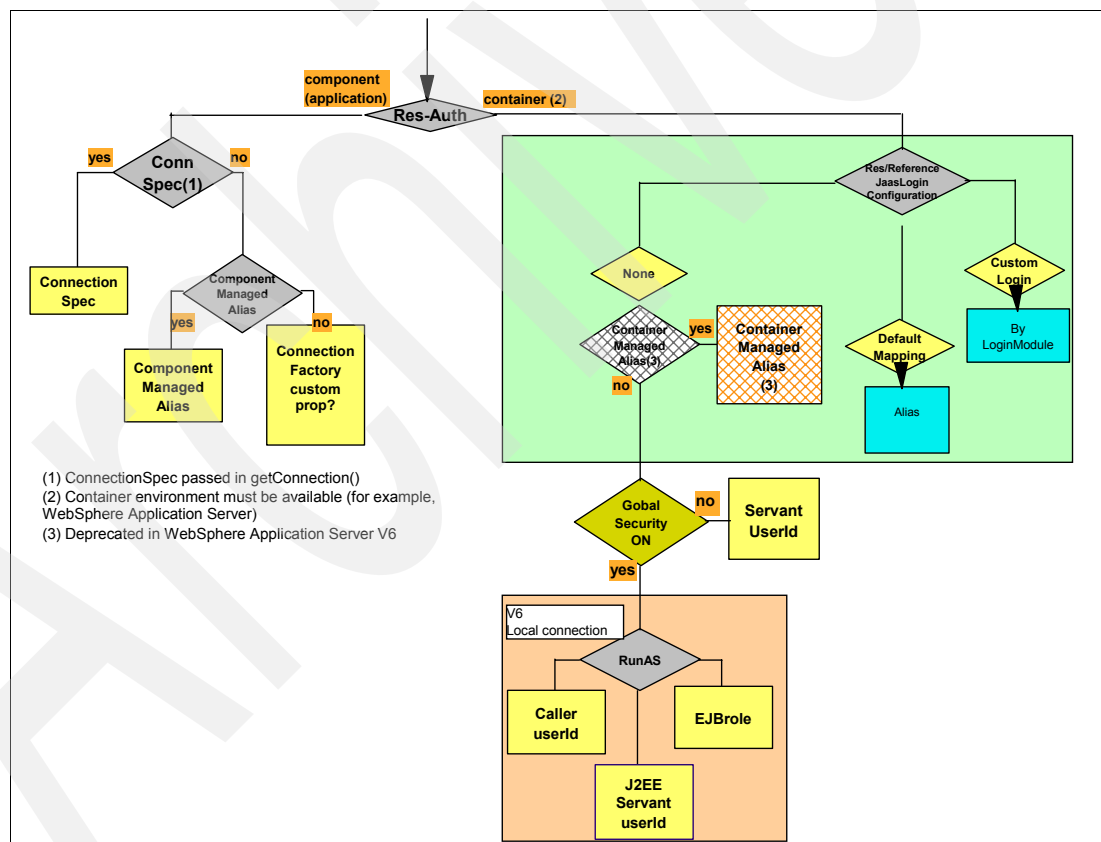


Figure 12-5 Security information overwrite

In Figure 12-5 on page 235, we distinguish two sides:

► *Component (application)-managed authorization*

Based on the Res-Auth option in the resource reference section of the deployment descriptor, we follow the left path. Use the following steps:

- If we specify credential information (user ID/password) with the ConnectionSpec object in the getConnection() call on the ConnectionFactory, this credential information is taken.
- Otherwise, check if a component-managed alias was specified. This alias is specified in the factory definition in the container resources for the cell. User aliases are defined earlier in the global security section of the WebSphere Application Server cell through the administrative functions.
- Otherwise, custom properties of the factory are taken if they exist.

If *no* credentials have been provided, they are blank. An exception will be thrown if authentication information is required by the resource.

This type of authorization behaves the same as in WebSphere Application Server Version 5.

► *Container-managed authorization*

Based on the Res-Auth option in the resource reference of the deployment descriptor, we follow the right path. This option changed a lot with WebSphere Application Server Version 6. Use the following steps:

- If the authentication method is set to none, the behavior is similar to WebSphere Application Server Version 5. A container-managed alias can be indicated. This alias belongs to the factory definition in the container resources for the cell.
- Aliases are defined earlier in the global security section of the WebSphere Application Server cell through the administrative functions.
- In this case, if no alias was specified, as for WebSphere Application Server Version 5:
 - If global security is off, the servant user ID is used.
 - If global security is on, but only with a local option, the RunAs identity is selected.
- If the authentication method is set as *default*, a default JAAS login configuration *DefaultPrincipalMapping* is used. This configuration takes the Java class (com.ibm.ws.security.auth.j2c.WSPPrincipalMappingLoginModule) as a login module. The module takes a parameter, an alias. The reference to this alias is done in the resource reference section of the deployment descriptor and travels with the deployment. An alias that is valid in the development tool probably will not be valid on the z/OS system. This alias reference can be adapted during the deployment phase of the J2EE application (EAR file), or later through the administration console.
- If the authentication method is set as *custom JAAS login*, we have the opportunity to specify our own JAAS login module. This provides a lot of freedom. In this module, we have to respect a predefined implementation. It is possible to extract the “Utoken/thread” identity from the J2EE security and propagate it to connection. As such, the connection to IMS can be executed under the RunAs (caller, role, and so on) identity. This mechanism is active outside global security. If you want to develop a new mapping LoginModule in WebSphere Application Server Version 6, use the programming interface described in the “Developing your own J2C principal mapping module” topic of *IBM WebSphere Application Server for z/OS, Version 6.0.1: Securing applications and their environment*, SA22-7961.

Service point

This information specifies how IMS Connect can be reached from IMS Connector for Java. The information is part of the ConnectionFactory instance.

- ▶ Host name

Mandatory for TCP/IP connections: The IP address or host name of the machine running the target IMS Connect.

- ▶ Port number

Mandatory for TCP/IP connections: The number of a port used by the target IMS Connect for TCP/IP connections. Multiple sockets can be open on a single TCP/IP port.

- ▶ IMS Connect name

Mandatory for local option connections: The job name of the target IMS Connect. The IMS Connect name overrides host name and port number, so it must not be specified for TCP/IP connections.

Transport security

The user ID security information on which the authorization in IMS is based is passed over the local option connection or over TCP/IP. In the latter case, we often have to secure the transport layer of this connection with Secure Sockets Layer (SSL). When we refer to the word client in the context of this SSL discussion, this is *not* the end-user client accessing IMS. In this context, the client is IMS Connector for Java, because the connection that is being secured is only the connection between IMS Connector for Java and IMS Connect. The setup of the SSL protected socket connection is only done once by an initial handshake. This socket connection is persistent and continues to be used in this protected state. SSL exists in two flavors:

- ▶ SSLv2: Only the server proves its identity.
- ▶ SSLv3: Both the client and server prove their identity.

IMS Connector for Java only supports *SSLv3* through its successor Transport Layer Security (TLS) v1. The identity on the client side used to establish the protected connection is determined by properties in IMSConnectionFactory.

In SSL, a keystore is a password-protected database that contains key entries that consist of an entity's identity and its private key. A truststore is a key database (keystore) intended to contain only trusted certificate entries, that is, the identity and public key of known and trusted entities. Typically, a keystore is used to hold your own certificates and private keys, while a truststore is used to hold other entities' certificates and public keys. You might elect to provide more protection for your private keys by storing them in a keystore with restricted access, while making your truststore more accessible. However, you can choose to keep both your and others' private and public keys and certificates in the same keystore or truststore.

If no value is specified for the KeyStore Name property, the value of the TrustStore Name property is used as both the KeyStore Name and the TrustStore Name. Likewise, if no value is specified for the TrustStore Name property, the value of the KeyStore Name property is used as both the KeyStore Name and the TrustStore Name. If neither the KeyStore Name nor the TrustStore Name values are specified, ICO0096I exceptions are thrown for both the KeyStore Name and the TrustStore Name: Warning. Invalid value provided for SSL parameter. In addition, an IC00003E: Failed to connect to host exception and the underlying FileNotFoundException occur. This is because IMS Connector for Java has no name to use when it tries to open a Java Keystore (JKS) format keystore as it attempts to initialize an SSL socket connection to the host.

On z/OS, RACF can be used as the key repository for the keyrings. A keyring does exactly what its name implies—it contains keys. It holds the private key (default) for the owner of the

keyring and public keys (certificates) of various certificate authorities, used for the verification of partner certificates. A keyring also contains the certificate of the keyring owner for exchanging it with possible partners during an SSL handshake. A z/OS keyring can also be used as a truststore, in which case, it contains trusted certificate entries. Note that, even when WebSphere Application Server for z/OS and IMS Connect are in the same z/OS image and use the same instance of RACF, you probably want to use separate keyrings for WebSphere Application Server and IMS Connect.

The following IMSConnectionFactory properties are used to define SSL connections:

- ▶ **SSL Enabled**

The default is false. This property is only valid for TCP/IP connections. A value of true indicates that IMS Connector for Java will create an SSL socket connection to IMS Connect using the HostName and PortNumber specified in these connection properties. This port must be configured as an SSL port by IMS Connect.

- ▶ **KeyStore Name**

For non-z/OS platforms, specify the fully qualified path name of your JKS keystore file. For z/OS, specify the name of your JKS keystore file as previously discussed, or a special string that provides the information needed to access your RACF keyring. Private keys and their associated public key certificates are stored in password-protected databases called keystores.

The keystore name can be used to specify either a JKS keystore or a RACF keyring when running on z/OS. A RACF keyring is specified as: *keystore_type:keyring_name:racfid*. The *keystore_type* must be either JCERACFKS when software encryption is used for SSL, or JCE4758RACFKS if hardware encryption is used. Replace *keyring_name* with the name of the RACF keyring that you are using as your keystore and *racfid* with a RACF ID that is authorized to access the specified keyring. Examples of RACF keyring specifications are:

```
JCERACFKS:myKeyring:kruser01  
JCE4758RACFKS:myKeyring:kruser01
```

When running in z/OS, if the keystore name matches the above RACF keyring format and points to a valid RACF keyring, IMS Connector for Java uses the specified RACF keyring as its keystore. If the RACF keyring format is used and the keystore type specified is anything other than JCERACFKS or JCE4758RACFKS, IMS Connector for Java attempts to interpret the keystore name specified as the name of a JKS keystore file.

- ▶ **KeyStore Password**

Specify the password for the keystore. Private keys and their associated public key certificates are stored in password-protected databases called keystores. The KeyStore password property is used with both JKS keystores and RACF keyrings.

- ▶ **TrustStore Name**

For non-z/OS platforms, specify the fully-qualified path name of your JKS truststore file. For z/OS, specify the JKS name or the RACF keyring of the truststore. The same format is used for the values of the KeyStore Name and TrustStore Name properties. See the description of the KeyStore Name property for a discussion of this format. A truststore file is a key database file (keystore) intended to contain public keys or certificates of trusted entities.

- ▶ **TrustStore Password**

Specify the password for the truststore. A truststore file is a key database file that contains public keys. The KeyStore Password property is used with both JKS keystores and RACF keyrings.

► Encryption Type

Select the encryption type. Strong and weak are related to the strength of the ciphers, that is, the key length. All those ciphers that can be used for export come under the weak category and the rest go into the strong category. By default, the encryption type is set to weak.

On the server side, IMS Connect initialization for SSL is done through the file referenced by SLENVAR parameter in the IMS Connect configuration. SSL also requires the availability of protected TCP/IP ports, which are specified through the SSLPORT parameter in the IMS Connect configuration. Example 12-4 shows an example of an IMS Connect SSL configuration file (this is also the default setup of SSL).

Example 12-4 SSL configuration

```
#####
This is an SSL interface configuration file
#####
GSK_PROTOCOL_SSLV2=GSK_PROTOCOL_SSLV2_ON
GSK_PROTOCOL_SSLV3=GSK_PROTOCOL_SSLV3_ON
GSK_PROTOCOL_TLSV1=GSK_PROTOCOL_TLSV1_ON
GSK_KEYRING_FILE=SSLRING
GSK_KEYRING_LABEL=IMS_CONNECT
GSK_KEYRING_PW=
GSK_KEYRING_STASH_FILE=
GSK_CLIENT_AUTH_TYPE=GSK_CLIENT_AUTH_FULL_TYPE
GSK_SESSION_TYPE=GSK_SERVER_SESSION
GSK_V2_CIPHER_SPECS=642
GSK_V3_CIPHER_SPECS=0906030201
```

In Example 12-4, the following fields are important:

- GSK_KEYRING_FILE=SSLRING: The qualified keyring name is userid.SSLRING, where userid is the ID under which IMS Connect is running.
- GSK_KEYRING_LABEL=IMS_CONNECT: The label of the certificate in the keyring.

Figure 12-6 on page 240 shows how the system is retrieving information for setting up a secure TCP/IP channel:

- In IMS Connector for Java, each connection factory can be configured independently to create SSL connections.
- IMS Connect can be configured to accept SSL connections through the IMS Connect and SSL configuration members.

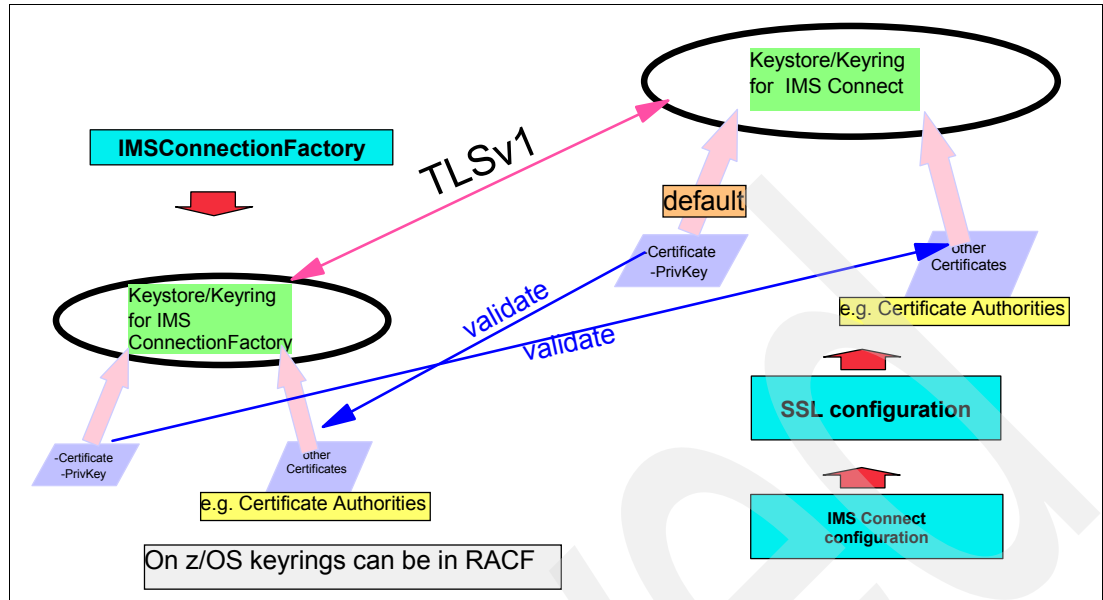


Figure 12-6 TLSv1 elements

A complete example, “Configuring SSL for IMS Connector for Java and IMS Connect,” which explains all the required steps including generating certificates, is available at:

<http://www.ibm.com/software/data/ims/examples/exHome.html>

Dedicated or shareable persistent sockets

The CM0Dedicated property of the IMSConnectionFactory object determines whether the acquired socket connection is dedicated or shareable. A dedicated connection requires the setting of a clientID through the connectionSpec object passed in the getConnection method.

The default is false. A value of false indicates that the connection factory will generate shareable persistent socket connections and IMS Connector for Java will generate a clientID to identify the socket connection. These connections can be used by commit mode 0 and commit mode 1 interactions. A value of true indicates that the connection factory will generate dedicated persistent socket connections, which require user-specified clientIDs to identify the socket connections. A dedicated persistent socket connection is reserved for a particular clientID, and only commit mode 0 interactions are allowed. This property applies to TCP/IP connections only.

Mandatory reference to IMS

Datastore name is the name of the target IMS datastore. It must match the ID parameter of the datastore statement that is specified in the IMS Connect configuration member. It also serves as the XCF member name for IMS during internal XCF communications between IMS Connect and IMS OTMA.

Optional trace

Trace level is the level of information to be traced. Four levels are supported:

- ▶ RAS_TRACE_OFF (0)
- ▶ RAS_TRACE_ERROR_EXCEPTION (1)
- ▶ RAS_TRACE_ENTRY_EXIT (2)
- ▶ RAS_TRACE_INTERNAL (3)

In a non-managed environment, if tracing is needed, the application must configure the `IMSManagedConnectionFactory` object.

Example 12-5 Setting trace information in a non-managed environment

```
// Set the trace level.
mcf.setTraceLevel(new Integer(IMSTraceLevelProperties.RAS_TRACE_INTERNAL));
// Provide a value for the file to be used for the trace.
try {
    PrintWriter pr = new PrintWriter(new java.io.BufferedWriter(
        new java.io.FileWriter("c:/temp/CCISampleLog.log", false)));
    mcf.setLogWriter(pr);
} catch (Exception ex) {

}
```

In the WebSphere test environment of the development environment or WebSphere Application Server, ensure that the **Enable trace** check box is selected. To enable logging and tracing in the IMS resource adapter, enter the following in the Trace string field:

```
com.ibm.connector2.ims.*=all=enabled
```

Other combinations of trace strings will enable tracing in other components. For example, with the following trace string, the string `com.ibm.ejs.j2c.*` provides you with logging and tracing of the WebSphere implementation of the J2EE connector architecture and the string `com.ibm.connector2.*` provides you with logging and tracing of all of the resource adapters, including IMS:

```
com.ibm.ejs.j2c.*=all=enabled:com.ibm.connector2.*=all=enabled
```

Note: Trace strings can vary slightly with different versions of WebSphere Application Server.

Transaction enlistment

In “To enlist, or not to enlist” on page 227, we explain transaction enlistment as provided by the JCA v1.5 IMS resource adapter.

`TransactionResourceRegistration` (optional) indicates the type of transaction resource registration (enlistment). Valid values are either “static” (immediate) or “dynamic” (deferred). If this property is set to dynamic, the enlistment of the resource to the transaction scope is deferred until the resource is used for an interaction for the first time.

Note: This is a property of the JCA v1.0 IMS resource adapters. For the JCA v1.5 IMS resource adapter, transaction enlistment does not require the specification of a property value.

MFS support

The following properties only apply for applications generated by development environments that support MFS. At the time of the publication of this book, only WebSphere Studio Application Developer Integration Edition supports MFS.

► **MFS XMI Repository ID**

This property is used by applications generated from MFS source. This field contains a unique name for identifying the repository location. This ID must match the repository field defined in the generated format handler of your application. The default for this field is default.

► MFS XMI Repository URI

This property is used by applications generated from MFS source. This field specifies the physical location of the XMI repository. Valid formats for this field include:

- `file://path_to_xmi`, where `path_to_xmi` is a directory on the local file system containing the XMI files, for example, `file://c:/xmi`.
- `http://url_to_xmi`, where `url_to_xmi` is a valid URL that resolves to a directory containing the XMI files, for example, `http://sampleserver.com/xmi`.
- `hfs://path_to_xmi`, where `path_to_xmi` is the HFS directory on the host z/OS. This format is only supported for WebSphere Application Server for z/OS.

12.3.3 Interaction properties

The execution of an IMS transaction is invoked through the `execute` method on an `IMSInteraction` object. One of the parameters that has to be specified on the `execute` method is the `IMSInteractionSpec` object, describing the characteristics of this IMS interaction.

Figure 12-7 shows the set methods of the `IMSInteractionSpec` class that are used to give a value to a property.



- `setAsyncOutputAvailable(boolean arg0) void - IMSInteractionSpec`
- `setCommitMode(int arg0) void - IMSInteractionSpec`
- `setConvEnded(boolean arg0) void - IMSInteractionSpec`
- `setExecutionTimeout(int arg0) void - IMSInteractionSpec`
- `setImmsRequestType(int arg0) void - IMSInteractionSpec`
- `setInteractionVerb(int arg0) void - IMSInteractionSpec`
- `setLtermName(String arg0) void - IMSInteractionSpec`
- `setMapName(String arg0) void - IMSInteractionSpec`
- `setPurgeAsyncOutput(boolean arg0) void - IMSInteractionSpec`
- `setReRoute(boolean arg0) void - IMSInteractionSpec`
- `setReRouteName(String arg0) void - IMSInteractionSpec`
- `setSocketTimeout(int arg0) void - IMSInteractionSpec`

Figure 12-7 *IMSInteractionSpec* setters

The following sections describe all the properties of the `IMSInteractionSpec` object.

Output only

These output only properties are not set by the application component:

► `asyncOutputAvailable` (output only)

This property is used by a Java application on return from a commit mode 0 interaction on a dedicated or shareable persistent socket to determine if there is queued output for the associated `clientId`. `clientId` is a property of `IMSConnectionSpec` and can be a user-specified value or an IMS Connector for Java generated value. The value of `asyncOutputAvailable` is true if there are messages in the queue. The `asyncOutputAvailable` property is not set on input by the application component.

Note: If your Java application uses this property, it must be exposed as an output property of `IMSInteractionSpec`.

- convEnded (output only)

This property is used by a Java application to determine if a conversation has ended (true). The convEnded property is *not* set on input by the application component.

Note: If your Java application uses this property, it must be exposed as an output property of IMSInteractionSpec. A conversation, as previously referenced, is a Java application that invokes an IMS transaction with a scratch pad area (SPA) defined.

commitMode

The following two commit modes are available:

- 0 (commit_then_send): The transaction is committed or aborted by the IMS Transaction Manager and the result message is queued on a Tpipe. Commit_then_send uses sync level 1.
- 1 (send_then_commit): The transaction status is waiting for a synchronization message. Note that the message processing regions are occupied as long as the synchronization message is received.

The value is used by the IMS resource adapter to indicate the type of commit mode processing to be performed for an IMS transaction.

The commitMode property can be set to 0 or 1 when interactionVerb is set to SYNC_SEND_RECEIVE.

When interactionVerb is set to SYNC_RECEIVE_ASYNCOUTPUT, SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT, SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT, or SYNC_SEND, IMS Connector for Java uses commitMode 0.

CommitMode 1 is required when interactionVerb is set to SYNC_END_CONVERSATION. If commitMode is 0 and a dedicated persistent socket is used for the interaction, the clientID property of the IMSConnectionSpec must be provided for the interaction. If commitMode is 0 and a shareable persistent socket is used for the interaction, the clientID must not be specified. If commitMode 0 or 1 is specified for an interaction on a shareable persistent socket, the output message from a transaction can be purged or rerouted. The output message is recoverable by using a subsequent interaction within the same application with interactionVerb set to SYNC_RECEIVE_ASYNCOUTPUT, SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT, or SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT.

If a dedicated persistent socket is used for a commitMode 0 interaction, the output message from a transaction cannot be purged or rerouted, but the output message is recoverable using a subsequent interaction with interactionVerb set to SYNC_RECEIVE_ASYNCOUTPUT, SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT, or SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT.

imsRequestType

imsRequestType indicates the type of IMS request and determines how output from the request is handled by the IMS resource adapter. Integer values are:

- 1 (IMS_REQUEST_TYPE_IMS_TRANSACTION)

The request is an IMS transaction. Normal transaction output returned by IMS is used to populate the application's output message. If IMS returns a "DFS" message, the IMS resource adapter throws an IMSDFSMessageException. This value for imsRequestType is used for applications that are not generated using WebSphere Studio MFS support.

► 2 (IMS_REQUEST_TYPE_IMS_COMMAND)

The request is an IMS command. Command output returned by IMS, including “DFS” messages, is used to populate the application’s output message. The `IMSDFSMessageException` is not thrown. This value for `imsRequestType` is used for applications that submit IMS commands.

► 3 (IMS_REQUEST_TYPE_MFS_TRANSACTION)

This value for `imsRequestType` is reserved for applications that are generated using WebSphere Studio MFS support. Normal transaction output returned by IMS, as well as “DFS” messages, are used to populate the application’s output message. The `IMSDFSMessageException` is not thrown.

interactionVerb

`interactionVerb` defines the mode of interaction between the Java application and IMS. The values currently supported by the IMS resource adapter are:

► 0 (SYNC_SEND)

The IMS resource adapter sends the client request to IMS through IMS Connect and does not expect a response from IMS. With a `SYNC_SEND` interaction, the client does not need to synchronously receive a response from IMS. `SYNC_SEND` is supported on both shareable and dedicated persistent socket connections and is only allowed with `commitMode 0` interactions. If the `interactionVerb` is set to `SYNC_SEND`, execution timeout and socket timeout values are ignored. Note that `imsRequest` type 2 is not allowed with `SYNC_SEND` and will generate an exception.

► 1 (SYNC_SEND_RECEIVE)

The execution of an IMS Interaction sends a request to IMS and receives a response synchronously. A typical `SYNC_SEND_RECEIVE` interaction is the running of a non-conversational IMS transaction in which an input record (the IMS transaction input message) is sent to IMS and an output record (the IMS transaction output message) is returned by IMS. `SYNC_SEND_RECEIVE` interactions are also used for the iterations of a conversational IMS transaction. A conversational transaction requires `commitMode 1`. A non-conversational transaction can run using either `commitMode 1` or `commitMode 0`. If `commitMode 0` is used on a dedicated persistent socket, a value for the `clientId` property of `IMSConnectionSpec` must be provided. If `commitMode 0` is used on a shareable persistent socket, a value for the `clientId` property of `IMSConnectionSpec` must not be provided.

► 3 (SYNC_END_CONVERSATION)

If the application executes an interaction with `interactionVerb` set to `SYNC_END_CONVERSATION`, the IMS resource adapter sends a message to force the end of an IMS conversational transaction. For `SYNC_END_CONVERSATION`, `commitMode 1` is required. The `clientId` is not allowed.

► 4 (SYNC_RECEIVE_ASYNCOUTPUT)

`interactionVerb SYNC_RECEIVE_ASYNCOUTPUT` is valid on both shareable persistent and dedicated persistent socket connections. `SYNC_RECEIVE_ASYNCOUTPUT` is used to retrieve asynchronous output that was not delivered. When `SYNC_RECEIVE_ASYNCOUTPUT` is used on a dedicated persistent socket, a value must be provided for the `clientId` property of `IMSConnectionSpec`.

With this type of interaction, the Java client can only receive a single message. If there are no messages in the IMS OTMA asynchronous queue for the `clientId` when the request is made, no further attempts are made to retrieve the message. No message is returned and a timeout will occur after the length of time specified in the `executionTimeout` property of the `SYNC_RECEIVE_ASYNCOUTPUT` interaction.

► 5 (SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT)

interactionVerb SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT is valid on both shareable and dedicated persistent socket connections. It is used to retrieve asynchronous output. With this type of interaction, the Java client can only receive one single message. If there are no messages in the IMS OTMA asynchronous queue for the clientID when the request is made, no further attempts are made to retrieve the message. No message will be returned and a timeout will occur after the length of time specified in the executionTimeout property of the SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT interaction.

Note: The interactionVerbs, SYNC_RECEIVE_ASYNCOUTPUT and SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT, perform the same function. However, we recommend that you use SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT with Rational Application Developer V6.0.0.2 with the IMS resource adapter V9.1.0.1.1 or V9.1.0.2.

► 6 (SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT)

interactionVerb SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT is used to retrieve asynchronous output. It is valid on both shareable and dedicated persistent socket connections. With this type of interaction, the Java client can only receive one single message. If there are no messages in the IMS OTMA asynchronous queue for the clientID when the request is made, IMS Connect waits for OTMA to return a message. IMS Connect waits the length of time specified in the executionTimeout property of the SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT interaction before returning an exception.

Note: The J2EE Connection architecture (JCA) value SYNC_RECEIVE (2) is currently not supported.

Timeouts

Two types of timeout can be defined. The socketTimeout needs to be larger than the executionTimeout, as shown in Figure 12-8. The value that you need to set depends on from your environment and the response times that you can expect.

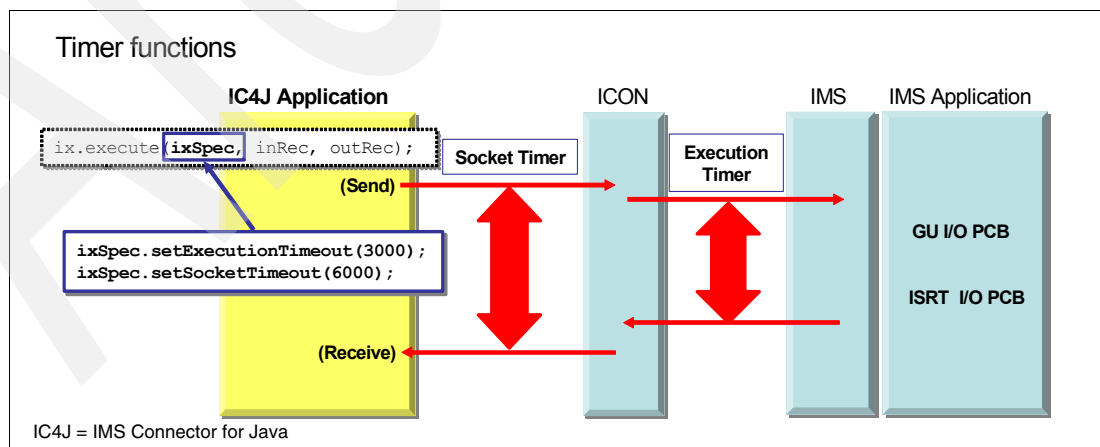


Figure 12-8 Execution and socket timer

The two types are:

- ▶ **socketTimeout**

The maximum amount of time IMS Connector for Java will wait for a response from IMS Connect before disconnecting the socket and returning an exception to the client application. The `socketTimeout` value is represented in milliseconds. To use socket timeout, the value must be greater than zero. If a socket timeout is not specified for an interaction, or it is supplied with a socket timeout value of zero milliseconds, this results in no socket timeout or an infinite wait.

- ▶ **executionTimeout**

The maximum amount of time allowed for IMS Connect to send a message to IMS and receive a response. The `executionTimeout` value is represented in milliseconds and must be a decimal integer that is either -1 or between 1 and 3,600,000, inclusively. That is, the `executionTimeout` value must be greater than zero and less than or equal to one hour. If a -1 value is set for this property, the interaction runs without a time limit.

ltermName

This is the LTERM name used to override the value in the LTERM field of the IMS application program's I/O PCB. See *IMS Version 9: IMS Connect Guide and Reference*, SC18-9287, for a description of how to use the LTERM override. The value of this property can be set if the client application wants to provide an LTERM override name. This name will be in the IMS application program's I/O PCB, with the intent that the IMS application will make logic decisions based on this override value.

mapName

The `mapName` field typically contains the name of a Message Format Service (MFS) control block. MFS is the component of IMS that performs online formatting of transaction input and output messages. Because IMS Connect uses IMS OTMA to access IMS, MFS online formatting is bypassed. However, the `mapName` field can still be used by a Java application to input the name of an MFS control block to an IMS application program or to retrieve the name of an MFS control block provided by an IMS application program.

On input, typically the value of the `mapName` property is the name of an MFS message output descriptor, or MOD. The MOD name is provided to the IMS application program in the I/O PCB. On output, the value of the `mapName` property is the name of an MFS message output descriptor. This is the MOD name that the IMS application program specified when inserting the transaction output message to the I/O PCB.

Note: The `mapName` field should not be used by Java applications that use an enterprise service whose input and output messages are generated by WebSphere Studio MFS support.

purgeAsyncOutput

This is an input property. This property determines whether or not IMS Connect purges undelivered output. This property is only valid for interactions on shareable persistent socket connections that use the IMS interaction verb `SYNC_SEND_RECEIVE`. It is not valid for any interactions on dedicated persistent socket connections. It applies to commit mode 0 interactions. It does not apply to commit mode 1 interactions. However, if a commit mode 1 interaction executes a program-to-program switch, the spawned program will run commit mode 0, and therefore, the property will apply.

If the `purgeAsyncOutput` property is not specified on a `SYNC_SEND_RECEIVE` interaction on a shareable persistent socket connection, the default is `TRUE`. If this property is set to `TRUE`, the following output messages are purged:

- ▶ Undelivered output message inserted to the I/O PCB by the primary IMS application program
- ▶ Output messages inserted to the I/O PCB by secondary IMS application programs invoked by a program-to-program switch

reRoute

This is an input property. This property is only valid for `SYNC_SEND_RECEIVE` interactions on shareable persistent socket connections with either a `commit mode 0` or a `commit mode 1` interaction that spawns a program-to-program switch that invokes another `commit mode 0` interaction and results in undeliverable secondary output. This property determines if undelivered output is to be rerouted to a named destination specified in the `reRouteName` field.

If `reRoute` is `TRUE`, the asynchronous output is not queued to the `Tpipe` of the generated `clientID`. Instead, the asynchronous output is queued to the destination specified in the `reRouteName` field. The default value for `reRoute` is `FALSE`. If both `reRoute` and `purgeAsyncOutput` are set to `TRUE`, an exception is thrown.

reRouteName

This property provides the name of the destination to which asynchronous output is queued. If `reRoute` is `TRUE`, this property provides the named destination. If `reRoute` is `FALSE`, the `reRouteName` property is ignored. If the `reRoute` property is set to `TRUE`, and no `reRouteName` is provided, the value for the `reRouteName` property is:

- ▶ The value specified in the IMS Connect configuration file.
- ▶ If no value is specified in the IMS Connect configuration file, the value `__HWS$DEF__` is used.

The property, `reRouteName`, is only valid for `SYNC_SEND_RECEIVE` interactions on shareable persistent socket connections. It is not valid for any interactions on dedicated persistent socket connections.

12.3.4 Use considerations

The most used interactionVerbs are `SYNC_SEND` and `SYNC_SEND_RECEIVE`. The other verbs are related to the retrieval of asynchronous or non-delivered output.

SYNC_SEND: Programming model

Always execute this interactionVerb with *commitMode 0*. In general, no output is expected because it normally addresses a *non-response* mode transaction. This is the normal way of using it.

If this verb is used with a *response mode* transaction, an output message is inserted to the I/OPCB and queued on the `Tpipe` with a `clientID`. Here, we distinguish the following cases:

- ▶ Shareable persistent socket
The `clientID` has been generated by the system. In this case, *no* messages can be retrieved, and after a while, there is overflow on the `Tpipe` queue.

- Dedicated persistent socket

The clientID is mandatory and has been provided by the IMS Connect client in the *IMSConnectionSpec* object. The messages queued on the Tpipe can be retrieved by issuing a SYNC_RECEIVE_ASYNCOUTPUT interaction.

If the program is inserting an output message to an alternate PCB, which is associated with an Tpipe destination, those messages can be retrieved under the following conditions:

- The *CM0Dedicated* property was set to TRUE in the connection factory.
- SYNC_RECEIVE_ASYNCOUTPUT interaction, commitMode 0, clientID of alternate PCB.

SYNC_SEND_RECEIVE: Programming model

This interactionVerb can be executed with commitMode 0 and 1.

commitMode 0

To better understand the meaning of this commit mode, we use the expression *commit_then_send*. In other words, the commit/backout is decided by IMS at the end of the transaction execution, the message processing region (MPR) is freed immediately, and a message, inserted by the message processing program, is put on the Tpipe queue. This probably is the best way to trigger stand-alone transactions with update intent because the queued message will stay on the queue as long as a dequeue has not been confirmed by an ACK from the client, which is sent automatically by the resource adapter after receiving the output message.

A commitMode 0 with SYNC_SEND_RECEIVE cannot participate in a global unit of work under RRS control. IMS is always the coordinator, and the message processing region is freed as soon the IMS sync point terminates.

Figure 12-9 shows the flow.

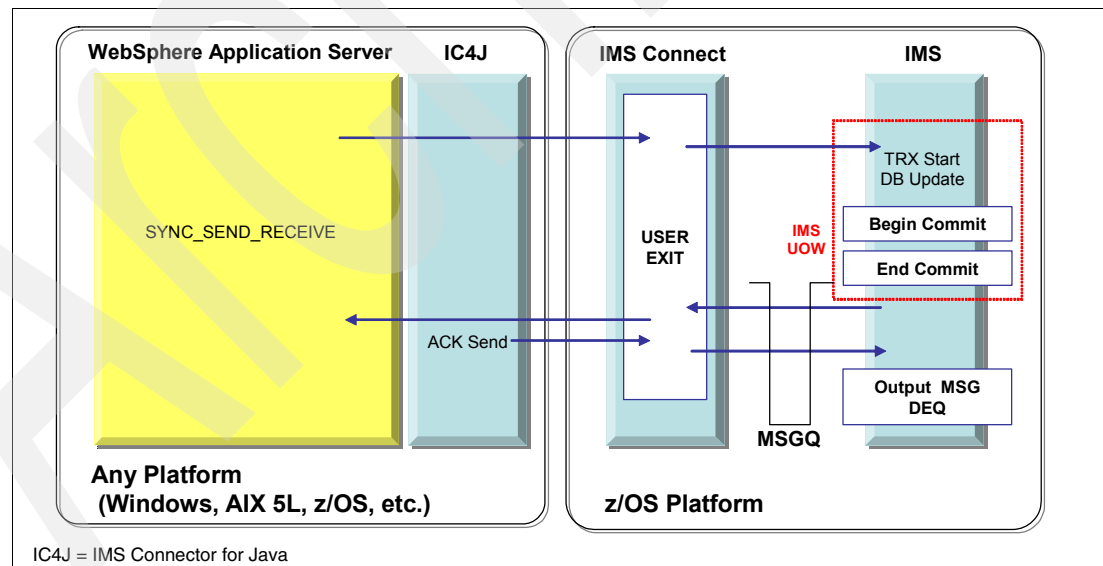


Figure 12-9 IMS Connector for Java flow for commit mode 0

The confirm (ACK) is automatically sent by IMS Connector for Java after receiving the message, after which the message is dequeued. We can work both with *shareable* and *dedicated* persistent sockets. In this case, there is a difference in the error processing and handling of undelivered output. Undelivered output might also be caused by an execution

time when the socket connection timed out before the execution was producing any output. Here, we distinguish the following cases:

► **Shareable persistent socket**

This socket type implies an automatic generated clientID.

All errors result in a resource exception being thrown to the client application. In this case, the output message cannot be delivered anymore to the client and it will not be queued to the Tpipe. This undelivered output can be handled in a couple of ways depending on the specified options:

- `purgeAsyncOutput=TRUE` (default)

The following output is purged:

- Undelivered output message inserted to the I/O PCB by the primary MPP (the first program invoked by the transaction).
- Output messages inserted to the I/O PCB by a secondary program, invoked by a program switch.

- `reroute=true`, `reRouteName=xxxxx`, `purgeAsyncOutput=false`

- Undelivered output messages are queued to the named reroute destination.
- Non-delivered messages because of Socket timeout are queued to the Tpipe corresponding to the generated clientID.

- `reroute=true`, `reRouteName=xxxxx`, `purgeAsyncOutput=false`

- Undelivered output messages are queued to the named reroute destination.

► **Dedicated persistent socket**

This socket type requires that a clientID is specified through the `IMSConnectionSpec` object. All errors result in a resource exception being thrown to the client application. If the output message cannot be delivered to the client, it is queued for later retrieval by `SYNC_RECEIVE_ASYNCOUTPUT`.

Figure 12-10 shows the `SYNC_RECEIVE_ASYNCOUTPUT` flow. The properties `purgeAsyncOutput` and `reRoute` are not applicable for this type of socket.

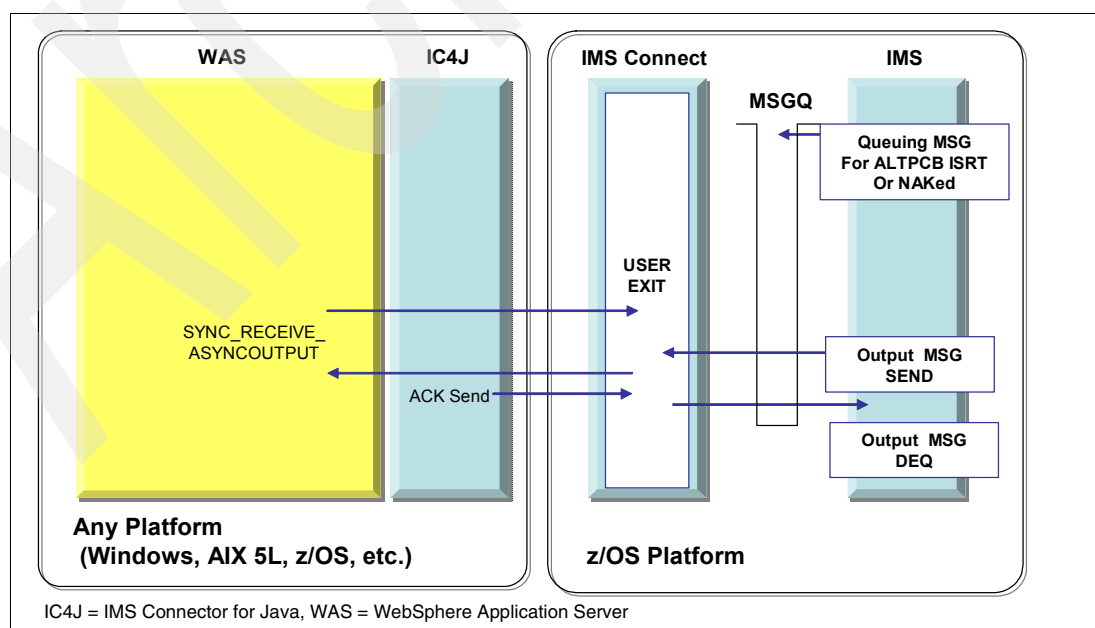


Figure 12-10 `SYNC_RECEIVE_ASYNCOUTPUT` flow for commit mode 0

commitMode 1

With this mode, only *shareable persistent sockets* are supported. To better understand the meaning of this commit mode, we use the expression `send_then_commit`. After sending the output message directly to the client, the message processing region (MPR) in which the transaction ran waits on a commit/abort hint, and consequently the MPR stays occupied with the message processing program (MPP). The synchronization level will be defined by the transactional context in which the transaction was invoked and determine whether the transaction is part of a global unit of work or not. Basically, we have the following cases:

- No transactional context

We are in this situation when the client does not run in a container, or if the external container method (for example, the method of an Enterprise JavaBeans, or EJB) that invokes the IMS transaction was tagged in its container with the attribute `TX_NOTSUPPORTED`. In this case, the invoked transaction is a stand-alone transaction, and the selected sync level will be *none*. Under this sync level, the output message is not put on the Tpipe queue but send immediately to the client. The MPR in which the transaction ran waits for an acknowledgment, which with sync level *none* will be the confirmation that the output message was sent correctly over XCF to the IMS Connect, even before it was passed to the IMS client. This is depicted in Figure 12-11.

With this scenario, be aware that the occupancy time of the MPR has been increased slightly and that the fact that the message was acknowledged by the XCF in the IMSConnector is not a guarantee for a correct arrival at the client, for example, in WebSphere Application Server. Only use this type for query only transactions.

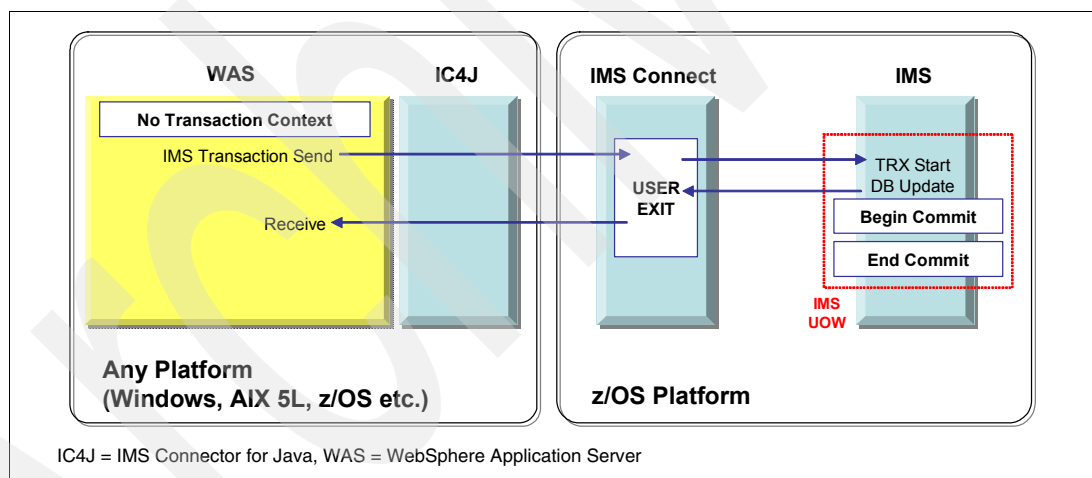


Figure 12-11 IMS Connector for Java flow for commit mode 1, sync level none

- Transactional context

A transactional context can be set in a container, for example, a client container or a WebSphere container. Transactional context means that a global unit of work (GUOW) has begun, which for z/OS is registered with RRS, and as a result of this, the invoked transaction will be part of this GUOW. Several other accesses from the client through the resource managers can be part of the same GUOW. The synchronization level used in this case is `SYNCPT`. RRS is responsible for coordinating the total synchronization point processing, but the hint to start this processing is given by the container.

Next, we explain briefly how a transactional context can be obtained and when the global commit/abort signal is fired. The transaction type is set in the deployment descriptor of the Enterprise JavaBeans (EJB). Example 12-6 on page 251 provides an example.

Example 12-6 EJB transaction type description

```
<enterprise-beans>
  <session id="IMSPhoneSess">
    <ejb-name>IMSPhoneSess</ejb-name>
    <home>j2cEJB.IMSPhoneSessHome</home>
    <remote>j2cEJB.IMSPhoneSess</remote>
    <ejb-class>j2cEJB.IMSPhoneSessBean</ejb-class>
    <session-type>Stateless</session-type>
    <transaction-type>Container</transaction-type> <== or Bean ==>
  </session>
</enterprise-beans>
```

– Declarative (transaction type Container)

EJBs are objects with several entry points, called methods. All methods have by default or explicitly a *transactional* attribute. This attribute determines the transactional behavior of everything invoked from within this method. We distinguish the following values:

- Required: A new transaction created if one does *not* exist; otherwise, continue with the existing transaction.
- Supports: No new transaction created; eventually continue with the existing transaction.
- RequiresNew: A new transaction is created.
- Mandatory: We should already be in a transaction.
- Never: Method will not be part of any transaction.

There are many conditions for which we can be in a transactional context. See Example 12-7.

Example 12-7 Declarative settings for transaction type Container

```
<container-transaction>
  <method>
    <ejb-name>IMSPhoneSess</ejb-name>
    <method-intf>Remote</method-intf>
    <method-name>getPhoneInfoSyncpt</method-name>
    <method-params>
      <method-param>java.lang.String</method-param>
      .....
      <method-param>java.lang.String</method-param>
    </method-params>
  </method>
  <trans-attribute>Required</trans-attribute> <=====
</container-transaction>
<container-transaction>
  <method>
    <ejb-name>IMSPhoneSess</ejb-name>
    <method-intf>Remote</method-intf>
    <method-name>getPhoneInfo</method-name>
    <method-params>
      <method-param>java.lang.String</method-param>
      .....
      <method-param>java.lang.String</method-param>
    </method-params>
  </method>
  <trans-attribute>Supports</trans-attribute> <=====
</container-transaction>
```

The location where the global transaction will be committed/aborted is where the transactional context implicitly (based on the attribute) was set up.

– Programmatic (transaction type Bean/Component)

In this case, the transactional context is created by code. The J2EE application uses the JTA `javax.transaction.UserTransaction` interface to demarcate a transaction boundary to a set of changes to the protected resource programmatically. Component-managed transactions can be used in both the *servlet* and the *EJB* environment. For an EJB, you set the transaction attribute in its deployment descriptor as `TX_BEAN_MANAGED`.

In the session bean method, after instantiating a `UserTransaction` object, a transaction normally begins with a `UserTransaction.begin()` call. When the application component is ready to commit the changes, it invokes a `UserTransaction.commit()` call to coordinate and commit the changes. If the application component must roll back the transaction, it invokes `UserTransaction.rollback()` and all changes are backed out. Here, the global transaction demarcation is under the control of the program.

The code in Figure 12-8 shows how this can be coded. Remember that this can be done both in a servlet, an EJB with a `BEAN_MANAGED` transaction type, and eventually in code running in a client container.

Example 12-8 Code excerpt for controlling a UserTransaction in a servlet or an EJB

```
// Get User Transaction from EJB context
UserTransaction transaction = ejbcontext.getUserTransaction();
// Get User Transaction in a servlet
Context ic = new InitialContext();
UserTransaction ut = (UserTransaction)ic.lookup("java:comp/UserTransaction");
// Start transaction
transaction.begin();
// Make changes to the protected resources.
// For example, use the J2EE/CA's CCI Interaction interface
// to submit changes to an EIS system(s)
interaction.execute(interactionSpec, input, output);
if (/* decide to commit */) {
    transaction.commit(); // commit the transaction
} else {
    /* decide to roll back */
    transaction.rollback(); // rollback the transaction
}
```

The sync point processing that is initiated by the application code or implicitly in case of `container_MANAGED` transaction type can be two-phase commit (2PC) or one-phase commit (1PC). When several resources are involved, it is 2PC, as shown in Figure 12-12 on page 253.

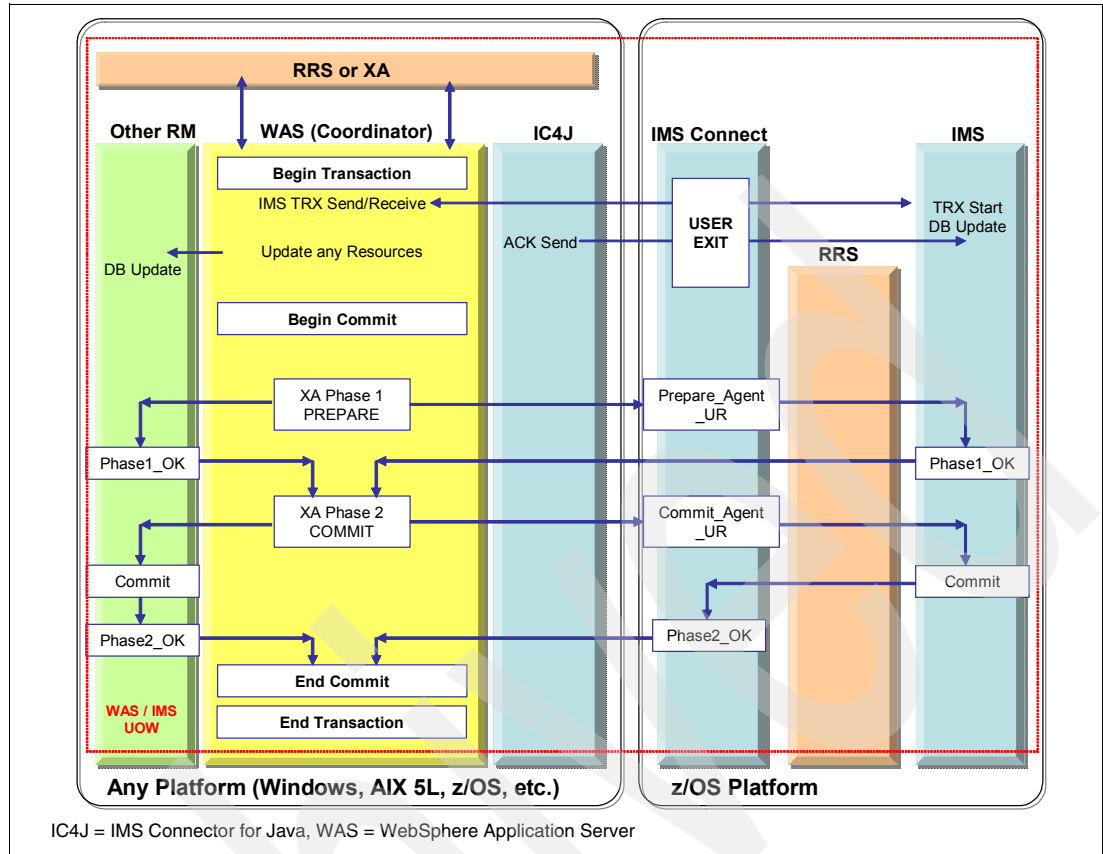


Figure 12-12 Commit mode 1 flow, sync level SYNCPT with multiple resources 2PC

The connection between the WebSphere platform on the left and the IMS platform will be over TCP/IP. The local option (a program call) is also possible if both are in the same z/OS. In Figure 12-12, you clearly see the *begin commit* in the coordinating system, which uses the local RRS or XA to coordinate the local resources and drives over the session (TCP/IP or PC) with IMS Connect, the sync point to IMS. IMS Connect acts as a remote coordinator for IMS using the RRS on that platform.

If only one resource has been accessed, for example, only one IMS transaction, WebSphere Application Server is able to apply some optimization and drive a *one-phase commit*, as shown in Figure 12-13 on page 254.

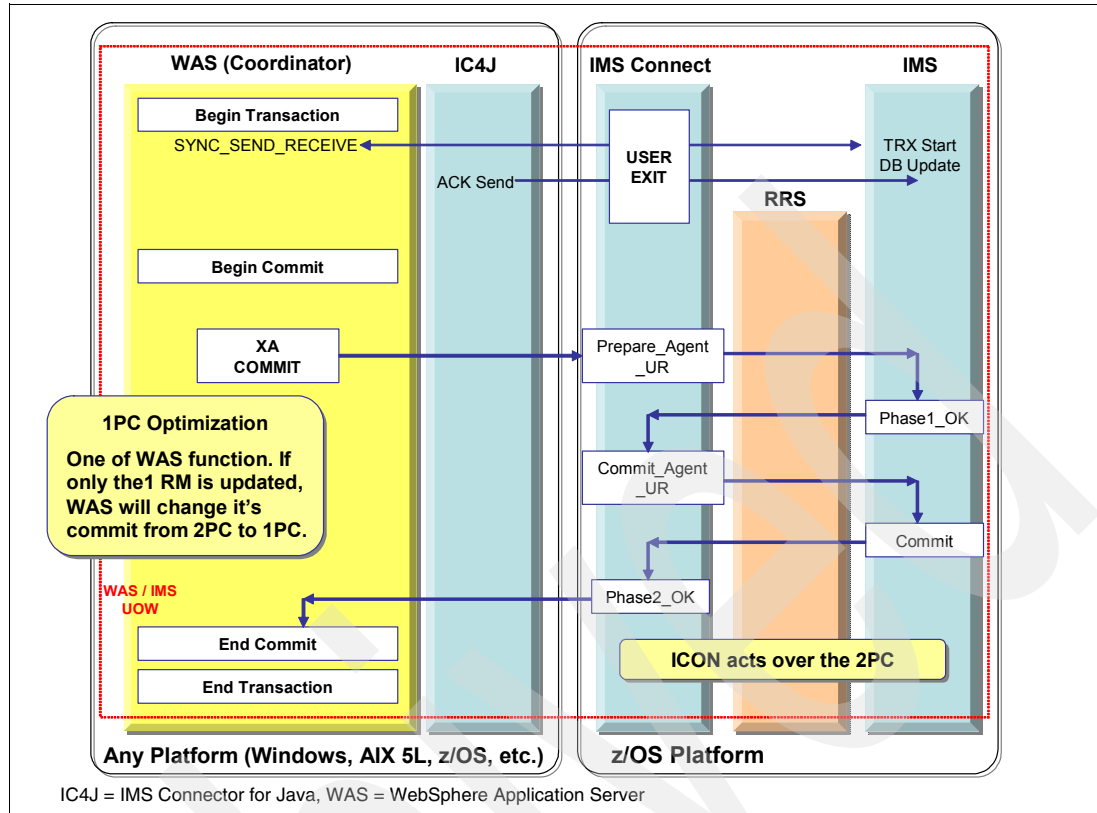


Figure 12-13 Commit mode 1, sync level sync point with one resource 1PC

The sync point processing is again driven by WebSphere Application Server but, as the optimization has been detected, there is no interference with the local RRS or XA coordinator. Remotely, it works as before.

To run a two-phase commit application, or even the one-phase commit, consider the following suggestions:

- It is best to have as many MPP regions as possible running to ensure that two-phase commit applications do not contend for a region, because a transaction that is within a two-phase commit application uses an MPP region for the duration of the entire two-phase commit transaction.
- To safeguard against a transaction that might be waiting for an extensive amount of time for resources, we recommend that you set an appropriate timeout value for each interaction taking place within the global transaction.
- Avoid having an excessive number of database interactions performed in one two-phase commit transaction. If multiple IMS transactions are used within a two-phase commit transaction, they might contend or lock in an attempt to update or modify the same data. To avoid this, it is best to write an application that will prevent a user from accessing duplicate entries within the same two-phase commit operation.
- If multiple interactions are performed using the same IMS transaction on the same IMS database within a global transaction (unit of work), each interaction with that IMS transaction must run on a separate MPP region. The IMS transaction must have a SCHDTYP=PARALLEL and a PARLIM=0 value to indicate that the IMS transaction can run on multiple MPP regions and that it will always meet the scheduling requirements (the number of messages will be greater than zero) to process every interaction on a new MPP region.

- If an external transaction is calling an IMS transaction as part of a GUOW, consider putting this call as close as possible toward the global commit/backout so that the elapsed time of freezing the MPR and the locks on resources (DB2, DLI, WebSphere MQ) are as short as possible.
- Avoid using global transactions if it is not required. Verify the transactional attributes of the EJBs.
- All calls within a global transaction time frame do *not* have to participate in this global transaction.

12.3.5 Summary

Figure 12-14 shows the different options possible for IMS Connector for Java.

Commit	TRX ^(b)	Shareable Sockets		Dedicated Sockets	
		Non_MGT	Managed	Non_MGT	Managed ^(c)
0	YES				
	NO	synclvl->confirm tpipe=HWSxxxx provide pooling ^(a)	synclvl->confirm tpipe=HWSxxxx	synclvl->confirm tpipe=clientID	synclvl->confirm tpipe=clientID
1	YES	synclvl->syncpt tpipe= portnr	synclvl->syncpt tpipe= portnr		
	NO	synclvl->none tpipe= portnr	synclvl->none tpipe= portnr		

Synclevel is defined by Adapter
 Only cases with text are applicable
^(a) Tpipes are created with System generated names
 program should provide pooling/reuse to avoid wildgrow
^(b) Transaction means entered with Transactional Context
^(c) Managed means Connection Factory obtained with lookup

Figure 12-14 Summary chart

Note the following information from Figure 12-14:

- Use option 1 for IMS update transactions that are not part of a global transaction.
This is only possible with commit mode 0 and only in non-transactional context. Verify the timer options and eventually care for the retrieval of non-delivered output.
- Use option 2 when an IMS update transaction is part of a global transaction.
The transactional context has to be set up. Consider the consequences.
- Use option 3 for query only transactions.
This is really the fastest option. It is chosen by IMS Connector for Java for commit mode 1, no transactional context.

Use sync point processing with one-phase commit if IMS is only participant; otherwise, two-phase commit. This requires the most resources, can take a long time, and blocks and locks several resources (message processing regions, locks on databases, and so on).

IMS Connector for Java rerouting and timeout support

In Versions 2.2.3, 9.1.0.1.1, and 9.1.0.2, IMS Connector for Java introduced some enhancements to the asynchronous messages support. These enhancements consist of:

- ▶ The option to automatically purge (discard) the asynchronous messages, instead of queuing them.
- ▶ The option to automatically reroute the asynchronous messages to a Tpipe specified by the client application, instead of queuing them under the Tpipe that is associated with the IMS Connector for Java generated clientID.
- ▶ The option to build a specialized client to request asynchronous messages without the need to poll the Tpipe continuously.

This chapter discusses these enhancements and the way you can implement them in your IMS Connector for Java client application.

13.1 Asynchronous message processing

We define an *asynchronous message* as an item of information sent by IMS Connect to a client asynchronously relative to its normal flow of execution.

In a normal send-receive interaction, the client application expects to read an IMS response just after it has sent the actual transaction message. An IMS Connector for Java application does not do anything explicit to get a transaction response. The `execute()` method of the Interaction object takes care of:

- ▶ Sending the transaction to IMS Connect
- ▶ Getting the IMS response
- ▶ Sending, if needed, the positive acknowledgement (ACK) to IMS Connect so that the response is dequeued (for commit mode 0 interactions) or the transaction is committed (for commit mode 1 interactions)
- ▶ Reading the IMS Connect response to the ACK to check whether the process has ended correctly

This flow of execution is based on the idea that the IMS transactions send messages to their clients in a synchronous, predictable way. Unfortunately, this is not always the situation in real applications:

- ▶ A transaction can insert a message intended for a third-party terminal or a client different from the one that sent the original transaction. This is done at the IMS application level by using an alternate PCB (ALTPCB) and issuing a sequence of Transaction Manager calls against the alternate IOPCB (ALTPCB): CHNG, ISRT, and PURG. These messages are asynchronous by their own nature, because the destination terminal or client cannot predict when such an operation will take place.
- ▶ A transaction, or a sequence of transactions chained by means of program-to-program switches, can insert more than one response to the originating terminal or client. Excepting the first response, which will be read by the client in its normal flow of execution, these messages are asynchronous, because the client has no way to know if the IMS application will send more than one response.
- ▶ An interaction can timeout due to an exceptional high response time from IMS, for example, when the transaction stays in the message queue for a long time before it gets processed. In this case, when the transaction gets processed, the client will not be waiting for the response any more.

Refer to 7.4, “Asynchronous output support” on page 100 for the full details about how to retrieve the asynchronous messages. In this chapter, we take a deeper look in the ways that those messages can be generated and routed before a client application can get them.

13.2 Messages inserted to ALTPCB

When an application program issues a CHNG call against an ALTPCB, it can use a transaction code or an LTERM name as new destination. If the destination is a logical LTERM, IMS lets you determine the real destination using the DFSYPRX0 and DFSYDRU0 exits. Refer to Chapter 9, “IMS Connect user exit support” on page 115 for the details. At this point, we should just notice that:

- ▶ You can send the response to an OTMA client of your choice using the DFSYPRX0 exit.
- ▶ You can specify the final Tpipe destination of the message using the DRU0 exit (which can be different for each OTMA client you use).

The DFSYPRX0/DRU0 combination gives you enough flexibility to manage the ALTPCB output according to your needs. As simple suggestions, you can use DFSYPRX0/DRU0 to:

- ▶ Send all your ALTPCB output to a single (or a few) known IMS Connect Tpipe. The messages get queued in the asynchronous hold queue of that Tpipe, and you could use a specialized IMS Connector for Java client to retrieve them and send to the final clients.
- ▶ Send each ALTPCB output message to an IMS Connect Tpipe, according to the LTERM, using a different Tpipe for each LTERM. In that case, each of your clients recover the messages by themselves. Consider the memory requirements of each Tpipe if you choose this option. The Tpipes use storage, and they do *not* free this storage when you stop using them.
- ▶ Send all your ALTPCB output to another OTMA client, such as the IBM WebSphere MQ IMS bridge, and use any WebSphere MQ client to retrieve the messages.

13.3 Multiple and timed out IOPCB responses

In this section, we discuss the actions you can instruct IMS Connect to take *before* queuing the asynchronous messages under their destination Tpipe. Basically, you have the following choices:

- ▶ You can do *nothing*, and the messages are discarded.
- ▶ You can *discard* the messages, so they do not get queued at all.
- ▶ You can tell IMS Connect to *reroute* the messages, changing the name of the Tpipe where they will be queued.

The issues discussed in this section do not apply to the ALTPCB inserted messages. We discuss the management of those messages in 13.2, “Messages inserted to ALTPCB” on page 258.

13.3.1 Discarding the non-delivered messages

You can ask IMS Connector for Java to purge the asynchronous, undelivered output, setting the Boolean property `purgeAsyncOutput` to true. This is a property of the `IMSInteractionSpec` object and can be set in different ways, depending on the programming environment you are using:

- ▶ You can set it specifically if you are coding your own Java client using the CCI interface using the setter `setPurgeAsyncOutput` of the `IMSConnectionSpec` class.
- ▶ You can use the WebSphere Application Studio Developer Integration Edition or Rational Application Developer wizards and property sheets to set this value. Figure 13-1 on page 260 shows the WebSphere Studio Application Developer Integration Edition property sheet.

Property	Value
commitMode	1
convEnded	
socketTimeout	0
executionTimeout	0
reRoute	false
mapName	
ltermName	
interactionVerb	1
purgeAsyncOutput	true
imsRequestType	1
reRouteName	
asyncOutputAvailable	

Figure 13-1 WebSphere Studio Application Developer Integration Edition operation property sheet

In any case, you have to be aware of the restrictions summarized in Table 13-1.

Table 13-1 Results of enabling *purgeAsyncOutput*

Type of message	Commit mode 1	Commit mode 0	
	Shareable connection	Shareable connection	Dedicated connection
Non-timed out insert to IOPCB	Ignored	Ignored.	Runtime exception
Timed out insert to IOPCB	The output message is lost. Because IMS Connector for Java uses sync level=NONE, the transaction is <i>not</i> rolled back. The net effect is equivalent to the message being purged.	The message is purged.	Runtime exception
Second or later insert to IOPCB by second or later transaction as result of program-to-program switch	The message is purged. The output of second and later transactions is treated as commit mode 0.	Non-predictable. See 13.3.2, "Rerouting the non-delivered messages" on page 261.	Runtime exception
Insert to IOPCB by a transaction sent using a SEND ONLY interaction	Not applicable.	The message is <i>not</i> purged, but queued under the system-generated clientID.	Runtime exception

13.3.2 Rerouting the non-delivered messages

If you let IMS Connector for Java and IMS Connect manage the non-delivered IOPCB responses in their default way, and you want to retrieve those messages, you need to issue RESUME TPIPE interactions to the same clientIDs that sent the original transactions. This can be done in two different ways, depending on the kind of connections you are using:

- Using *shareable connections*

You must issue the SYNC_RECEIVE_ASYNCOUTPUT using the same Connection object that you used to send the original transaction. The clientID for this type of connections is automatically generated by IMS Connector for Java and is associated with each physical Connection object.

Restriction: If you are running your application in a managed environment (for example, under WebSphere Application Server), you cannot be sure if you will again get an instance of a Connection after you close it. In a managed environment, the Connection instances are pooled, and the pool manager can decide to destroy a specific one. In that case, you will not be able to retrieve any asynchronous output queued under the Tpipe associated with that connection.

- Using *dedicated connections*

You must issue the SYNC_RECEIVE_ASYNCOUTPUT interaction using whatever Connection object you want, but you have to specify the same clientID that you used to send the original transaction.

Restriction: You *must* take the appropriate precautions to make sure that one specific clientID is used by just one application thread or instance at a specific moment of time. If you are issuing a SYNC_RECEIVE_ASYNCOUTPUT against a clientID that is being used at the same time to send a transaction to IMS Connect, you will get an exception.

The two restrictions noticed above can make it impractical to use the same Tpipe you used to send the original message to retrieve the asynchronous output. The first restriction means that you should design your application in such a way that the SYNC_RECEIVE_ASYNCOUTPUT interactions immediately follow the transmission of the original transactions. One such application designs might use SYNCH_SEND interactions (that is, SEND ONLY messages) to send the transaction to IMS and get all the output using SYNC_RECEIVE_ASYNCOUTPUT (that is, RESUME TPIPE). Even in this case, you might lose messages if the responses are delayed and the pool manager decides to destroy a Connection instance with pending asynchronous output before the application has the chance to retrieve it.

However, the second restriction imposes the burden of implementing a serialization algorithm designed in such a way to avoid collisions (each clientID should be used just by one process instance or thread), starvation (all clientIDs should be polled for asynchronous output in a reasonable interval of time), and race conditions.

Starting with IMS Connector for Java Versions 2.2.3, 9.1.0.1, and 9.1.0.2, you can avoid this problem using the reRoute flag and the optional reRoute destination name. You can set both the flag and the destination name using the corresponding attribute setters, setReroute() and setRerouteName(), of the IMSInteractionSpec object, or the Rational Application Developer or WebSphere Studio Application Developer Integration Edition property sheet, as shown in Figure 13-1 on page 260.

The effects and restrictions of setting the reRoute flag are similar to the ones noted for purgeAsyncOutput. Table 13-2 shows the resulting outcome.

The name of the Tpipe where a rerouted message is queued is determined as follows:

- ▶ If you specify a non-blank, valid value for reRouteName, that value is used as the Tpipe name.
- ▶ If you leave reRouteName blank, then if the HWSCFGxx member contains a value for the RRNAME, that value will be used as the Tpipe name.
- ▶ Otherwise, the value HWS\$DEF will be used.

Table 13-2 Results of enabling reRoute

Type of message	Commit mode 1	Commit mode 0	
	Shareable connection	Shareable connection	Dedicated connection
Non-timed out insert to IOPCB	Ignored.	Ignored.	Runtime exception
Timed out insert to IOPCB	The output message is lost. Because IMS Connector for Java uses sync level=NONE, the transaction is <i>not</i> rolled back.	The message is rerouted.	Runtime exception
Second or later insert to IOPCB by second or later transaction as result of program to program switch	The message is rerouted. The output of second and later transactions is treated as commit mode 0.	Not predictable. See below.	Runtime exception
Insert to IOPCB by a transaction sent using a SEND ONLY interaction	Not applicable.	Not rerouted. The message gets queued under the system-generated ClientId.	Runtime exception

At the time of writing, the rules applied to decide if a commit mode 0 message will be rerouted are based on the capability of IMS Connect of delivering the message to its client. *If the message is delivered, it will not be rerouted.* IMS Connect considers a message to be delivered if it gets an ACK or a NAK from the client. IMS Connector for Java sends an ACK for the first response message to a transaction, and NAKs all the ulterior responses. IMS Connect will try to send all the response messages until:

- ▶ It gets a NAK from IMS Connector for Java.
- ▶ The socket is disconnected.

Then, consider the following scenario:

1. IMS Connector for Java sends to IMS Connect a response mode IMS transaction (TRANA), using commit mode 0, and requesting reRoute.
2. TRANA inserts another transaction (TRANB) using a program-to-program switch (CHNG-ISRT-PURG against an ALTPCB).
3. TRANA writes its response to the IOPCB and terminates.
4. TRANB writes its response to the IOPCB and terminates.
5. IMS Connector for Java receives the TRANA response (RESPA).
6. IMS Connector for Java ACKs RESPA.
7. IMS Connector for Java issues a recv() after the ACK.

The final outcome of RESPB depends on the order in which these events take place:

- If TRANB inserts RESPB after IMS Connector for Java issues the `recv()` following the ACK, that `recv()` will time out, because there will be no message to retrieve at the time it is issued. Consequently, IMS Connector for Java will not get RESPB. IMS Connect will consider RESPB not deliverable, and thus will reroute it. Figure 13-2 depicts this case.

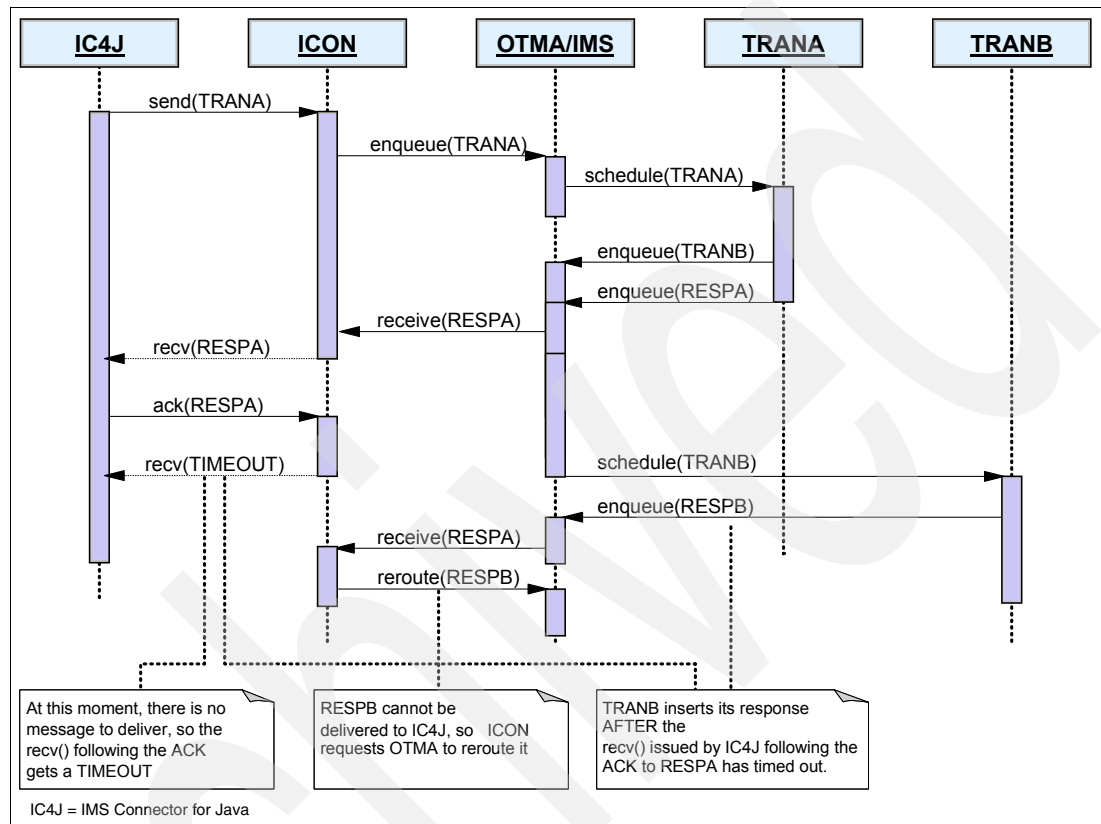


Figure 13-2 Sequence of events that lead to a rerouted secondary message

- If TRANB inserts RESPB before IMS Connector for Java issues the `recv()` following the ACK, that `recv()` will get RESPB. Because IMS Connector for Java cannot deliver RESPB to the application, it will send a NAK to IMS Connect. However, IMS Connect will consider RESPB as delivered, because IMS Connector for Java actually received and NAKed it. IMS Connect then will not reroute RESPB, which will be left enqueued under the Tpipe associated with the original clientId. Figure 13-3 on page 264 shows this case.

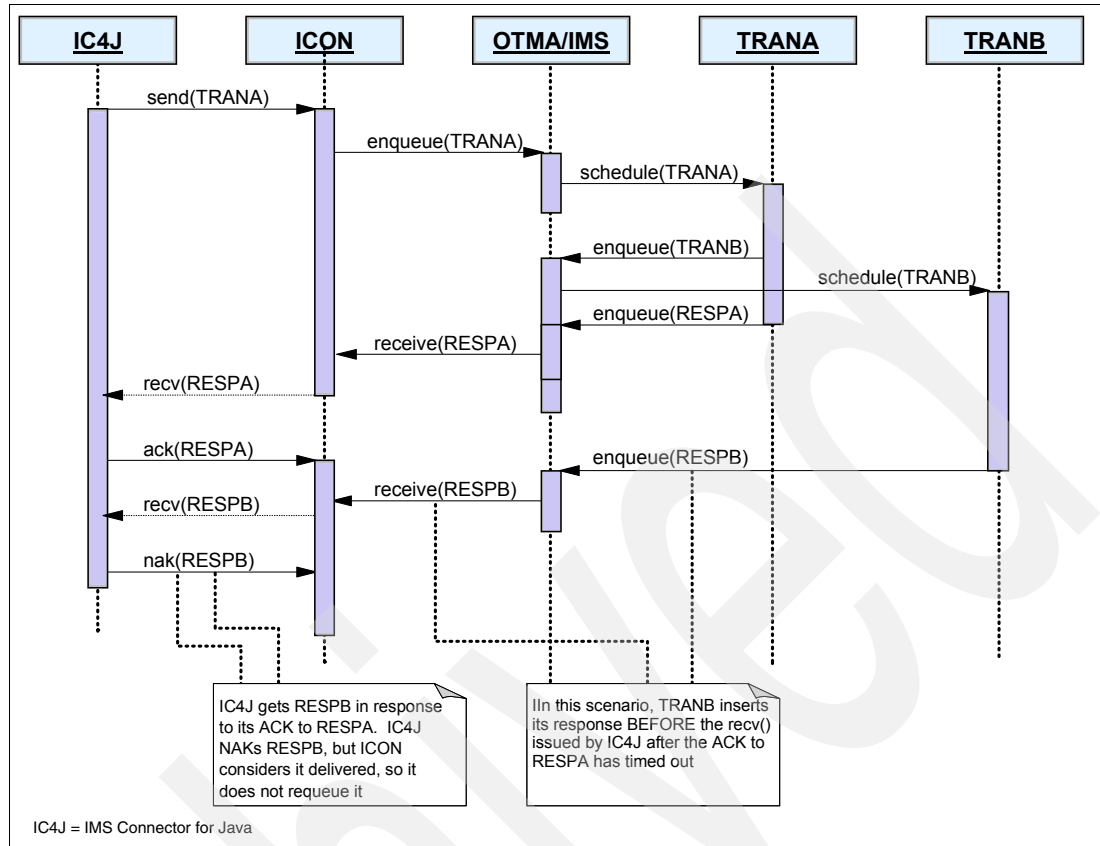


Figure 13-3 Sequence of events that lead to a non-rerouted secondary message

It is interesting to notice that this does not apply if the primary transaction (TRANA) is sent using a commit mode 1 interaction. In that case, the secondary transaction (TRANB) will be executed in commit mode 0. IMS Connector for Java uses sync level none for CM1 interactions, so there is no ACK, or a second rcv() call. IMS Connect will not try to deliver TRANB in any case, so it will be considered non-deliverable, and thus will be rerouted.

Restriction: Because the outcome of any secondary output sent using a CM0 interaction and specifying the reRoute flag is not predictable, we do not recommend using this feature. Periodically check the IMS Connector for Java Web site for changes and enhancements related to the reRoute mechanism.

Building roll your own clients

In this chapter, we describe how can you send transactions and commands to IMS using IMS Connect, but without using IMS Connector for Java. We call these roll your own (RYO) clients. We provide two detailed examples, written in the C and Java programming languages. These examples are based on the IMS Connect sample code, which you can download from the corresponding Web page, available at:

<http://www.ibm.com/software/data/db2imstools/imstools/imsconnect.html>

The sample code we provide is meant to be used as a base to build your own application. Although we tried to make it useful, we do not pretend to cover all the functionality and possibilities you can obtain from IMS Connect. Refer to *IMS Connect Guide and Reference*, SC18-9287, to get the full details.

14.1 Basic structure of a simple IMS Connect client program

If you have any experience writing TCP/IP client programs, you will feel comfortable with the IMS Connect programming model. It works in a similar fashion to an HTTP client. Let us take a look at the logical flow of a simple IMS Connect client. This is, by the way, the flow used in the sample applications mentioned earlier.

A simple IMS Connect client has to:

1. Obtain a stream socket connection to the IMS Connect server.
2. Build a header structure containing the parameters of the interaction we want to do with IMS.
3. If needed, build a data structure that contains the transaction code and data we are sending to IMS.
4. Write the header and the message data to the socket.
5. Get the response back.
6. If needed, build and send a positive acknowledgement (ACK) or a negative acknowledgement (NAK) message to the server.
7. If needed, get the response from IMS Connect to the ACK or NAK.
8. Close the socket connection.

Figure 14-1 shows a high-level overview of an IMS Connect socket application.

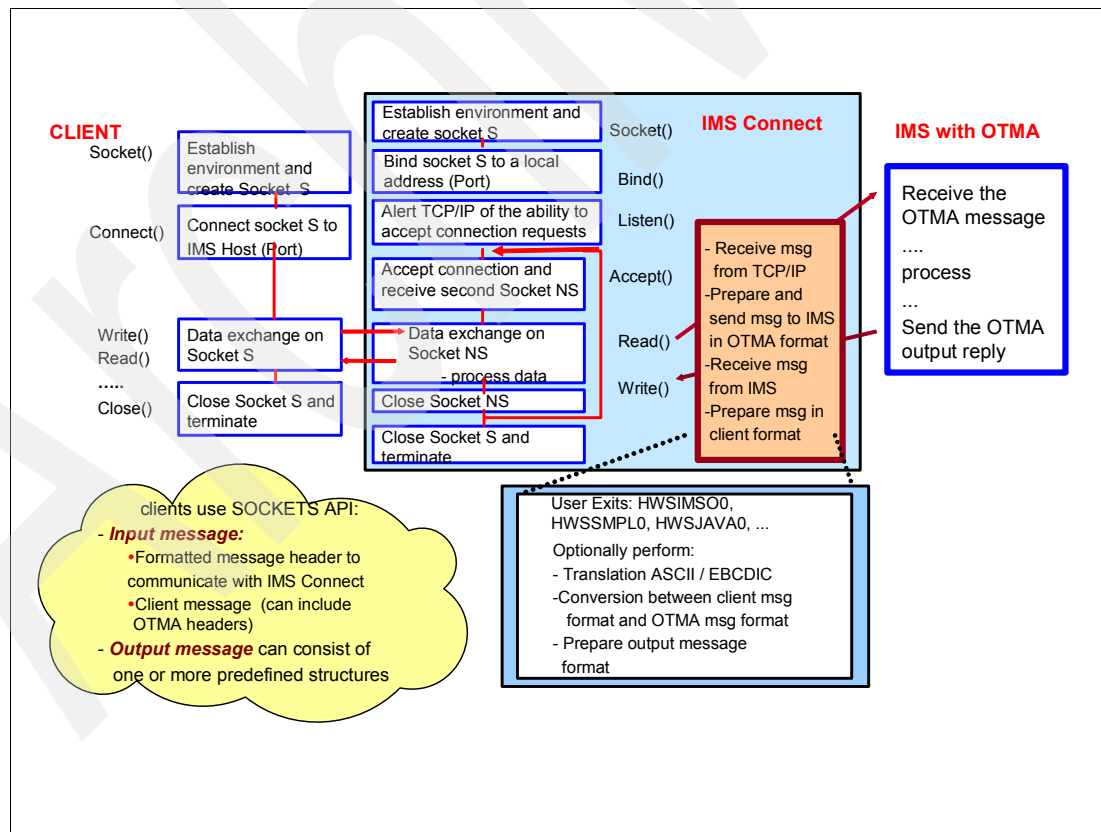


Figure 14-1 Overview of an IMS Connect sockets based application

As we will see later, this is the simplest way to talk to IMS Connect. Notice that this flow corresponds to a *non-connected* protocol, in which the socket connection lasts only long enough to complete one interaction. With IMS Connect, we can also use a *connected* protocol, where we keep the socket open during more than one interaction. The basics, however, are the same. The difference is that we would simply not close the socket after sending the ACK.

IMS Connect interaction types

There are three basic types of interaction between a client and IMS Connect:

- **Send-receive**

This is the usual message interchange, in which the client application sends a message to IMS Connect that contains a response mode IMS transaction. The client issues a `write()` or `send()` call to the IP socket to send the message to IMS Connect and immediately does a `read()` or `recv()` call to get the response. The message sent to IMS Connect contains the IMS Request Message (IRM) prefix and at least one transaction segment. We provide a full description of the IRM prefix in “IMS Request Message (IRM) prefix” on page 269.

- **Send-only**

In this case, we send IMS Connect a message that contains a non-response IMS transaction, so the client application issues the `send()` call but does not follow it with the corresponding `recv()` call. If the transaction sent does send any message through the IOPCB, the response can be discarded or placed on an asynchronous hold queue, depending on the combination of parameters used (commit mode, purge not delivered output, and so on). In this case, as in the previous one, the message we send to IMS Connect has the IRM prefix and at least one transaction segment.

- **RESUME TPIPE**

The term Tpipe comes from OTMA. Refer to Chapter 2, “Open Transaction Manager Access” on page 7.

This interaction type is used to recover the undelivered output queued in the asynchronous hold queue belonging to a Tpipe. Those messages can be created by:

- Messages written by an IMS application using an alternate PCB
- Normal transaction responses (written to the IOPCB) received by IMS Connect after the timeout period expired for the message that generated them
- Messages sent to the IOPCB by any transaction in a program-to-program switch chain when a previous transaction has already written to the IOPCB

This message consists of just the IRM prefix. The application issues a `read()` just after the `write()`, and it will get the first (oldest) undelivered asynchronous message waiting in the queue. There are several options to recover the rest of the messages, which we cover in “IMS Request Message (IRM) prefix” on page 269.

There are also a few other types of interactions, which are always related to one of the three we have just described:

- **ACK/NAK (acknowledgement/negative acknowledgement)**

Used in response to a message where synchronization level confirm was specified.

- **CANCEL TIMER**

Used to request the cancellation of the timer associated with the wait of data from IMS. See 7.2.6, “The CANCEL TIMER request” on page 97 for details.

- **DEALLOCATE**

Used to terminate a conversation rather than complete it.

The type of interaction we want is determined by the content of one of the IRM prefix fields, namely IRM_F4, which we cover in the next section.

14.2 IMS Connect message structures

The structure of an IMS Connect message is simple, as we just described. It is composed of a *message header* followed by the *message data*. We now describe what is inside those information blocks.

The first thing you need to know is that the header structures you use must correspond to what the IMS Connect user message exit expects to find. In this chapter, we assume that the sample exits provided with IMS Connect, HWSSMPL0 and HWSSMPL1, are in use. If your installation uses a modified exit, ask your systems programmer if the message header structures have been changed and modify the samples accordingly with those changes.

Note: IMS Connect expects to find the binary data in *big-endian* order, also called *network* order.

The big-endian order means that binary magnitudes longer than one byte must come with their most significant byte first. This is the same order used on the IBM @server zSeries, POWER™, PowerPC®, and SPARC machines, but it is just the opposite of the Intel® x86 convention. If you are using the C programming language, you can invoke the htons(), htonl(), ntohs(), and ntohl() to convert short (16-bit) and long (32-bit) magnitudes from host (native) to network format and back. If you are using the Java programming language, you are safe as long you use the writeShort() and writeLong() methods of the DataOutputStream class to write the data to the network connection.

We distinguish four types of messages:

- ▶ **IMS Request Message (IRM)**

These are the messages sent from the client to IMS Connect.

- ▶ **Request Status Message (RSM)**

This is the message returned by IMS Connect to the client when an error has occurred. The RSM contains a return code and a reason code identifying the type of status.

- ▶ **Complete Status Message (CSM)**

IMS Connect sends this message as the last structure of an output message if the input message was processed successfully.

- ▶ **Request Mod Message (RMM)**

IMS Connect returns the RMM as the first structure of an output message if the MFS MOD name is requested and there is output data present.

14.2.1 The IMS Connect input message

Just to clarify, we are using the words “input” and “output” relative to the IMS Connect side. Therefore, the input message is actually what the client application sends to IMS Connect, and the output message what IMS Connect sends back to the client application.

The structure of the input message is always the same:

```
LLLL IRM LLZZdata1 LLZZdata2 ... 0004 0000
```

That is:

- ▶ Four bytes containing the total size of the message
- ▶ The IRM structure, described later
- ▶ One or more optional data segments, each one beginning with a two-byte length and two flag bytes, which should be zero
- ▶ An end of message indication, consisting of the four bytes 00040000

IMS Request Message (IRM) prefix

The IRM prefix contains the information needed by IMS Connect to process the message. It establishes the type of message that we are sending and what options need to be applied to it. Table 14-1 describes the IRM prefix as expected by the sample exits, HWSSMPL0 and HWSSMPL1.

Table 14-1 IRM prefix structure for HWSSMPL0 and HWSSMPL1

Offset	Length (bytes)	Field	Meaning
0	4	IIII	Total length of the message. This total includes <i>these</i> four bytes.
4	2	IRM_LL	Message prefix length, including itself.
6	1	IRM_ARCH	Architectural level: <ul style="list-style-type: none"> ▶ X'00' Base support. ▶ X'01' Required space for IRM_REROUT_NM field. See "IRM_REROUT_NM: Reroute name" on page 274 for details about IRM_REROUT_NM.
7	1	IRM_F0	Reserved. Initialize to binary zeros.
8	8	IRM_ID	EXIT identifier. For HWSSMPL0, it should be "SAMPLE"
16	4	IRM_RES	Reserved. Must be zero.
20	1	IRM_F5	Input message type. See "IRM_F5: Input message type" on page 270.
21	1	IRM_TIMER	Time delay that IMS Connect will wait for IMS to return data before a timeout occurs. See "IRM_TIMER: Timeout control" on page 271.
22	1	IRM_SOCT	Socket connection type. Use the following values: <ul style="list-style-type: none"> ▶ X'00' Transaction socket. The socket connection lasts across a single transaction. ▶ X'10' Persistent socket. The socket connection lasts across multiple transactions. ▶ X'40' Non-persistent socket. The socket connection lasts for a single exchange consisting of an input and an output.
23	1	IRM_ES	Unicode encoding schema: <ul style="list-style-type: none"> ▶ X'01': UTF8 ▶ X'02': UCS2 ▶ X'02': UTF16 Set to binary zeros. See 14.3, "IMS Connect Unicode support" on page 275.

Offset	Length (bytes)	Field	Meaning
24	8	IRM_CLIENTID	Unique identification for each client. See “IRM_CLIENTID: clientID” on page 272 for details and warnings about this field.
32	1	IRM_F1	MFS MOD name request. Use X'80' to request IMS Connect to return a MOD name in the output message.
33	1	IRM_F2	Commit mode. Use X'40' for commit mode zero and X'20' for commit mode one.
34	1	IRM_F3	Sync level. Use the following values: <ul style="list-style-type: none"> ▶ X'00' for NONE ▶ X'01' for CONFIRM ▶ X'02' for SYNCPT This field is also used to set the following options: <ul style="list-style-type: none"> ▶ X'04' for purge not deliverable ▶ X'08' for reroute request
35	1	IRM_F4	Message type. See description below.
36	8	IRM_TRNCOD	IMS transaction code we want to execute.
44	8	IRM_IMSDESTID	Name of the datastore, as defined in the IMS Connect configuration member.
52	8	IRM_LTERM	IMS LTERM override. See “IRM_LTERM: Logical terminal override” on page 273.
60	8	IRM_RACF_USERID	RACF user ID. Optional if IRM_ARCH = X'00'.
68	8	IRM_RACF_GRPNAME	RACF group name. Optional if IRM_ARCH = X'00'.
76	8	IRM_RACF_PW	RACF PassTicket or password. Optional if IRM_ARCH = X'00'.
84	8	IRM_APPL_NM	RACF APPL name, defined to RACF on the PKTDATA definition. This field is optional when IRM_ARCH = X'00'.
92	8	IRM_REROUT_NM	Reroute name for the client reroute request. See “IRM_REROUT_NM: Reroute name” on page 274. Optional if IRM_ARCH = X'00'.

IRM_F5: Input message type

This field is actually a bit map that we use to tell IMS Connect about some processing options that we want it to apply to our message. Table 14-2 shows the available options.

Table 14-2 Values for IRM_F5

Value	Meaning
X'80'	The client builds the OTMA headers. We usually will not do that in a user-written client application, but IMS Connector for Java does.
X'40'	Translation (ASCII/EBCDIC) done by client. Specify this if you are running on an EBCDIC machine, or if you want to use a nonstandard translation table.

Value	Meaning
X'10'	Single message with wait option. Applies only to RESUME TPIPE interactions. If no message is present in the asynchronous hold output queue, IMS Connect waits for the timeout value specified in IRM_TIMER for a new message to arrive and then sends it to the client.
X'00'	No option flow for messages. See the explanation for the X'04' value to understand what this means. This is the default.
X'01'	Single message. Applies only to RESUME TPIPE interactions. This means that we only do one read() on the communication socket after issuing the write() with the RESUME TPIPE request.
X'02'	Auto flow of messages. Applies only to RESUME TPIPE interactions. This means that the client application enters a loop of read() calls, each one of which will return a message present at the asynchronous hold queue. The read() issued after getting the last message waits for the next message according to the IRM_TIMER setting. Use this only if the client is a dedicated output client.
X'04'	No auto flow of messages. This means that the client application enters a loop of read() calls, each one of which will return a message present at the asynchronous hold queue. Use this only if the client is a dedicated output client.

The difference between AUTO and NO AUTO is related to the messages that might arrive during the message retrieval (between the RESUME TPIPE sent to IMS Connect and the retrieval of the last message present at that moment). If you are using the AUTO option, those new messages will be delivered to your client. If you use the NO AUTO option, IMS Connect will not deliver any new messages. For that reason, specify a short timeout value if you are using NO AUTO, and a long timeout value for AUTO. It does not make sense to wait with NO AUTO, while it makes sense to do it for AUTO.

IRM_TIMER: Timeout control

This field contains an encoded byte that specifies how much time IMS Connect will wait for IMS to return data after a write() to the socket. The following functions support the IRM_TIMER settings:

- ▶ TCP/IP SEND of a RESUME TPIPE
- ▶ TCP/IP SEND of an ACK or a NAK
- ▶ TCP/IP SEND of data
- ▶ PC SEND of a RESUME TPIPE
- ▶ PC SEND of an ACK or a NAK
- ▶ PC SEND of data

The actual timeout period depends on the specific value you have set, the default value specified in the IMS Connect configuration member, and an internal IMS Connect default:

- ▶ If the IRM_TIMER is set to X'00', the following defaults are used:
 - The default for all RESUME_TPIPE is two seconds.
 - The default for all RESUME_TPIPE non-single ACK is 0.25 seconds.
 - The TIMER setting in the configuration member is used for all other cases.
- ▶ The X'FF' value means WAIT FOREVER. It is intended to support the AUTO option of the asynchronous output function (RESUME TPIPE).
- ▶ The X'E9' (character 'Z') value means NO WAIT. Nevertheless, IMS Connect only honors the NO WAIT request on:
 - SENDONLY transactions
 - ACKs or NAKs associated with a RESUME_TPIPE with asynchronous option SINGLE

In any other case, IMS Connect enforces the NO WAIT option as follows:

- There is a two second delay for:
 - RESUME TPIPE requests
 - Conversational transactions (code and data)
 - ACKs or NAKs associated with conversational transactions
 - Non-conversational transactions (code and data)
- There is a 0.25 second delay for:
 - ACKs and NAKs associated with non-conversational transactions commit mode 1 confirms
 - ACKs and NAKs associated with non-conversational transactions commit mode 0 confirms
 - ACKs and NAKs associated with RESUME TPIPEs with asynchronous options AUTO or NOAUTO

Be careful about the use of the X'E9' value. Its misuse can result in several problems, including socket disconnections, losing messages, and hang conditions due to IMS Connect, the client, and OTMA being in different states, all of them waiting for input.

- The rest of IRM_TIMER values correspond to specific timeout settings, according to Table 14-3.

Table 14-3 Specific IRM_TIMER values

IRM_TIMER value	Wait value
X'01' through X'19'	Range from 0.01 to 0.25 second, 0.01 increments
X'19' through X'28'	Range from 0.25 to 1 second, 0.05 increments
X'28' through X'63'	Range from 1 to 60 seconds, 1 second increments
X'63' through X'9E'	Range from 1 to 70 minutes, 1 minute increments

IRM_CLIENTID: clientID

The clientID identifies one particular instance of work inside IMS Connect and OTMA. If you are using commit mode 0, it is also used to build the name of the Tpipe that IMS uses for the input and output messages.

Important: It is your responsibility to make sure that in each moment a clientID is used only once. This applies both to the SEND type interactions and the RESUME TPIPE ones. If you use a clientID that is currently “alive” in IMS Connect, you get an error message (an RSM with a decimal value of 56, DUPECLNT, in its reason code field).

IRM_F4: Message type

The content of this field defines what kind of interaction we have between our client and IMS Connect. Table 14-4 on page 273 contains the possible values.

Table 14-4 Message and interaction types

Value	Meaning
' ' (blank)	SEND. Transaction code and data (for conversational and non-conversational response mode transactions) or just data (for conversational transactions) follows the IRM prefix.
'S'	SENDONLY. Transaction code and possibility data for a non-response mode transaction follows.
'A'	ACK: Positive acknowledgement. This is used in response to a message sent to the client when sync level=confirm was specified in the input message. The ACK tells IMS Connect that the message was successfully received, so the transaction can be committed (when using commit mode 1), or the output message can be dequeued (when using commit mode 0).
'N'	NAK: Negative acknowledgement. Same as ACK. In this case, we tell IMS Connect that we were unable to process the output message. If we are using commit mode 1, IMS issues an abend 119 and rolls back the transaction. If we use commit mode 0, the output message is placed in the asynchronous hold queue associated with the used Tpipe so that it can be recovered using a RESUME TPIPE.
'D'	DEALLOCATE. Use this value to terminate a conversation rather than a complete transaction.
'R'	RESUME TPIPE. Use this value to request asynchronous output data from IMS. The commit mode <i>must</i> be zero. IMS Connect delivers one or more messages enqueued in the asynchronous hold queue associated with the Tpipe corresponding to the specified IRM_CLIENTID. The quantity of messages delivered depends on the value of IRM_F5.
'C'	CANCEL TIMER. Use this value to request to cancel the timer associated with the wait of data from IMS.

IRM_LTERM: Logical terminal override

This is a character string value that contains the IMS LTERM override. You can set it to a valid value or to blanks.

If you specify a non-blank value, the HWSSMPL* exit uses it as the LTERM name passed to OTMA. OTMA then places this value in the IOPCB LTERM name, so it is seen by the IMS host application as the LTERM name. If you fill this field with blanks, OTMA uses the Tpipe name as LTERM name, with the values:

- ▶ The ClientId *(IRM_CLIENTID) if you are using commit mode 0.
- ▶ The port number (expressed in characters) if you are using commit mode 1.

If you specify your own value in IRM_LTERM, it must follow the rules governing valid IMS LTERM names.

Note: There are security issues regarding IRM_LTERM. If your system setup or your applications restrict the access to transactions or to options within the transactions in base of the LTERM name, you must be sure to control who and what can use a specific IRM_LTERM value. One place you can do that is in the user message exits. See Chapter 9, “IMS Connect user exit support” on page 115 for more details about the user message exits.

IRM_REROUT_NM: Reroute name

This field is intended to contain an 8-character alternate Tpipe name, which is used if you specify X'08' in IRM_F3. To use this field, you must also set IRM_ARCH to X'01'. If you specify X'08' in IRM_F3, the undelivered asynchronous option goes to:

- ▶ The Tpipe you specify in IRM_REROUT_NM
- ▶ The Tpipe you specify for the RRNAME parameter in the IMS Connect configuration member
- ▶ The default value, HWS\$DEF, if you did not specify either

14.2.2 The IMS Connect output message

The structure of the output message (remember, output means information sent by IMS Connect to the client) depends on whether we requested a MOD name to be present (setting IRM_F1 to X'80') and the outcome (success or failure) of the interaction. We also have to take into account that if we use the HWSSMPL1 exit, IMS Connect puts before the whole output message a four-byte field with the total length, including those 4 bytes:

```
[LLLL] RMM LLZZdata1 LLZZdata2 ... CSM
[LLLL] LLZZdata1 LLZZdata2 ... CSM
[LLLL] RSM
```

These three formats correspond to:

- ▶ A successful interaction, with MOD NAME requested
- ▶ A successful interaction, without MOD NAME requested
- ▶ A non-successful interaction

Request Status Message (RSM)

The Request Status Message (RSM) contains the status and reason codes set by IMS Connect when an interaction does not complete successfully. Table 14-5 describes the structure of this block.

Table 14-5 Request Status Message (RSM)

Offset	Length (bytes)	Field	Meaning
0	2	LL	Length of RSM
2	1	RSM_FLG1	Flag byte one: <ul style="list-style-type: none">▶ X'80' Asynchronous message queued in IMS▶ X'40' Conversational output message▶ X'20' ACK/NAK required
3	1	Reserved	Reserved (set to binary zeros)
4	8	IRM_ID	Char value of *REQSTS*
12	4	RSM_RETCOD	Return code
16	4	RSM_RSNCOD	Reason code

Complete Status Message (CSM)

The Complete Status Message (CSM) structure appears at the end of the message IMS Connect sends to the client when an interaction has completed successfully. Table 14-6 on page 275 describes the structure.

Table 14-6 Complete Status Message (CSM)

Offset	Length (bytes)	Field	Meaning
0	2	LL	Length of CSM
2	1	CSM_FLG1	Flag byte one: <ul style="list-style-type: none"> ► X'80': asynchronous message queued in IMS. ► X'40': conversational output message ► X'20': ACK/NAK required
3	1	Reserved	Reserved (set to binary zeros)
4	8	CSM_ID	Char value of *CSMOKY*

Request Mod Message (RMM)

The Request Mod Message (RMM) structure appears at the beginning of the message IMS Connect sends to the client when the interaction has completed successfully and a MOD name was requested. Table 14-7 describes this structure.

Table 14-7 Request Mod Message (RMM)

Offset	Length (bytes)	Field	Meaning
0	2	LL	Length of RMM
2	2	ZZ	Reserved (set to binary zeros)
4	8	ID	Char value of *REQMOD*
12	8	MOD	Char value of the requested MFS MOD name

14.3 IMS Connect Unicode support

Unicode is a character encoding, established by the Unicode Consortium, that provides a unique number for every character without depending on the platform, application, or language. The Unicode Standard defines three encoding forms that allow the same data to be transmitted in a byte, word, or double-word-oriented format (that is, in 8-, 16-, or 32-bits per code unit). All three encoding forms encode the same common character repertoire and can be efficiently transformed into one another without a loss of data. The Unicode Standard has been widely adopted by industry leaders and it is used by modern standards such as XML and Java. For more information about Unicode, the Unicode Consortium, and the different encoding schemas, see:

<http://www.unicode.org/>

IMS Connect provides Unicode support in the form of allowing users to selectively send Unicode data. The following areas are *excluded* from containing Unicode data:

- LLLL
- IRM
- LLZZ (all data length fields)
- OTMA HEADERS
- RMM
- CSM
- RSM

Only the data portion of the message can be of Unicode format, but the transaction code can be ASCII, EBCDIC, or Unicode. (It depends on your IRM header setting.)

IMS Connect Unicode support includes following Unicode encoding schemas:

- ▶ UTF-8
- ▶ UTF-16
- ▶ UCS-2

IMS Connect supports the following language (script) groups:

- ▶ Group 1 (Western Europe and United States)
- ▶ Group 2 (Central Europe)
- ▶ Group 3 (Baltic)

Important: The Unicode support is based on the cooperation and understanding between the client application and the IMS host application (such as a Java application running in an IMS dependent region). Also, it is based on an understanding that both applications can deal with Unicode data and that the two have agreements about the supported Unicode encoding schema and the structure and content of the message being sent and received.

Currently, IMS Connect supports ASCII and EBCDIC data streams from and to the client. The input to IMS Connect from the client is translated to EBCDIC if ASCII data is received; if the input from the client is EBCDIC, no translation takes place. Output from IMS Connect to the client is translated from EBCDIC to ASCII if ASCII data is received. This support allows, in addition to ASCII and EBCDIC data to be received from the IMS Client by IMS Connect and sent by IMS Connect to the client, the receipt of Unicode (UTF-8, UTF-16, or UCS-2) data.

There is not a transformation of output to the client by IMS Connect for Unicode messages received from IMS. All IMS error messages, such as DFS555 and DFS0064, are sent as EBCDIC or ASCII, based on the code type specified by the IRM_MSGID content of the IRM received from the client. As an example, the IRM_MSGID of *SAMPLE*, which means HWSSMPL0, will either be ASCII or EBCDIC, and that setting will determine the code type of the IRM and OTMA header.

14.3.1 Transaction code translation

The IMS Connect user message exits will translate the IMS transaction code from Unicode to EBCDIC when the client sends the transaction code as Unicode. A valid IMS transaction code can be constructed from the following conditions:

- ▶ A through Z (uppercase only).
- ▶ Special characters #, \$, @, and 0 to 9.
- ▶ It must begin with an alphabetic character.

It is assumed that any IMS application that supports Unicode is a new IMS application that has the capability to process Unicode data and deal with an 8-byte transaction code.

IMS Connect transforms the IMS transaction code to EBCDIC if it is Unicode. The client application can send an EBCDIC, ASCII, or Unicode IMS transaction code, but the transaction code must be an 8-byte transaction code field (left justified and padded with blanks), followed by Unicode data, in which case the IMS transaction code will only be translated to EBCDIC, if sent as ASCII and if the Unicode data is sent to IMS.

If the transaction code is sent as Unicode, the transaction code will be translated to EBCDIC. The host application must define an 8-byte field to contain the transaction code. If the installation wants to have a blank following the 8-byte transaction code field, it will be delivered to the host application as a Unicode blank, not an EBCDIC blank.

14.3.2 Output message including Unicode data from IMS Connect

The data section of output message is not transformed for those outputs that are a result of Unicode data received from the client. If Unicode data is sent to IMS Connect, the related output is treated as Unicode. For RESUME TPIPE requests, the client must specify in the IRM if the output is to be treated as Unicode or non-Unicode. For message switching, it is the IMS host application's responsibility to ensure that the output message is formatted using the correct Unicode encoding schema or EBCDIC for the destination. All IMS error messages (for example, DFS555I) are sent as either ASCII or EBCDIC. The client application uses the IRM_MSGID field of the IRM to tell IMS which type to send. IMS Connect does not transform messages to Unicode. For example, if IRM_MSGID is specified as EBCDIC, the IMS error message (DFSnnnn) is sent as EBCDIC; if IRM_MSGID is specified as ASCII, the IMS error message (DFSnnnn) is translated from EBCDIC to ASCII by the user exit.

14.3.3 Message structures for Unicode support

Table 14-8 shows new fields and flags in the IRM header have been defined for Unicode support.

Table 14-8 New fields and flags in IRM header for Unicode support

Field	Value	Meaning
IRM_F1 field (has new flags).		
IRM_F1_UC	X'20'	Unicode message
IRM_F1_UCTC	X'10'	Unicode transaction code
IRM_ES field (This is a new field. Previous name was IRM_RSV02, reserved field.)		
IRM_ES_UTF8	X'01'	UTF8 encoding schema
IRM_ES_UCS2	X'02'	UCS2 encoding schema
IRM_ES_UTF16	X'02'	UTF16 encoding schema

For Unicode data, the input message structure sent by the client must adhere to the structure shown in Table 14-9.

Table 14-9 The input message structure for Unicode support

Message structure contents	Consideration
LLLL (full word total length)	It must be binary.
IRM header	It cannot be Unicode. It must be: <ul style="list-style-type: none"> ► The binary data for defined binary fields. ► The data fields defined as character fields and can be either ASCII or EBCDIC.
OTMA headers	If your own user exit provides OTMA header, it cannot be Unicode. It must be: <ul style="list-style-type: none"> ► The binary data for defined binary fields. ► The data fields defined as character fields and can be either ASCII or EBCDIC.

Message structure contents	Consideration
LL (halfword message data length)	It must be binary.
ZZ (halfword reserved field)	It must be binary zeros.
IMS transaction code	It can be either: <ul style="list-style-type: none"> ► Unicode and must be defined as an 8-character transaction code field, left aligned, and padded with blanks. ► ASCII and must be defined as an 8-character transaction code field, left aligned, and padded with blanks. ► EBCDIC and can be defined as an 8-character transaction code field, left aligned, and padded with blanks.
User Data	Unicode data for the supported Unicode encoded schema.
LL (halfword message data length)	It must be binary.
ZZ (halfword reserved field)	It must be binary zeros.
User Data	Unicode data for the supported Unicode encoded schema.

For Unicode data, the output message structure sent by the client must adhere to the structure shown in Table 14-10.

Table 14-10 The output message structure for Unicode support

Message structure contents	Consideration
RMM	It cannot be Unicode. It must be: <ul style="list-style-type: none"> ► The binary data for defined binary fields. ► The data fields defined as character fields and can be either ASCII or EBCDIC.
LLLL (full word total length)	It must be binary.
LL (halfword message data length)	It must be binary.
ZZ (halfword reserved field)	It must be binary zeros.
User Data	Unicode data for the supported Unicode encoded schema.
LL (halfword message data length)	It must be binary.
ZZ (halfword reserved field)	It must be binary zeros.
User Data	Unicode data for the supported Unicode encoded schema.
CSM	It cannot be Unicode. It must be: <ul style="list-style-type: none"> ► The binary data for defined binary fields. ► The data fields defined as character fields and can be either ASCII or EBCDIC.
RSM	It cannot be Unicode. It must be: <ul style="list-style-type: none"> ► The binary data for defined binary fields. ► The data fields defined as character fields and can be either ASCII or EBCDIC.

14.4 Complete pseudocode samples

In this section, we provide you complete pseudocode for several useful execution flows. In 14.5, “Detailed code examples” on page 282, we provide two implementations of these programming models in the C and Java programming languages.

14.4.1 Commit mode 1 send-receive programming

In this sample, we assume that you are interested in writing a program to send one transaction to an IMS system using IMS Connect, under the commit mode 1 (send-then-commit) protocol, acknowledging the response (synchronization level confirm), and using a transaction socket.

This program model corresponds to the simplified example presented at the beginning of the chapter. Example 14-1 shows the pseudocode to implement this model.

Example 14-1 Pseudocode for a commit mode 1, sync level confirm, transactional socket

```
PROGRAM send_receive_cm1(host, port, datastore, transaction_data)
    socket := open_socket(host, port)
    IRM := build_irm(commit_mode=1, synch_level=CONFIRM, data_store=datastore,
                    type=SEND_RECEIVE, transaction=transaction_data);
    send(socket,IRM);
    response := receive(socket);
    if (response is a RSM)
        print_error_details(response.RSM_data);
    else
        if (response contains a CSM)
            process_response(response.segment_data);
            if (response.ack_needed)
                IRM := build_irm(commit_mode=1, synch_level=CONFIRM, data_store=datastore,
                                type=ACK, transaction=NULL);
                send(socket,IRM);
                response := receive(socket);
                if (response.RSM IS NOT "deallocate commit")
                    print_error_details(response.RSM_data);
                    <prepare abnormal termination>
                else
                    <everything OK, prepare normal termination>
                endif
            endif
        else
            print_error_details(response.RSM_data);
            <prepare abnormal termination>
        endif
    endif
    close(socket);
END send_receive_cm1;
```

The following numbers correspond to the numbers in Example 14-1:

1. To communicate to IMS Connect, we need to create and open a TCP socket bound to the host and port where IMS Connect is listening.
2. Next, we prepare and send the input message structure, according to 14.2.1, “The IMS Connect input message” on page 268.
3. Now, we read the response from the socket. IMS Connect waits for OTMA up to the maximum time we specified in the IRM_TIMER field of the IRM prefix. We can also use the TCP/IP timeout features to avoid blocking too much time.

4. If the interaction ended normally, IMS Connect sends us a CSM structure; otherwise, we receive an RSM. If we receive an RSM, we will probably want to notify the user with an appropriate error message.
5. If we receive a CSM, we can process the output message from IMS. We also check if we need to send an ACK, looking at the corresponding bit flag, according to 14.2.2, “The IMS Connect output message” on page 274. In this sample, we use sync level=confirm, so we always need to ACK a correct response. However, it is a good programming practice to check the flag, so if in the future we modify the program to use another sync level, it will not stop working.
6. We have to read the IMS Connect response to our ACK. IMS Connect answers with an RSM that we must examine to know if the transaction was committed successfully. In that case, the return code is X'04' and the reason is X'61' (decimal 97, “deallocate commit”). If the commit was not successful, the reason code is X'62' (decimal 98, “deallocate abort”).
7. Because we are using a transaction socket, we close the communication with IMS Connect. Actually, at this point, IMS Connect has already closed its side of the socket.

14.4.2 Commit mode 0 send-receive programming

Now, we take a look at a program to use commit mode 0 and a permanent socket. This program has to take into account the possibility of IMS sending more than one response message to a transaction. The sync level we use is imposed by the commit mode 0 protocol and has to be *confirm*.

Example 14-2 shows the pseudocode for a interactive program that prompts for transactions until the user signals it to end the process.

Example 14-2 Pseudocode for a commit mode 0 interaction, permanent socket

```

PROGRAM send_receive_cm0(host,port,datastore)
  socket := open_socket(host, port)                                1.
  prompt_user(transaction_data, end_flag);                          2.
  while (not end_flag)
    IRM := build_irm(commit_mode=0, synch_level=CONFIRM, data_store=datastore,
                     type=SEND_RECEIVE, transaction=transaction_data);
    send(socket,IRM);
    response := receive(socket);                                    3.
    while(response is NOT a RSM)                                     4.
      display_response(response.segments);
      if (response.ack_needed)                                       5.
        IRM := build_irm(commit_mode=0, synch_level=CONFIRM, data_store=datastore,
                         type=ACK, transaction=NULL);
        send(socket,IRM);
      endif
      response := receive(socket);                                   6.
    endwhile
    if (response.RSM is not “timeout expired”);                     7.
      print_error_details(response.RSM_data);
    endif
    prompt_user(transaction_data, end_flag);
  endwhile
  close(socket);                                                    8.
END send_receive_cm0;

```

The following numbers correspond to the numbers in Example 14-2:

1. To communicate to IMS Connect, we need to create and open a TCP socket bound to the host and port where IMS Connect is listening.

2. Next, enter a loop to send the messages to IMS Connect. We prompt the user for a transaction, prepare the IRM structure according to 14.2.1, “The IMS Connect input message” on page 268, and send the IRM to IMS Connect.
3. Now, we read the response from the socket. IMS Connect waits for OTMA up to the maximum time we specified in the IRM_TIMER field of the IRM prefix. We can also use the TCP/IP timeout features to avoid blocking too much time.
4. IMS Connect tries to send us all the output generated by the sent transaction and any other transaction chained by it using program-to-program switch, so we must loop reading responses until IMS Connect notifies us that there are no more using an RSM block.
5. Because we are using commit mode zero, we *must* ACK each response to tell IMS Connect to dequeue the messages from the output queue, but it is a good programming practice to check the corresponding bit in the CSM structure.
6. We try to get the next response from IMS Connect and loop again.
7. When there are no more messages to get or an error has occurred, IMS Connect sends us an RSM, which signals the end of the loop. If the reason for sending the RSM was simply the end of pending messages, the RSM contains a return code of X'20' or X'24', meaning that a timeout has occurred. If the return code is not X'20' or X'24', we have to deal with an error condition and notify the user accordingly.
8. Because we are using a permanent socket, we do not close until the main loop has ended.

14.4.3 Commit mode 0 RESUME TPIPE programming

In this example, we show the pseudocode for a dedicated asynchronous message reader client. This program is intended to be run as a started task, in z/OS terminology, or a daemon, in UNIX terms, continuously waiting for asynchronous messages in a specific Tpipe.

This kind of a client can *only* be used to recover messages sent to a dedicated Tpipe by means of ALTPCB messages or rerouted non-delivered messages. See Chapter 13, “IMS Connector for Java rerouting and timeout support” on page 257 for an explanation about the different types of asynchronous messages. It *must not* be used to recover asynchronous output sent to ordinary clients. The reason is that this program will be continuously waiting for messages. This means that the clientID for which it is waiting will not be usable to send transactions. IMS Connect will refuse to send a transaction to that clientID and will issue a DUPECLNT error.

Example 14-3 shows the pseudocode for a dedicated asynchronous message processor. This program would continue to run until IMS reports an error, or the program is cancelled or killed. If you plan to build this type of client, you should provide a way to shut down the program in an ordered way. In a UNIX environment, that can be done using a signal trap. In an z/OS environment, you can use the console interface.

Example 14-3 Pseudocode for a dedicated asynchronous message processor

```

PROGRAM process_asynchronous(host,port,datastore,clientid);
    socket := open_socket(host,port);                                1.
    IRM := build_irm(commit_mode=0, synch_level=CONFIRM, data_store=datastore,
                                type=SEND_RECEIVE, transaction=NULL, client=clientid,
                                flow=AUTO, timeout=x'FF');          2.
    send(socket,IRM);
    response := receive(socket);                                     3.
    while (response is NOT a RSM);
        process_response(response.segments);
        if (response.ack_needed);                                   4.
            IRM := build_irm(commit_mode=0, synch_level=CONFIRM, data_store=datastore,
                                type=ACK, transaction=NULL, timeout=x'FF');
```

```

        send(socket,IRM);
    endif;
    response := receive(socket);
endwhile;
close(socket);
END process_asynchronous;

```

5.

The following numbers correspond to the numbers in Example 14-3 on page 281:

1. To communicate to IMS Connect, we need to create and open a TCP socket bound to the host and port where IMS Connect is listening.
2. Next, we prepare and send an IRM to IMS Connect, specifying that we want to retrieve asynchronous output using the AUTO flow control and a X'FF' or "wait forever" IRM_TIMER. This combination of values tells IMS Connect to send every message present in the asynchronous hold queue in the moment that we send the IRM and to wait forever for further messages. See 14.2.1, "The IMS Connect input message" on page 268 for details about the IRM structure.
3. Now, we read the first asynchronous message from the socket. If there is no such message waiting to be retrieved, IMS Connect waits until there is one. This wait proceeds forever, due to the AUTO flow control we specified in the IRM. Afterward, we enter a loop until IMS Connect sends us an RSM, indicating that a error has been detected.
4. Because we are using commit mode zero, we *must* ACK each response to tell IMS Connect to dequeue the messages from the output queue, but it is a good programming practice to check the corresponding bit in the CSM structure.
5. We try to get the next response from IMS Connect and loop again.

14.5 Detailed code examples

After providing the basics about the execution flow of an IMS Connect client, we take a more deeper look at two code examples. The first code sample is written in the C programming language and should compile under any reasonably standard environment. The second code sample is written in the Java language and should compile and run under a JVM™ with a version equal to or later than 1.4.2.

We wrote both samples trying to make them functionally equivalent. Both are invoked the same way, and both can send any arbitrary transaction to an IMS Connect host or send a RESUME TPIPE command:

```

sample -h hostName -p portNumber [-d dataStoreName] [-u userId [-g groupName]
    [-w password]] [-c clientid] [-t ltermName] [-o timer ]
    [-x] [-y [reRouteName]]
    [-m {0|1}] [-l {N|C|S}] [-s] [-n] transcode transdata...
sample -h hostName -p portNumber [-d dataStoreName] [-u userId [-g groupName]
    [-w password]] [-o timer] -c clientId -r [-n]

```

The **-s** switch makes the interaction SEND ONLY. The **-m** switch enables you to select commit mode 0 or commit mode 1, and the **-l** switch lets you specify the sync level (none, confirm, synch). The **-n** switch forces a NAK, unless the sync level is set to none. You can specify a value for the IRM_TIMER field using the **-o** flag and its hexadecimal, encoded value as specified in "IRM_TIMER: Timeout control" on page 271. The **-x** flag enables the purge asynchronous output feature, and the **-y** flag allows you to ask the asynchronous output to be rerouted and, optionally, to specify the reroute name.

Of course, you must add the **java** command at the beginning of the line and capitalize the name of the program (Sample instead of sample) when you execute the Java version of the program.

14.5.1 C example

The complete listing of the source code is in Example A-1 on page 470. In this section, we review some excerpts of the code that are relevant to this chapter.

Compiling and linking the C sample program

This program should compile, link, and run in any reasonable POSIX-compliant system. That includes Linux, IBM AIX 5L, the Berkeley Software Distribution (BSD)-based systems, and Mac OS X. It will *not* run as is in the UNIX System Services environment of z/OS: The HWSSMPL0 will expect to find the data coded in ASCII, and UNIX System Services is EBCDIC based.

If you want to run the sample under Microsoft Windows, will need a POSIX layer emulator such as Cygwin. You can download Cygwin from the following Web site:

<http://www.cygwin.com>

Note: Cygwin is licensed under an free software license (GNU General Public License, or GPL). This license contains provisions that govern its use and distribution in a commercial environment. Carefully check those conditions before using or deploying Cygwin.

To compile the sample program, place it into a directory of your choice, change (**cd**) to that directory and issue the following command:

```
cc -o client client.c
```

Change **cc** to the name of the C compiler in your system. For example, if you are using Linux, you must type:

```
gcc -o client client.c
```

That should leave you with a client executable, which can be run using the syntax described earlier in this section, 14.5, “Detailed code examples” on page 282. Remember to add the current directory prefix to the command if your execution PATH does not include it (**./client**).

Structure of the program

This program shows the general structure presented in 14.1, “Basic structure of a simple IMS Connect client program” on page 266. Lets see which part of the code corresponds to each step described there.

Obtain a stream socket connection to the IMS Connect server

This is done in the function `sample_connect`, as shown in Example 14-4.

Example 14-4 Opening a socket connection to IMS Connect in C

```
int sample_connect(struct sampletran *sample) {
    int sockfd;
    struct hostent * host;
    struct sockaddr_in socketAddress;

    /* get host info */
    if ((host = gethostbyname(sample->hostName)) == NULL) {
        perror("gethostbyname"); (A)
    }
}
```

1.

```

        exit(1);
    }

    /* initialize the socket descriptor */
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    /* set some socket address values */
    socketAddress.sin_family = AF_INET;
    socketAddress.sin_port = htons(sample->portNumber); /* network byte order */
    socketAddress.sin_addr = *((struct in_addr *)host->h_addr);
    memset(&(socketAddress.sin_zero), 0, 8); /* zero out the rest of the struct */

    /* connect the socket */
    if (connect(sockfd, (struct sockaddr *)&socketAddress, sizeof(struct sockaddr)) == -1)
    {
        perror("connect");
        exit(1);
    }

    /* return the socket descriptor */
    return sockfd;
}

```

This function is fairly standard, and its structure will be recognized by anyone who has written a TCP/IP client program. The following numbers correspond to the numbers in Example 14-4 on page 283:

1. We translate the host name to a binary IP address using the `gethostbyname()` function.
2. We prepare to create a STREAM socket using the IP address we just obtained and the port number specified in the call.
3. Finally, we open a connection with the IMS Connect server and get a socket descriptor.

Preparing and sending the message to IMS Connect

The function `sample_send()` prepares the whole message in a dynamically allocated buffer and sends it to the server in a single `send()` call. If you use multiple calls to send the data, you have to be aware that the z/OS TCP/IP stack is by default configured to wait for 200 milliseconds before sending back each ACK. In that case, we would incur in a great performance penalty doing several `send()` calls. The best way to prevent this is to use a single `send()`. If you must do multiple `send()` calls, be sure that the TCP/IP *NODELAYACK* option is in use.

Example 14-5 shows the source code for the `sample_send()` function. This function can send IMS Connect any kind of interaction, based on the value of the `msgType` parameter. In the sample, code we are only sending ' ' (SEND), 'A' (ACK), 'S' (SEND ONLY), and 'R' (RESUME TPIPE) interactions.

Example 14-5 Preparing and sending a message to IMS Connect in C

```

/*
 * Sends the input data to the host
 *
 * Parameters:
 * sockfd    Connected socket descriptor
 * sample    Address of a sampletran structure with the interaction data
 * msgType   Value of IRM_F4 (' ', 'A', 'S', 'R')

```

```

*
* Notice: This code has been modified to send all the data to IMS Connect in a
* single write to enhance performance. The old IMS Connect sample does a write for
* each field. YOU SHOULD NOT DO THAT. Build first the whole message in memory and
* send it in one call.
*/

void sample_send(int sockfd, struct sampletran *sample, char msgType) {
    int totalLength, totalLengthBE;
    short segmentLength;
    short prefixLength = PREFIX_LENGTH;
    char irm_f3 = (char) 0;
    char irm_arch = (char) 0x01;    /* Arch level 1 to use Reroute Name */
    char *message = NULL;
    char *currPtr = NULL;
    int zero = 0; /* need this so we can pass a pointer to zero */

    /* +4 for first LL, ZZ and final LL, ZZ */
    totalLength = 4 + PREFIX_LENGTH + 4; 1.

    /* add in segment length, if segment is defined */
    if (sample->tranText != NULL) {
        totalLength += strlen(sample->tranText) + 12; /* +12 for LL, ZZ, tranCode */
    }

    /* Compute the IRM_F3 value */
    irm_f3 = sample->syncLevel; 2.
    if (sample->purge) {
        irm_f3 |= SL_PURGE;
    }
    if (sample->reroute) {
        irm_f3 |= SL_REROUTE;
    }

    message = malloc(totalLength); 3.
    if (message == NULL) {
        perror("Could not allocate memory for the whole message.");
        exit(32);
    }
    memset(message, 0, totalLength); /* Clean up the new allocated space */
    currPtr = message; /* Current write pointer set to beginning of message space */

    /* convert lengths to big endian */
    totalLengthBE = htonl(totalLength);
    prefixLength = htons(prefixLength);

    /* Build the message structure in the allocated buffer */ 4.
    /* Build the IRM prefix first */
    currPtr = addBuffer(currPtr, &totalLengthBE, 4); /* Total message length */
    currPtr = addBuffer(currPtr, &prefixLength, 2); /* IRM_LL */
    currPtr = addBuffer(currPtr, &irm_arch, 1); /* IRM_ARCH */
    currPtr = addBuffer(currPtr, &zero, 1); /* IRM_F0 */
    currPtr = addBuffer(currPtr, sample->exitID, 8); /* IRM_ID */
    currPtr = addBuffer(currPtr, &zero, 4); /* IRM_RES */
    currPtr = addBuffer(currPtr, &zero, 1); /* IRM_F5 - No option flow */
    currPtr = addBuffer(currPtr, &sample->timer, 1); /* IRM_TIMER */
    currPtr = addBuffer(currPtr, &zero, 1); /* IRM_SOCT - Transaction socket
*/
    currPtr = addBuffer(currPtr, &zero, 1); /* IRM_ES */
    currPtr = addBuffer(currPtr, sample->clientID, 8); /* IRM_CLIENTID */

```

```

currPtr = addBuffer(currPtr, &zero, 1); /* IRM_F1 - No MODNAME request */
currPtr = addBuffer(currPtr, &sample->commitMode, 1); /* IRM_F2 - Set commit mode */
currPtr = addBuffer(currPtr, &irm_f3, 1); /* IRM_F3 - Set sync level... */
currPtr = addBuffer(currPtr, &msgType, 1); /* IRM_F4 - Set message type */
currPtr = addBuffer(currPtr, sample->tranCode, 8); /* IRM_TRNCOD */
currPtr = addBuffer(currPtr, sample->datastoreID, 8); /* IRM_IMSDESTID */
currPtr = addBuffer(currPtr, sample->ltermName, 8); /* IRM_LTERM */
currPtr = addBuffer(currPtr, sample->racfUserID, 8); /* IRM_RACF_USERID */
currPtr = addBuffer(currPtr, sample->racfGroupName, 8); /* IRM_RACF_GRPNAME */
currPtr = addBuffer(currPtr, sample->password, 8); /* IRM_RACF_PW */
currPtr = addBuffer(currPtr, BLANK8, 8); /* IRM_APPL_NM */
currPtr = addBuffer(currPtr, sample->rerouteName, 8); /* IRM_REROUT_NM */

/* Add the transaction segment (trancode + tranText) just for IRM_F4 = ' ' or 'S' */ 5.
if (msgType == ' ' || msgType == 'S') {
    /* + 12 for LL and ZZ and tranCode */
    segmentLength = (short) (strlen(sample->tranText) + 12);
    /* convert to big endian */
    segmentLength = htons(segmentLength);
    currPtr = addBuffer(currPtr, &segmentLength, sizeof(short)); /* Transaction LL */
    currPtr = addBuffer(currPtr, &zero, sizeof(short)); /* Transaction ZZ */
    currPtr = addBuffer(currPtr, sample->tranCode, 8); /* Transaction code */
    currPtr = addBuffer(currPtr, sample->tranText, strlen(sample->tranText)); /* data */
}

/* send final LL ZZ to signal no more data to IMS Connect */ 6.
segmentLength = 4;
/* convert to big endian */
segmentLength = htons(segmentLength);
currPtr = addBuffer(currPtr, &segmentLength, sizeof(short)); /* End of message LL */
currPtr = addBuffer(currPtr, &zero, sizeof(short)); /* End of message zeros */

/* Send the built message to IMS Connect and free the malloc() */ 7.
send(sockfd, message, totalLength, 0);
free(message);
}

```

The following numbers correspond to the numbers in Example 14-5 on page 284:

1. First, we compute the length of the whole message. For a starter, we know the IRM PREFIX length, which, as we can see in Table 14-1 on page 269, is exactly 96 bytes (the offsets that you can see in Table 14-1 on page 269 are relative to the beginning of the message and have not yet accounted for the four-byte length field). We have to add to this 96 bytes:
 - Four bytes for the total message length prefix.
 - Four bytes for the trailing 0004 0000 block.
 - If we are sending data, 12 bytes for its LL, ZZ (2 bytes each) and the transaction code (8 bytes more).
 - The actual length of the data we are sending.
2. IRM_F3 is a bit map value, and we must build it combining the bit values of the appropriate flags.
3. When we know the total length, we allocate heap memory to build the message we want to send to IMS Connect.

As explained in 14.2, “IMS Connect message structures” on page 268, IMS Connect expects the binary quantities to be expressed in big endian byte order. We use the htonl()

and `htons()` macros to put integers and shorts, respectively, in the correct representation. Those macros take care of the details of the architecture in which we are running our program, so the bytes are swapped only when it is needed.

Note: The Intel x86 architecture used in the IBM @server xSeries® and common PC-compatible machines uses *little endian order*, while the zSeries, POWER, PowerPC, and SPARC processors are *big endian*. Nevertheless, do not rely on what you know about the target architecture for your application and instead make use of the aforementioned `htonl()`, `htons()`, `ntohl()`, and `ntohs()` macros to make your program more easily portable.

4. We build the IRM prefix field by field. To accomplish this, in this sample, we use an utility function (`addBuffer`) that copies a number of bytes to an specified address and returns the address to which we should copy the next block. See Example A-1 on page 470 for the code for that function. There you will find also the definition for the structure `sampletran`, which we use to keep all the interaction parameters.
5. If we are doing a SEND or SEND ONLY interaction (IRM_F4 is blank, or 'S'), we need to add the actual transaction data to our message. Notice how, once again, we use `htons()` to put the length of the segment in network, that is, big endian order.
6. If we need to send more than one segment in our message, do it at this point. In this sample, we are sending a unique segment, so we signal IMS Connect that there are no more segments to process, just sending four bytes with the values X'00040000', that is, the big endian representation of the length (4 bytes) and two binary zero octets.
7. At this point, we can send the message through the opened socket. For the message length, be sure to use a variable that holds that magnitude in the *native* byte order, or the results will be unpredictable. We also have to be careful with the dynamically allocated memory and free whatever buffer we used `malloc` with previously.

Getting the IMS Connect response

Unless we requested a SEND ONLY interaction, we should request and wait for a response from IMS Connect. This response will be formatted according to what we see in 14.2.2, “The IMS Connect output message” on page 274. We have to be prepared to get a success response, which will be composed by one or more data segments followed by a CSM, or a failure response, which will be formed by a single RSM.

In this sample, we did not request a MOD name, so we can safely ignore the RMM structure (we will never get that one), but we coded the corresponding support just to make it easier to add that capability to the sample program.

Example 14-6 shows the source code for the `sample_receive()` function, which takes care of all this issues.

Example 14-6 Obtaining the IMS Connect response in C

```
/*
 * Receive the data from the host, and parses it into segment strings
 * We are using HWSSMPL0, so we don't have a total-length prefix.
 * If we were using HWSSMPL1 we should take that into account.
 *
 * Parameters:
 * sockfd    Connected socket descriptor
 * sample    Address of a sampletran structure with the interaction data
 *
 * Warning: if you are adapting this code to be used in a production
 * program, please notice that this function DOES NOT free the heap
```

```

* memory it allocates, namely the array sample->response and its
* components.
*/
void sample_receive(int sockfd, struct sampletran *sample) {
    int totalLength, recordLength, segmentIndex, returnCode, reasonCode;
    char xsm_flg;
    char reserved;
    char * * segments;
    char * identifier;
    char * modName;
    char * segment;
    int done = 0;

    sample->nakRequired = 0;
    sample->ackRequired = 0;
    /*
     * malloc() up the segment array
     */
    segments = (char * *) malloc(sizeof(char *) * MAX_SEGMENTS);
    for (segmentIndex = 0; segmentIndex < MAX_SEGMENTS; segmentIndex++)
        segments[segmentIndex] = NULL;
    segmentIndex = 0;

    /* read total length */
    recv(sockfd, &totalLength, sizeof(short), 0);
    /* read flags byte */
    recv(sockfd, &xsm_flg, sizeof(char), 0);
    /* read reserved byte */
    recv(sockfd, &reserved, sizeof(char), 0);

    /* convert total length from big endian */
    totalLength = ntohs(totalLength);

    /* read identifier (exitID) */
    if (totalLength < 12) {
        identifier = (char *) malloc(sizeof(char) * (totalLength - 4) + 1);
        memset(identifier, 0, sizeof(char) * (totalLength - 4) + 1);
        recv(sockfd, identifier, totalLength - 4, 0);
    } else {
        identifier = (char *) malloc(sizeof(char) * 8 + 1);
        memset(identifier, 0, sizeof(char) * 8 + 1);
        recv(sockfd, identifier, 8, 0);
    }

    /* check first segment for possible errors / alerts */
    if (strcmp(identifier, "REQMOD") == 0) {
        /* read mod name */
        modName = (char *) malloc(sizeof(char) * 8);
        recv(sockfd, modName, 8, 0);
        /* add mod name to segment array */
        segments[segmentIndex++] = modName;
    } else if (strcmp(identifier, "REQSTS") == 0) {
        /* read return code and reason code */
        recv(sockfd, &returnCode, sizeof(int), 0);
        recv(sockfd, &reasonCode, sizeof(int), 0);
        /* convert them from big endian */
        returnCode = ntohl(returnCode);
        reasonCode = ntohl(reasonCode);
        /* add them to the segment array */
        segments[segmentIndex] = (char *) malloc(sizeof(char) * 20);
    }
}

```

```

    sprintf(segments[segmentIndex++], "RETURN CODE: %i", returnCode);
    segments[segmentIndex] = (char *) malloc(sizeof(char) * 20);
    sprintf(segments[segmentIndex++], "REASON CODE: %i", reasonCode);
    done = 1;
    /* return since there should be no more data */
    /* return segments; */
} else {
    if (totalLength <= 12)
        segments[segmentIndex++] = identifier;
    else {
        /* read in the rest of the segment data */
        segment = (char *) malloc(sizeof(char) * (totalLength - 12) + 1);
        /* + 1 for trailing zero */
        memset(segment, 0, sizeof(char) * (totalLength - 12) + 1); /* Clear buffer */
        recv(sockfd, segment, sizeof(char) * (totalLength - 12), 0);
        segments[segmentIndex] = (char *) malloc(sizeof(char) * (totalLength - 4));
        sprintf(segments[segmentIndex++], "%s%s", identifier, segment);
    }
}

/* continue trying to read in data till we come across *CSMOKY* */
if (done == 0) {
    while ((strcmp(segment, "CSMOKY") != 0) && (done == 0)) {
        /* read next segment */
        /* read LL */
        recv(sockfd, &recordLength, sizeof(short), 0);
        /* read ZZ */

        /* read flags byte */
        recv(sockfd, &xsm_flg, sizeof(char), 0);
        /* read reserved byte */
        recv(sockfd, &reserved, sizeof(char), 0);

        /* convert record length from big endian */
        recordLength = ntohs(recordLength);
        /* read in segment data */
        segment = (char *) malloc(sizeof(char) * (recordLength - 4) + 1);
        /* + 1 for trailing zero */
        memset(segment, 0, sizeof(char) * (recordLength - 4) + 1); /* Clear buffer */
        recv(sockfd, segment, sizeof(char) * (recordLength - 4), 0);
        /* add it to the segment vector */
        segments[segmentIndex++] = segment;
        if (segmentIndex >= MAX_SEGMENTS) {
            fprintf(stderr, "Maximum number of segments exceeded.\n");
            done = 1;
            sample->nakRequired = 1;
        }
    }
}

if (sample->nakRequired == 0) {
    /* Now we have reached the CSM or the RSM */
    /* Check if ACK required */
    if (xsm_flg & 0x20) {
        sample->ackRequired = 1;
    } else {
        sample->ackRequired = 0;
    }
}

sample->response = segments;
free(identifier);

```

}

The following numbers correspond to the numbers in Example 14-6 on page 287:

1. First, we must allocate storage in which to get the response. Because we are using the HWSSMPL0 exit, we do not know how much space we will need, or how many output segments will we get. In this sample application, we guess that we will have, at most, 100 segments, so we allocate an array of 100 pointers, which will support each segment. In a real-world application, we can use a linked list to avoid guessing the maximum segment number. If we run out of space and do not get all the data, we will have issues, so do not do it in a production environment.
2. Now, we read the header of the first segment. We read two length bytes, which come in network order and have to be translated to host order using `ntohs()`, the flags byte, which can be a CSM_FLG or RSM_FLG, and one reserved byte.
3. At this point, the next 8 bytes we read *might* be a message identifier, or might just be part of the IMS output message. In any case, we have to allocate storage to read those 8 bytes or whatever quantity of information follows.
4. If the segment we received is an RMM, we allocate more storage to get the MOD NAME, read it, and store it as the response first segment. In a real-world application, we might want to do some treatment based on the MOD NAME.
5. If the segment is an RSM, IMS Connect is informing us about an error condition. We get the return and reason codes, which are integers in big endian order. After translating them to host order, we allocate storage for two lines of text, which will contain the formatted error text. In this sample, we just print the numeric codes in decimal. A more sophisticated error handling routine can print a more informative text.

If there is an RSM in a message, it is the only component, so we set the done variable to a non-zero value.
6. If we reach this point, we know that our first segment is actually an output data segment. We allocate storage enough to contain the information, plus one byte for the required trailing zero (we assume we are dealing with character string data), and put into the newly allocated buffer the first 8 bytes we got previously, concatenated with the rest of this first data segment.
7. Now, we have processed the first segment, so we enter a loop until we find the CSM, or run out of space in the segments array. We know any segment we read will be a data segment or a CSM, so we keep reading until we find the *CSMOKY* literal.
8. If we run out space in the segments array, we stop reading and take note in the sampletran structure to send a NAK back to IMS Connect. Of course, we do not really know if we will have to send an actual NAK, because we have not read yet the CSM, and we will not. Therefore, the main function logic takes into account what is our sync level and does only send the NAK if it is not NONE.
9. At this point, we have just read a CSM, an RSM, or we ran out of segment space. In the first two cases, we can look at the CSM_FLG or RSM_FLG fields to check if we need to send an ACK to IMS Connect.
10. Finally, we reference the segments array in our sampletran structure and free the memory we allocated for the segment identifier (which was copied into the first of our segments if needed).

Building and sending the ACK or NAK

The `sample_send` function, described in detail in “Preparing and sending the message to IMS Connect” on page 284, can send any kind of messages, including ACK for positive acknowledgements and NAK for negative acknowledgements.

Closing the socket connection

To close the socket connection, use the `close()` function using the socket descriptor as a parameter.

In the sample program, we use a *transaction socket*. That means that the IMS Connect host will close the connection as soon as the transaction has completed. Therefore, if we try to send a second transaction to IMS, we receive an input output error, because we are trying to write to a socket that has the remote end closed. If we want to send more than one transaction without reopening the socket, we use a *permanent socket*. See Chapter 7, “IMS Connect programming model” on page 91 to learn more about transaction and permanent sockets.

If you have used IMS Connector for Java, you are familiar with *shared* and *dedicated* connections. This concept has nothing to do with non-IMS Connector for Java clients, and it is specific to the JCA implementation.

14.5.2 Java example

The complete source code listing is in Example A-2 on page 484. In this section, we review some code excerpts that are relevant to this chapter.

Program structure

The Java sample consists on a single class (Sample), which contains methods to prepare, send, and receive information to and from IMS Connect, and a static main method to parse the command line to get parameters and data to build the transaction. Even though this is not the *correct* way of doing things in Java, it will suffice to show you the IMS Connect programming specifics so that you can build on it.

Obtain a stream socket connection to the IMS Connect server

This is straightforward in Java. Example 14-7 shows the way to do it. The `Socket` constructor takes care of the host name translation and returns an opened, ready-to-use socket object reference.

Example 14-7 Opening a socket connection in Java

```
/**
 * Establish a socket connection with the IMS Connect host
 */
public void connect() {
    try {
        // open a socket for the transaction
        socket = new Socket(hostName, portNumber);
    } catch (Exception e) {
        System.err.println(e);
        System.exit(1);
    }
}
```

Preparing and sending the message to IMS Connect

Example 14-8 shows the code we use to send a message to IMS Connect in Java. This code is very similar to the code we reviewed in Example 14-5 on page 284. The only relevant differences are:

- We do not need to do any translation between the network and host byte orders. We use a `DataOutputStream` object to prepare the IRM, and the `write()` method of that Class already converts the binary integers to big endian order.

- We use `ByteArrayOutputStream` as a buffer to build and hold the message until it is complete so that we can send it to the IMS Connect server in a single `write()`.
- Of course, we do not have to worry about memory management: The garbage collector takes care of that.

Example 14-8 Preparing and sending a message to IMS Connect in Java

```
/**
 * Build and send a message to IMS Connect
 * This method sends the message using a single write() call
 * to improve efficiency and performance.
 * The z/OS TCP/IP stack can be configured in such way that
 * there is a 200ms delay for every ack message (TCP ack, not
 * IMS Connect ACK). If such is the case, then you should expect
 * a severe performance penalty if you use multiple write() calls.
 * The old IMS Connect sample used a write for each field. DO
 * NOT DO THAT.
 * @param msgType Value for the IRM_F4 field (' ','R','A','N')
 */
public void send(char msgType) {
    int totalLength;
    String segment = null;
    response = null;
    byte irm_f3 = 0;

    if (msgType == ' ')
        segment = this.tranText;

    // Compute the total length of the message.
    // In java we don't care about byte order, since the write() methods
    // of DataOutputStream always work in network order.

    // +4 for first LL, ZZ and final LL, ZZ
    totalLength = 4 + prefixLength + 4;

    // add in segment length, if segment is defined
    if ((segment != null) && (segment.length() > 0)) {
        totalLength += segment.length() + 12; // +12 for LL, ZZ, tranCode
    }

    try {
        // Allocate a conveniently sized byte stream.
        ByteArrayOutputStream messageBuffer =
            new ByteArrayOutputStream(totalLength);
        // Associate a DataOutputStream to this newly created byte stream.
        DataOutputStream out = new DataOutputStream(messageBuffer);
        // Prepare another DataOutputStream to send the real message
        // to our IP socket.
        DataOutputStream outsocket =
            new DataOutputStream(socket.getOutputStream());

        // Compute the IRM_F3 byte
        irm_f3 = syncLevel;
        if (purgeAsync) {
            irm_f3 |= SL_PURGE;
        }
        if (reRoute) {
            irm_f3 |= SL_REROUTE;
        }
    }
}
```

1.

2.

3.

```

// Build the IRM prefix
out.writeInt(totalLength); // total message length
out.writeShort(prefixLength); // IRM_LL
out.writeByte(0x01); // IRM_ARCH
out.writeByte(0x00); // IRM_F0
// out.writeShort((short) 0); // IRM_RSV
out.writeBytes(exitID); // IRM_ID
out.writeInt(0); // IRM_RES
out.writeByte(IRM_F5_SINGLE); // IRM_F5
out.writeByte(timer); // IRM_TIMER
out.writeByte(0x40); // IRM_SOCKET - Transaction socket
out.writeByte(0); // IRM_ES
out.writeBytes(clientID); // IRM_CLIENTID
out.writeByte(0); // IRM_F1 - No MODNAME request
out.write(commitMode); // IRM_F2 - Set commit mode
out.write(irm_f3); // IRM_F3 - Set sync level et al
out.writeByte(msgType); // IRM_F4 - Set message type
out.writeBytes(tranCode); // IRM_TRNCOD
out.writeBytes(datastoreID); // IRM_IMSDESTID
out.writeBytes(ltermName); // IRM_LTERM
out.writeBytes(racfUserID); // IRM_RACF_USERID
out.writeBytes(racfGroupName); // IRM_RACF_GRPNAME
out.writeBytes(password); // IRM_RACF_PW
out.writeBytes(" "); // IRM_APPL_NM
out.writeBytes(reRouteName); // IRM_REROUT_NM

// Add the transaction segment (trancode + tranText) if required
if (msgType == ' ' || msgType == 'S') {
    // 12 for LL and ZZ and tranCode
    short recordLength = (short) (segment.length() + 12);
    out.writeShort(recordLength); // Transaction LL
    out.writeShort((short) 0); // Transaction ZZ
    out.writeBytes(tranCode); // Transaction code
    out.writeBytes(segment); // Transaction data
}

// send final LL ZZ to signal no more data to IMS Connect
out.writeShort((short) 4); // send LL
out.writeShort((short) 0); // send ZZ

// Send the built message to IMS Connect
out.flush();
outsocket.write(messageBuffer.toByteArray());
outsocket.flush();
} catch (Exception e) {
    System.err.println(e);
    System.exit(1);
}
}

```

The following numbers correspond to the numbers in Example 14-8 on page 292:

1. First, we compute the length of the whole message. For a start, we know the IRM PREFIX length, which, as we can see in Table 14-1 on page 269, is exactly 96 bytes (the offsets that you can see in Table 14-1 on page 269 are relative to the beginning of the message and have not yet accounted for the four-byte length field). We have to add to this 96 bytes:
 - Four bytes for the total message length prefix.
 - Four bytes for the trailing 0004 0000 block.

- If we are sending data, 12 bytes for its LL, ZZ (2 bytes each) and the transaction code (8 bytes more).
 - The actual length of the data we are sending.
2. We want to send all the data using a single write, so we create an output stream based on a byte array in which we will build the IRM prefix and the transaction data. The constructor we use allocates the entire array at once. This is convenient for performance reasons, but it is not required, because Java can grow the buffer as needed.
 3. We build the IRM_F3 value using the bit values of the flags we want to apply.
 4. We build the IRM prefix using the ByteArrayOutputStream. Notice that we do not care about byte order transformation, because the write(short) and write(long) methods always use big endian order.
 5. We add the transaction data, in LLZZdata format, to the output message.
 6. If we need to send more than one segment in our message, do it at this point. In this sample, we are sending a unique segment, so we signal IMS Connect that there are no more segments to process, just sending four bytes with the values X'00040000', that is, the big endian representation of the length (4 bytes) and two binary zero octets.
 7. Now that we have built our message using the ByteArrayOutputStream, all we need is to flush the buffer and write the message to the real OutputStream, associated with the IMS Connect socket.

Getting the IMS Connect response

Unless we requested a SEND ONLY interaction, we should request and wait for a response from IMS Connect. This response is formatted according to what we saw in 14.2.2, “The IMS Connect output message” on page 274. We have to be prepared to get a success response, which will be composed by one or more data segments followed by a CSM, or a failure response, which will be formed by a single RSM.

In this sample (Example 14-9), we did not request for a MOD name, so we can safely ignore the RMM structure (we will never get that one), but we coded the corresponding support just to make it easier to add that capability to the sample program.

Example 14-9 Obtaining the IMS Connect response in Java

```

/**
 * Read and parse the response from IMS Connect
 * We are using HWSMPL0, so we won't have a total-length
 * prefix. If we were using HWSMPL1 instead, we should have
 * taken that into consideration.
 */
public void receive() {
    byte xsm_flg = 0; // Flag byte
    boolean done = false; // End of loop indication

    // initialize segment vector
    Vector segments = new Vector();

    RSM = false;
    try {
        DataInputStream in = new DataInputStream(socket.getInputStream());

        // read total length. We don't care about byte order
        int totalLength = (int) in.readShort();
        // read flags
        xsm_flg = in.readByte();
        // read and ignore reserved byte

```

1.

```

in.readByte();

// read identifier
byte[] identifierBytes = null;
if (totalLength < 12) {
    identifierBytes = new byte[totalLength - 4];
} else {
    identifierBytes = new byte[8];
}
in.readFully(identifierBytes);
String identifier = new String(identifierBytes);

// check first segment for possible errors / alerts
if (identifier.equals("*REQMOD*")) { // Parse RMM
    // read mod name
    byte[] modBytes = new byte[8];
    in.readFully(modBytes);
    String modName = new String(modBytes);
    // add mod name to segment vector
    segments.add("MOD NAME: " + modName);
} else if (identifier.equals("*REQSTS*")) { // Parse RSM
    // read return code and reason code
    RSM = true;
    returnCode = in.readInt();
    reasonCode = in.readInt();
    done = true; // No more data after RSM
} else {
    if (totalLength <= 12)
        segments.add(identifier);
    else {
        // read in the rest of the segment data
        byte[] segmentBytes = new byte[totalLength - 12];
        // identifier was actually
        in.readFully(segmentBytes);
        // part of the data. So we add it
        segments.add(identifier + new String(segmentBytes));
    }
}

String segmentData = "";
// continue trying to read in data till we come across *CSMOKY*
while (!segmentData.equals("*CSMOKY*") && !done) {
    // read next segment
    // read LL, XSM_FLG and reserved byte
    short recordLength = in.readShort();
    xsm_flg = in.readByte();
    in.readByte();
    // read in segment data
    byte[] segmentBytes = new byte[recordLength - 4];
    in.readFully(segmentBytes);
    segmentData = new String(segmentBytes);
    // add it to the segment vector
    segments.add(segmentData);
}

// At this point, we have just read an RSM or a CSM.
// Check if an ACK will be required

if ((xsm_flg & 0x20) == 0x20)
    ackRequired = true;

```

```

    } catch (Exception e) {
        System.err.println(e);
        System.exit(1);
    }

    // return segment vector
    this.response = segments;
}

```

The following numbers correspond to the numbers in Example 14-9 on page 294:

1. First, we read the header of the first segment. We read two length bytes, which come in network order, and are correctly read by the `readShort()` method, the flags byte, which can be a CSM_FLG or RSM_FLG, and one reserved byte. Notice that if we were using HWSSMPL1, we should first read the 4-byte total length prefix.
2. At this point, the next 8 bytes we read *might* be a message identifier, or might just be part of the IMS output message. In any case, we have to allocate storage to read those 8 bytes or whatever quantity of information follows.
3. If the segment we just received is an RMM, we allocate more storage to get the MOD NAME, read it, and store it as the response first segment. In a real-world application, we would probably want to do some treatment based on the MOD NAME.
4. If the segment is an RSM, IMS Connect is informing us about an error condition. We get the return and reason codes, which are integers in big endian order. In this sample, we move the return and reason codes to a pair of attributes of the Sample class.
5. If we reach this point, we know that our first segment is actually an output data segment. We allocate storage enough to contain the information and put into the newly allocated buffer the first 8 bytes we got previously, concatenated with the rest of this first data segment.
6. Now, we have processed the first segment, so we enter a loop until we find the CSM or run out of space in the segments array. We know any segment we read will be a data segment or a CSM, so we just keep reading until we find the *CSMOKY* literal.
7. At this point, we have just read a CSM, an RSM, or we ran out of segment space. In the first two cases, we can look at the CSM_FLG or RSM_FLG fields to check if we need to send an ACK to IMS Connect.



IMS Connect client diagnostics

In this chapter, we discuss some issues you might encounter developing and using IMS Connect and IMS Connector for Java client applications.

If you have IMS Connect Extensions installed on your system, refer too to 11.7, “IMS Connect problem determination” on page 205.

15.1 No response from IMS or IMS Connect

We distinguish the following different situations for which we do get the response for which we are waiting: hanging clients, TCP/IP socket timeouts, and IMS Connect execution timeouts.

15.1.1 Hanging clients

This case is usually related to some kind of client programming error. The most common causes are:

- ▶ You are specifying an incorrect length for the message you are sending. If you specify a length greater than the actual length of the message, IMS Connect will block waiting for the data it thinks you are still going to send. If you specify a length shorter than the actual length you send, IMS Connect will read the first two additional bytes and interpret them as the length of the following message. It can result in a protocol error status being returned or a blocking condition occurring.

Tip: One way to view the contents of the messages that are sent back and forth between IMS Connector for Java and IMS Connect is to enable WebSphere Application Server tracing, set the trace specification to `com.ibm.connector2.ims.*=all=enabled` in WebSphere Application Server Version 5 or `com.ibm.connector2.ims.*=enabled` in WebSphere Application Server Version 6, and set the trace level to 3 (`RAS_TRACE_INTERNAL`) in the J2C connection factory.

Alternatively, you can use a TCP/IP tracing tool to see the messages actually being sent on the wire, but be aware of your company's policy regarding the use of such tool, because it can be considered a security risk.

- ▶ You specified a value of `X'FF'` (wait forever) for the `IRM_TIMER` field in a `RESUME TPIPE` interaction with a flow control other than `AUTO` or `SINGLE_WAIT`. In this case, IMS Connect will not send you any message because of the flow option, but you will not get a timeout notification.
- ▶ Your program has lost the protocol synchronization with IMS Connect. Both your client program and IMS Connect are blocked in a `recv()` operation, waiting for the other end to send something through the socket.

Tip: If you have IMS Connect Extensions, the trace report is the best tool to diagnose this kind of problem. See Chapter 11, "IMS Connect Extensions" on page 155 for details. If you do not have IMS Connect Extensions, the IMS Connect recorder trace combined with the information you can get from a TCP/IP trace can help you to diagnose this situation.

- ▶ Your program has a logic error, such as a loop or an erroneous recovery after an I/O error. Use the debugging tools available in your client platform to diagnose and fix the problem.

Usually, the first thing you will want to know if you find yourself dealing with an apparently hung client is determine if your program has reached IMS Connect or if it is in some sort of internal loop. To know if your program is already talking to IMS Connect, you can use the `VIEW PORT` command. Example 15-1 on page 299 shows the command issued as an z/OS `MODIFY` command. The corresponding IMS Connect command would be issued to the IMS Connect WTO reply number (nnn):

```
nnn,VIEWPORT 7003
```

Example 15-1 Using the VIEW PORT command to look for an active client

```
/F IMSGCONN, VIEW PORT NAME(7003)
```

HWSC0001I	PORT=7003	STATUS=ACTIVE					
HWSC0001I	CLIENTID	USERID	TRANCODE	STATUS	SECOND	CLNTPORT	IP-ADDRESS
HWSC0001I	JORDI		IVTJJ	CONN	500	4879	009.001.039.119
HWSC0001I	TOTAL CLIENTS=1	RECV=0	CONN=1	XMIT=0	OTHER=0		

In this example, we can see just one active client, using the clientID JORDI, active from the IP address 9.1.39.119. This client has been active for 500 seconds, so we know that our program has *at least* established communication with IMS Connect and has sent it a request to execute a transaction whose transaction code is IVTJJ. We can also tell from the connection status that IMS Connect has sent the transaction request to IMS and is still waiting for the output from that transaction to be returned by IMS.

Of course, this a relatively easy case to spot. If we were trying to locate an IMS Connector for Java client that was using a generated clientID (in other words a client using a sharable persistent socket) in a heavily used environment, we might have found several active clients to identify our hung program.

15.1.2 TCP/IP socket timeouts

A socket timeout occurs when the maximum wait time for a socket operation to complete is exceeded. You can specify this maximum wait time in several ways. If you are using IMS Connector for Java:

- ▶ If your application is built with generated code, you can set the socket timeout property value in the Web Services Description Language (WSDL) for the application or in the J2C connection factory.
- ▶ If you are coding directly to the CCI API, you can set the socket timeout value for an interaction using the `setSocketTimeout()` method of the `IMSInteractionSpec`.

If a timeout then occurs, your program will get the exception shown in Example 15-2.

Example 15-2 Exception thrown when a socket timeout occurs

```
javax.resource.spi.CommException: IC00113E:  
com.ibm.connector2.ims.ico.IMSTCIPManagedConnection@7f462c4a.receive() error. Socket  
Timeout has occurred for this interaction. The Socket Timeout value specified was [50]  
milliseconds. [java.net.SocketTimeoutException: Read timed out]
```

- ▶ You can use the `setSoTimeout()` method of a `Socket` object if you are programming a non-IMS Connector for Java client in the Java programming language.

Example 15-3 shows the exception thrown by the `Socket` class when the socket timeout in the `Socket` class is triggered. Your program should catch and handle this exception appropriately.

Example 15-3 Exception thrown by the socket class when a timeout exception occurs

```
java.io.InterruptedIOException: Read timed out
```

- ▶ You can also use the `setsockopt()` function with the `SO_SNDTIMEO` and `SO_RCVTIMEO` options if you are writing a C client program in a UNIX or UNIX-like environment.

If a socket timeout occurs, your `recv()` call will return fewer bytes than requested, or you will get an error with `errno=EWOULDBLOCK` if there were no bytes to read.

- Finally, you can use the equivalent function in your programming and execution environment.

If you choose to use socket timeouts, be aware that IMS Connect ignores disconnect requests when its own timeout period (set using the `IRM_TIMER` field) is still active. Design your application considering the socket timeout as a safety net protecting against delays caused by the network, not delays caused by IMS Connect or IMS.

Important: Set the value for the socket timeout to be *greater* than the timeout specified in the IMS Connect `IRM_TIMER`. In this way, your application receives an IRM timeout message (or an Execution Timeout message in the case of IMS Connector for Java) if IMS Connect fails to receive a response from IMS, rather than a socket timeout message. This enables your application to differentiate between an IRM or execution timeout caused by IMS not returning an output message and a socket timeout due perhaps to network problems. Failure to do so can result in unpredictable behavior in IMS Connect, which might be difficult to diagnose, possibly including duplicate client errors.

15.1.3 IMS Connect execution timeouts

Figure 15-1 depicts a non-IMS Connector for Java client scenario in which you will get an IMS Connect timeout notification. The situation depicted corresponds to a commit mode 1 interaction, in which the value specified for `IRM_TIMEOUT` was X'2C', corresponding to 5 seconds. In this case, an execution timeout has occurred, so the commit mode 1 output, when it is available from OTMA, will be NAKed by IMS Connect and then discarded by OTMA.

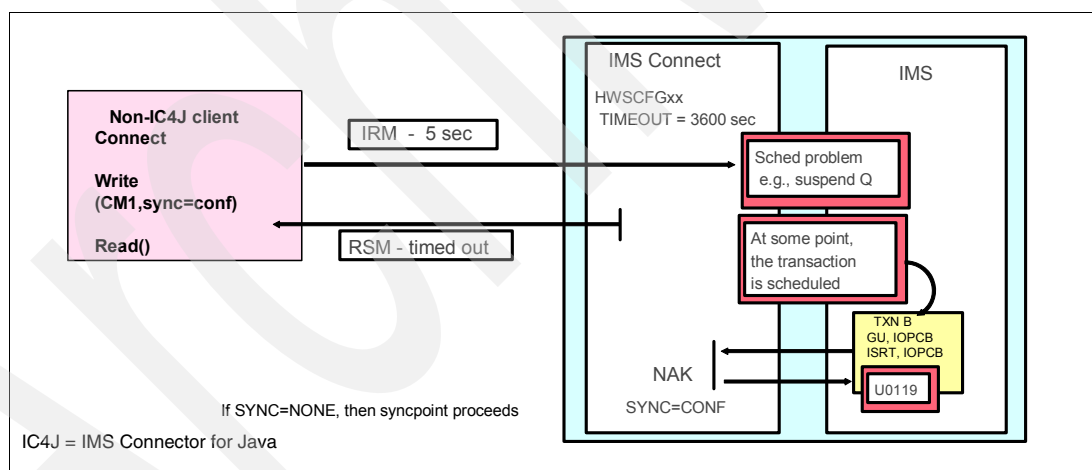


Figure 15-1 IMS Connect timeout

Depending on the interface you use to access IMS Connect, you will get notified of a execution timeout in the following ways:

- If you are using IMS Connector for Java, the adapter throws an exception, as shown in Example 15-4 on page 301. If the value shown as `executionTimeout` is zero, the timeout was based on the default value specified in the IMS Connect configuration member (`HWSCFGxx`). Notice that in this case we specified a value of 1 millisecond, but IMS Connect used 10, due to the manner in which timeout values are converted by IMS Connect. See the IMS Connector for Java execution timeout documentation, which describes how IMS Connect converts the execution timeout value specified into the actual timeout value that it uses. Access this documentation at:

<http://www.ibm.com/software/data/db2imstools/imstools-library.html#imsconjav-lib>

Example 15-4 Exception thrown when a execution timeout occurs

```
javax.resource.spi.EISSystemException: IC00081E:  
com.ibm.connector2.ims.ico.IMSTCPIPManagedConnection@25cdec5c.processOutput0TMAMsg(byte[],  
IMSInteractionSpec, int) error. Execution timeout has occurred for this interaction. The  
executionTimeout value specified was [1] milliseconds. The value used by IMS Connect was  
[10] milliseconds.
```

- ▶ If you are using a non-IMS Connector for Java client with the socket interface, you will get an RSM block, with the following content:
 - The RSM_RETCD value will be X'20' or X'24'. The latter value means that the timeout specified was X'00' or an invalid value, so the default value from the IMS Connect configuration member was used. The former indicates that the specified value, after conversion by IMS Connect, was used.
 - RSM_RSNCOD contains the value that was specified for IRM_TIMER.

Notice that, even if you are using the Java programming language, you will *not* get an exception in this case, just a return code and a reason code in the RSM block of the response message. It is up to the client code to determine whether or not to throw an exception based on the contents of the response message. You will also find a message in the SYSPRINT of the IMS Connect JOB or STC, as shown in Example 15-5.

Example 15-5 Message on the IMS Connect JES2 log after an execution timeout

```
HWSD0252W UNABLE TO SEND RESPONSE FROM DS=IMSG      TO CLIENT=HWSWKY5E; R=4, S=LATEMSG ,  
M=DREC
```

Perform the following steps to diagnose a situation where multiple execution timeouts occur:

1. Check if the default TIMEOUT value specified in the IMS Connect configuration member is adequate for the usual transaction response times that you have in your system. Remember that the value is specified in hundredths of seconds. A value of 500 means 5 seconds. The default value needs to be:
 - Greater than the response time of your slowest transaction
 - Lower than the value used for the socket timeout (if any)If you experience abnormal transaction response times, consider modifying the IMS Connect TIMEOUT value and recycle your IMS Connect. If you are using IMS Connect Extensions, you also consider using its pacing feature to diminish the IMS workload, rejecting the overload before it gets into IMS and thus allowing the accepted transactions to complete in a reasonable time.
2. Check if the client application is using an appropriate value for IRM_TIMER. The exception message or the RSM values will tell you what value was used to determine the timeout period. If you find that the value is too low, change the application or the application configuration. Set the value for IRM_TIMER using the same reasoning as was used for the default TIMEOUT value specified in IMS Connect configuration member.
3. Check if the transactions you are experiencing problems with are in a correct state. Use the IMS commands /DIS TRANSACTION and /DIS PROGRAM to verify that the transaction and the application program are active and ready to be scheduled. If an IMS application program ends abnormally (abend), IMS stops both the transaction and the program. If the transaction is STOPPED, you get DFS065 messages. However, if the transaction is started but the program is stopped, the transaction will not be scheduled.

4. Check if IMS is experimenting global problems. The IMS troubleshooting procedures are beyond the scope of this book. If you know that the IMS problems will not be solved in a short period of time, and you are experiencing a lot of execution timeouts, lengthening the IMS Connect TIMEOUT value or using the IMS Connect Extensions pacing feature to diminish the IMS workload should alleviate this problem.

Unless you find a configuration problem, there are very few things that can be done on the IMS Connect side to fix an execution timeout situation. The problem will be either in IMS or on the application side. However, ensure that the timeout values used in your applications and in IMS Connect are set appropriately.

15.2 Error messages from IMS

You can get IMS error messages in the form of literals, which begin with an error code, expressed as DFSnnnn, where nnnn is a code that specifies the kind of error.

If you are using IMS Connector for Java, the way DFS messages are handled depends on the type of `imsRequestType` that was used for the interaction. If the `imsRequestType` was `IMS_REQUEST_TYPE_IMS_COMMAND` or `IMS_REQUEST_TYPE_MFS_TRANSACTION`, DFS messages are treated as normal output and, assuming that there are no other problems with the output, no exception is thrown. If the `imsRequestType` was `IMS_REQUEST_TYPE_IMS_TRANSACTION`, DFS messages are treated as errors. In this last case, IMS Connector for Java throws an exception, as shown in Example 15-6.

Example 15-6 Exception thrown when IMS sends a DFSxxxx message

```
com.ibm.connector2.ims.ico.IMSDFSMessageException: IC00079E:
com.ibm.connector2.ims.ico.IMSTCIPManagedConnection@4b7cac41.getOutputData(InteractionSpec
) error. IMS returned DFS message: DFS065 17:58:27 TRAN/LTERM STOPPED
```

If you are using the socket interface with a non-IMS Connector for Java client, you will get the DFS message as a normal IMS response (followed by the CSM block). However, there is an important difference between a normal IMS response and *some* of the DFS status messages: In some cases, you must *not* send an ACK after you get a DFS message. That is the reason why the recommended way to decide if you need to send an ACK is to check the flag byte present both in the CSM and RSM blocks, instead of relying on your own program logic.

15.3 Wrong status codes from IMS Connect

IMS Connect uses the RSM message structure to notify non-IMS Connector for Java client applications of any special situation. This refers not only to error or exception situations, but also to informational codes associated with a normally ended interaction. The RSM structure contains a return code and a reason code that describe the detected situation.

In this section, we analyze some of the error conditions reported in an RSM, the method to diagnose it, and the measures you can take to correct the situation.

15.3.1 Duplicate clientID (reason code 56)

This error occurs when you send IMS Connect an interaction on a given port using a clientID that is already in use on that port. There can be several reasons that this can occur:

- ▶ You are executing two concurrent interactions using the same clientID. This can happen when you are running more than one instance of an application in the same or different machines, or in cloned environments, especially if you are using dedicated persistent sockets, which require user-specified clientIDs, with IMS Connector for Java.

You must ensure that the same clientID is not associated with more than one connection to a given IMS Connect port. In particular, it is not possible to both send transactions and recover asynchronous output messages (RESUME TPIPE) on different connections to the same IMS Connect port using the same clientID at the same time. For example, if you are executing a receive asynchronous output interaction (RESUME TPIPE) using clientID 12345abc on a socket connected to PORT 5678 of IMS Connect ICON01, you are not able to submit transaction requests using the clientID 12345abc on another socket connected to PORT 5678 of IMS Connect ICON01.

Note that it is possible to use the same clientID on two different sockets connected to the same IMS Connect, provided that the sockets are connected to different ports on that IMS Connect.

- ▶ You are executing an interaction with a clientID that is the same as that used by another interaction that ended as a result of a socket timeout. If this new interaction is received by IMS Connect while IMS Connect is still waiting for a response from IMS for the original interaction that received the socket timeout, a duplicate clientID error might occur.

This can happen if you set the socket timeout to a value that is less than the timeout set by the IRM_TIMER (or ExecutionTimeout in the case of IMS Connector for Java) or the IMS Connect default timeout set in the HWSCFGxx member if that is the timeout being used for the original interaction. IMS Connect is not aware that the original socket has been disconnected as a result of the socket timeout until it does a subsequent read on that socket, so it would consider the original socket still active, even though that socket has already been disconnected from the client end. Figure 15-2 on page 304 shows the sequence of events that lead to this problem. After you get to this situation, you will receive DUPECLNT errors until:

- The IRM_TIMER expires on the IMS Connect side.
- You issue a STOPCLNT command to kill the client on the IMS Connect side. A DELETE PORT command also kills the client, but we do not recommend this, because it disables that port for all clients.

Example 15-7 shows the message IMS Connect writes in the system log when a DUPECLNT occurs.

Example 15-7 IMS Connect JES2 DUPECLNT error message

```
HWSS0742W MESSAGE FAILED, ORIGIN=7003 _CLIENT01 TO DESTID=IMSG ; R=8, S=DUPECLNT,
M=SRE4
```

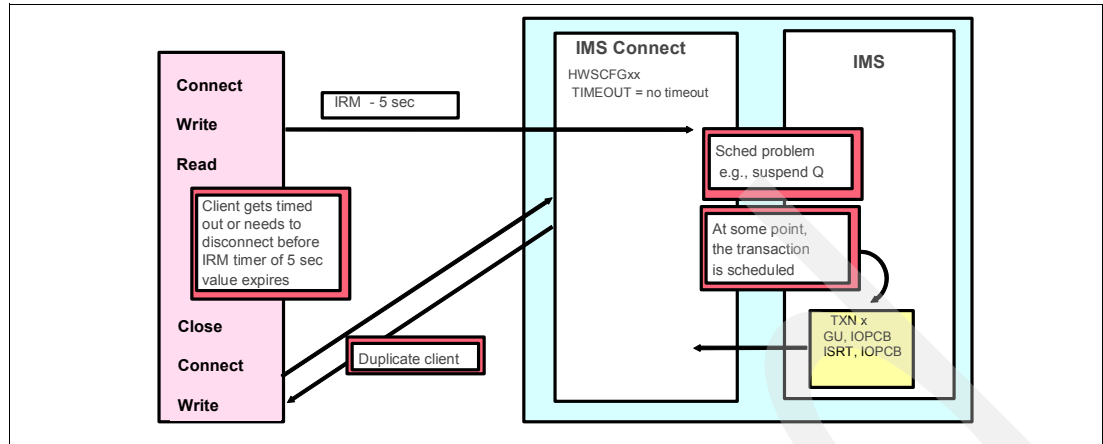


Figure 15-2 Duplicate clientID caused by an erroneous timeout setting

APARs PQ96500 and PQ96501 introduce a way to circumvent this problem for non-IMS Connector for Java applications, as shown in Figure 15-3. In the event that you receive a DUPECLNT error response from IMS Connect caused by the premature disconnection of a socket while the IRM_TIMER timeout is still active, you can send IMS Connect a new type of interaction, CANCEL TIMER, setting IRM_F4 to C.

For example, when IMS Connect *sees* a CANCEL TIMER command on a socket with clientID CLIENT01, it “pops” the IRM_TIMER on the original CLIENT01 socket, attempts to send a timeout error back on the original CLIENT01 socket, sends a “CANCEL TIMER success” response back on the socket from which the CANCEL TIMER command was issued, and then disconnects that socket from which the CANCEL TIMER command was issued.

The attempt by IMS Connect to send a timeout error back on the original CLIENT01 socket will, of course, fail because the socket has already been disconnected from the client end. As a result, IMS Connect will go through its disconnect processing to clean up all of its resources associated with that socket. At this point, both CLIENT01 sockets, the original one and the one that was used for the CANCEL TIMER request, will have been disconnected, so you are able to create and use another connection with clientID CLIENT01.

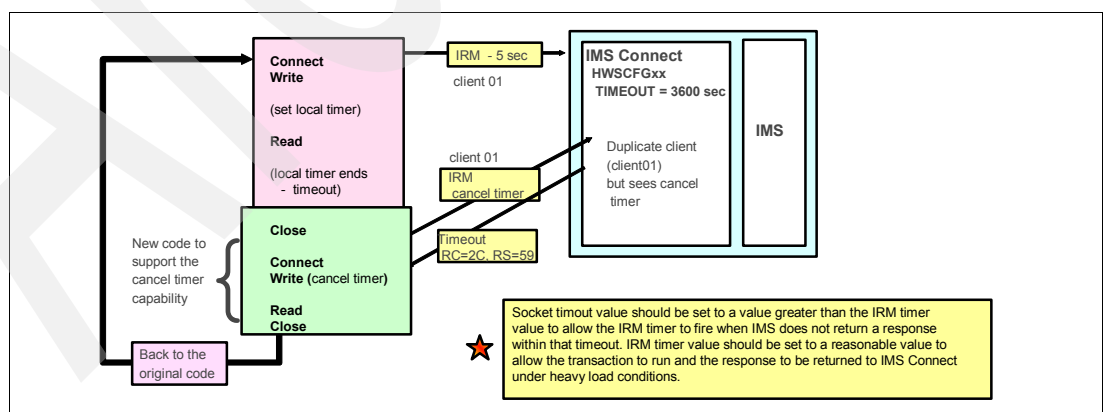


Figure 15-3 CANCEL TIMER used to avoid DUPECLNT after premature disconnect

The response for the original, pending interaction is considered non-deliverable, so:

- If the original interaction was running under commit mode 1, and the sync level was set to NONE, the transaction is committed and the response is discarded.

- If the original interaction was running under commit mode 1, and the sync level was set to CONFIRM or SYNCPT, the transaction is backed out and the region abends with U0119.
- If the original interaction was running under commit mode 0, the transaction is committed and the response will be deemed undeliverable. The response is:
 - Purged (discarded) if you specified PURGE NOT DELIVERABLE in the IRM header.
 - Rerouted to the specified or default destination if you specified REROUTE in the IRM header.
 - Queued to the asynchronous hold queue of the Tpipe corresponding to the clientID used for the interaction.

Note: The CANCEL TIMER mechanism is intended to free a clientID that is blocked because a client program was prematurely disconnected, perhaps because of a network failure. Do *not* rely on CANCEL TIMER to avoid the DUPECLNT error after a socket timeout caused by an incorrect timeout value. Always set the socket timeout to a value greater than the IRM_TIMER value. If you use CANCEL TIMER to systematically address problems due to the incorrect settings of the socket timeout, IRM_TIMER/ExecutionTimeout, or IMS Connect TIMEOUT, you will encounter performance problems.

15.3.2 OTMA protocol error (reason code 36)

You will encounter this error when your program sends a message to IMS Connect that is out of sequence with the input that IMS Connect expects. For example, IMS Connect will send this error if you send a new request when it is waiting for an ACK for the previous request.

Example 15-8 shows an example of such an error, as it appears in the IMS Connect system log.

Example 15-8 IMS Connect JES2 log OTMA protocol error message

```
HWSP1495E OTMA PROTOCOL VIOLATION; R=20, C=CLIENT01, M=SDRC
```

If you have IMS Connect Extensions installed, use its trace analysis features to diagnose this problem. If you do not have IMS Connect Extensions, you can use the recorder trace to learn the sequence of events previous to the error.

Note: If you are using IMS Connector for Java Version 2.2.2 or earlier, you will see this error message (and the corresponding IMS Connector for Java exception) if you send a transaction that generates multiple IOPCB responses under commit mode 0. Update your IMS Connector for Java resource adapter to Version 2.2.3, 9.1.0.2, 9.1.0.1.1, or later.

15.3.3 Other errors

In general, the steps to diagnose a failing IMS Connect application are more or less the same:

- Look at the SYSLOG and the SYSPRINT of the IMS Connect you are using. If you find an error message there, look for this message in *IMS Connect Guide and Reference*, SC18-9287.

- ▶ Try to reproduce the problem after enabling the IMS Connect recorder trace (see Chapter 5, “IMS Connect operations” on page 63 for details) and analyze the trace records.
- ▶ If you are using IMS Connector for Java in WebSphere Application Server, try to reproduce the problem after enabling WebSphere Application Server tracing for IMS Connector for Java (see the “Logging and tracing with the IMS resource adapter” topic in the online help of WebSphere Studio Application Developer Integration Edition or Rational Application Developer. This document can also be found in PDF format on the IMS Web site under IMS Connector for Java → Publications:
<http://www.ibm.com/ims>
- ▶ Use a TCP/IP packet logger to analyze the network traffic between your client machine and IMS Connect.
- ▶ Use the trace analysis features of IMS Connect Extensions (see Chapter 11, “IMS Connect Extensions” on page 155 for details).

If you decide to report the error to IBM, you need to provide the following material:

- ▶ IMS Connect recorder trace log
- ▶ IMS Connector for Java and J2EE Connector architecture (J2C) trace logs if applicable

To enable the IMS Connector for Java and J2C trace logs, perform the following steps:

- ▶ If you are running your application under the development environment (for example, WebSphere Application Developer Studio Integration Edition), you must open the Server perspective and double-click your test environment server to open the configuration page. Then, select the **Trace notebook**. Ensure that the **Enable trace** check box is selected and add the following string to the Trace String:

```
com.ibm.connector2.ims.*=all=enabled:com.ibm.ejs.j2c.*=all=enabled
```

The first part of the string enables tracing in the IMS Connector for Java classes, while the second part enables tracing in the J2C classes. This string can be *added* to or replace the existing Trace String.

Next, select the **J2C notebook** and highlight (select) the IMS Connector for Java resource adapter and then the connection factory you are using. Scroll down to TraceLevel under Resource Properties and set the value to 3, as shown in Figure 15-4.

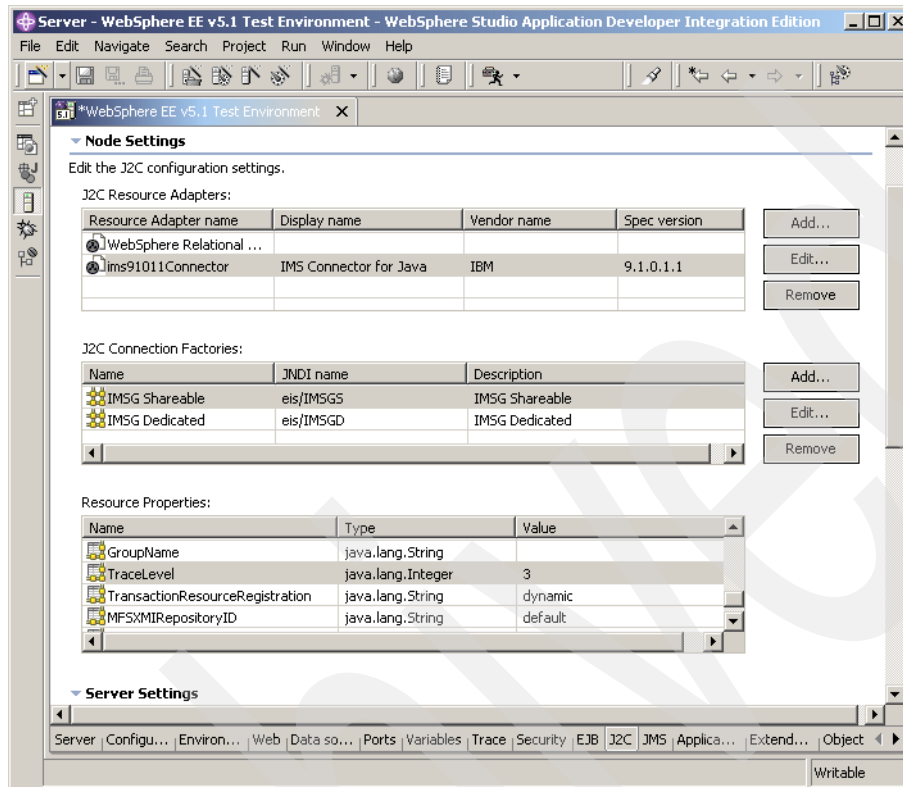


Figure 15-4 Setting the trace level in the IMS Connector for Java J2C adapter

If you are using the default values, your trace.log files will be under your workspace directory in:

`${WORKSPACEDIR}\.metadata\.plugins\com.ibm.etools.server.core\tmp0\logs\server1`

The log file can grow very fast, so you will want to reset the trace settings after collecting the required trace data.

- If you are running your application in a WebSphere Application Server production environment, you must enable the traces using the administrative console. Select **TroubleShooting** → **Logs and Trace** and then select your server instance name and **Diagnostic Trace**, as shown in Figure 15-5.

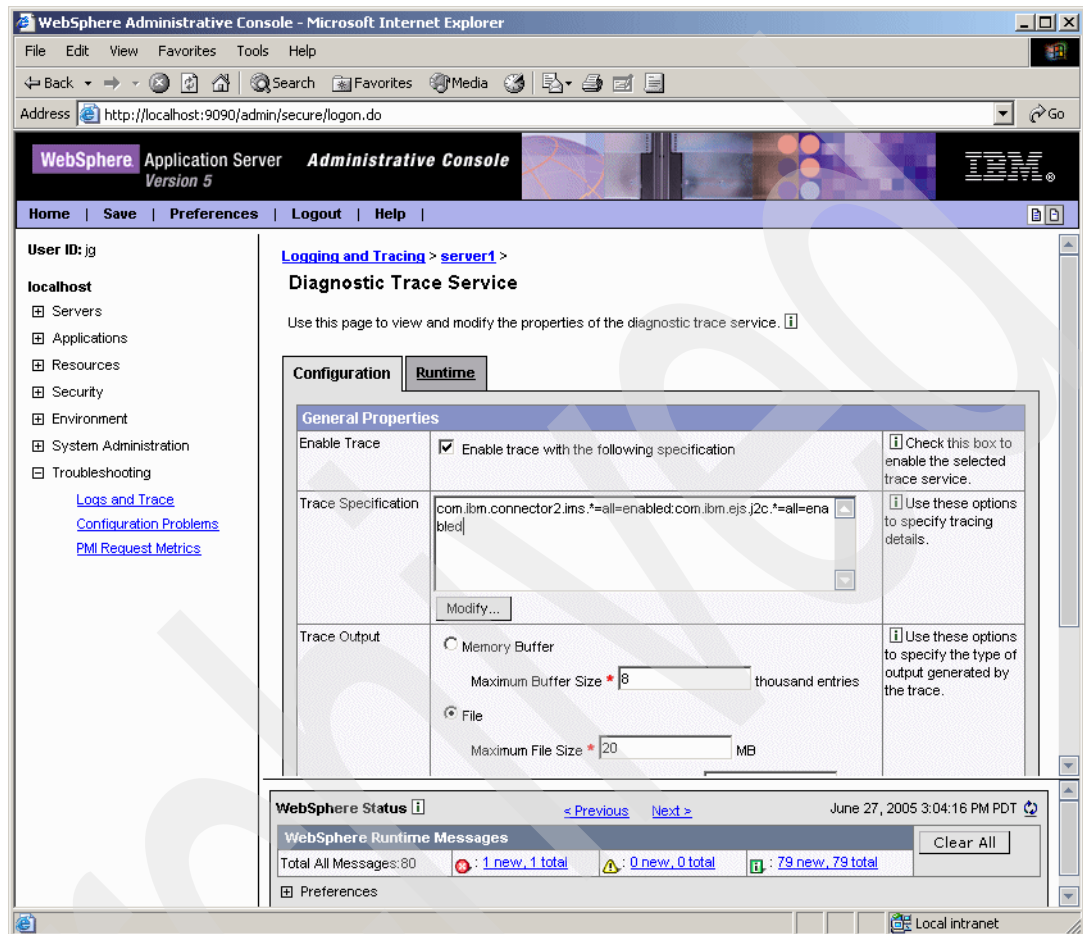


Figure 15-5 Enabling the traces using the WebSphere Application Server Administrative Console

As in the previous example, you must ensure that the **Enable Trace** check box is selected and add the following values to whatever is present in the Trace Specification text box:

```
com.ibm.connector2.ims.*=all=enabled:com.ibm.ejs.j2c.*=all=enabled
```

After that, set the trace level to 3 in the connection factory properties page (expand the **Resources** → **Resource Adapters** and then select the IMS Connector for Java resource adapter and the connection factory that you are modifying, as shown in Figure 15-6 on page 309. Scroll down the property list until you find the Trace Level property and set it to 3.

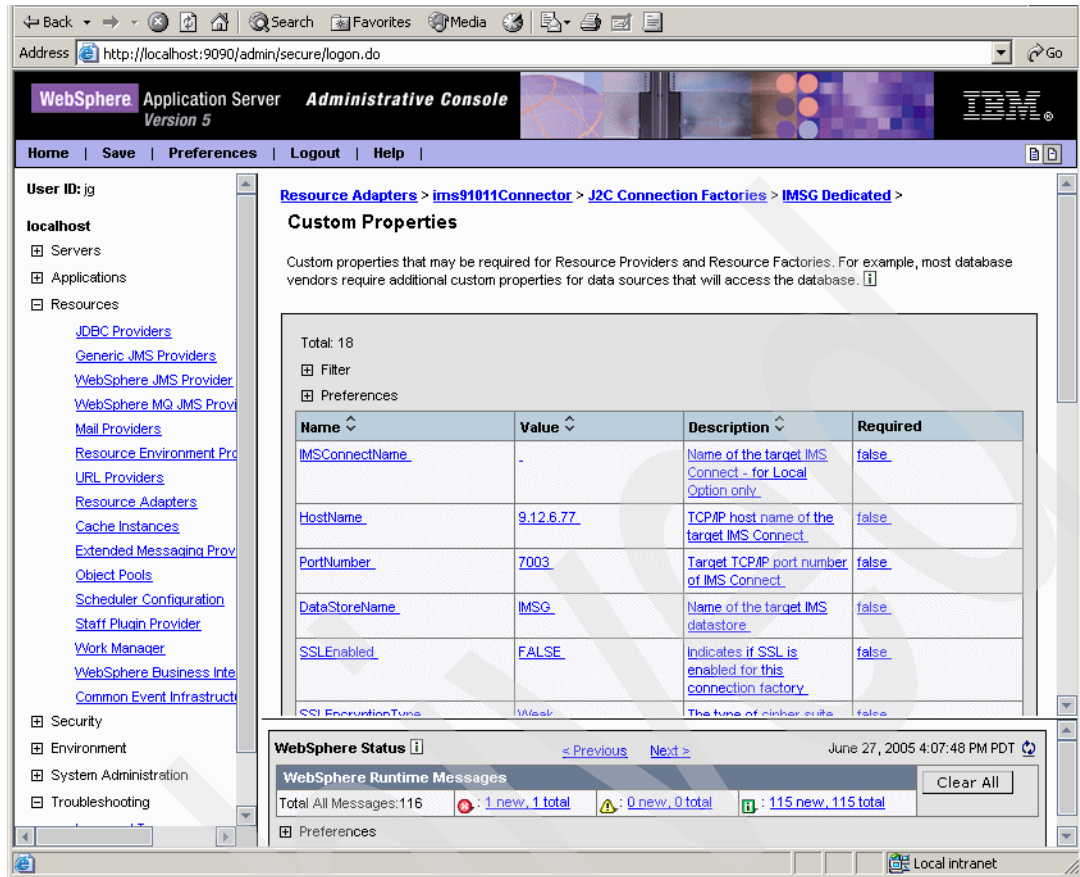


Figure 15-6 Setting the trace level using the WebSphere Application Server Administrative Console

15.4 Exceptions in IMS Connector for Java applications

If you are using the IMS Connector for Java J2C adapter, IMS Connect will not send an RSM. IMS Connector for Java will detect IMS Connect error messages and throw exceptions that your code should catch and analyze. For example, Example 15-9 shows the exception received by the client when a duplicate clientID error occurs.

Example 15-9 Exception thrown when a DUPECLNT error occurs

```
javax.resource.spi.EISSystemException: IC00001E:
com.ibm.connector2.ims.ico.IMSTCPManagedConnection@43f76c57.processOutputOTMAMsg(byte [],
InteractionSpec, Record) error. IMS Connect returned error: RETCODE=[8],
REASONCODE=[DUPECLNT]. Duplicate client ID was used; the client ID is currently in use.
```

The tools and procedure to diagnose IMS Connector for Java exceptions are exactly the same as you would use for a non-IMS Connector for Java client. Refer to 15.3, "Wrong status codes from IMS Connect" on page 302 for details. In this section, we cover only the IMS Connector for Java specific exceptions.

15.4.1 Naming (JNDI)-related errors

Java Naming and Directory Interface (JNDI) is the technology and API used to resolve logical names to specific objects in a Java runtime environment, such as the WebSphere Application Server execution environment.

In an IMS Connector for Java application, you use JNDI to look up and get a reference to an instance of the connection factory that you use to obtain the IMSConnection objects needed to send requests to IMS Connect.

To do this, you need to use two different names (although you can assign the same value to both names):

- ▶ A string, which we call the “resource reference name,” that is internal to your application, and that is used in your Java classes to do the lookup. This string can be either hard-coded in the source file, or obtained from a resource file or similar source.
- ▶ Another string, which we call the “connection factory JNDI name,” that is external to your application and that is defined at the connection factory level when the administrator deploys the IMS Connector for Java resource adapter.

Note: You need to recycle the server instance if you modify the JNDI bindings of a deployed application. You need to do this even if you undeploy and then redeploy an application after changing the JNDI bindings.

The binding between the applications, resource reference names, and the connection factory JNDI names is done in the deployment descriptor and can be changed by the package builder through the application assembly tools, the application deployer using the deployment interface (scripted or administrative console-based), or the WebSphere Application Server administrator using the administrative console. Figure 15-7 shows the components involved in the JNDI name definition and their relationship.

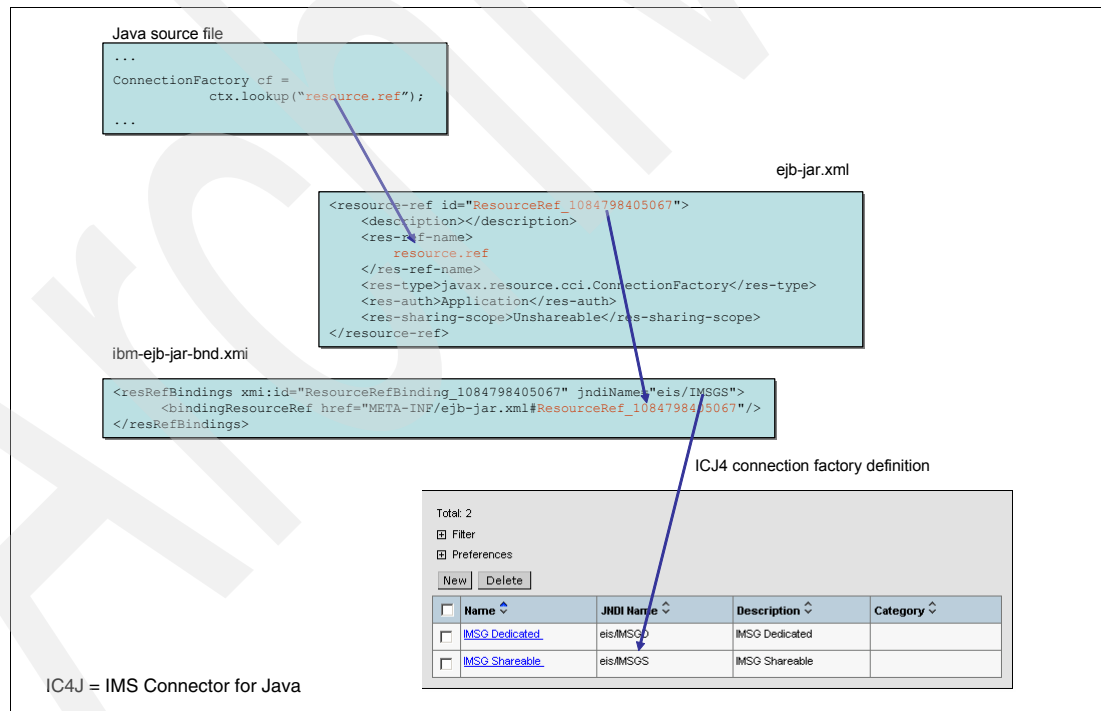


Figure 15-7 JNDI names and configuration elements

Two different exceptions can occur related to the JNDI names and binding. Example 15-10 on page 311 shows the exception thrown when the resource reference name is not found.

Example 15-10 Exception thrown when the JNDI reference name is not found

```
javax.naming.NameNotFoundException: Name
"com/ibm/sg246794/sample/ProcessTxBean/IC4JShareable" not found in context "java:comp/env".
```

To fix this problem, look at the deployment descriptor using the assembly tools, the administrative console, or the development tools and check that the reference name used in the code is exactly the same as the one used in the descriptor. If you cannot use one of these tools, you can look at the descriptor itself, in a file named `ejb-jar.xml`, and search the resource reference definitions, similar to the one shown in Example 15-11.

Example 15-11 A JNDI resource reference definition

```
<resource-ref id="ResourceRef_1084798405067">
  <description></description>
  <res-ref-name>com/ibm/sg246794/sample/ProcessTxBean/IC4JShareable</res-ref-name>
  <res-type>javax.resource.cci.ConnectionFactory</res-type>
  <res-auth>Application</res-auth>
  <res-sharing-scope>Unshareable</res-sharing-scope>
</resource-ref>
```

In this case, the code should be looking for:

```
java:comp/env/com/ibm/sg246794/sample/ProcessTxBean/IC4JShareable
```

The development tools usually generate the correct resource reference names, so unless you edited the deployment descriptors by hand, the usual cause of this problem is a change in the Java code without regenerating the deployment descriptor or a mistake in an external resource file containing resource reference names.

The second kind of JNDI problem you can find is related to the JNDI connection factory name. If JNDI cannot find it at run time, you will get the exception shown in Example 15-12.

Example 15-12 Exception thrown when the JNDI connection factory name is not found

```
Reference Factory Class Name: com.ibm.ws.util.ResRefJndiLookupObjectFactory
Reference Factory Class Location URLs: <null>
Reference Class Name: java.lang.Object
Type: ResRefJndiLookupInfo
Content: com.ibm.ws.util.ResRefJndiLookupInfo@a1b467 ResRefJndiLookupInfo: Look up
Name="com/ibm/sg246794/sample/ProcessTxBean/IC4JShareable";JndiLookupInfo:
jndiName="eis/IMSGS"; providerURL=""; initialContextFactory=""
```

Exception data follows:

```
javax.naming.NameNotFoundException: Context: localhost/nodes/localhost/servers/server1,
name: eis/IMSGSA: First component in name IMSGSA not found. Root exception is
org.omg.CosNaming.NamingContextPackage.NotFound:
IDL:org.omg.CosNaming/NamingContext/NotFound:1.0
```

In this case, we bound the following resource reference name:

```
java:comp/env/com/ibm/sg246794/sample/ProcessTxBean/IC4JShareable
```

This resource reference name is bound to the following JNDI connection factory name:

```
java:comp/env/eis/IMSGSA
```

The latter name was not found by the JNDI run time. Usually, the problem is that the person who deployed the application made a mistake at the JNDI binding phase. Check also that the IMS Connector for Java resource adapter has been properly deployed in the target runtime

environment and that the connection factory you are intending to use has been correctly created in that environment.

You will typically use the WebSphere Application Server administrative console to check and fix this error by deploying the missing connection factory, changing the JNDI name of the connection factory, or correcting the application bindings. Remember to recycle the application server instance after applying changes to the JNDI naming and binding.

Note: Be careful if you change the JNDI connection factory name, because it might be used by another application. If you change it, the other application will continue to work only until you recycle the application server instance.

15.4.2 Connection pool-related errors

If you are using IMS Connector for Java in a managed environment such as WebSphere Application Server, the creation and destruction of the physical connection objects is done by a connection manager. When you invoke the `getConnection()` method of the `ConnectionFactory` instance you obtained through JNDI, the Connection Manager gets one of the existing connections from the connection pool for that connection factory, or creates a new connection if dictated by the connection pool properties and status.

This can lead to a situation in which:

- ▶ All the existing connections in the pool are being used by other applications or threads in your application.
- ▶ The connection pool properties do not allow the pool manager to create an additional connection to satisfy your request.

In this event, the thread that is requesting the connection waits for the time specified in the Connection Timeout of the connection pool properties for that connection factory. If that time period ends without being able to get a connection, you get an exception, as shown in Example 15-13.

Example 15-13 Exception thrown when IMS Connector for Java cannot obtain a connection


```
com.ibm.websphere.ce.j2c.ConnectionWaitTimeoutException: Connection not available, Timed out waiting for 30003
```

To help you better understand the reason for this error and the measures you can take to prevent it, a brief description of the connection pool properties follows.

Connection pool properties

The WebSphere Application Server administrator can set the connection pool properties using the administrative console. Figure 15-8 on page 313 shows the connection pool properties window in a WebSphere Application Server Version 5.1 administrative console. As you can see, each field has an explanation in the right column.

Connection Pools

Connection pool properties that can be modified to change the behavior of the J2C connection pool manager. Default values are provided for non-production use. Reviewing and possible modification of these configuration values is recommended. 

Configuration

General Properties










Scope	cells:localhost:nodes:localhost	 The scope of the configured resource. This value indicates the configuration location for the configuration file.
Connection Timeout	<input type="text" value="10"/> seconds	 Interval, in seconds, after which a connection request times out and a <code>ConnectionWaitTimeoutException</code> is thrown.
Max Connections	<input type="text" value="10"/> connections	 The maximum number of <code>ManagedConnections</code> that can be created in this pool.
Min Connections	<input type="text" value="1"/> connections	 The minimum number of <code>ManagedConnections</code> that should be maintained.
Reap Time	<input type="text" value="10"/> seconds	 Interval, in seconds, between runs of the pool maintenance thread.
Unused Timeout	<input type="text" value="20"/> seconds	 Interval, in seconds, after which an unused connection is discarded by the pool maintenance thread.
Aged Timeout	<input type="text" value="30"/> seconds	 Interval, in seconds, after which an unused, aged connection is discarded (regardless of recent usage activity) by the pool maintenance thread.
Purge Policy	<input type="text" value="EntirePool"/> 	 Specifies how to purge connections when a "stale connection" or "fatal connection error" is detected.

Figure 15-8 Connection pool properties in the WebSphere Application Server administration console

Let us look at the values configured in this example:

- ▶ We set a Connection Timeout value of 10 seconds. This means that when we run out of connections, the application threads will wait for up to 10 seconds before throwing the exception in Example 15-13 on page 312.
- ▶ The value for Max Connections is 10. This means that the connection manager will respond to demand, creating connections as necessary until the number of live connections totals 10. At this point, new requests are queued and wait for the Connection Timeout setting of 10 seconds.
- ▶ The value for Min Connections is 1. The connection manager can destroy (delete) unused connections until there is just one connection left in use or in the connection pool.
- ▶ The value for Reap Time is 10. This means that the pool maintenance thread will run at 10 second intervals, checking the pool status (and deleting any unused or aged connections).
- ▶ We set the Unused Timeout value to 20. This value tells the connection manager to destroy any connections that have not been used for 20 or more seconds.
- ▶ The value for Aged Timeout is 30. This means that the connections older than 30 seconds will be destroyed at the next reap time as soon as they are free, regardless of the length of time they have been idle.

- ▶ The Purge Policy is set to EntirePool. This means that when a severe connection error occurs, the entire pool will be purged on the assumption that the same error will occur on the other connections.

Note: The sample values are far too low and have been set for demonstration purposes only. You will want to have as many available connections as possible. Of course, it does not make sense to have more connections than possible threads in your WebSphere Application Server servant process, so adjust Max Connections to no more than the maximum number of threads.

You will probably want to adjust your connection pool values according to the following guidelines:

- ▶ Set Max Connections and Min Connections to the same value. Use a value not greater than:
 - The maximum number of threads in your application server.
 - The number of message processing regions in your IMS or set of IMS systems, unless you prefer to have IMS transaction queues instead of WebSphere Application Server threads waiting for connections.
 - The MAXSOC parameter value specified in the IMS Connect configuration member for the TCP port this connection factory uses. If you set a value greater than MAXSOC, the connection will be created but will not be able to establish communication to IMS Connect and will then be put into a wait state. See 15.5, “Diagnosing problems related to sockets” on page 319.
- ▶ Set Reap Time to a value that is less than both Unused Timeout and Aged Timeout. The Reap Time interval affects the accuracy of the Unused Timeout and Aged Timeout settings. The smaller the interval, the greater the accuracy. If Reap Time is greater than the Unused Timeout and Aged Timeout values, the Reap Time will effectively override the two timeout values. The Reap Time interval also affects performance. Smaller intervals mean that the pool maintenance thread runs more often and degrades performance.
- ▶ The value of Unused Timeout will make no difference if you set Max Connections and Min Connections to the same value. Otherwise, use a value for Unused Timeout that will let the number of connection in a pool decrease slowly to avoid having to create new connections at peak times. Use your knowledge about the transaction arrival pattern in your installation to compute a good value. For example, if you have peaks every hour, set Unused Timeout to 2 hours, so your connections will survive two valleys without being purged and having to be re-created a short time later.
- ▶ You should set the value of Aged Timeout to zero *unless* you use Sysplex Distributor. If you use Sysplex Distributor, configure your Aged Timeout and Reap Time so that the pool maintenance thread periodically removes aged connections. This allows the connection manager to automatically redistribute connections from more heavily loaded servant regions and IMS Connect instances to those that are less heavily loaded. Determine the appropriate value as function of your workload. If your workload follows a day shift/night shift pattern (you have a high workload during normal business hours, and lower workload after business hours with peaks around each hour), a good value Aged Timeout might be 90 minutes and a good value for Reap Time might be 30 or 45 minutes.

Connection pool monitoring

You can use IBM Tivoli® Performance Monitor, included in the Windows version of WebSphere Application Server to monitor the connection pools. You can see how the different pools evolve (or all pools as a whole) using a numerical, historical display or a real-time graph.

To use Tivoli Performance Monitor, you must enable the Performance Monitoring Service in your WebSphere Application Server, using the administrative console. To do so, select your server by clicking **Servers** → **Application Servers** and then your server name. Scroll through the list of server properties until you find and then click the **Performance Monitoring Service** entry. Figure 15-9 shows the window that opens and the values you should set to enable the standard monitoring levels.

Note: You must recycle your server in order for any changes you make in this configuration window to take effect.

[Application Servers](#) > [server1](#) >

Performance Monitoring Service

Configuration and Runtime Settings for Performance Monitoring Infrastructure (PMI) ⓘ

Runtime **Configuration**

General Properties

Startup	<input checked="" type="checkbox"/>	ⓘ Specifies whether the server will attempt to start the specified service when the server starts.
Initial specification level	<input type="radio"/> None - All modules below set to "N" (None). <input checked="" type="radio"/> Standard - All modules below set to "H" (High) <input type="radio"/> Custom - Modify, add or remove the modules from the below list.	ⓘ A Performance Monitoring Infrastructure (PMI) specification string that stores PMI specification levels for all components in the server. Levels N,L,M,H,X represent None,Low,Medium,High,Maximum respectively.

alarmManagerModule=H
beanModule=H
cacheModule=H
connectionPoolModule=H
j2cModule=H

Apply OK Reset Cancel


Additional Properties

[Custom Properties](#) Additional custom properties for this service which may be configurable.

Figure 15-9 Enabling Performance Monitoring Service using the administrative console

You must also know the port number assigned to the WebSphere Application Server SOAP service. Using the administrative console, find the End Points entry in the Server Attribute list, as you did for Performance Monitoring Service. Click the **Server Attribute** list and you will find a list of services or “End Points” defined in your WebSphere Application Server instance. Click **SOAP CONNECTOR ADDRESS** and take note of the Port number shown, as shown in Figure 15-10 on page 316.

[Application Servers](#) > [server1](#) >
SOAP_CONNECTOR_ADDRESS

Configure important TCP/IP ports which this server uses for connections. 

Configuration

General Properties




End Point Name	SOAP_CONNECTOR_ADDRESS	
Host	* localhost	 The IP address, DNS host name with domain name suffix, or just the DNS host name, used by a client to request a Web application resource (such as a servlet, JSP, or HTML page).
Port	* 8880	 The port for which the Web server has been configured to accept client requests. Specify a port value in conjunction with the host name.

Figure 15-10 Finding the SOAP Port number using the administrative console

After you know the port number (and the IP address of your server), you can start the Tivoli Performance Viewer and select the JCA connectors section to look at your connection pools. You will find one entry for each active connection factory, which you can select individually. You can also select all the factories of a JCA adapter (such as IMS Connector for Java) or all factories active in this server instance.

Figure 15-11 on page 317 shows a graph corresponding to a well-behaved application. In this example, a “burst” of 55 transactions was sent using 11 concurrent threads. You can see in the graph how the number of connections goes from zero to 10 (the scale is multiplied by 5). The grey line shows the number of allocated connections (connection handles) going from zero to almost 10, and after that, going back to zero. That means that the application correctly returns the connections to the pool so that they can be reused. The number of free connections (red line) climbs to 10 after the activity burst has ended, and after some time (set by the Unused Timeout parameter value) the unused connections are freed.

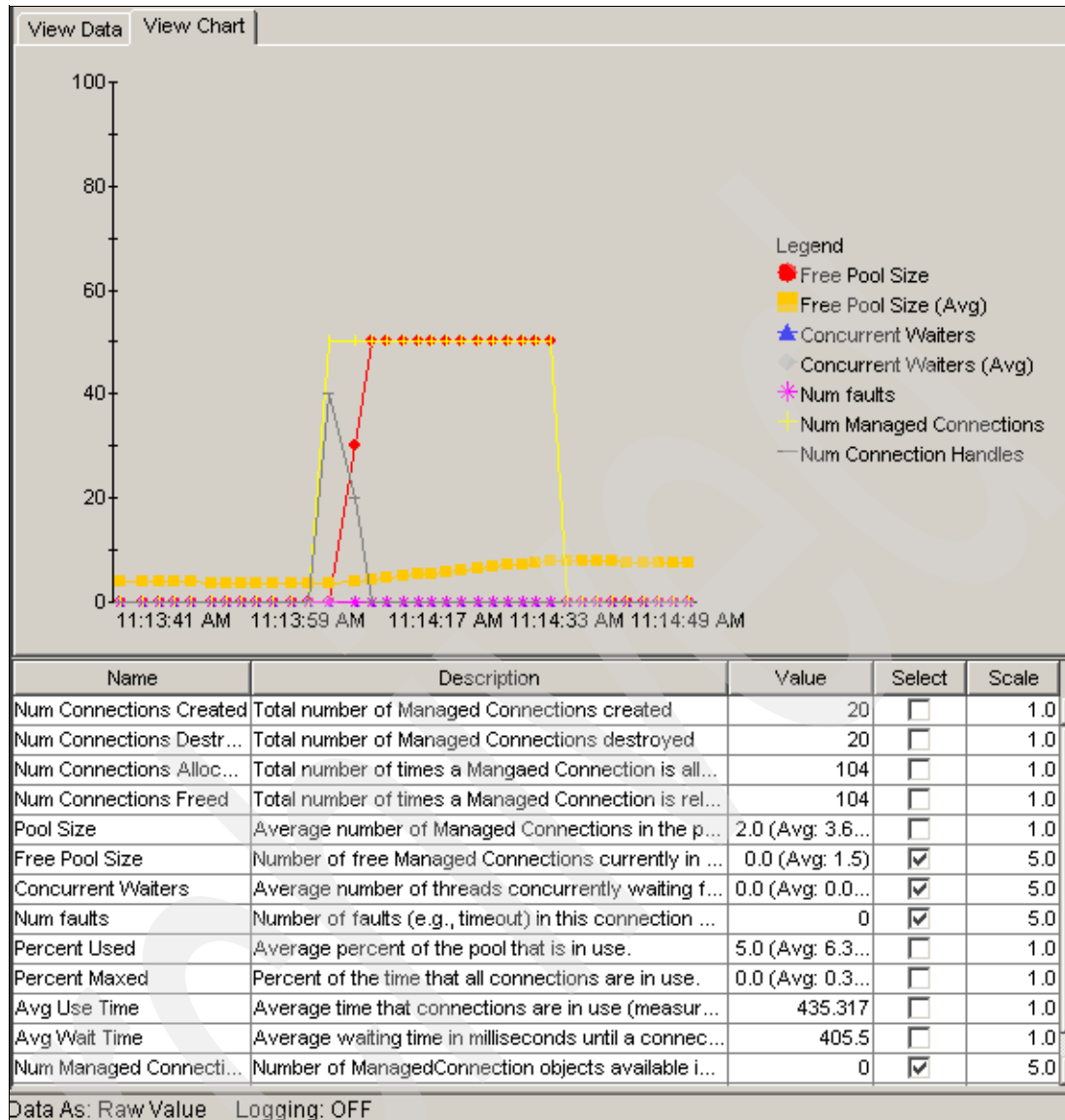


Figure 15-11 A well-behaved application seen through Tivoli Performance Monitor

Figure 15-12 on page 318 shows the graph for an ill-behaved application. In this case, the application does not return the connections to the pool. The application does not invoke the `close()` method for the `Interaction` or the `Connection` objects, so the connection manager does not know if the application will reuse the same objects later, and thus does not mark them free. In this graph, you can see the pool size increasing to 10, which is the Max Connections value. Then, you can see that some threads start to wait (pink line), and start to get connection timeouts (Num faults line). This is a symptom of a poorly-written program or poorly-configured resource. You should check:

- ▶ The use of `close()` for the `Interaction` and `Connection` objects is necessary to reuse the connections.
- ▶ If you are using bean-managed transactions, you should `commit()` them.

- Be careful to not *own* a connection handle outside the scope of a transaction. That can happen if you save a reference to a Connection object as a class or object variable, or if you save it in the session object or application context. You should not save references to Connection objects in a managed environment. It is a much better programming practice to leave the job of reusing and recycling connections to the connection manager in a managed environment. If you want to manage your connections in your own code, you should be using non-managed connections. Otherwise, there is a high probability that what you do will conflict with the what the connection manager is doing, leading to possible severe errors and performance problems.

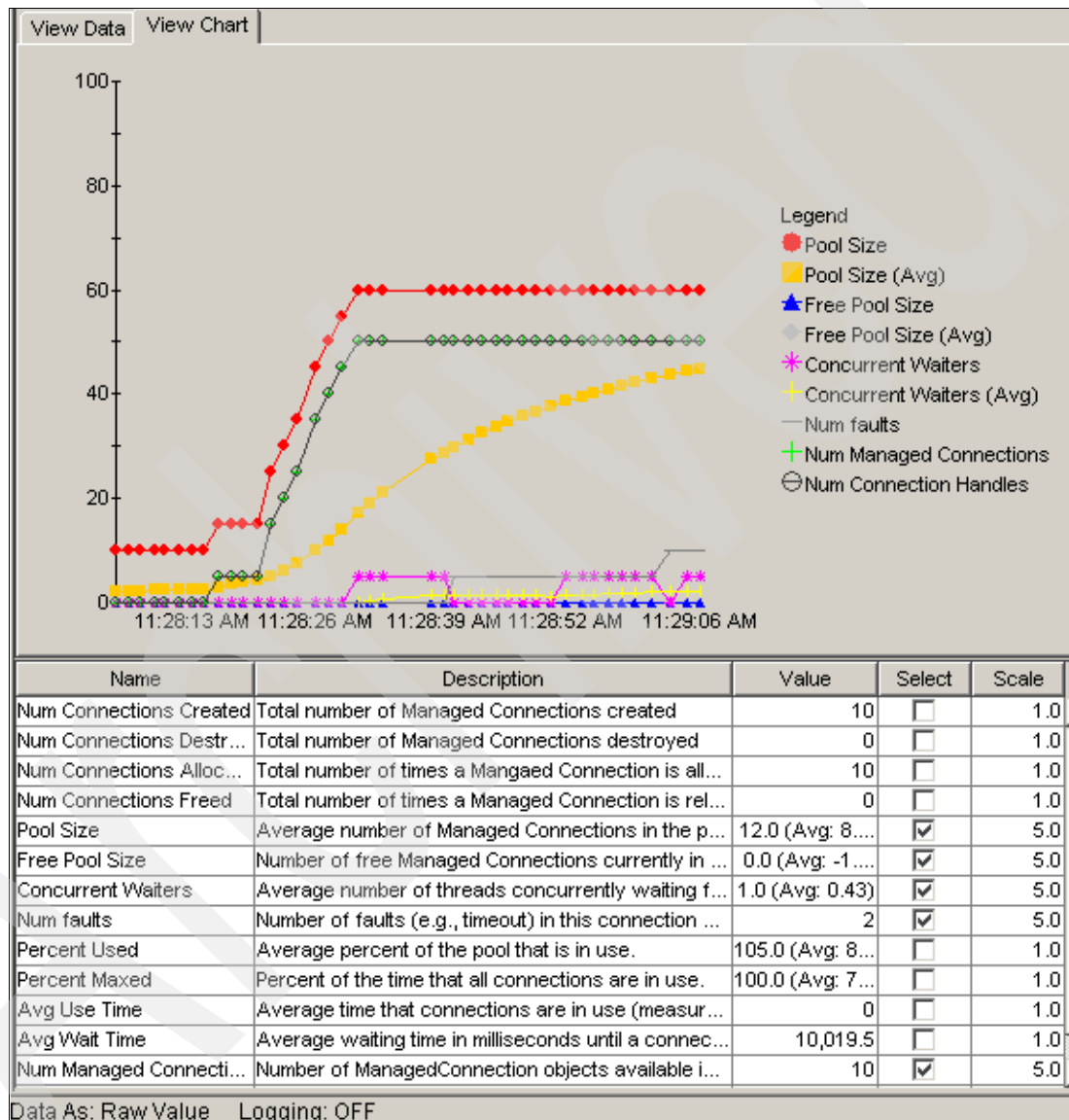


Figure 15-12 An ill-behaved application seen through Tivoli Performance Monitor

15.5 Diagnosing problems related to sockets

If the IMS Connect environment appears to be suspended or if the Java application or servlet (using a valid host name and port for the target IMS Connect system) fails to make a connection to the IMS Connect system, you might have reached the maximum number of sockets. The maximum number reached can either be due to incorrect parameter definitions in IMS Connect, IMS Connector for Java, or in z/OS UNIX System Services.

15.5.1 IMS Connect and IMS Connector for Java parameters for sockets

The number of sockets that can exist between a Java client (a Java application or servlet) and the host component, IMS Connect, is controlled by two parameters that need to be synchronized. These parameters are MAXSOC in the TCPIP statement of the IMS Connect configuration member and Max Connections, a property of the connection factory, as shown in 15.4.2, “Connection pool-related errors” on page 312:

► **MAXSOC**

The maximum number of sockets for an IMS Connect port. MAXSOC applies to all of the ports associated with an instance of IMS Connect. MAXSOC is displayed in the MAXSOC field of the IMS Connect commands VIEWHWS and QUERY MEMBER. The default value for MAXSOC is 50.

The value displayed in the MAXSOC field includes one socket dedicated to the Listen State and the remainder available for connections. For example, if MAXSOC is 50, only 49 can be used by connections from Java clients (Java applications or servlets).

Example: If a particular host machine has a single instance of IMS Connect with two ports, and if MAXSOC is 50, each port can have 49 sockets (connections) from Java clients, yielding a maximum of 98 sockets into the IMS Connect instance.

When the MAXSOC limit is reached, IMS Connect enters a wait state and returns at regular time intervals to check for available sockets. If the Java applications or servlets are not releasing their sockets, your environment can appear to be suspended. If this situation occurs, take one of the following actions:

- Consider increasing the maximum number of sockets.
- Make sockets available by cancelling one or more of the Java applications or servlets that are holding sockets to the particular IMS Connect instance.
- Assign a non-zero value to the Aged Timeout or Unused Timeout of the connection pool, or both.

► **Max Connections**

This is a property of the connection pool properties of the connection factory and applies only when you are running your application in a managed environment, such as in WebSphere Application Server. See 15.4.2, “Connection pool-related errors” on page 312 for information about this and other connection pool properties.

Recommendation: We recommend that you set Max Connections to a value less than MAXSOC.

15.5.2 z/OS UNIX System Services parameters for sockets

There are also some z/OS UNIX System Services parameters that you need to review if IMS Connect is unable to open new socket. These parameters usually reside in the BPXPRMxx member in SYS1.PARMLIB. The parameters MAXFILEPROC and MAXSOCKETS are critical in order to be able to open socket connections. You should also review the other parameters, but these two are the most common source for problems when opening socket connections. Consult with your system programmer responsible for these parameters to ensure that these parameters have values that are large enough to accommodate your maximum projected workload.

- **MAXFILEPROC**

MAXFILEPROC specifies the maximum number of files that a single user is allowed to have concurrently active or allocated. Because the socket is considered a generic file in UNIX, the MAXFILEPROC setting applies to sockets as well as files. That means that one user, in our case, the IMS Connect address space, might reach the MAXFILEPROC value if the value is set too low. This is more likely to happen in cases when persistent sockets are used (because persistent sockets are longer lasting by their nature).

You can view the actual value of MAXFILEPROC by using the D OMVS,O command and then you can change the value dynamically using the SETOMVS command. For more information about these commands, refer to *z/OS V1R6.0 MVS System Commands*, SA22-7627.

- **MAXSOCKETS**

MAXSOCKETS specifies the maximum number of sockets that can be opened by the users of a particular TCP/IP stack. It is specified in the network statement for a TCP/IP stack's physical file system and should be set large enough to allow new sockets to be opened as needed. Note that this number limits the number of sockets of all users of a particular TCP/IP stack. Exceeding MAXSOCKETS is also more likely to happen if the socket connections are long-lasting. You can reach this number even if there are not many IMS Connect sockets if there are other users such as Telnet sessions.

You can display the actual value of MAXSOCKETS by using the D OMVS,P command. Note that the command output also includes a field showing the highest number of concurrent sockets (HIGHUSED) used since the start of this TCP/IP stack. If HIGHUSED is equal to MAXSOCKETS, it indicates that you have experienced a situation where new sockets were unable to be opened. In this case, consider increasing the MAXSOCKET value.

For more information about setting the MAXSOCKETS value, refer to *z/OS V1R6.0 CS: IP Configuration Reference*, SC31-8776.

IMS MFS Web Services

IMS MFS Web Services lets you define a service from a Message Format Service (MFS) source and then deploy it to WebSphere Application Server and make it available as an EJB service and a Java service.

You build an MFS-based service by importing MFS source files into the WebSphere Studio Application Developer Integration Edition and generating Web Service Definition Language (WSDL) files. The WSDL files are based on application interface information that is stored in the MFS source. The WSDL files contain details about the service including:

- ▶ Interface elements that expose the operations and messages provided by the service
- ▶ Provider-specific binding elements that describe how the service interface is implemented
- ▶ Service and port elements that enable client applications to locate the service

In this chapter, we provide an overview of MFS Web Services.

16.1 IMS MFS Web Services introduction

MFS Web Services are built by generating Web Service Definition Language (WSDL) files. The MFS WSDL files contain only device information. For the input part of the WSDL message, only device fields that map to input message fields are used; and on the output side, only device fields that map to output message fields are used.

A Web services client, which can be a Java or EJB application using a Web Services Invocation Framework (WSIF), looks up the WSDL file generated from the MFS importer. Based on the WSDL, the Web services client starts the service. The service request then invokes the MFS Adapter.

The MFS Java or EJB clients use the MFS Adapter to translate a service message into a byte stream to input to IMS Connector for Java and also to translate a response in byte stream to a service message, as shown in Figure 16-1.

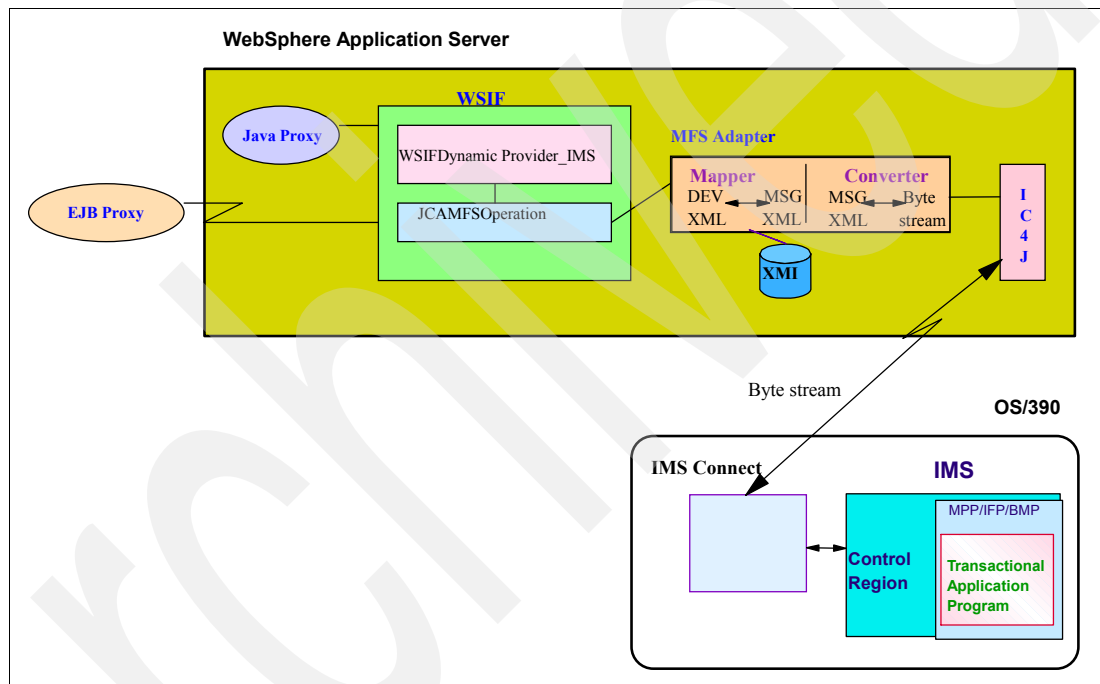


Figure 16-1 MFS Web Services in WebSphere Application Server

16.2 IMS MFS Web Services development process overview

The following steps show the typical development processing steps for IMS MFS Web Services:

1. Download valid MFS source files from z/OS and place them in a directory that is accessible to WebSphere Studio Application Integration Edition. If the MFS source is not available, you can create it from MFS format libraries by running the IBM IMS Message Format Services Reversal Utility. For more information, see *IMS Message Format Services Reversal Utilities for z/OS User's Guide*, SC27-0823.
2. Import the MFS source files into the workbench and generate the service:
 - a. The Create Service wizard generates the service WSDL files (Interface WSDL, Binding WSDL, and Service WSDL).

- b. The MFS Importer wizard specifies the input and output messages for your new operation. The wizard reads the MFS source file and parses it. If the MFS source file successfully parses, the MFS Importer wizard generates XML Metadata Interchange (XMI) files. The XMI files represent information about the application interface that is encapsulated by the MFS source, including:
 - Input and output message fields (MID/DIF, MOD/DOF)
 - Display information
 - MFS flow control
 - Device characteristics
 - Operation semantics
 - c. The MFS Importer wizard populates the input and output XML Schema Definition Language (XSD) fields in the interface WSDL file and adds binding information to the binding WSDL file.
3. Generate the deploy code for the service. The deploy code includes the session bean that handles client requests to the service and deploy classes that allow your session bean to operate on an EJB server.
4. Deploy the service and supporting files to a production server and publicize the service by a Universal Description, Discovery, and Integration (UDDI) registry. Client applications can then use the service to access information in your MFS-based transaction.

16.3 IMS MFS Web Services supported features

IMS MFS Web Services supports the following MFS features:

► Application output with MOD name

At runtime, an application program can change the format that is used to map the current output message by passing back a MOD name. The MFS runtime component uses this MOD name to format the output.

► Logical pages

You can select a specific logical page that is associated with the MID to use for input. For output, an output message is defined with one or more logical pages (LPAGE). Each LPAGE relates one segment, or a series of segments, produced by an application program to a corresponding device format.

► Physical pages

A logical page can contain one or more physical pages. Physical paging allows data from a logical page to be displayed in several physical pages on the device. For display devices, the size of a physical page is defined by the screen capacity (the number of lines and columns that can be referred to).

► Message option 1 and 2 (input)

Message option 1 depends on whether a fill character of NULL has been defined. When NO field in an option 1 message is defined as having a fill character of null:

- The message always contains the defined number of segments.
- Each segment is of the defined length and contains all defined fields.
- All fields are filled with data, data and fill characters, or fill characters.

When fields in an option 1 message are defined as having a fill character of null:

- Each field with null fill and no input data from the device is eliminated from the message segment. If all fields in a segment are eliminated in this manner and no literals (explicit or default) are defined, the segment is eliminated; otherwise, the length of the segment is reduced and relative position of succeeding fields in the segment is altered.
- Fields with null fill that receive data that does not fill the field are not padded. The number of characters received for the device field becomes the number of characters of the input data. This alters the length of the segment and the relative position of all succeeding fields in the segment.

Message option 2 formats messages in the same way that option 1 does, unless the segment contains no input data from the terminal after the literals have been removed. If a segment does not contain any data and there are subsequent segments that contain data, a null segment is created. A null segment contains only a X'3F' character.

► Message option 1 and 2 (output)

All fields in option 2 output segments are defined as fixed length and fixed position. The data in the fields can be truncated or omitted by two methods:

- Inserting a short segment
- Placing a null character (X'3F') in the field

Fields are scanned from left to right for a null character. The first null character that is encountered terminates the field. If the first character of a field is a null character, the field is omitted (depending on the fill character used). Positioning of all fields in the segment remains the same regardless of null characters. Fields truncated or omitted are padded as defined to the MFS Language Utility.

If ATTR=YES is specified in the MFLD definition, and if X'3F' is the first or second byte of the attribute portion of the field, the field is omitted and the attributes that are specified on the device field (DFLD) statement are used.

► System literals for date, time, and LPAGENO

MFS Web Services supports system literals for date, time, and LPAGENO.

► Function keys that map to literal data

MFS Web Services supports function keys that map to literal data. MFS Web Services does not support PF keys that map to control functions or PA keys. If an IMS command is specified as a PF key that maps to literal data, the command must be supported by IMS OTMA.

► Double-byte character sets (DBCS)

Double-byte character sets (DBCS) are graphical character sets in which each character is represented by two bytes. The valid DBCS code range is X'4040' or X'41' through X'FE' for byte 1, and X'41' through X'FE' for byte 2. DBCS is used when the number of characters in some written languages is more than 256 characters.

Because DBCS is a subset of the Extended Graphic Character Set (EGCS), DBCS fields are specified by using EGCS keywords and parameters, and they are treated by MFS in much the same way as EGCS data. However, DBCS data can be used in two field types, a DBCS field and a mixed DBCS and EBCDIC field. The DBCS field accepts only DBCS data and no special control characters are needed with this type of field. But, in a mixed field, where DBCS data is mixed with EBCDIC data, the DBCS data must be enclosed by Shift Out (SO) and Shift In (SI) control characters. To specify a DBCS and EBCDIC mixed field, the keyword MIX or MIXD should be used for the EATTR parameter in a DFLD statement.

Because the WebSphere Studio Integration Application Integration tools for building enterprise services target local and LAN files, download MFS source files (which normally reside on z/OS) to a workstation or LAN server, and then import the source files into the workbench. You can download MFS source files in either binary or text format; however, if an MFS source file contains double-byte characters, you must download the file in binary format. Downloading a DBCS file in text format will result in a parser error when the source file is parsed by the MFS Importer wizard.

16.3.1 Supported device types

The MFS Transformer only supports MFS source formats for 3270 and 3270-An devices.

16.3.2 Supported MFS statements

The following MFS statements are supported:

- ▶ MSG
- ▶ LPAGE
- ▶ PASSWORD
- ▶ SEG
- ▶ MFLD
- ▶ FMT
- ▶ DEV
- ▶ DIV
- ▶ DPAGE
- ▶ DFLD
- ▶ TABLE
- ▶ IF

The following statements are not modeled but implicitly supported through parser:

- ▶ ALPHA
- ▶ COPY
- ▶ DO
- ▶ END
- ▶ ENDDO
- ▶ EQU
- ▶ FMTEND
- ▶ MSGEND
- ▶ RESCAN
- ▶ STACK
- ▶ TABLEEND
- ▶ UNSTACK

16.4 IMS MFS Web Services limitations

The following features are not supported:

- ▶ Conditional logical paging
- ▶ EGCS (NLS Support)
- ▶ IMS system-generated MFS parameters such as PAGDEL
- ▶ Magnetic strip reading device
- ▶ Message option 3
- ▶ MFS Bypass
- ▶ MFS Buffer Pool

- ▶ MFS Field Exit routine
- ▶ MFS Pool Manager
- ▶ MFS Segment Exit routine
- ▶ Multiple physical page input
- ▶ Operator control table
- ▶ Operator logical paging
- ▶ PA2 key to advance to the next message
- ▶ PA3 key (copy to the local printer)
- ▶ Password
- ▶ PF keys defined for anything besides transactions and commands
- ▶ Programmed symbols (for example, scientific or technical symbols)
- ▶ Selector pen
- ▶ System Control Area (SCA)
- ▶ System literal defined for anything other than date and time
- ▶ \$\$IMSDIR (resident directory)

The following types of messages are not supported:

- ▶ Asynchronous messages (due to IMS Connector for Java restriction)
- ▶ Conversational transactions (due to IMS Connector for Java restriction)

16.5 Adding operations, messages, and bindings

For adding operations, messages, and bindings from MFS source, refer to the WebSphere Studio Application Developer Integration Edition Information Center, available at:

<http://publib.boulder.ibm.com/infocenter/adiehelp/index.jsp>

To find this topic, navigate with your Web browser as follows:

1. Click **WebSphere Studio**.
2. Expand **Developing** → **Expand Enterprise services** → **IMS services** → **Developing your application** → **Expand Building an IMS service** → **Generating an enterprise service for an IMS transaction**.
3. Select **Adding operations, messages, and bindings from MFS source**.

16.6 Creating an enterprise service

For creating an enterprise service for an MFS-based IMS transaction, refer to the WebSphere Studio Application Developer Integration Edition Information Center, available at:

<http://publib.boulder.ibm.com/infocenter/adiehelp/index.jsp>

To find this topic, navigate with your Web browser as follows:

1. Click **WebSphere Studio**.
2. Expand **Developing** → **Expand Enterprise services** → **Expand IMS services** → **Expand Samples**.
3. Select **Creating an enterprise service for an MFS-based IMS transaction**.

16.7 Deploying an MFS-based IMS enterprise service

For deploying an MFS-based IMS enterprise service to a production server, refer to the WebSphere Studio Application Developer Integration Edition Information Center, available at:

<http://publib.boulder.ibm.com/infocenter/adiehelp/index.jsp>

To find this topic, navigate with your Web browser as follows:

1. Click **WebSphere Studio**.
2. Expand **Developing** → **Expand Enterprise services** → **Expand IMS services** → **Expand Samples**.
3. Select **Deploying an MFS-based IMS enterprise service to a production server**.

Archived

IMS MFS Web Enablement

IMS MFS Web Enablement modernizes existing MFS-based IMS transactions in a business-to-consumer (B2C) environment. The support Web-enables existing or new IMS MFS-based applications in IBM WebSphere Application Server and interactively renders them for display in standard browsers such as Microsoft Internet Explorer and Mozilla Firefox.

First, we describe the user experience of MFS Web Enablement. Then, we look at the details of the MFS Web Enablement tooling and the runtime components. The MFS Web Enablement tooling utility support consists of the MFS XML Utility and the MFS Importer. MFS Web Enablement runtime support consists of application instance servlets, sample cascading style sheets, the MFS Servlet, and the MFS Adapter.

17.1 How does IMS MFS Web Enablement work?

At development time, the IMS MFS XML Utility invokes the MFS Importer to parse MFS source files of existing MFS-based IMS transactions and generates Java classes and metadata files for runtime processing. The Java classes are packaged into a Web application archive (WAR) file and deployed on WebSphere Application Server.

During run time, the HTTP request from a browser is sent to the Web application, which works in conjunction with the IMS MFS Web Enablement runtime component. If this request is the initial request, a new HTTP session is created and the initial Web page is returned, which simulates the 3270 type terminal blank screen. Subsequent requests containing input data, which can be transaction or command, are transformed into an input byte stream to be sent across to the IMS host application. The output byte stream coming back from the application is transformed into XML data objects that are stored in the HTTP session. The XML data object contains one or more physical pages to be displayed as Web pages. Web pages are returned in the HTTP response, one page per 3270-PA1-equivalent paging request.

As depicted in Figure 17-1, the IMS Installation Verification Program Phonebook application is invoked to display a phonebook entry. The input request, coming in from the Web browser, is transformed into an input byte stream and sent to the IMS host application. The output byte stream is transformed, and the Web page containing output data is generated and returned.

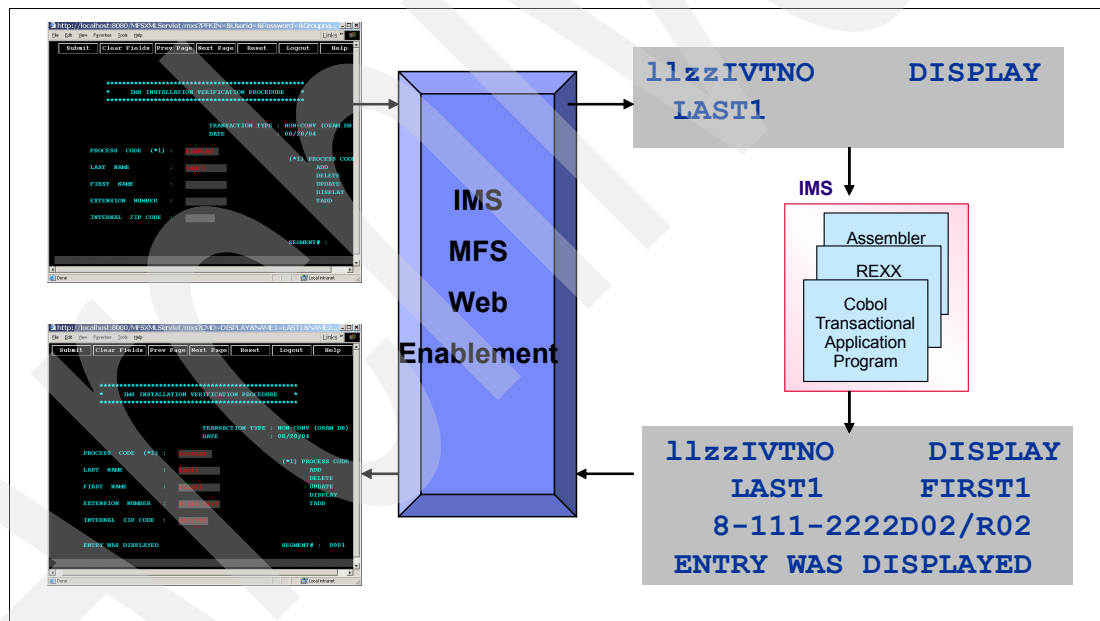


Figure 17-1 IMS IVP Phonebook application as an MFS Web-enabled Web service

17.2 IMS MFS XML Utility

The MFS XML Utility invokes the MFS Importer to parse MFS source files and generates XML Metadata Interchange (XMI) files for each message input descriptor (MID) or device input format (DIF) pair, message output descriptor (MOD), or device output descriptor (DOF) pair, and MFS table.

17.2.1 Overview of the MFS XML Utility

The MFS XML Utility is a command line development-time tool that runs on an MS-DOS-based command prompt. This utility generates all of the necessary files needed to Web enable MFS-based IMS transactions. It takes MFS source files as input and produces metadata XML files and Web application archive (WAR) files as output. In addition, the utility provides FTP client support to transport the generated output to an application server such as WebSphere Application Server.

17.2.2 User modes

You can run the MFS XML Utility in three different modes:

► Novice

Novice mode is the default user mode. In novice mode, you are prompted on each input instruction until all instructions are complete. Novice mode is designed for new users and users that prefer more guidance.

► Expert

In expert mode, you have the flexibility of specifying all of the input values as flag parameters in one command. To use expert mode, you must know the device type and device feature in advance. If you do not know the available device types and features from the parse result of the MFS source files, run in novice mode so that you can see the available selections. Each input field is specified by a flag that precedes the value. The expert mode set of parsing uses the parameters shown in Table 17-1.

Table 17-1 MFS XML Utility step 1 expert mode parameters

Input	Syntax
Device Characteristics Table (optional)	-d or -deviceTableFile
Binary source files (True or False)	-b or -binarySource
Host codepage	-hc or -hostCodepage
Source codepage	-sc or -sourceCodepage
Device type	-dt or deviceType
Device feature	-df or -deviceFeature
Output directory	-o or -outputDirectory
MFS source files	-f or -sourceFile

All of the parameters shown in Table 17-1 are required except for the Device Characteristics Table, which is optional. All of the required parameters have a default value except for the MFS source files. You can also see more detailed information by typing /help on the command prompt after step 1 is selected in the MFS XML Utility menu. The expert mode uses the set of servlet flag parameters shown in Table 17-2.

Table 17-2 MFS XML Utility step 2 expert mode parameters

Input	Syntax
Name of the instance servlet	-n or -instanceServletName
Platform where WebSphere Application Server is located	-sp or -serverPlatform
Name and location of style sheet	-ls or -localStylesheet

Input	Syntax
Name of the host machine	-ht or -hostname
Port number	-p or -port
IMS name	-i or -ims
RACF user name (optional)	-u or -rUserName
RACF group name (optional)	-g or -rGroup
RACF password (optional)	-pw or -rPassword
Trace level for IMS Connector for Java (optional)	-t or -traceLevel
Execution timeout for IMS Connector for Java (optional)	-e or -executionTimeout
Socket timeout for IMS Connector for Java (optional)	-s or -socketTimeout

► Batch

In batch mode, you can rerun previously saved commands from Step 1: Generating XMI files from MFS source files or Step 2: Generating the instance servlet and the web.xml files to generate the XMI and instance servlet files.

17.2.3 Invoking the MFS XML Utility

The MFS XML Utility is invoked by starting the mfsxml.bat file using an MS-DOS-based command prompt window.

Step 1: Generating XMI files from MFS source files

The MFS XML Utility invokes the MFS Importer and uses the Eclipse Modeling Framework to serialize each MID/DIF pair, MOD/DOF pair, and MFS TABLE into an XMI file. The XMI files contain all of the application metadata information from the MFS source, including the input and output device descriptors, message descriptors, MID-MOD chaining, device characteristics, and operation semantics. The generated XMI files are transferred to an XMI repository on WebSphere Application Server and are read for data transformation during run time.

The optional device characteristics table file specifies the screen size of certain device types. Transfer the file in binary format from MVS™ to the system running the MFS XML Utility. You can transfer the MFS source files from MVS in either text or binary format. Because of this, you must indicate which mode you are using to transfer the files.

The MFS Importer is invoked after you specify the host codepage. If the parsing of a specific MFS source file results in a warning, the XMI files will still be generated. However, if the parsing resulted in errors, the XMI files will not be generated, and you return to the MFS XML Utility menu. Example 17-1 shows the MID, MOD, DIF, and DOF blocks in an MFS source file.

Example 17-1 MID, MOD, DIF, and DOF blocks in an MFS source file

```

IVTNOMI1 MSG TYPE=INPUT,SOR=(IVTNOF,IGNORE),NXT=IVTNO
...
IVTNOMI2 MSG TYPE=INPUT,SOR=(IVTNOF,IGNORE),NXT=IVTNO
...
IVTNO MSG TYPE=OUTPUT,SOR=(IVTNOF,IGNORE),NXT=IVTNOMI1
...
IVTNOF FMT
...

```

Parsing this file, three XMI files are generated:

- ▶ IVTNOMI1.xmi containing MID (IVTNOMI1) and DIF (IVTNOF) metadata
- ▶ IVTNOMI2.xmi containing MID (IVTNOMI2) and DIF (IVTNOF) metadata
- ▶ IVTNO.xmi containing MOD (IVTNO) and DOF (IVTNOF) metadata

After the XMI files are parsed, you are prompted to select a device type and feature. The output directory specifies where to save the generated XMI files on the local system.

Step 2: Generating the instance servlet and the web.xml files

In step 2, you generate the instance servlet class files and web.xml files for packaging into a Web application archive (WAR) file in step 3. Note that you can run through step 2 multiple times to generate different connection settings.

Here is some information about the different parameters in step 2. These parameter values are stored in the web.xml file, which is packaged into the WAR file for deployment on WebSphere Application Server:

- ▶ **Server platform**

The server platform is the platform where WebSphere Application Server is located. Select 1 for a Microsoft Windows-based server or select 2 for UNIX-based server.

- ▶ **MFSStyleSheet**

The style sheet URI to use for rendering the HTML page.

- ▶ **hostname**

The IMS host name to which to connect.

- ▶ **portNumber**

The port number of the IMS host.

- ▶ **dataStore**

The datastore of the IMS host.

- ▶ **traceLevel**

The trace level settings for IMS Connector for Java:

- Trace level 0: IMS trace level RAS_TRACE OFF for no tracing.
- Trace level 1: Lists only errors and exceptions.
- Trace level 2: Adds entry and exit methods.
- Trace level 3: Prints the contents of buffers sent to and received from IMS Connect.

If the trace level is set 1, 2, or 3, the trace output is sent directly to WebSphere Application Server's trace log file.

- ▶ **executionTimeout (optional)**

The IMS resource adapter execution timeout. The execution timeout value for the IMS resource adapter is defined as the maximum amount of time allowed for IMS Connect to send a message to IMS and receive a response from IMS. The execution timeout value is represented in milliseconds and must be a decimal integer in the range of 1 to 3600000. The recommended value is 5000 milliseconds.

- ▶ **socketTimeout (optional)**

The IMS resource adapter socket timeout. The socket timeout is the maximum amount of time IMS Connector for Java will wait for a response from IMS Connect before disconnecting the socket and returning an exception to the client application.

With the socketTimeout parameter, you can set individual timeout values for a particular interaction using a socket. The value, in milliseconds, can be set on the socketTimeout

parameter in IMSInteractionSpec. If the socketTimeout property is not specified for an interaction or if it is set to zero milliseconds, there is no socket timeout and the connection will wait indefinitely. The default socket timeout value is zero. The recommended value is 5000 milliseconds.

- ▶ **userName** (optional)
The RACF user name.
- ▶ **password** (optional)
The RACF password.
- ▶ **groupName** (optional)
The RACF group name.

Example 17-2 provides a sample web.xml file.

Example 17-2 Sample web.xml file

```
<!DOCTYPE web-app (View Source for full doctype...)>
<web-app>
<servlet>
<servlet-name>PhoneBookServlet</servlet-name>
<servlet-class>PhoneBookServlet</servlet-class>
<init-param>
<param-name>hostName</param-name>
<param-value>www.example.com </param-value>
</init-param>
<init-param>
<param-name>dataStore</param-name>
<param-value>IMS1</param-value>
</init-param>
<init-param>
<param-name>portNumber</param-name>
<param-value>9999</param-value>
</init-param>
<init-param>
<param-name>serverPlatform</param-name>
<param-value>2</param-value>
</init-param>
<init-param>
<param-name>MFSStyleSheet</param-name>
<param-value>file:/c:/stylesheets/exampleIEN6.xsl</param-value>
</init-param>
<init-param>
<param-name>traceLevel</param-name>
<param-value>3</param-value>
</init-param>
<init-param>
<param-name>executionTimeout</param-name>
<param-value>5000</param-value>
</init-param>
<init-param>
<param-name>socketTimeout</param-name>
<param-value>5000</param-value>
</init-param>
<init-param>
<param-name>userName</param-name>
<param-value>KEVIN</param-value>
</init-param>
<init-param>
<param-name>Password</param-name>
```

```
<param-value>L0</param-value>
</init-param>
<init-param>
<param-name>groupName</param-name>
<param-value>KGROUP</param-value>
</init-param>
</servlet>
<servlet-mapping>
<servlet-name>PhoneBookServlet</servlet-name>
<url-pattern>/PhoneBookServlet</url-pattern>
</servlet-mapping>
</web-app>
```

Step 3: Generating the Web application archive (WAR) file

In step 3, the MFS XML Utility generates a Web application archive (WAR) file from one or more instance servlets and the web.xml file that you generated in step 2.

The MFS XML Utility packages the generated instance servlets and deployment descriptor web.xml files into a J2EE-compliant WAR file that is deployable on WebSphere Application Server. The deployment web.xml file stores a specific style sheet, XMI repository, host connection information, and timeout settings related to the IMS resource adapter (IMS Connector for Java). The J2EE-compliant WAR file contains one or more Java files, class files, and web.xml files.

Example 17-3 shows an example of what the J2EE-compliant WAR files generated by the MFS XML Utility can contain.

Example 17-3 WAR files generated by the MFS XML Utility

```
/WEB-INF/classes/servlet1.class
/WEB-INF/classes/servlet2.class
/WEB-INF/classes/servlet3.class
/WEB-INF/classes/servlet1.java
/WEB-INF/classes/servlet2.java
/WEB-INF/classes/servlet3.java
.
.
/WEB-INF/web.xml
```

Step 4: Uploading WAR and XMI files using an FTP client

The MFS XML Utility provides an FTP client for uploading the Web application archive (WAR) and XML Metadata Interchange (XMI) files onto WebSphere Application Server. To use the FTP client, you must know the host name, user ID, password, server-side directory path, and directory that contains the files to upload.

Step 5: Running a batch file (optional)

The MFS XML Utility lets you use a previously-saved batch file for “Step 1: Generating XMI files from MFS source files” on page 332 and “Step 2: Generating the instance servlet and the web.xml files” on page 333, instead of repeatedly invoking these steps.

17.3 IMS MFS Web Enablement runtime support

MFS Web Enablement runtime support consists of the instance servlets, MFS Servlet, and MFS Adapter. The support runs on WebSphere Application Server and requires IMS Connect and IMS Connector for Java.

- ▶ MFS instance servlet

Instance servlets are generated by the MFS XML Utility tool. An instance servlet records user-specified IMS Connector for Java connection properties, MFS style sheet file path, MFS XMI repository file path, and WebSphere Application Server platform information. The specifications are used by the MFS Servlet.

- ▶ MFS Servlet

The MFS Servlet is the super class of all MFS instance servlets. The MFS Servlet handles HTTP requests and responses to and from Web browsers. The MFS Servlet is responsible for the connection state management, interaction with the MFS Adapter, and the rendering of MFS XMI objects using the style sheet to dynamically generate Web pages.

- ▶ MFS Adapter

The MFS Adapter loads metadata XMI files into XML data objects for data transformation to and from byte stream that IMS applications understand. It works in conjunction with the MFS Servlet and IMS Connector for Java to supply input and to handle output for MFS-based IMS transactions.

Figure 17-2 on page 337 depicts data flowing through MFS Web Enablement runtime components, IMS Connector for Java, and IMS Connect to IMS application programs and back.

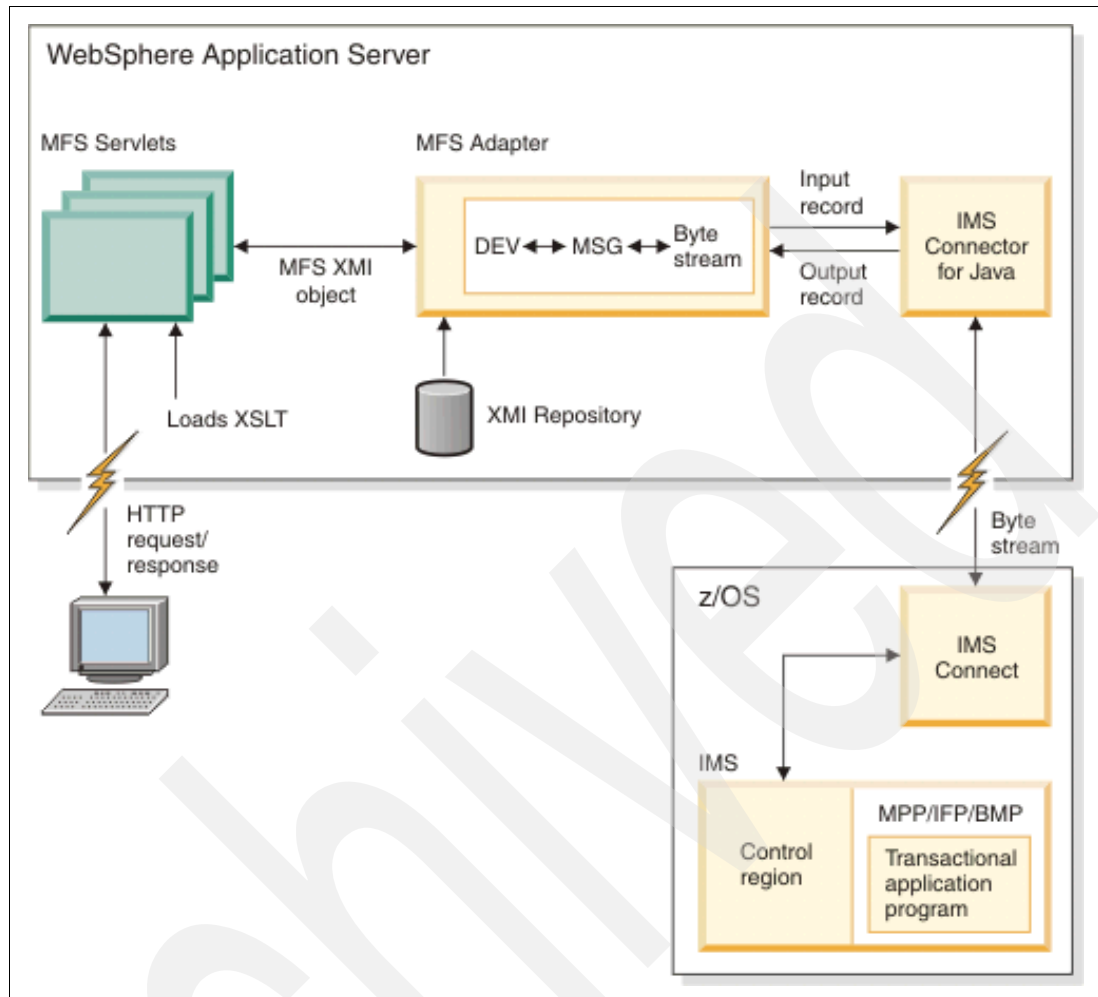


Figure 17-2 Data flowing through MFS Web Enablement runtime components

17.3.1 MFS Web Enablement features and functions

The following core MFS features are supported:

- ▶ 3270 type devices
- ▶ Attribute bytes
- ▶ Cursor positioning
- ▶ Extended attributes bytes (blinking only supported in Mozilla)
- ▶ Multiple physical pages input
- ▶ Multiple logical and physical pages output
- ▶ Message options 1 and 2 only for input and output
- ▶ PA1 key equivalent to advance to the next physical page
- ▶ PF keys with literal data (transaction code and two commands, /FOR and /EXIT, and two control functions, EXTPP (next physical page) and ENDMPPPI (end multiple physical pages input))
- ▶ System literals only for date, time, and LPAGENO

- System default MIDs and MODs, including DFSMI1, DFSMI2, DFSMO1, DFSMO2, DFSMO3, DFSMO5, and the blank screen

Other functional characteristics of MFS Web Enablement include:

- Conversation support
The host connection is created and managed by the MFS Servlet for the duration of the HTTP session. The connection object is reused in a conversation for the same session. MFS Web Enablement runtime support handles /EXIT command requests to properly terminate the conversation on the host.
- Instance servlet Web application archive (WAR) file
The generated WAR file is deployable to WebSphere Application Server. Each WAR file contains a deployment descriptor file and one or more instance servlets, which contain WebSphere Application Server platform information, specific IMS Connector for Java connection properties, XMI repository location, style sheet location, and the IMS Connector for Java interaction properties execution timeout and socket timeout.
- Style sheet
The two sample MFS style sheets render the XML data stream into HTML for display in a Web browser. One style sheet contains templates for generating 3270-like Web pages and the other style sheet is for generating stylized Web pages.

WebSphere Application Server provides the following features:

- Xalan Extensible Stylesheet Language Transformation (XSLT)
The XSLT processor converts XML data into HTML by applying an XSL cascading style sheet, which is a well-formed XML file that contains template information.
- Secure Sockets Layer (SSL) and HTTPS
SSL can be configured to encrypt the data transmitted between the Web browsers and the Web server.
- User authentication
User authentication can be configured so that clients accessing a particular servlet for the first time must log in.

17.3.2 MFS Servlet

The MFS Servlet works with HTTP session objects to:

- Load state information associated with the unique session ID. Create a new session if the request comes in with new session ID.
- Manage and update the state information in each HTTP request.
- Invalidate sessions when an HTTP session becomes unbound (upon logout, browser closed, or session timeout).

Upon receiving the HTTP request, the MFS Servlet checks if this request is associated with an existing HTTP session. If a match cannot be found, the HTTP session is created and the initial Web page, simulating the 3270 blank screen, is generated and returned, as shown in Figure 17-3 on page 339.

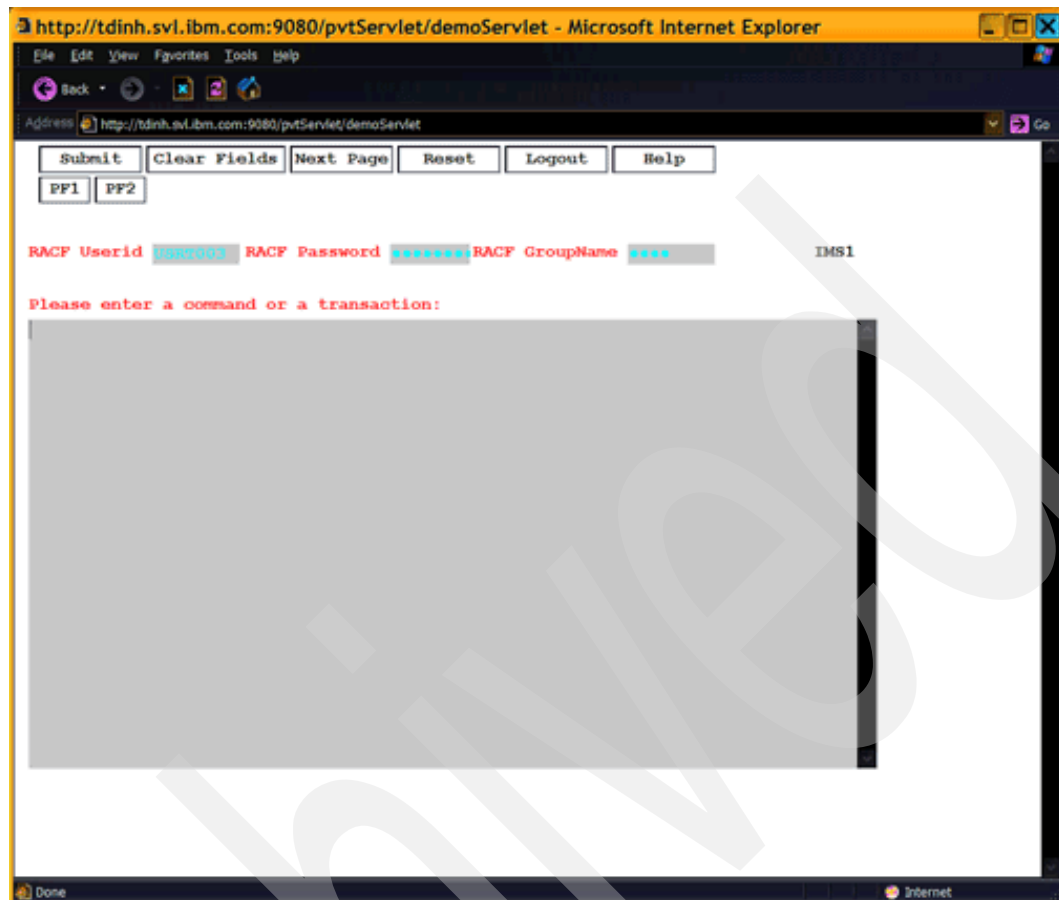


Figure 17-3 Initial Web page

From the initial Web page, you can enter the following inputs:

- RACF user ID, password, and group name

The RACF credentials specified by the user while running the MFS XML Utility are displayed as the default value. You can choose to supply a different set of credentials. The supplied credentials are valid for the entire session. The session is terminated after you log out, close the browser, or the session times out. When new credentials are specified, the previously active connection and pending conversation are automatically terminated, and a new IMS Connector for Java connection object created. The RACF user ID, password, and group name are converted to uppercase text by the MFS Adapter.

- /FOR or /FORMAT modname command

The format command is processed by the MFS Adapter, which attempts to load the MOD /DOF XMI file, based on the modname, from the XMI repository. The MFS Servlet then renders the DOF metadata with the MFS style sheet and returns the formatted Web page to the client browser. If the specified modname cannot be found, the system returns the message IXFT003E: REQUESTED XMI NOT FOUND: MODNAME using system default DFSMO3.xmi.

- ▶ Transaction code followed by data

The transaction code and optional data are written to the input byte array and are then sent to IMS. Trancode is converted to uppercase text by the MFS Adapter. The data remains unchanged (mixed cased allowed). The output execution follows the same flow as in the processing execution in “transaction data” from a Web page other than the initial page.

- ▶ /EXIT command

The exit (/EXIT) command is processed by the MFS Servlet and the MFS Adapter to end the current pending conversation. The MFS Servlet determines if the client is in the middle of a conversation. To end a conversational message, the MFS Adapter sets the SYNC_END_CONVERSATION in the IMSInteractionSpec, and then sends an empty request through IMS Connector for Java to terminate the host conversation. The system returns one of the following messages:

- If in a conversation:

DFS058I HH:MM:SS EXIT COMMAND COMPLETED.

- If not in a conversation:

DFS180 HH:MM:SS NO ACTIVE CONVERSATION IN PROCESS, CANNOT PROCESS COMMAND.

From a Web page other than the initial page, you can enter the following inputs:

- ▶ Transaction data

The MFS Servlet sends the user's input data in one or more physical pages to the MFS Adapter. If multiple physical pages of input (MPPI) is specified for the device page in the MFS source file, the MFS Servlet displays one physical page at a time to collect the data belonging to the same device page and sends them all at once to the MFS Adapter.

- ▶ MFS function keys

The MFS Servlet supports function keys defined from PF1 to PF36. If the request is a function key request, the MFS Servlet either fills the literal value into a device field, as specified in the function key definition, or performs the specified control function. The literal value can be field data, format, or exit commands. The supported control functions are next physical page and end multiple physical pages input.

- ▶ MFS paging

The MFS Servlet supports the NEXT PAGE paging request similar to PA1 on a 3270 terminal. The MFS Servlet keeps track of the current logical and physical page position and displays one physical page at a time for every NEXT PAGE request. If the MFS Servlet receives a NEXT PAGE request on the last physical page on a logical page, it returns the next logical page's first physical page and iterates through until the last physical page of the last logical page. When a NEXT PAGE request is received on the last logical and physical page, the same page is displayed.

- ▶ Reset

Upon receiving the reset request, the MFS Servlet clears the current state and redirects the user to the initial blank page.

- ▶ Logout

Upon receiving the logout request, the MFS Servlet invokes MFS Adapter to end the conversation if in a conversation, closes the connection, dumps all state data associated with the session ID, and redirects the user to the logout page.

17.3.3 MFS Adapter

The MFS Adapter runs inside WebSphere Application Server and transforms data between MFS device data and message data. The MFS Adapter is invoked by the MFS Servlet. Using the Eclipse Modeling Framework, the MFS Adapter loads the appropriate MFS XMI resource from the repository, invokes the transformer routine to handle the data conversion, and submits the IMS transaction using the IMS Connector for Java Common Client Interface (CCI) method calls.

Based on the information contained in the DIF/MID XMI file, the transformer routine first maps the input device data into message data, and then into an input byte array. The input byte array is sent across using IMS Connector for Java. Upon successful execution, the output byte array comes back on the return route. The MFS Adapter then loads the DOF/MOD XMI file, specified in mapName (capable of handling the case with the application program switches the MODNAME), and invokes the transformer routine to first map the output byte array into message data, and then into output device data. The resulting data object is returned to the MFS Servlet.

Note: MFS Web Enablement does not support asynchronous send-only message requests.

The MFS Adapter transformer routine implements both the J2EE CCI Record and Streamable interfaces. The `javax.resource.cci.Record` interface is the base interface for the representation of an input or output to the execute methods defined on a J2EE interaction. The `javax.resource.cci.Streamable` interface enables a resource adapter to extract data from an input record or set data into an output record as a stream of bytes. Table 17-3 describes various scenarios that the MFS Adapter supports.

Table 17-3 Supported MFS Adapter scenarios

Scenario	Description
MFS Adapter receives format request.	Loads and returns specified modname XMI file (load DFSMO3 if not found).
MFS Adapter receives exit request.	Ends the conversation and returns the status using DFSMO2.
Adapter receives transaction request.	Loads and parses using the input MID XMI file.
MFS Adapter receives transaction response where the IMS application does not replace modname.	Loads and processes using the input MID next MOD XMI file (default is DFSMO2 if unspecified).
MFS Adapter receives transaction response where the IMS application replaces modname.	Loads and processes using MOD XMI file specified in the InteractionSpec's mapName.
MFS Adapter receives transaction response where the output byte array begins with "DFS".	Loads and processes using DFSMO1 (for single segment output) or DFSMO5 (for multiple segment output) XMI file.
MFS Adapter receives transaction response where a runtime exception occurred (MFS Adapter, IMS Connect, IMS Connector for Java, or IMS).	Loads and processes using DFSMO2.

17.4 Installing the instance servlet WAR file

To install the instance servlet WAR file:

1. Start WebSphere Application Server.
2. Open the WebSphere administrative console.
3. Expand **Applications** and select **Install New Application**.
4. Click **Browse**, and then under Local path, select the WAR file that you want to deploy.
5. Enter the Context Root. The text you enter will be a part of the URL.
6. Click **Next** to go to the next window and accept the default values.
7. Click **Next** to go to the Application Security Warnings window.
8. Click **Continue** to go to the Install New Application window and accept the default values.
9. Click **Next** and accept the defaults three more times.
10. Click **Finish**. You should get the message “Application [application name] installed successfully,” as shown in Figure 17-4.

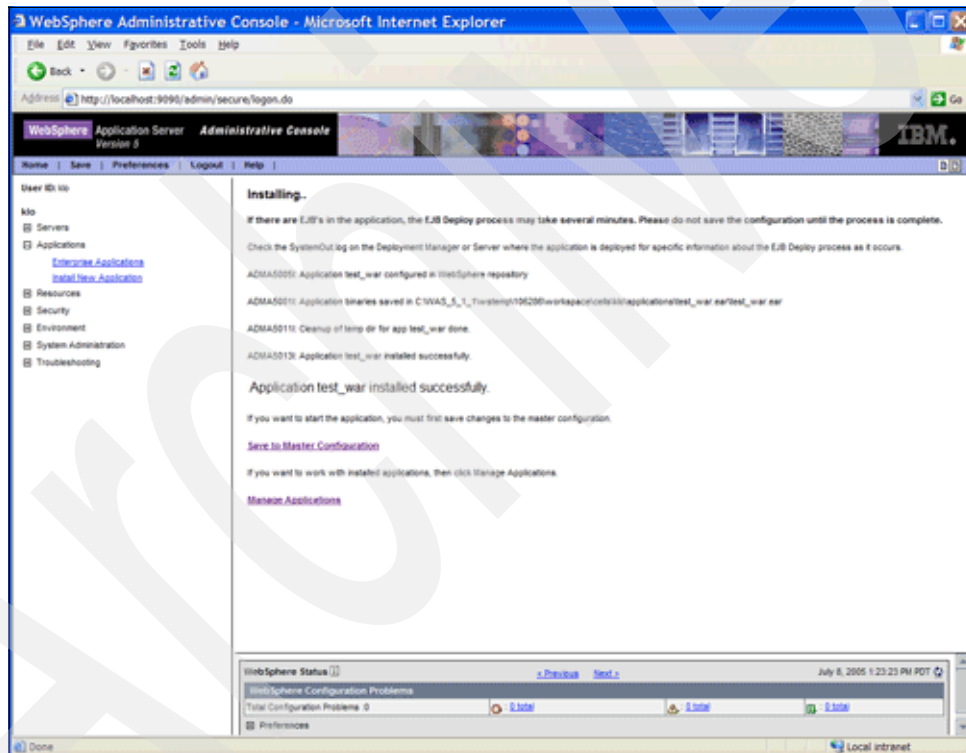


Figure 17-4 Application installed successfully

11. Click the **Save to Master Configuration** link to go to the Save window.
12. Click **Save**.
13. To start the application, select **Enterprise Applications**.
14. Select the WAR file that you installed, and click **Start**.

15. You should get the message “Application [application name] on server [server name] and node [node name] started successfully,” as shown in Figure 17-5. The application status color changes from red to green.

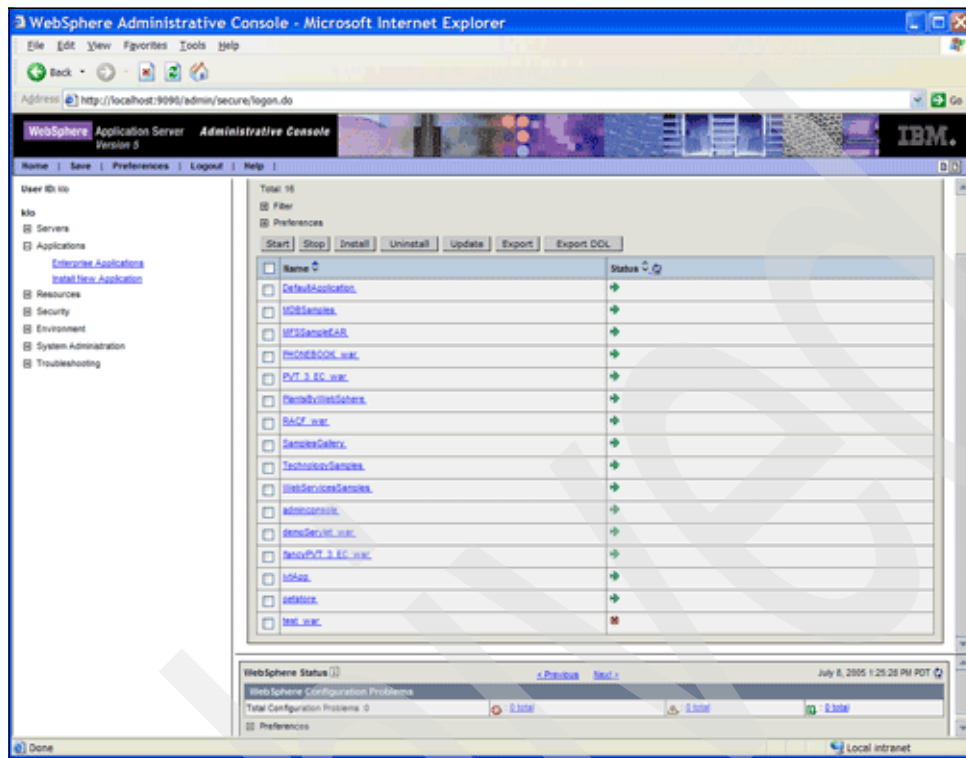


Figure 17-5 Application started successfully

17.5 Accessing the deployed instance servlet

Before accessing the deployed instance servlet, make sure that your Web server is running. To access the instance servlet:

1. From a Microsoft Internet Explorer or Mozilla Firefox browser, enter the URL of the instance servlet (for example, <https://localhost:9080/test/testServlet>). If user authentication is on, proceed to step 2; if not, proceed to step 4.
2. Enter your user ID and password.
3. If the user authentication is on the Security Alert prompt opens. Click **Yes** to indicate that you want to proceed.
4. After WebSphere Application Server authenticates, you are then redirected to the initial Web page.
5. You can now enter your RACF information, command, or transaction request in the same way that you would using a 3270 type terminal. The RACF information displayed defaults to the information specified in the web.xml file. However, you can choose to overwrite the default values. The RACF information is used for creating a connection to IMS Connect. The servlet is timed out in fixed intervals. Make sure that you properly log out to release the connection when you are finished.

Browser-specific tips:

- ▶ Microsoft Internet Explorer

If you open a new browser window by pressing Control+N or by clicking **File** → **New** → **Window**, and then go back to using the previous browser window, you will get the Session ID Error page. However, this restriction does not apply to browser windows that are opened by double-clicking the Microsoft Internet Explorer icon, because a new session ID will be associated with every new instance of the browser.

- ▶ Mozilla Firefox

If you open a new browser window to access the same instance servlet, and then go back to using the previous browser window, you will get the Session ID Error page. This restriction does not apply when you are using multiple Mozilla browsers, each invoking a different instance servlet.

17.6 Sample MFS style sheets

The sample MFS style sheets are provided for demonstration purposes and are customizable. The MFS Servlet loads an MFS style sheet to render Web pages using the XSLT processor. The MFS style sheets supply information about how to render the data in a Web browser. The MFS style sheets provide functionality similar to that of using a 3270 type terminal, including:

- ▶ A Submit button on the top of the page that is analogous to pressing the Enter key on a 3270 type terminal.
- ▶ The Next Page button, which is equivalent to the PA1 function. Clicking this button advances you to the next physical page. When you get to the last physical page, clicking the Next Page button simply displays the same page.
- ▶ The PF keys PF1 through PF36 are displayed as buttons on the HTML pages. Only PF keys with literal data (transaction code and the two commands, /FOR or /FORMAT, and /EXIT and the two control functions, NEXTPP (next physical page), and ENDMPP (end multiple physical pages input), are supported.
- ▶ A Clear Fields button that clears the contents of all input fields.
- ▶ A Reset button that enables you to return to the blank page.
- ▶ A Logout button that closes all connections and exits.
- ▶ A Help button opens the *MFS Web Enablement Version 9.1.0 User's Guide and Reference*.
- ▶ Attribute bytes support, including:
 - Protected
Data cannot be entered into this field. Setting this attribute to true changes it into a label text field.
 - Modified
Data in this field can be modified. Setting this attribute to true changes it into an input text field.
 - High-intensity
Data displayed in this field appears in bold font (default).

- Non-displayable
Data entered into this field is non-displayable. In the case of label text field, the foreground color is set to the background color. In the case of input text field, the input type is set to hidden.
- Extended attribute bytes support:
 - Highlighting:
 - Default: This field gets the default font and color assignments.
 - Blink: This field is blinking.
 - Reverse video: This field's foreground and background colors are reversed.
 - Underline: This field is underlined.
 - Color
 - Sets a field's color. Eight colors are used, as described in Table 17-4.

Table 17-4 Extended attribute colors

Color name	Color displayed on a classic 3270 type terminal simulation	Color displayed on a stylized 3270 type terminal
Blue	Blue	Blue
Red	Red	Red
Green	Lime green	rgb(33,70,40)
Turquoise	Aqua	rgb(53,126,124)
Yellow	Yellow	rgb(244,122,0)
Pink	Fuchsia	rgb(160,50,140)
Default	Aqua	rgb(100,50,0)
Neutral	White	rgb(111,111,111)

- Outlining
 - Sets a border around a field.
 - Box: Sets the border over, under, left, and right. This overrides other outlining extended attributes.
 - Over: Sets the border on the top of the field.
 - Under: Sets the border on the bottom of the field.
 - Left: Sets the border to the left of the field.
 - Right: Sets the border to the right of the field.

Two types of sample MFS style sheets are provided:

- Classic 3270 type terminal simulation
The following page is rendered with the sample classic 3270 type terminal simulation and shows the IMS installation verification procedure (IVP).
- Stylized 3270 type terminal
This displays a Web page interface.

The customizable attributes of the MFS style sheets are:

- Font attributes:
 - Color
 - Family

- Size
- Weight
- Background color.
- Button style.
- JavaScript™ code can be added.

You can specify additional graphics in the style sheet and add them into the WAR file. Do not modify the rest of the code in the style sheets.

17.7 Instructions to Web-enable IMS Phonebook application

This section provides step-by-step guidelines to Web-enable, generate, deploy, and invoke the IMS IVP Phonebook application Phonebook MFS source file. The text under each step shows the instruction displayed by the utility.

17.7.1 Step 1: Parsing the MFS source file with MFS XML Utility

To parse the MFS source files:

1. From the MFS XML Utility window, choose selection 1 and press Enter:

```
Please enter your selection here: 1
Step 1: Generate XMI files that represent MID/DIF and MOD/DOF of the MFS source
This step requires the following information:
-MFS source files
-Device Characteristics Table file (Optional)
-Whether source files are in text or binary format (default to text)
-Codepage for source files (default to MS950)
-Codepage for host environment (default to Cp037)
-Device type to format (default to 3270-A02)
-Device feature to enable (default to ignore)
-Output directory for generated XMI files (default to installation directory)
```

2. Press Enter to run in novice mode:

```
Enter arguments here or press enter to run novice mode. Type '/help' for more
information or 'q' to quit anytime: >>
Beginning Step 1: Generating XMI files...
```

3. Specify c:\MFSXMLUtility+\dfsivf1.mfs as the MFS source files and press Enter:

```
Specify MFS source files or directory containing MFS source files:
c:\MFSXMLUtility+\dfsivf1.mfs
You selected c:\MFSXMLUtility+\dfsivf1.mfs
```

4. Press Enter to indicate no Device Characteristics Table:

```
Specify device characteristics table (Optional):
No device characteristics table selected
```

5. Press Enter to indicate the default value of n:

```
Is the source in binary mode (y|n ; Default is no):
>> You entered no by default
```

6. Press Enter to specify the source codepage as Cp1252 (default based on system locale):

```
Specify codepage for source (your system default is Cp1252):
>> You entered MS950 by default.
```

7. Press Enter to specify the host codepage as Cp037 (for EBCDIC United States):
Specify codepage for host (Default is Cp037):
>> You entered Cp037 by default.
8. Specify PHONEBOOK as the output directory and press Enter:
Specify output directory (Default is C:\MFSXMLUtilityGA\): PHONEBOOK
>> You entered C:\MFSXMLUtilityGA\PHONEBOOK\ Parsing files...
9. Press Enter to select the default device type:
Choose one of the following device features (Default is Ignore):
1) Ignore
=>
You selected Ignore
parsing c:\MFSXMLUtility+\dfsivf1.mfs
Writing to C:\MFSXMLUtilityGA\device_types.log completed
10. Press Enter to specify that you do not want to see the parse output:
Parse successfully. Would you like to see the parse output? (y|n; default is no):
Parse output log will be created.
Writing to C:\MFSXMLUtilityGA\parse.log completed
The following XMI files were generated: C:\MFSXMLUtilityGA\PHONEBOOK\IVTNOMI1.xmi
C:\MFSXMLUtilityGA\PHONEBOOK\IVTNO.xmi XMI files generated.
11. Type y and press Enter to indicate that you want to save your input values to a batch file:
Do you wish to save your input values to a batch file? (y|n ; Default is no) y
Writing batch file to C:\MFSXMLUtilityGA\PHONEBOOK\IVTNOMI1_step1.txt...
Step 1 batch file created
The following batch file was generated: C:\MFSXMLUtilityGA\PHONEBOOK\IVTNOMI1_step1.txt
Step 1 completed.

17.7.2 Step 2: Generating an instance servlet

In step 2, you generate the PHONEBOOK instance servlet.

To generate an instance servlet:

1. From the MFS XML Utility window, select 2 and press Enter:
Please enter your selection here: 2
Step 2: Generate and compile instance servlet used during runtime for the backend MFS application;
This step requires the following information:
-Name of the this instance servlet
-Platform (Windows or Unix based System) where WebSphere Application Server is located
-Location of XMI repository on web server (default to last value)
-Name and location of stylingsheet on local machine to copy to web server (default to last value)
-Host name or IP address of IMS (default to last value)
-Port number of host (default to last value)
-IMS datastore name (default to last value)
-RACF username (optional)
-RACF group (optional)
-RACF password (required if RACF username is specified)
-Trace Level of IMS Connect for Java (default to 0)
Begin Servlet Generation....
2. Press Enter to generate the servlet in novice mode:
Press Enter to generate servlet in novice mode, otherwise enter servlet arguments for expert mode or type '/help' or 'q':
>>

3. Enter PHONEBOOK as the name of your instance servlet and press Enter:
Please enter the name of this instance servlet:PHONEBOOK
4. Specify the output directory for your instance servlet and press Enter (the last output directory specified or the MFS XML Utility installation directory):
Specify output directory (default is C:\MFSXMLUtilityGA\PHONEBOOK\):
You have selected an existing directory! Files with the same name will be over-written without warnings!
Continue? (y|n; default is yes)
>> You entered C:\MFSXMLUtilityGA\PHONEBOOK\
5. Specify 1, Windows platform, as the platform where WebSphere Application Server is located and press Enter:
Please select the platform where WebSphere Application Server is located:
1) WINDOWS
2) UNIX Systems (AIX, Linux, Solaris, and so forth)
>> 1
You chose WINDOWS.
6. Specify /c:\xmi as the file path URI of the XMI repository on WebSphere Application Server and press Enter:
Specify target location of XMI repository on web server ('?' for help): c:\xmi
>> You entered file:/c:\xmi
7. Specify c:\\$Projects\MFSXML\source\sample3270.xml as the target location of your style sheet on WebSphere Application Server and press Enter:
Specify location of styling sheet ('?' for help):
c:\\$Projects\MFSXML\source\sample3270.xml
>> You entered file:/c:\\$Projects\MFSXML\source\sample3270.xml
8. Specify ecdb31.svl.ibm.com as the IMS host name or IP address and press Enter:
Specify IMS hostname or IP address ('?' for help): ecdb31.svl.ibm.com
>> You entered ecdb31.svl.ibm.com
9. Specify 9999 as the host port number and press Enter:
Specify a port number ('?' for help): 9999
>> You entered 9999
10. Specify IMS1 as the IMS datastore name and press Enter:
Specify IMS1 as the IMS datastore name ('?' for help): IMS1
>> You entered IMS1
11. Skip this step by pressing Enter:
Note that the following RACF information will be used if no RACF information is specified during runtime.
Specify RACF user name (Optional; '?' for help):
No value entered
12. Specify 3 for the trace level for IMS Connector for Java and press Enter:
Specify trace level for IMS Connector for Java from 0 to 3 (default is 0; '?' for help):
3
>> You entered trace level 3
13. The instance servlet is generated and compiled in the output directory that is specified:
Generating servlet.....completed
Servlet is being compiled.....completed.

14.Type y to save your input values for later execution in batch mode and press Enter:

```
Do you wish to save your input values to a batch file (RACF information will NOT be
saved)?
(y|n ; Default is no)y
Writing batch file to C:\MFSXMLUtilityGA\PHONEBOOK\PHONEBOOK_step2.txt...
Batch file created
```

15.The deployment descriptor and web.xml files are generated:

```
Generating servlet deployment descriptor.....generated.
Starts to put files in the WEB-INF directory...
Compile servlet to be packaged into WAR file...
Servlet is being compiled.....completed.
The following servlet file was generated: C:\MFSXMLUtilityGA\PHONEBOOK\PHONEBOOK.java
The following servlet class file was generated:
C:\MFSXMLUtilityGA\PHONEBOOK\PHONEBOOK.class
The following batch file was generated: C:\MFSXMLUtilityGA\PHONEBOOK\PHONEBOOK_step2.txt
The following segment of web.xml file was generated:
C:\MFSXMLUtilityGA\PHONEBOOK\PHONEBOOKWeb.xml
Step 2 completed.
```

17.7.3 Step 3: Generating a WAR file

In step 3 you, generate the PB.war file.

To generate a WAR file:

1. From the MFS XML Utility window, select 3 and press Enter:

```
Enter your selection here: 3
Step 3: Generate a WAR (Web Application aRchive) file containing one or more instance
servlets
To generate a WAR file, you must first complete step 2running through step 2 in advance
is mandatory.
This step requires the following information:
-Previously generated instance servlet class file(s) in the WEB-INF\classes directory
-Previously generated deployment descriptor (web.xml) in the WEB-INF directory
*Examine the content of the web.xml file in C:\MFSXMLUtilityGA\WEB-INF\ and make any
necessary additions.*
*The web.xml file that will be packaged into the WAR file is in
C:\MFSXMLUtilityGA\WEB-INF\
This WAR file is going to be generated with the following instance servlets.
1) .\WEB-INF\classes\PHONEBOOK.class
2) .\WEB-INF\classes\PHONEBOOK.java
```

2. Enter PB for the name of your WAR file and press Enter:

```
Enter the name of this WAR file: PB
```

3. Indicate if you would like to include additional files in your WAR file, for example, GIF or JPG files, (default is no) and press Enter:

```
Do you want to package additional files such as pictures with this WAR file? (y|n;
default is no)
```

4. The WAR file is generated:

```
Adding manifest
Adding_WEB-INF/ (reading=0)(writing=0)(saving 0%)
Adding_WEB-INF/classes/ (reading=0)(writing=0)(saving 0%)
Adding_WEB-INF/classes/PHONEBOOK.class (reading=379)(writing=270)(saving 28%)
Adding_WEB-INF/classes/PHONEBOOK.java (reading=614)(writing=404)(saving 34%)
Adding_WEB-INF/web.xml (reading=1060)(writing=373)(saving 64%)
WAR file generated.
```

The following WAR file was generated:
C:\MFSXMLUtilityGA\WAR\PB.war
Step 3 completed.

17.7.4 Step 4: Configuring WebSphere Application Server

To configure MFS Web Enablement support on the WebSphere Application Server resource adapter, complete the following tasks. To begin configuring the WebSphere Application Server resource adapter, you must:

- ▶ Download and install IMS Connector for Java Version from:
<http://www.ibm.com/software/data/db2imstools/imstools/imsjavcon.html>
- ▶ Download the MFS Web Enablement ZIP files from:
<http://www.ibm.com/software/data/ims/toolkit/mfswebsupport/index.html>
- ▶ Unzip and place the JAR files in an accessible location on WebSphere Application Server.

If you FTP the JAR files to the WebSphere Application Server system, make sure that it is first set to binary mode.

To configure the WebSphere Application Server resource adapter:

1. Start WebSphere Application Server, and then open the administrative console.
2. From the Contents pane, expand **Resources**, and then click **Resource Adapters**.
3. Create a directory named mfsweb under your WAS_INSTALL_ROOT directory. Copy and paste the MFSRuntime.jar and MFSTDTDLang.jar files into this new folder.

You can look up the WAS_INSTALL_ROOT directory from **Environment** → **WebSphere Variables** on the WebSphere Application Server administrative console.

4. Select the IMS resource adapter (IMS Connector for Java). The current class path shows the path:

\$(CONNECTOR_INSTALL_ROOT)/ims91011.rar

Important: Add the MFS JAR files above the IMS resource adapter entry and add each entry on a new line.

\${WAS_INSTALL_ROOT}/mfsweb/MFSRuntime.jar
\${WAS_INSTALL_ROOT}/mfsweb/MFSTDTDLang.jar

Figure 17-6 on page 351 shows the MFS JARS files located in the WAS_INSTALL_ROOT/mfsweb directory, where WAS_INSTALL_ROOT is the root install directory of WebSphere Application Server.

Figure 17-6 MFS JAR file locations

- ### 17.7.5 Step 5: Deploying the application WAR file

1. With WebSphere Application Server started, go to your WebSphere Application Server administrative console. Expand **Applications** from the left and then click **Install New**.
2. Click **Browse** under Local path to select the PB.war file generated in “Step 3: Generating a WAR file” on page 349.
3. Enter the Context Root (for example, demo). The text that you enter here will be a part of the URL.
4. Click **Next** to go to the next window. Keep the default values.
5. Click **Next** to go to the Application Security Warnings window.
6. Click **Continue** to go to the Install New Application window. Keep the default values.
7. Click **Next** to go through the next three windows.
8. Click **Finish** and you should receive the message “Application PB_war installed.”
9. Click **Save**.

10. Select **Enterprise Applications**.
11. Select the check box of the Web application archive (WAR) file that you just installed and click **Start**.
12. You should receive the message “Application PB_war on server server 1 and node [node name] started successfully,” and the application status icon should become green.

17.7.6 Step 6: Invoking the instance servlet

With WebSphere Application Server started, open a Web browser and enter the URL `http://localhost:9080/demo/PHONEBOOK`. Then on the initial Web page, enter /FOR IVTN0 and click **Submit**.

Note: localhost:9080 is the WebSphere Application Server node, demo is the context root you entered when you installed the Web application archive (WAR) file, and PHONEBOOK is the instance servlet that was generated in “Step 2: Generating an instance servlet” on page 347.

17.7.7 Step 7: Invoking the Phonebook application

To invoke the Phonebook application sample:

1. Enter DISPLAY in PROCESS CODE field.
2. Enter LAST1 in LAST NAME field.
3. Click **Submit**.
4. Verify that the output is correct.

17.7.8 Step 8: Logging out

The final step is to log out. Click **Logout**.

IMS SOAP Gateway

IBM IMS SOAP Gateway is a Web services solution that enables IMS applications to interoperate outside of the IMS environment through Simple Object Access Protocol (SOAP) to provide and request services independently of platform, environment, application language, or programming model.

The IMS SOAP Gateway can assist an organization in the following areas:

- ▶ Enterprise modernization
- ▶ Application development
- ▶ Business-to-business (B2B) integration
- ▶ Service-oriented architecture (SOA) implementation

Note: You can download IMS SOAP Gateway from the IMS Web site:

<http://www.ibm.com/ims>

This chapter provides a high-level description about how to use IMS SOAP Gateway to make your IMS application to become a Web service. The IMS SOAP Gateway documentation provides detailed instructions, including a sample.

For the latest information about IMS SOAP Gateway, visit:

<http://www.ibm.com/software/data/ims/soap/>

18.1 IMS SOAP Gateway introduction

IMS SOAP Gateway allows you to enable your IMS application to become a Web service. Different types of client applications, such as Microsoft .NET, Java, and third-party applications, can submit SOAP requests into IMS to drive the business logic of your IMS applications.

IMS SOAP Gateway is compliant with the industry standards for Web services, including SOAP/HTTP 1.1 and WSDL 1.1. This allows your IMS assets to interoperate openly with various types of applications.

With IMS SOAP Gateway, you can be flexible about how you want your data to be handled. For example, you can have the data send the XML data from the client to your IMS environment that can be handled by your new or enhanced IMS application. The data can be stored directly in XML using the IMS XML DB function, or you can transform the XML data into data bytes.

IMS SOAP Gateway consists of two main components:

- ▶ IMS SOAP Gateway deployment utility

The end-to-end deployment utility enables you to set up properties and create runtime code that IMS SOAP Gateway uses to enable IMS applications as Web services.

- ▶ IMS SOAP Gateway server

The IMS SOAP Gateway server processes SOAP messages. It receives the SOAP message from the client application, converts it to an IMS input message, and sends it to IMS through IMS Connect. It then receives the output message from IMS and converts it to a SOAP message to send back to the client.

The diagram in Figure 18-1 shows how IMS SOAP Gateway helps you process the SOAP message from your client at run time.

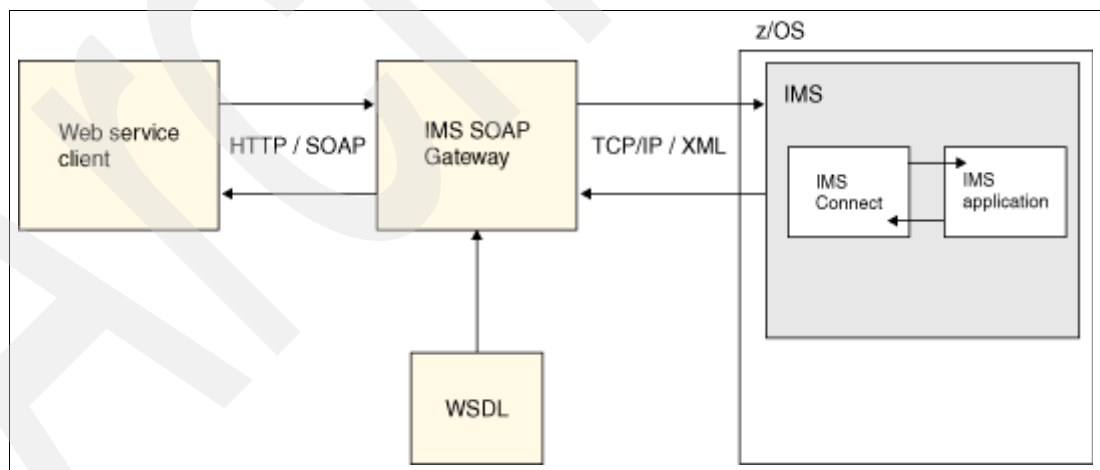


Figure 18-1 IMS SOAP Gateway

18.2 Making your IMS application a Web service

You can easily make your IMS application accessible a Web service with IMS SOAP Gateway by using the following steps:

1. Create a Web service interface, which is a Web Service Description Language (WSDL) file, for your IMS application.
2. Deploy the Web service interface to IMS SOAP Gateway and define the connection and correlation information by using the deployment utility.

After you deploy the WSDL file, the IMS application is available as a Web service. You can create your desired client application to send SOAP messages to your IMS application through IMS SOAP Gateway.

18.2.1 Creating a WSDL file for your IMS application

To make an IMS application a Web service, you need a WSDL file. The WSDL file is the Web service interface for the IMS application. It describes where the Web service is located and what the input and output message looks like for invoking your IMS application.

You can easily create a WSDL file by hand or using an application development tool (for example, IBM Rational Application Developer) and then customize it to be used with IMS SOAP Gateway.

Another method for creating a WSDL file for IMS SOAP Gateway is to use IBM WebSphere Developer for zSeries tool, which can help you generate a WSDL file from the COBOL copybook of the IMS application.

Using IBM WebSphere Developer for zSeries tool

IBM WebSphere Developer for zSeries V6 is an application development tool that helps with the development of traditional mainframe applications. It helps you to easily generate the artifacts needed to transform your IMS application into a Web service to be used with the IMS SOAP Gateway run time. By simply taking a COBOL copybook for your IMS application that describes the input and output message format, it generates the following Web service artifacts:

- ▶ Web Services Description Language (WSDL) file, which provides a Web service interface of the IMS application so that the client can communicate with the Web service
- ▶ COBOL converters and driver file, which help you to transform the XML message from the client into COBOL bytes for the IMS application and then back to XML
- ▶ Correlator file, which contains information that enables IMS SOAP Gateway to set IMS properties and call the IMS application

To generate Web services artifacts for IMS SOAP Gateway using WebSphere Developer for zSeries, perform the following steps:

1. Start WebSphere Developer for zSeries and open the **z/OS Projects** perspective. Make sure that you have enabled.
2. Create a local COBOL project.
3. Import the COBOL copybook that describes the format of the input and output messages of your IMS application into the project.
4. Start the Enable Web Service wizard:
 - a. Right-click the COBOL copybook file.

- b. Select **Enable Web Services** → **Generate enablement code**.
5. Select the data structures for the inbound and outbound converters:
 - a. Click **Change COBOL Options**. The COBOL Import Properties panel opens.
 - b. In the Platform field, select **z/OS**.
 - c. Click **Finish**. The Data structures panel opens.
 - d. For the inbound data structure, select the COBOL data structure that corresponds to the input message of the IMS application.
 - e. Go to the Outbound data structure tab.
 - f. Select the COBOL data structure that corresponds to the output message of the IMS application.
 - g. Click **Next** to continue.
6. Specify generation options:
 - a. In the Converter type field, select **IMS SOAP Gateway**.
 - b. In the Host code page field, select the code page that the host uses. IMS SOAP Gateway supports only UTF-8 encoding for the inbound and outbound code pages. Therefore, you cannot change these settings.
 - c. Specify any additional properties.
 - d. Go to the WSDL and XSD Options tab.
 - e. In the Endpoint URI field, change the host and port name to the location of IMS SOAP Gateway. This URI specifies the address of the Web service.
 - f. Specify any additional properties.
 - g. Click **Next** to continue.
7. Specify the IMS SOAP Gateway correlator properties and click **Next**.
8. Specify location and names of the Web service artifacts:
 - a. If necessary, change the default location and names of the COBOL converters and driver.
 - b. Ensure that **Generate all to driver** is selected.
 - c. Go to the WSDL and XSD tab.
 - d. If necessary, change the default location and names of the WSDL file.
 - e. Ensure that the WSDL file name is selected.
 - f. Optionally, select the inbound and outbound XSD files to be generated. These files are not required by IMS SOAP Gateway.
 - g. Click **Finish**.
9. The following files are generated:
 - COBOL converters and driver file
 - Correlator file
 - WSDL file
 - Inbound and outbound XSD files (optional)

After you create the Web services artifacts, you can deploy the WSDL and the correlator files to IMS SOAP Gateway.

18.2.2 Deploying WSDL and configuring properties with IMS SOAP Gateway

The deployment step help you set up properties and create runtime code that will be used by IMS SOAP Gateway to make your IMS application accessible as Web services.

IMS SOAP Gateway provides you with a deployment utility that can help you to perform the following tasks:

- ▶ Easy end-to-end deployment.
- ▶ Set up and modify connection and correlation information (for example, host name/port, trancode, timeout).
- ▶ Generates runtime code from WSDL that will be used by IMS SOAP Gateway to enable the IMS application as a Web service.

The IMS SOAP Gateway documentation provides detailed instructions about how to use the deployment utility.

After you deploy to IMS SOAP Gateway, your IMS applications are ready to be accessible to a client as a Web Service.

18.2.3 Writing the client application

After you have the WSDL deployed to IMS SOAP Gateway, you can write a client application, for example, Java or Microsoft .NET, to send a SOAP message to invoke the IMS application as a Web service.

Archived

Open Database Access

This chapter provides the information about the IBM IMS Open Database Access (ODBA) function. ODBA is a callable interface to access databases managed by the IMS Database (DB) Manager.

ODBA can be used by any z/OS application program that uses the Recovery Resource Services (RRS) of z/OS as a sync point manager to access the IMS full function databases and data entry databases (DEDBs). This allows IMS DB and z/OS application programs to be developed, installed, and maintained independently of each other. ODBA also provides for failure isolation and independent resource recoverability.

19.1 Accessing IMS databases through the ODBA

ODBA uses the IMS database resource adapter (DRA) function to establish a connection to the IMS subsystem specified by IMSID or DBCTLID coded in the DRA startup table. ODBA consists of a startup table for each IMS system and a group of modules that make up the ODBA callable interface.

The DRA startup table specifies how to connect to a particular IMS subsystem, such as the subsystem name and the number of threads allowed.

The ODBA callable interface modules manage threads connecting an ODBA application to an IMS system and also provide some recovery services. They provide a set of Data Language/I (DL/I) database calls and system services calls. Application programs establish connections to IMS systems and make IMS database and system function calls using AERTDLI. The syntax is very similar to AIBTDLI.

ODBA requires the use of the application interface block (AIB). The AIB is a storage area obtained by the application program and is passed to IMS as a parameter in the AERTDLI call list when the application program issues DL/I calls. For ODBA, the AIB also provides the area the application program uses to keep track of the thread token.

ODBA applications use z/OS Resource Recovery Services (RRS) and its application programming interfaces (APIs) to invoke commit processing. Figure 19-1 shows application access to IMS using the ODBA interface.

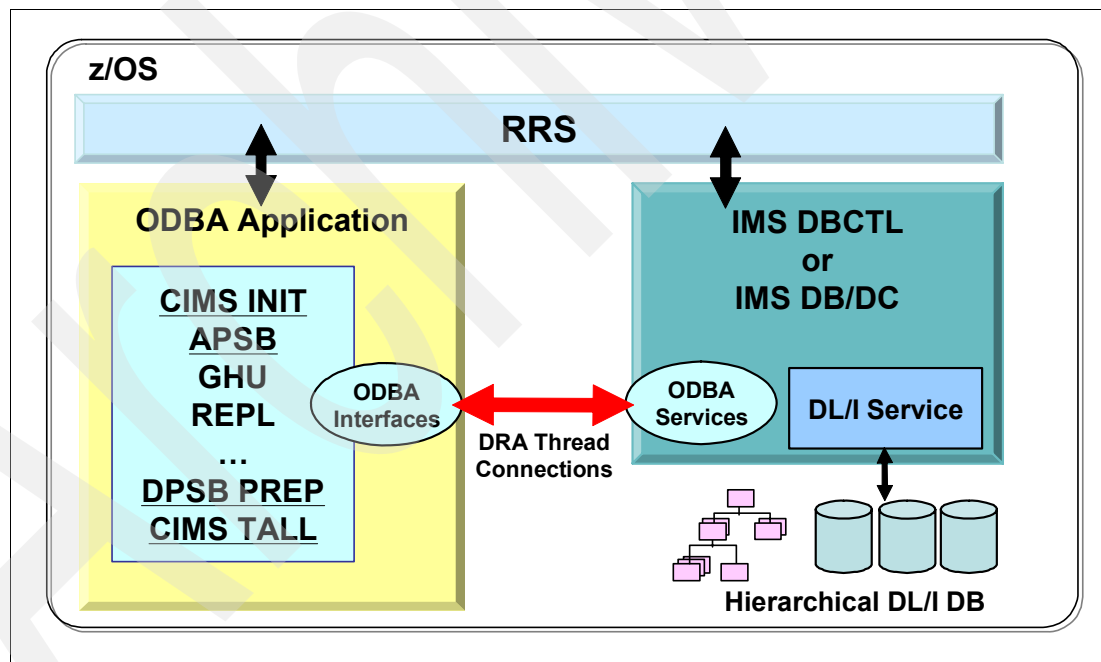


Figure 19-1 Application access to the IMS ODBA interface

19.2 The database resource adapter (DRA)

The database resource adapter (DRA) is an interface to the IMS DB full function database and DEDBs. The DRA can be used by a coordinator controller (CCTL), such as IBM CICS® Transaction Server for z/OS, or a z/OS application program that uses the ODBA interface. The DRA function implements thread concepts for communicating between the IMS DB

server task and the application task. A single DRA thread is associated with every CCTL or ODBA thread. CCTL or ODBA threads cannot establish a connection with more than one DRA thread at a time. In the meantime, the DRA is capable of processing more than one thread at the same time, known as multithreading. This means that multiple CCTL or ODBA threads can be using the DRA at the same time. Multithreading applies to all CCTL requests and ODBA requests.

An ODBA DRA thread does not have TCB affinity like a CCTL DRA thread. For ODBA, you can have multiple ODBA DRA threads on a TCB, so you can have multiple partition specification tables (PSTs) allocated to same TCB. Figure 19-2 shows the thread concepts of the DRA interface.

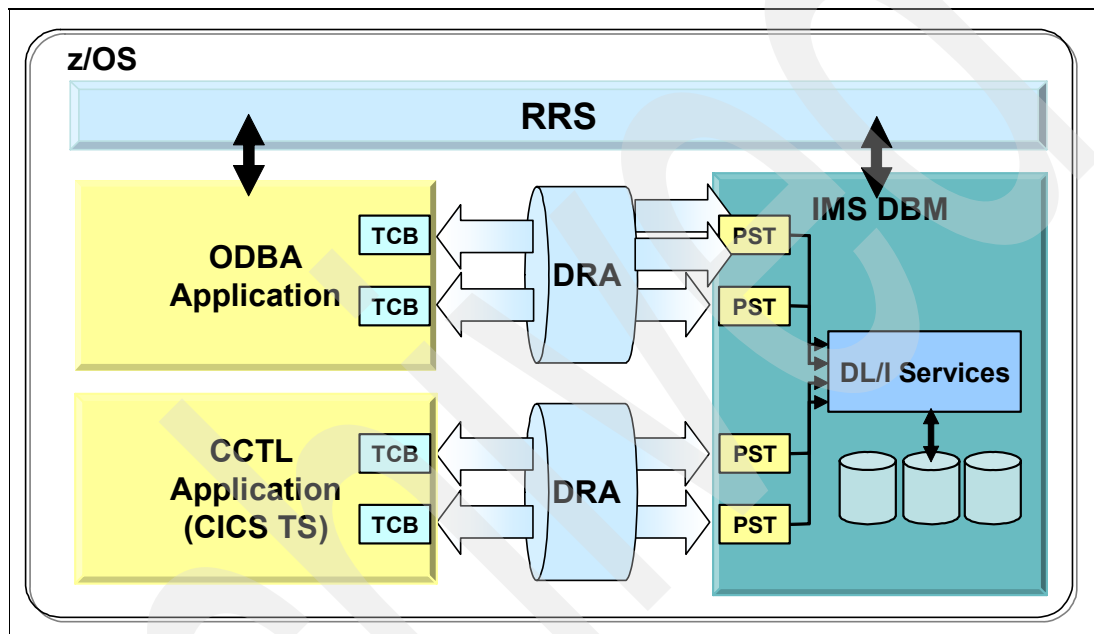


Figure 19-2 Thread concepts of the DRA interface

19.3 Setting up the DRA and the ODBA interface

To set up ODBA and the DRA, perform the following tasks:

1. Create the ODBA DRA startup table.
2. Place the ODBA and DRA modules in STEPLIB or JOBLIB in the z/OS application region.
3. Bind the ODBA application programs with DFSCDLI0.
4. Set up security.

19.3.1 Creating the ODBA DRA startup table

The DRA startup table contains values used to define the characteristics of the DRA. The DRA startup table is created by assembling the DFSxxxx0 module for the ODBA's use. For naming the DRA load module, we recommend the naming convention shown in Table 19-1 on page 362.

Table 19-1 Recommended ODBA DRA startup table naming convention

Characters one to three	Characters four to seven	Character eight
DFS	Specified 4-byte ID. Use the IMSID or DBCTLID of the system to which you want to connect. However, this is not a requirement.	0 (zero)

For example, if your IMSID is "IMSG," the DRA load module name will be "DFSIMSG0."

IMS does not ship a default DRA startup table in IMS.SDFSRESL. You must generate your own table by using the DFSPRP macro. You can refer to the IMS IVP member IV_E308J in IMS.INSTALIB that contains a sample generation JCL for the DRA startup module for the CICS/DBCTL environment. Ensure that the name of your DRA startup module is not the same as the name of CCTL environment; you at least have to change the MBR= parameter in the JCL to satisfy your requirement.

The DRA parameters are specified as keywords on the DFSPRP macro invocation. Table 19-2 shows a list of the DFSPRP keywords and their descriptions.

Table 19-2 The DFSPRP macro keywords

Keyword	Description
AGN=	A one to eight character application group name. This is used as part of the IMS DB and DB/DC Application Group Name (AGN) security function.
CNBA=	Total Fast Path NBA buffers for the ODBA's use.
DBCTLID=	The four character name of the IMS DB or DB/DC region. This is the same as the IMSID parameter in the DBCTL/IMS procedure.
DDNAME=	A one to eight character ddname used with the dynamic allocation of the IMS DB execution library. The default ddname is CCTLDD.
DSNAME=	A one to 44 character data set name of the IMS DB execution library, which must contain the DRA modules and must be APF-authorized. The default DSNAME is IMS.SDFSRESL.
FPBOF=	The number of Fast Path DEDB overflow buffers (OBA) allocated per thread. The default is 00.
FPBUF=	The number of Fast Path DEDB normal buffers (NBA) allocated and fixed per thread. The default is 00.
FUNCLV=	Specifies the DRA level that the ODBA supports. The default is 1.
IDRETRY=	The number of times the z/OS application region is to attempt to IDENTIFY (or attach) to IMS after the first IDENTIFY attempt fails. The maximum number is 255. The default is 0.
MAXTHRD=	The maximum number of DRA thread TCBs available at one time. The maximum number is 999. The default number is 1.
MINTHRD=	The minimum number of DRA thread TCBs to be available at one time. The maximum number is 999. The default number is 1.
SOD=	The output class used for a SNAP DUMP in case of abnormal thread terminations. The default is A.
TIMEOUT=	(CCTL only.) The amount of time (in seconds) that a CCTL waits for the successful completion of a DRA TERM request. Specify this value only if the CCTL application is coded to use it. This value is returned to the CCTL upon completion of an INIT request.

Keyword	Description
TIMER=	The time (in seconds) between attempts of the DRA to identify itself to IMS DB or DB/DC during an INIT request. The default is 60 seconds.
USERID=	An eight character name of the CCTL region. This keyword is ignored for an ODBA region.

The DSNNAME from the DRA startup table is used to dynamically allocate the data set that contains the rest of the ODBA interface routines. The DDNAME is generated to allow multiple connections to IMS from the same z/OS application region. ODBA does not use the DDNAME parameter, so it is ignored if it is specified.

The default DSNNAME is IMS.SDFSRESL. This is the default name established by the IMS generation process. Make sure that this data set is APF-authorized.

19.3.2 Loading and running ODBA in the z/OS application region

Table 19-3 shows the modules that need to be made available to the z/OS ODBA application region by placing them in STEPLIB or JOBLIB. These modules are shipped with IMS in IMS.SDFSRESL.

Table 19-3 ODBA modules in IMS.SDFSRESL

Module name	Description
DFSCDLI0	This module is linked or loaded by an application program. DFSCDLI0 also contains the ALIAS name AERTDLI.
DFSAERG0	This module is loaded by DFSCDLI0.
DFSAERM0	This module is attached by DFSAERG0 in the z/OS application region.
DFSAERA0	This module is attached by DFSAERM0 for initialization to the specified IMS DB subsystem.

19.3.3 Linking application programs

Bind the ODBA application programs with DFSCDLI0 (AERTDLI). As an alternative, you can issue a load and branch command, passing the AIB call list in Register 1.

19.3.4 Establishing and defining security

IMS provides several options for establishing and defining security for application programs that use the ODBA interface. The options that you select depend on the type of security environment and authorization method that you plan to use. In general, the process that IMS uses to secure program specification blocks (PSBs) involves one of the following types of security checking.

AGN security

When a PSB is allocated, a security check is performed to determine:

- ▶ If the user is authorized to connect to the AGN name assigned to the dependent region or thread
- ▶ If the PSB name is defined in the AGN table entry with the authorized AGN name

Use the IMS ISIS execution parameter to control the authorization for a z/OS application region to connect to the IMS DB environment. If you specify ISIS=1 or ISIS=2, both the z/OS

application region connection and PSB scheduling are checked. If you specify ISIS= 0, neither is checked. Table 19-4 describes the actions that you need to perform to set up security when specific options are selected.

Table 19-4 Options for defining AGN security for applications that use ODBA

Specification	Actions to perform
ISIS=0	No action required. No connection security checking is performed.
ISIS=1	<ol style="list-style-type: none"> 1. Before the z/OS application region connects to IMS DB, make sure that the user ID and Application Group Name (AGN) from the DRA startup table are authorized to access the specified IMS DB. 2. Build RACF tables to define valid user ID-AGN combinations. If the DRA startup table values do not correspond to an entry in RACF tables, the z/OS application region cannot connect to IMS DB. 3. Use the AGN and AGPSB statements to define the AGN name and PSB name to the security maintenance utility. <p>Note: Connections to different IMS DB systems from the same JOB will not use the same user ID-AGN security because different DRA startup table load modules will be loaded.</p>
ISIS=2	<ol style="list-style-type: none"> 1. Create an application group name exit routine that is named DFSISIS0. This routine must determine whether the AGN passed to it is valid for the attempted connection. 2. Use the AGN and AGPSB statements to define the AGN name and PSB name to the security maintenance utility.

Resource Access Security (RAS)

IMS Version 9 introduces a new security function called Resource Access Security (RAS) for security maintenance utility (SMU) replacement. A security check is performed by RACF to determine if the user is authorized to use the PSB. RACF determines authorization by looking at the RACF security class profile defined for the dependent region or thread. Use the ISIS and ODBASE execution parameters to control the authorization for an z/OS application region to use a PSB. Table 19-5 describes the actions that you need to perform to set up security when specific options are selected.

Table 19-5 Options for defining RAS security for applications that use ODBA

Specification	Actions to perform
ISIS=0/N and ODBASE=N	No action required. No PSB security checking is performed.
ISIS=R and ODBASE=N	Define the PSBs that you want protected by RACF to the IIMS or Ixxxxxxx resource class, and then define the user IDs of the dependent region that you want to be authorized to access the PSBs. The ODBA support for IMS will use the security environment (ACEE) passed in the dependent region's task (TCBSENV) if present, or the dependent region's address space (ASXBSENV) if the ACEE is not present at the task level.
ISIS=C and ODBASE=N	Create a Resource Access Security exit routine named DFSRAS00. This routine must determine if the user is authorized to use the PSB.

Specification	Actions to perform
ISIS=A and ODBASE=N	<ol style="list-style-type: none"> 1. Define the PSBs that you want protected by RACF to the IIMS or lxxxxxxx resource class, and then define the user IDs of the dependent region that you want to be authorized to access the PSBs. The ODBA support for IMS will use the security environment (ACEE) passed in the dependent region's task (TCBSENV), if present, or the dependent region's address space (ASXBSENV) if the ACEE is not present at the task level. 2. Create a Resource Access Security exit routine named DFSRAS00. This routine must determine if the user is authorized to use the PSB. RACF is called first, and then the exit routine is called.

APSB security

A security check is performed to determine if the user is authorized to use the PSB. Use the ODBASE execution parameter to control the authorization for a user to use a PSB. Table 19-6 describes the actions that you need to perform to set up security when specific options are selected.

Table 19-6 Options for defining APSB security for applications that use ODBA

Specification	Actions to perform
ODBASE=N	No action required. No APSB security checking is performed.
ODBASE=Y	<ol style="list-style-type: none"> 1. Define the PSBs that you want protected by RACF to the AIMS or Axxxxxxx general resource class (where xxxxxxx is the value specified on the RCLASS= parameter of the IMS SECURITY macro). 2. Specify RCLASS=IMS xxxxxxx and TYPE=RACFAGN RACFCOM RACFTERM RASRACF RAS on the IMS SECURITY macro at IMS system definition time.

ICMD security

For ICMD authorization, the ODBA application must set the AIBRSNM1 field to the PSBNAME that was used on the previous APSB call. The PSBNAME is used by the existing ICMD support routines to issue the RACF call for security processing.

ODBA security summary

Table 19-7 summarizes the values that you need to specify to control data access for specific security implementations. The table also indicates the type of security checking that is performed for each set of specifications.

Table 19-7 Options for controlling data access for applications that use ODBA

Security implementation	Authorization method	ISIS= specification	ODBASE= specification	Connection security	PSB security
AGN security	RACF	1	N	Y	Y
	User exit (DFSISIS0)	2	N	Y	Y

Security implementation	Authorization method	ISIS= specification	ODBASE= specification	Connection security	PSB security
Resource Access Security	RACF	R	N	N	Y
	User exit (DFSRAS00)	C	N	N	Y
	RACF and DFSRAS00	A	N	N	Y
	None	0 or N	N	N	Y
APSB security	RACF	N/A	Y	N	Y

For more information about ODBA security options, see the *IMS Version 9: Administration Guide: System*, SC18-7807.

Note: IMS Version 9 is the last release that will support SMU security. You still have some choices using AGN security in IMS Version 9, but we strongly recommend that you use RAS or APSB security for ODBA resource access control, instead of AGN security.

19.4 Writing ODBA application programs

The z/OS ODBA application programs run in a separate z/OS address space that IMS regards as a separate region from the control region called the z/OS application region.

The ODBA interface gains access to the IMS DB through the database resource adapter (DRA). The ODBA application programs (which can access to any address space within the z/OS in which they are running) gain access to IMS DB databases through the ODBA interface.

19.4.1 General application program flow

z/OS ODBA application programs issue DL/I calls using an application interface block (AIB). No other interface is supported. The z/OS application must link-edit with a language module (DFSCDLI0), or this module can be loaded into the z/OS application region. The entry point for DFSCDLI0 is AERTDLI.

Figure 19-3 on page 367 shows a simple example of the program flow of an z/OS application program.

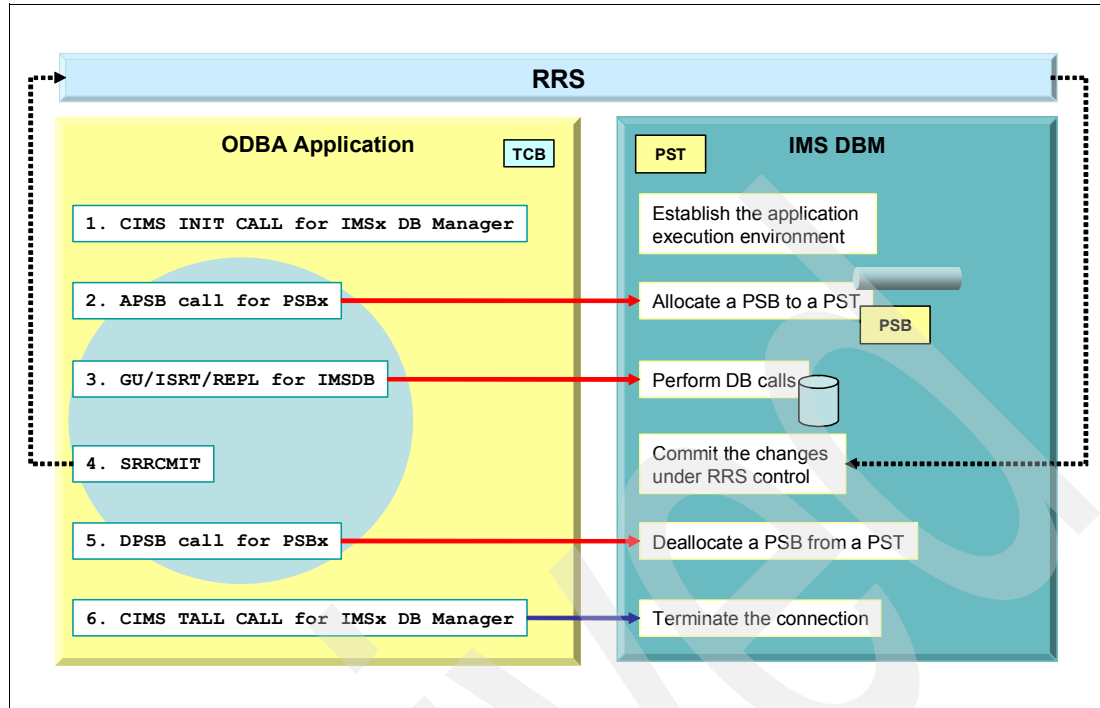


Figure 19-3 An example of the program flow of an ODBA application

19.4.2 Making calls to IMS

Your application program can be written in any language supported by IMS. You access IMS databases using the language-neutral CALL statement AERTDLI interface. Follow these considerations when using the AERTDLI interface:

- ▶ When using the AERTDLI interface for C/MVS™, COBOL, or PL/I language application programs, the language runtime options for suppressing abend interception (that is, NOSPIE and NOSTAE) must be specified. However, for Language Environment-conforming application programs, the NOSPIE and NOSTAE restriction is removed.
- ▶ The AERTDLI entry point for PL/I programs must be declared as assembler language entry (DCL AERTDLI OPTIONS(ASM)).
- ▶ For C language applications, you must specify env(IMS) and plis(IMS). These specifications enable the application program to accept the PCB list of arguments.
- ▶ AERTDLI must receive control with 31-bit addressability.

Example 19-1 shows the format of the CALL statement.

Example 19-1 The format of the AIB call

```
CALL AERTDLI, parmcount, function, AIB, .....
```

Where:

- ▶ parmcount is an optional parameter that specifies the address of a 4-byte field in the user-defined storage that contains the number of parameters in the parameter list that follows parmcount. This is kept for compatibility with the existing syntax for the IMS language interface module.

- ▶ function specifies the address of a 4-byte field in the user-defined storage that contains the function call. The function call must be left-aligned and padded with blanks, such as Gubb.
- ▶ AIB specifies the address of the application interface block.

19.4.3 The application interface block (AIB)

The application interface block (AIB) is used by application programs to issue IMS DL/I calls using a PCB name instead of a PCB address, or to issue calls that are not associated with a PCB. It provides a standard mechanism for IMS and the application to exchange information. The AIB mask enables application programs to interpret the control block that has been defined.

Note: The PSB used must have PCBNAME or LABEL defined.

Several conditions must be met for the AIB call to succeed:

- ▶ If an AIB is not passed in the call, a U261 abend is issued.
- ▶ If the AIB that is passed is not valid, a U476 abend is issued.
- ▶ If the AIB that is passed is not large enough (264 bytes), the AIB return and reason codes are set to X'104' and X'228'.
- ▶ If the AIB that is passed is not on a full word boundary, the z/OS system returns an abend S201.

If there are other internal problems with the call, other return and reason codes are passed back to the z/OS application program. Refer to *IMS Version 9: Messages and Codes Volume 1*, GC18-7827, for a complete list of these return and reason codes.

AIB structure

The AIB structure must be defined in working storage, on a full word boundary, and initialized according to the byte length and the field description column described here. Table 19-8 shows you the description of the AIB contents for ODBA AERTDLI calls.

Table 19-8 AIB fields for use of ODBA applications

AIB field	Offset	Length	Description
AIBID	0	8	This 8-byte field contains the AIB identifier. You must initialize AIBID in your application program to the value DFSAIBbb before you issue DL/I calls. This field is required. When the call is completed, the information returned in this field is unchanged.
AIBLEN	7	4	This field contains the actual 4-byte length of the AIB as defined by your program. You must initialize AIBLEN in your application program before you issue DL/I calls. The minimum length required is 264 bytes for ODBA. When the call is completed, the information returned in this field is unchanged. This field is required.
AIBSFUNC	11	8	This 8-byte field contains the subfunction code for those calls that use a subfunction. You must initialize AIBSFUNC in your application program before you issue DL/I calls. When the call is completed, the information returned in this field is unchanged.

AIB field	Offset	Length	Description
AIBRSNM1	19	8	<p>This 8-byte field contains the name of a resource. The resource varies depending on the call. You must initialize AIBRSNM1 in your application program before you issue DL/I calls. When the call is complete, the information returned in this field is unchanged. This field is required.</p> <p>For PCB-related calls where the AIB is used to pass the PCB name instead of passing the PCB address in the call list, this field contains the PCB name. The PCB name for the I/O PCB is IOPCBbb. The PCB name for other types of PCBs is defined in the PCBNAME= parameter in PSBGEN.</p>
AIBRSNM2	27	8	Specify a 4-character ID of ODBA startup table DFSxxxx0, where xxxx is a four-character ID.
Reserved	35	8	This 8-byte field is reserved.
AIBOALEN	43	4	This 4-byte field contains the length of the output area in bytes that was specified in the call list. You must initialize AIBOALEN in your application program for all calls that return data to the output area. When the call is completed, the information returned in this area is unchanged.
AIBOAUSE	47	4	This 4-byte field contains the length of the data returned by IMS for all calls that return data to the output area. When the call is completed, this field contains the length of the I/O area used for this call.
Reserved	51	12	This 12-byte field is reserved.
AIBRETRN	63	4	When the call is completed, this 4-byte field contains the return code.
AIBREASN	67	4	When the call is completed, this 4-byte field contains the reason code.
AIBERRXT	71	4	This 4-byte field contains additional error information depending on the return code in AIBRETRN and the reason code in AIBREASN.
AIBRSA1	75	4	When the call is completed, this 4-byte field contains call-specific information. For PCB-related calls where the AIB is used to pass the PCB name instead of passing the PCB address in the call list, this field returns the PCB address.
AIBRSA2	79	4	This 4-byte field is reserved for ODBA.
AIBRSA3	83	4	This 4-byte token, returned on the APSB call, is required for subsequent DLI calls and the DPSB call related to this thread.
Reserved	87	40	This 40-byte field is reserved.
Reserved	127	136	This 136-byte field is reserved for ODBA.

19.4.4 DL/I calls in the ODBA application

In this section, we discuss how you issue DL/I calls in the ODBA application, as shown in Figure 19-3 on page 367.

Establishing the application execution environment

The application execution environment must be initialized in the z/OS application region. Use the CIMS INIT call to initialize the environment. If the optional AIBRSNM2 field of the AIB contains a startup table ID, a connection to the IMS DB in the startup table is made. If the field is blank, connect to the IMS DB when you allocate a PSB.

Example 19-2 shows the form of the connection call.

Example 19-2 The form of the CIMS INIT call

```
CALL AERTDLI parmcount, CIMS, AIB
```

Where:

- ▶ CIMS is the required call function.
- ▶ AIB has the following fields:
 - AIBSFUNC
The subfunction is 'INITbbbb'. This field is mandatory.
 - AIBRSNM1
An optional field that provides an eye-catcher identifier of the application server that is associated with the AIB. This field is 8 bytes.
 - AIBRSNM2
Provides the optional 4-byte startup table ID. The ID is optional if the call is issued as preconditioning only. If the ID is given, the z/OS application region connects to the IMS DB specified in the DBCTLID parameter of the selected startup table. The characteristics of the connection are determined from the DRA startup table. The startup table name is DFSxxxx0, where xxxx is the startup table ID that is used in the CIMS and APSB calls. Each startup table defines a combination of connection attributes, one of which is a subsystem ID of the IMS DB.

Allocating a PSB

The APSB call is used with the ODBA interface to allocate a PSB for the z/OS application region. Example 19-3 shows the APSB call.

Example 19-3 The format of the APSB call

```
CALL AERTDLI parmcount, APSB, AIB
```

Where:

- ▶ APSB is the required call function.
- ▶ AIB is the name of the application interface block. The fields in the AIB must be filled in:
 - AIBRSNM1
The 8-character PSB name.
 - AIBRSNM2
The 4-byte startup table ID.

Performing DB calls

All DL/I calls, with a few exceptions, are supported through the AIB. The unsupported calls entail message handling (the IOPCB is available only for system calls), CKPT, ROLL, ROLB,

and INQY PROGRAM. Alternate destination PCBs cannot be used. Both full-function databases and DEDBs are available.

Existing DL/I calls fully supported by ODBA include:

- ▶ GU/GHU
- ▶ GN/GHN
- ▶ GNP/GHNP
- ▶ ISRT
- ▶ REPL
- ▶ DLET
- ▶ GMSG
- ▶ ICMD
- ▶ RCMD
- ▶ INIT
- ▶ INQY AIBSFUNC= FIND, DBQUERY, ENVIRON
- ▶ FLD
- ▶ POS
- ▶ LOG
- ▶ DEQ
- ▶ SETS/SETU
- ▶ ROLS
- ▶ SNAP
- ▶ STAT
- ▶ OPEN
- ▶ CLSE

The following calls are DL/I calls not supported by the ODBA interface. These calls are rejected with an AIB return code of X'104' and an AIB reason code of X'444':

- ▶ CHKP
- ▶ GSCD
- ▶ PCB
- ▶ XRST
- ▶ ROLL
- ▶ ROLB
- ▶ SYNC
- ▶ TERM
- ▶ INQY AIBSFUNC=PROGRAM

Committing changes

Synchronization is performed by issuing the distributed commit calls SRRCMIT or ATRCMIT, or possibly their rollback forms of SRRBACK or ATRBACK. IMS sync point calls are not allowed. ROLL and ROLB are not supported because RRS is used as the sync point coordinator. RRS provides the SRRBACK and ATRBACK calls to activate backout processing. The SRRBACK form of the call conforms to the Common Programming Interface (CPI) for Resource Recovery (RR) and is supported by RRS on the z/OS platform. The ATRBACK form of the call is provided by RRS and is only supported on the z/OS platform. Commit is effective for all RRS-controlled resources in the z/OS task.

Deallocating the PSB

The DPSB call is used when the work unit is complete. In the default case, a commit call must be issued before a DPSB call can be issued. No DL/I call, including system service calls, can be made between the commit and the DPSB call.

Example 19-4 on page 372 shows the DPSB call.

Example 19-4 The format of the DPSB call

CALL AERTDLI parmcount, DPSB, AIB

Where:

- ▶ DPSB is the required call function.
- ▶ AIB is the name of the application interface block. The fields in the AIB must be filled in:
 - AIBRSNM1
The 8-character PSB name.
 - AIBSFUNC
An optional field. Set it to 'PREPbbbb' when you want to deallocate the PSB before the initialization of commit processing and when the commit processing is provided from outside the application.

IMS performs phase one commit processing and returns control to the requestor, but holds the in-doubt work until RRS (the commit manager) requests full commit processing.

Terminating the connection

The CIMS TERM call is used for the connection termination between the IMS and the ODBA application.

Example 19-5 shows the CIMS call.

Example 19-5 The form of the CIMS call

CALL AERTDLI parmcount, CIMS, AIB

Where:

- ▶ CIMS is the required call function.
- ▶ AIB is the name of the application interface block. The fields in the AIB must be filled in:
 - AIBSFUNC
A mandatory field whose value is 'TERMbbbb' or 'TALLbbbb'. Use 'TERMbbbb' to sever a single IMS DB connection. Use 'TALLbbbb' to sever all connections for this z/OS application region and remove the DRA from the address space.
 - AIBRSNM1
An optional field that provides an eye-catcher of the application server associated with the AIB. This field is 8 bytes in length.
 - AIBRSNM2
When the subfunction equals TERM, provides the 4-byte startup table ID used in a previous APSB call. AIBRSNM2 is not needed when the subfunction equals TALL.

19.4.5 Server program structure and the unit of recovery

The commit scope within the z/OS application environment is all the work under the TCB from which the commit request is made to RRS. Each time the TCB expresses an interest in the unit of recovery (UOR), RRS creates the new unit of recovery identifier (URI) and assigns it to ODBA-related work for initiating the two-phase commit operation. Server environments, therefore, need a separate TCB under which the individual client requests will be managed. Each TCB will map to a PST for thread handling. Figure 19-4 on page 373 shows the TCB to PST relationship and the unit of recovery of ODBA-related work.

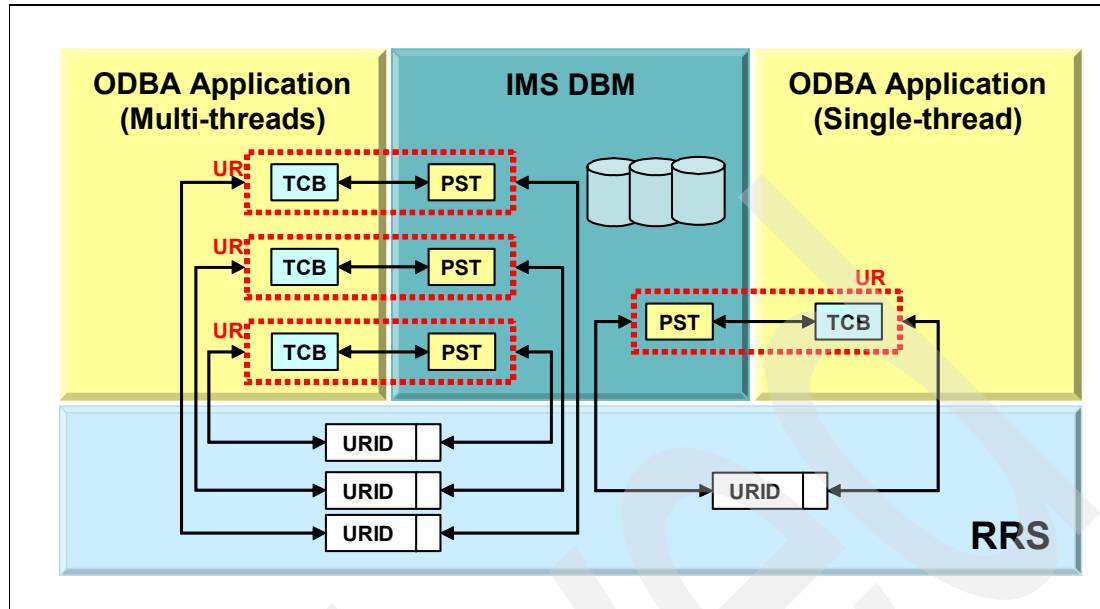


Figure 19-4 TCB to PST relationship and the unit of recovery

19.5 Considerations for using ODBA

In this section, we offer some considerations about the use of ODBA.

19.5.1 Restrictions

ODBA programs must execute on the same z/OS image (LPAR) as the IMS systems. Because GSAM databases are allocated by the dependent region (local), not by the IMS subsystem, GSAM databases are not allowed for ODBA.

19.5.2 Multiple access to IMS subsystems

Provided that you have coded the DRA startup table so that you can have more than one thread concurrently ($\text{MAXTHRD} > 1$), it is the application programmer's responsibility to keep track of each thread token after every APSB call by using the AIB.

Multiple PSBs can be active at the same time, which is typical for server environments. No token is specifically provided to identify which PSB is to be used for a given call to a given IMS DB, so the same AIB must be used for all calls to the same PSB instance (APSB, DB calls, DPSB). This enables multiple instances of the same PSB to be in use for the same IMS DB at the same time. The parallelism is controlled by the thread count specified in the startup table. The maximum number of threads and dependent regions supported by an IMS DB instance is 999.

Each APSB uses a free PST as the anchor for the IMS control blocks. If two APSB calls are issued from a single program (a single TCB), without an intervening CMIT/DPSB for the first PSB, there is a possibility that calls will be made to the same databases, and that the second PSTs locks will be in conflict with the first PSTs locks.

Because the first PST is active from an the IMS viewpoint, control in the application program is with the second PST. There will be a pseudo-deadlock, and the application will enter a permanent wait state that can only be broken by z/OS cancelling the program. Now, an abend

is detected during an IMS call, on the second PST, and the IMS system will terminate U113. This scenario also applies to calls made to shared databases across IMS. Rather than try to reuse existing PSBs for a set of databases, build and use a new super-set PSB.

19.5.3 IMS Fast Path resource usage

You need to specify the CNBA and FPB parameters in the DRA startup table for ODBA regions that require access to Fast Path resources. When the ODBA connects to DBCTL, the number of CNBA buffers is page-fixed in the Fast Path buffer pool. However, if CNBA buffers are not available, the connect fails. Each ODBA thread that requires DEDB buffers is assigned its Fast Path buffers out of the total number of CNBA buffers.

For planning and monitoring purposes, you can run the IMS monitor to check I/O to buffer pools, or use tools such as IMS Performance Analyzer to evaluate the impact of adding ODBA to your Fast Path buffer pools. During IMS control region initialization, parameters are read in and listed by DFS0578I messages.

The CNBA must be at least as large as the FPBUF x MAXTHRD, but small enough so as not to fail during identify:

$$\text{CNBA} = (\text{FPBUF} \times \text{MAXTHRD}) + \text{FPBOF}$$

A number of the IMS PROCLIB startup parameters in member DFSPBxxx relate to Fast Path. DBBF is the number of buffers in the IMS Fast Path buffer pool. DBFX is the number of IMS Fast Path buffers that are long-term page fixed during IMS startup (a subset of DBBF). BSIZ is the size of a single buffer. FPBUF is allocated from DBBF (CNBA must be less than DBBF).

All the FPBOF and OBA values for all dependent regions are compared. IMS gets a single OBA allocation based on the largest OBA value used in the IMS system (allocated out of DBBF). When one user needs it (watch for IMS status code GC), it becomes a serialized resource and it is something to avoid.

You can specify only one value on the DRA startup table for all thread connections. Therefore, if your applications need a different number of Fast Path buffers (for example, one application uses few Fast Path buffers for online processing, and another uses many Fast Path buffers for batch processing), you might have to create another DRA startup table for covering your application characteristics. For a detailed description of Fast Path DEDB buffer usage, see *IMS Version 9: Administration Guide: Database Manager*, SC18-7806.

19.5.4 The commit scope change and IMS resource occupancy

Generally, an ODBA application is designed as a server to provide IMS database access to its network client (TCP/IP client, DRDA® client, EJB client, and so on). In this environment, IMS is no longer the *coordinator* of the transaction but a *participant* controlled by RRS. In addition, RRS commit processing is initiated by the request of the network client. It means that now IMS commit processing time includes the coordinator (ODBA application and RRS) costs and the networking costs for two-phase commit messaging. From the IMS point of view, the longer lasting transactions affect to the following IMS resource occupancy:

- ▶ PSB/DMB pools
- ▶ Lock pools
- ▶ Database buffer pools (particularly Fast Path buffer pools)
- ▶ ODBA threads

In addition, it increases the possibility of lock request confliction or a deadlock situation between multiple PSTs, such as MPPs, IFPs, CCTL threads, and ODBA threads. When you introduce a new ODBA application to an existing IMS environment, you have to monitor your

IMS DB manager performance carefully and possibly take appropriate tuning actions when you find performance problems.

19.5.5 RRS logging performance

Because of the use of RRS, ODBA performance is related to the RRS logging performance. RRS uses z/OS logger and five log streams that can be shared by multiple systems in a sysplex. You have several choices of the z/OS logger implementation; you must consider that your choice should meet your performance and recovery requirements. For a description of configuring and defining RRS logging requirements, see *z/OS V1R6.0 MVS Programming: Resource Recovery*, SA22-7616.

19.6 Problem determination

Whenever your program terminates abnormally, you can take some actions to simplify the task of finding and fixing the problem. You already might have guidelines about what you should do if your program terminates abnormally. The suggestions given here are some common installation guidelines. Document the error situation to help in investigating and correcting it. Here is some of the information that can be helpful to document:

- ▶ Abends, return codes, or reason codes.
- ▶ Helpful messages: Depending on how your environment is set up, you might have to search the client location, local ODBA application messages, local z/OS messages (WLM, RRS, system), or the IMS subsystem.
- ▶ The program's PSB and PCB name.
- ▶ The call function.
- ▶ The contents of the application interface block (AIB).
- ▶ The contents of the I/O area when the problem occurred.
- ▶ The status of the database being accessed (stopped?).
- ▶ The date and time of day.

When your program encounters an error, it can pass all the required error information to a standard error routine.

You can send a message to the system log by issuing a LOG request.

19.6.1 Finding the problem

If your program does not run correctly, you need to isolate the problem. The problem might be anything from a programming error (for example, an error in the way you coded one of your requests), to a system problem. This section provides some guidelines about the steps that you, as the application programmer, can take when your program either fails to run, terminates abnormally, or gives incorrect results.

19.6.2 IMS initialization errors

Before your program receives control, IMS must have correctly loaded and initialized the PSB and DBDs used by your application program:

- ▶ Check for IMS error messages.

- ▶ Issue a /DIS PGM (psbname) and /DIS DB (dbname) to check the status of the IMS resources you need.
- ▶ Check to see if there have been any recent changes to the DBDs, PSBs, ACBs, or the IMS system maintenance.

If you have a problem in this area, you might need to talk to a DBA or IMS system personnel (or the equivalent specialist at your installation) to help with the problem.

19.6.3 Running errors

If you do not have any IMS initialization errors, check the following areas in your program:

- ▶ The output from the compiler. Make sure that all error messages have been resolved.
- ▶ The output from the linkage editor:
 - Are all external references resolved?
 - Have all necessary modules been included?
 - Was the language interface module correctly included?
- ▶ Your JCL: Is the information that described the files that contain the databases correct? If not, check with your DBA.
- ▶ If the program stalls and must be cancelled, check that there are CMIT/DPSB calls prior to a new APSB for the same databases.

19.6.4 The application interface block

For an explanation of the application interface block (AIB), refer to 19.4.3, “The application interface block (AIB)” on page 368. Always check the IMS status code, the reason code, and the return code after an IMS call. The reason code is contained in the AIBREASN field, and the return code is contained in the AIBRETRN field. Check for non-zero values for the reason and return codes, and unexpected non-blank spaces for the status code (for example, GE and GB might be valid status codes indicating no more segment found, or end of database reached).

19.7 IBM-supplied ODBA infrastructures

You do not have to code an ODBA application by yourself, because IBM provides several ODBA applications to meet your requirements. These ODBA applications work as infrastructure to execute your business logic that access to IMS databases. You can code the business logic by using various API such as DL/I or SQL through ODBC and JDBC interfaces, and you can run it on any platforms. In this section, we introduce the following IBM-supplied ODBA infrastructures.

19.7.1 DB2 stored procedure

The ODBA interface allows a DB2 stored procedure to directly connect to an system and issue DL/I calls to access IMS databases. A stored procedure can issue database DL/I requests through an ODBA callable interface. You need to update the startup procedure for the WLM-established stored procedure address space to add the ODBA data set names to the STEPLIB and DFSRESLB concatenations. Figure 19-5 on page 364 shows the general structure of ODBA from a DB2 stored procedure. After you have coded your stored procedure containing the logic for IMS database access, you can call this stored procedure from any platform that can access DB2 for z/OS stored procedure environment.

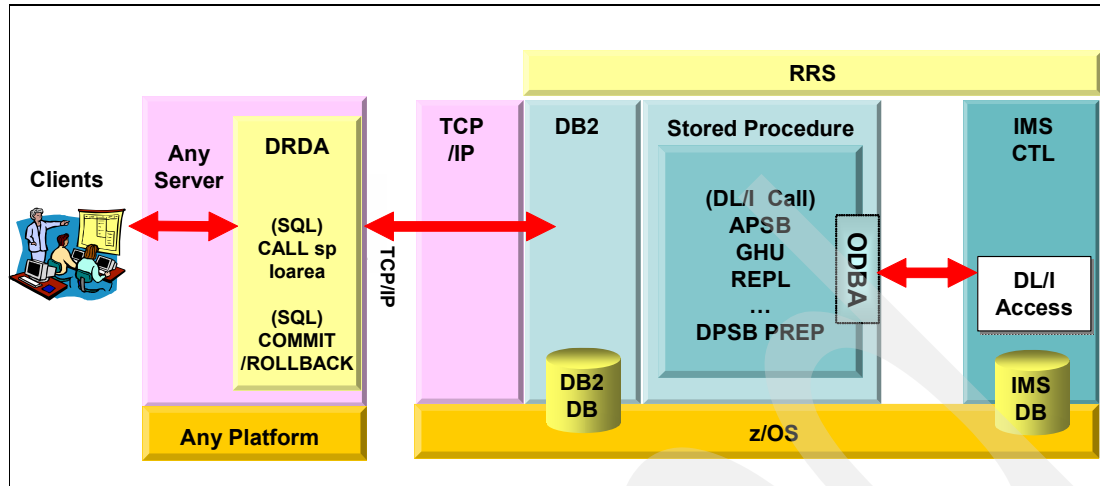


Figure 19-5 The general structure of ODBA from a DB2 stored procedure

For more information about the ODBA usage of DB2 stored procedure address space, see Chapter 20, “ODBA from DB2 stored procedures” on page 383.

19.7.2 WebSphere Application Server for z/OS and IMS Remote Data Access

WebSphere Application Server for z/OS Java applications can access IMS database directly through the IMS ODBA interface. You can use the classes in the IMS Java library to build a WebSphere Application Server for z/OS that accesses IMS data when WebSphere Application Server for z/OS and IMS are running on the same z/OS image. The WebSphere Application Server for z/OS application you build can access both the IMS full function and data entry databases using the IMS ODBA interface.

To provide this capability in WebSphere Application Server for z/OS, the IMS Java class library implements the J2EE Connector architecture resource adapter interfaces known as the IMS JDBC resource adapter.

With IMS Version 9, IMS Java Remote Database Services (RDS) support provides architected components on both the client and server side that allow a Java application deployed on a distributed WebSphere Application Server platform to issue JDBC calls for IMS data. These requests are sent (transparently to the application) across the network and processed in IMS.

For the client-side of the connection, IMS Java RDS provides an IMS distributed JDBC resource adapter. To condition the distributed WebSphere Application Server for JDBC access to IMS, the IMS JDBC resource adapter must first be installed. For the server-side (IMS side) of the connection, a server EJB is delivered as part of the IMS Java RDS support. This EJB must reside on a WebSphere Application Server for z/OS environment because it uses the IMS JDBC resource adapter.

Figure 19-6 on page 378 shows the ODBA from WebSphere Application Server for distributed platforms.

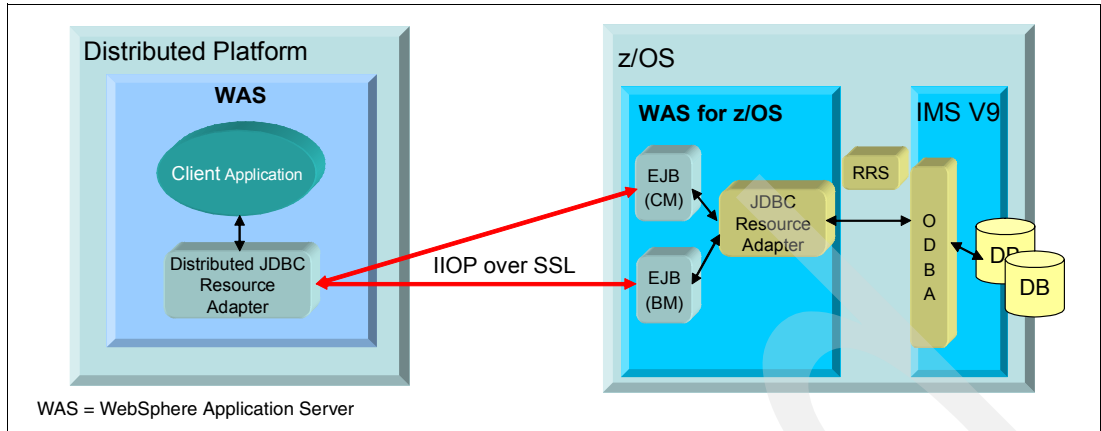


Figure 19-6 The ODBA from WebSphere Application Server for distributed platforms

For more information about IMS Remote Database Services, see Chapter 21, “IMS Remote Database Services” on page 405.

19.7.3 WebSphere Information Integrator Classic Federation for z/OS

WebSphere Information Integrator Classic Federation for z/OS provides SQL access to mainframe databases (including IMS databases) and files with transactional speed and enterprise scale without mainframe programming. Using WebSphere Information Integrator Classic Federation for z/OS applications can:

- ▶ Map mainframe assets into relational views.
- ▶ Build integrated views across mainframe databases and files co-resident on a single z/OS instance.
- ▶ Issue SQL SELECT, INSERT, UPDATE, and DELETE statements from ODBC, JDBC, or a call level interface (CLI) to access mainframes, databases, and files.

Figure 19-7 on page 379 shows an overview of WebSphere Information Integrator Classic Federation.

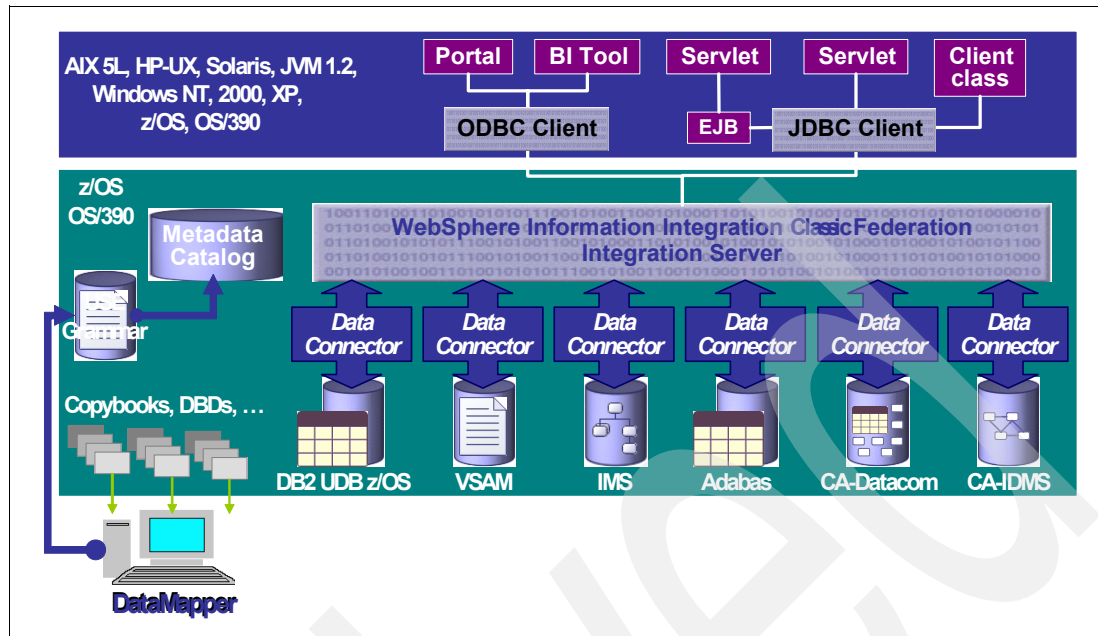


Figure 19-7 Overview of WebSphere Information Integrator Classic Federation for z/OS

In the next sections, we introduce the component parts of WebSphere Information Integrator Classic Federation.

Metadata catalog and data mapper

The metadata catalog holds a mapping between the virtual relational database catalog information and the underlying physical data constructs. The metadata catalog acts just like an RDBMS catalog for the tools and applications that will use the mainframe data. WebSphere Information Integrator Classic Federation creates an RDBMS catalog that holds the virtual relational table and view definitions that will be used by the SQL-enabled tools and applications to access data. Behind the scenes, the metadata catalog stores the mapping of the virtual RDBMS metadata to the physical database and file metadata. The metadata is used to put a fully functioning relational face on physically very non-relational data constructs. So depending on the type of databases that an organization has, they are going to have things such as COBOL copybooks. In case of an IMS user, they are going to have DBD, database definitions. All of these are sources of information for the utilities and data mapper that create and manage the WebSphere Information Integrator Classic Federation metadata.

Federation server

Residing on the z/OS platform, the federation server is responsible for communicating back and forth between the client applications and tools and the actual database access connectors. The server is responsible for determining what data needs to participate in the processing of the SQL statement that it receives and invoking appropriate data access connectors. It also provides for mainframe resource management and security.

After the server has determined which database connectors are needed to process the SQL statement, it provides the file-specific connectors control called *data connector*. These components then determine how to access the database.

Data connector

When your tools and applications access the data through the virtual relational structures, specific data connectors execute native database file access through database and file

system. Each of these data connectors is specifically built and therefore optimized for the type of database or file system that it accesses.

In case of IMS, the data connectors provide IMS-specific read/write services that parse SQL requests and translate them to native DL/I calls and issue against the IMS Database Manager. You can choose three kinds of data connector interfaces for IMS database access:

- ▶ **IMS BMP/DBB/DLI interface**

The BMP/DBB/DLI interface is a non-scalable interface that is primarily provided for initial development and testing purposes. With this interface, only a single PSB is used and all access is serialized through that PSB. When using this interface, you can access IMS data in a BMP or a DBB environment. In these environments, you must configure and activate an IMS BMP/DBB initialization service to access your IMS data.

- ▶ **IMS CCTL DRA interface**

The CCTL DRA interface is similar to the interface that CICS uses. DRA is the recommended interface to use when your client applications are not executing distributed transactions (two-phase commit). When using a DRA interface, you must configure and activate an IMS DRA initialization service to access your IMS data.

- ▶ **IMS ODBA DRA interface**

You must use the ODBA interface when your client applications are executing distributed transactions using two-phase commit protocols. The ODBA interface must be used in conjunction with the two-phase commit query processor (CACQPRRS), and you need to configure and activate an IMS ODBA initialization service. If you choose this interface, the federation server and the data connector work as the ODBA application.

Clients

WebSphere Information Integrator Classic Federation clients look like standard drivers to the applications and tools that interact with them. During installation, specific configuration parameters are set that control communication settings between this client and the federation server. You can use standard SQL 92 API through the JDBC/ODBC interface on any WebSphere Information Integrator Classic Federation supported platforms.

19.8 Summary of IBM-supplied ODBA infrastructures

Table 19-9 shows a summary of IBM-supplied ODBA infrastructures. It shows you an execution environment of your application, an interface of application, and the available APIs of the IMS database access for each infrastructures.

Table 19-9 Summary of IBM-supplied ODBA infrastructures

ODBA infrastructure	Execution environment	Interface	API
DB2 stored procedures	DB2 for z/OS with WLM	DL/I AIB interface JDBC	DLI call IMS Java SQL
DB2 stored procedures caller	Any platform that can access the DB2 for z/OS stored procedures environment	Any interface supported by the DB2 for z/OS stored procedures environment	Any stored procedures call supported by the DB2 for z/OS stored procedures environment
WebSphere Application Server for z/OS	WebSphere Application Server for z/OS	JDBC	IMS Java SQL

ODBA infrastructure	Execution environment	Interface	API
IMS RDS	WebSphere Application Server for distributed systems	JDBC	IMS Java SQL
WebSphere Information Integrator Classic Federation	Any platform supported by Information Integrator Classic Federation	JDBC/ODBC	Standard SQL 92

Archived

ODBA from DB2 stored procedures

The IBM DB2 stored procedures environment is a case of the general ODBA server structure described in Chapter 19, “Open Database Access” on page 359. A stored procedure can issue database DL/I requests through an ODBA interface that is implemented in DB2 stored procedures environment. In this chapter, we provide the information about the structure of ODBA from DB2 stored procedures, the general request and response flow from a DRDA client, and some examples of DB2 stored procedures accessing an IMS system through the ODBA.

20.1 A short introduction to DB2 stored procedures

A stored procedure is a compiled program, stored at a DB2 local or remote server, that can execute SQL statements. A typical stored procedure contains two or more SQL statements and some manipulative or logical processing in a host language. A client application program uses the SQL statement CALL to invoke the stored procedure.

A stored procedure can contain one or more SQL statements. You can replace your complex SQL statements with a single stored procedure. Because stored procedures are precompiled programs, they execute faster at the database server. Most of the time, stored procedures contain more than one SQL statement, thus the time to pass the individual SQL statements to the database server from the program is saved. A client issues just one call (to execute the stored procedure) and the DB server executes all the commands and returns the result. Therefore, the overall interaction time with the DB server reduces effectively. This can result in a large optimization in cases where the DB server is accessed through a slow network. Figure 20-1 shows processing with stored procedures.

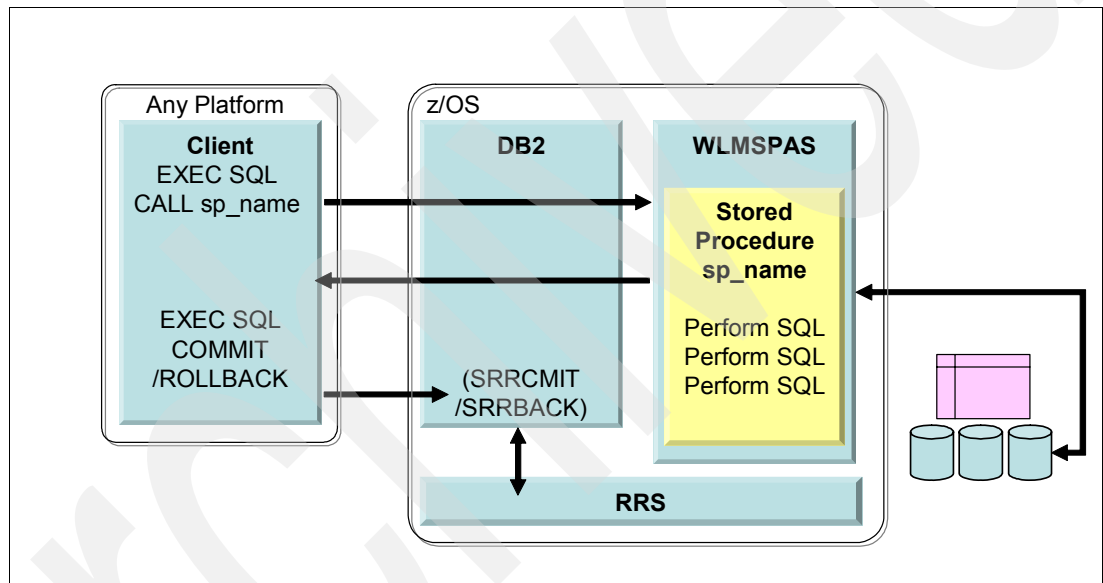


Figure 20-1 Processing with stored procedures

For more information about DB2 stored procedures, see the IBM Redbook *DB2 for z/OS Stored Procedures: Through the CALL and Beyond*, SG24-7083.

20.2 DB2 stored procedures' use of ODBA

DB2 stored procedures connecting to ODBA require DB2 Version 5 or later and must run in a Workload Manager (WLM)-managed stored procedures address space. Figure 20-2 on page 385 shows the flow of a DB2 stored procedure using ODBA.

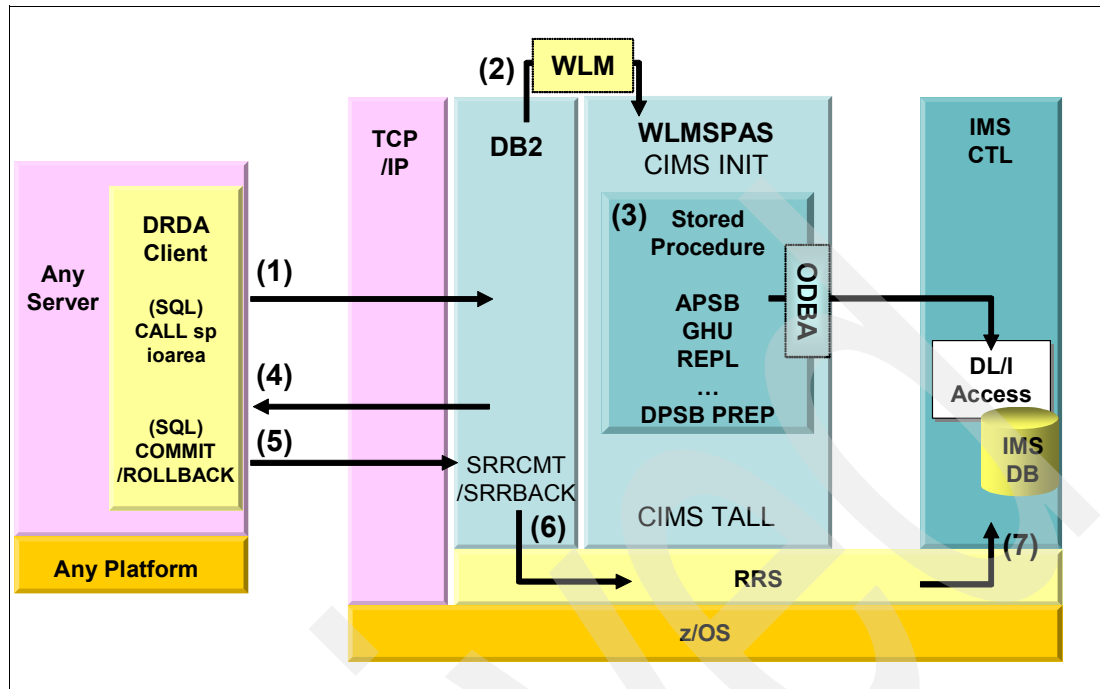


Figure 20-2 The flow of a DB2 stored procedure using ODBA

The following numbers describe the flow illustrated in Figure 20-2:

1. When a client application issues an SQL CALL statement, the stored procedure name and I/O parameters are passed to DB2.
2. When DB2 receives the SQL CALL statement, it searches in the SYSIBM.SYSROUTINES catalog table for a row associated with the stored procedure name. From this table, DB2 obtains the load module associated with the stored procedure and the run environment information. Then, the stored procedure is executed in the WLM-managed address space that runs fenced away from the DB2 code.
3. When the DB2 stored procedures address space (DSNX9WLM) starts up, the presence of a DFSRESLB DD statement causes it to issue a CIMS INIT call implicitly without specifying a DRA startup table. At this point, the address space does not know yet in advance which IMS subsystem to which the stored procedure wants to connect. The IMS subsystem to be accessed is determined based on the DRA startup table. When the ODBA program issues an explicit APSB call, it provides a 1-4 byte identifier that makes up the DRA startup table name and the PSB name to schedule. An APSB call establishes a thread to IMS, and ODBA returns a thread token associated with this thread. All subsequent IMS calls need to refer to this thread token. Each stored procedure running in the stored procedure address space runs under its own TCB that was established by DB2 when the stored procedure is initialized.
4. When the stored procedure execution is completed, DB2 returns control to the invoking program with the output parameters.
5. When the SQL commit call is issued by the client program, DB2 handles the commit process on behalf of the stored procedure. The only subfunction that should be issued by the stored procedure themselves is the PREP subfunction of the DPSB call.

6. When DB2 receives the commit request, DB2 invokes RRS Attachment Facility (RRSAF) to commit the changes. RRSAF works in conjunction with the application program, the resource manager (IMS/DB2), and the sync point manager (RRS) to ensure that updates to IMS/DB2 resources and other protected resources are synchronized across a unit of work. Either all work is committed, or all work is backed out.
7. Finally, IMS commits its updates under two-phase commit protocol of RRS.

20.3 Sample ODBA using DB2 stored procedures

The following sections describe how to implement a DB2 stored procedure to access an IMS system through the ODBA. We use the DSNTEJ61/DSNTEJ62 ODBA sample shipped with the DB2 product. This sample is used to access an IMS full function database to insert and retrieve segments. All sample code is in data set DSN810.SDSNSAMP.

We use DB2 UDB Version 8 and IMS Version 9 subsystems, which are already up and running (we do not describe here how to set up these subsystems).

The tasks can be divided into the following areas:

1. Setting up ODBA for the IMS subsystem: Customize the DRA startup table. It enables DB2 stored procedures and other applications to use ODBA for this IMS subsystem.
2. Setting up the DB2 stored procedure address space for ODBA: Set up a new WLM-established DB2 stored procedure address space, or modify an existing one.
3. Setting up a new WLM application environment for this address space.
4. Building the stored procedure: Compile, link-edit, and bind (if the stored procedure contains SQL statements). Define the stored procedure to DB2 using a CREATE PROCEDURE SQL statement.
5. We need to make sure that the IMS resources our stored procedure will reference are defined to IMS. This includes building the required IMS control blocks, DBD, PSB, and ACB, and performing MODBLKS system generation and activating the changes by making an online change.
6. Executing the application.
7. Analyzing the output.

Our example describes the first four steps. These examples do not require setting up any new IMS DBD or PSB, provided you already have a suitable PSB to use if you installed the IMS IVP system. Later, we illustrate steps 6 and 7 by executing the application.

20.3.1 Provided sample jobs

This sample consists of two jobs: DSNTEJ61 and DSNTEJ62. Because the stored procedure and the client program in these jobs are written by COBOL, you must have the COBOL for z/OS and Language Environment installed. The following sections provide some information about each job.

DSNTEJ61: Creating the sample stored procedure

This JCL creates a sample stored procedure application, DSN8EC1, that demonstrates a DB2 stored procedure for IMS ODBA. DSN8EC1 can be used to insert, retrieve, update, and delete rows in the IMS IVP telephone directory database, DFSIVD1.

The following dependencies apply:

- ▶ Run this job at the server site before running sample job DSNTEJ62 at the client site.
- ▶ The server site must have an IMS subsystem running.
- ▶ The IMS subsystem must have the following IMS IVP parts available:
 - DFSIVD1, the IMS IVP telephone directory database (HIDAM / OSAM)
 - DFSIVP64, the IMS IVP Cobol PSB for BMP access to DFSIVD1
- ▶ Specify the ID for this IMS subsystem in DB2 sample job DSNTEJ62, step PH062S03.
- ▶ The server site must also have a WLM environment started by a procedure that references the IMS SDFSRESL in both the STEPLIB DD and the DFSRESLB DD.
- ▶ Before running job DSNTEJ61, verify that this WLM environment is the one specified in the CREATE PROCEDURE statement in step PH061S01.

Table 20-1 shows the steps included in the DSNTEJ61 job and their purpose.

Table 20-1 DSNTEJ61 job

Step name	Description
PH061S01	Drop the sample ODBA stored procedure.
PH061S02	Create the sample ODBA stored procedure.
PH061S03	Precompile, compile, and link-edit the stored procedure.
PH061S04	Bind the stored procedure package (you must execute this step if you add SQL statements in the stored procedure).

DSNTEJ62: Executing the stored procedure

This JCL prepares and executes a sample application program, DSN8EC2, that demonstrates how to call a DB2 stored procedure for IMS ODBA. The results are directed to the SYSOUT DD. DSN8EC2 accepts a runtime parameter in step PH062S03 that specifies the DB2 server location name where the stored procedure is registered and the ID of the IMS subsystem where the ODBA activity is to occur. You must modify this job to provide the IMS subsystem ID.

The following dependencies apply:

- ▶ Run the sample job DSNTEJ61 at the server site before running this job; DSNTEJ61 prepares the sample stored procedure for IMS ODBA.
- ▶ Modify this job as directed in step PH062S03.
- ▶ Run this job at the client site.

Table 20-2 shows the steps included in DSNTEJ62 job and their purpose.

Table 20-2 DSNTEJ62 job

Step name	Description
PH062S01	Precompile, compile, and link-edit the client program.
PH062S02	Bind the client program package and plan.
PH062S03	Invoke the client program.

20.3.2 Provided sample source codes

This sample consists of two source codes referenced by sample jobs: DSN8EC1 and DSN8EC2. The following sections provide some information about each source code.

DSN8EC1: The stored procedure application for IMS IVP DB access

This stored procedure enables its client to add, retrieve, update, and delete entries in the IMS IVP telephone directory database thorough DL/I AIB interface. You also might want to add more displays for debugging purposes.

The following dependencies apply:

- You have to change the static variable APSBNME from 'DFSIVP6' to 'DFSIVP64'. See Example 20-1.

Example 20-1 APSBNME definition

```
* Initializers
77 SSA1          PIC X(9)  VALUE 'A1111111 ' .
77 APSBNME       PIC X(8)  VALUE 'DFSIVP64' .
77 DPCBNME       PIC X(8)  VALUE 'TELEPCB1' .
77 VAIBID        PIC X(8)  VALUE 'DFS AIB ' .
77 SFPREP        PIC X(4)  VALUE 'PREP' .
```

- The DPSB function should be performed to deallocate the thread token in case of some type of failure. In the A00000-ODBA-SP section, you have to comment out 'IF OKAY', which resides on 'PERFORM B40000-DEALLOCATE-AIB' so that the deallocate runs every time. See Example 20-2.

Example 20-2 A00000-ODBA-SP section

```
A00000-ODBA-SP.
  MOVE 'GOOD' TO RUN-STATUS.
  PERFORM B10000-ALLOCATE-AIB.
  IF OKAY THEN
    PERFORM B20000-PREPARE-REQUEST.
  IF OKAY THEN
    PERFORM B30000-PROCESS-REQUEST.
*  IF OKAY THEN
    PERFORM B40000-DEALLOCATE-AIB.

  STOP RUN.
```

DSN8EC2: The client application that calls the stored procedure

This client application enables you to access the stored procedure DSN8EC1. There is no dependency for this client application. You also might want to add more displays for debugging purposes.

20.4 Step-by-step instructions for using the sample

For the purpose of this sample, we use the parameters shown in Table 20-3.

Table 20-3 Parameters

Parameter	Value
IMSID	IM1B

Parameter	Value
DB2 server location	DB8P
WLM environment name	DB8PODBA

Figure 20-3 shows a rough flow of our DB2 stored procedure environment.

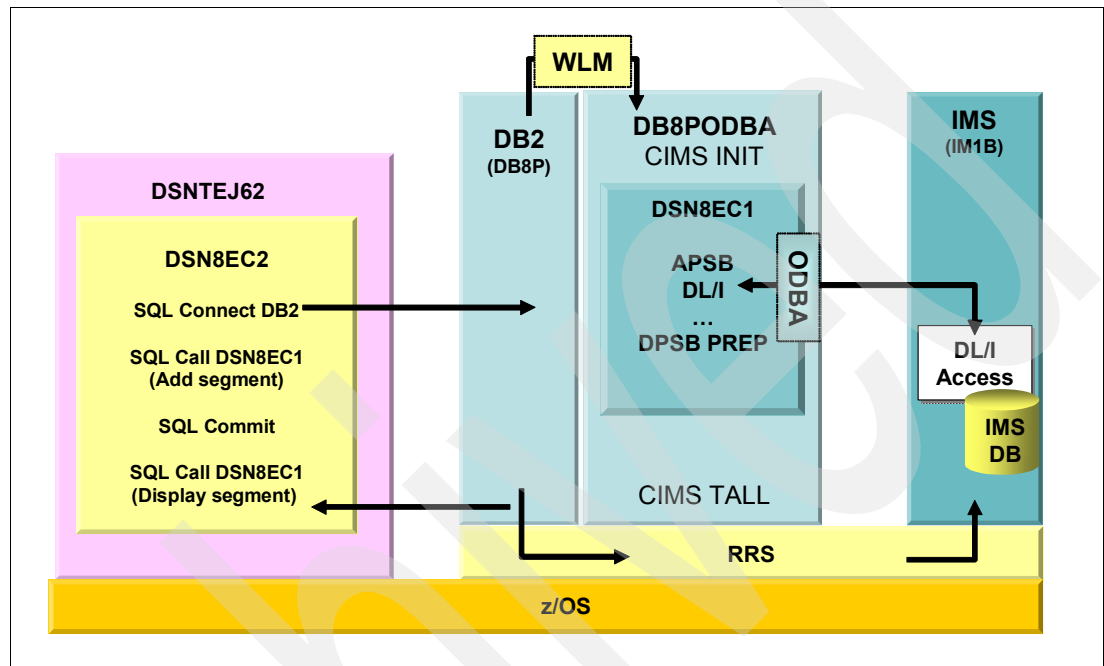


Figure 20-3 The flow of the DB2 stored procedure sample

20.4.1 Step 1: Creating an IMS DRA startup table

Application programs (in particular DB2 stored procedures) use information specified in the DRA startup table to establish an ODBA connection to an IMS system. The DRA startup table contains values used to define the characteristics of the DRA. The DRA startup table is created by assembling the DFSxxxx0 module for the ODBA's use.

The ODBA system programmer must make the required changes to these modules to correctly specify the DRA parameters desired. For more information about creating the DRA startup table, refer to 19.3.1, "Creating the ODBA DRA startup table" on page 361.

We customize the IMS IVP member IV_E308J for this sample. Example 20-3 is an example of the JCL to assemble and link-edit the DRA startup table.

Example 20-3 Assemble and link-edit the DRA startup table

```
//IV3E308J JOB (999,P0K),
// 'JJ',
// CLASS=A,
// MSGCLASS=H,MSGLEVEL=(1,1),
// NOTIFY=JOUK02,
// REGION=64M
//*
// JCLLIB ORDER=(IMSPSA.IMOB.PROCLIB)
/*JOBPARM L=9999,SYSAFF=*
```

```

/*
//ASMDRA  PROC MBR=TEMPNAME
/*
//ASM      EXEC PGM=ASMA90,PARM='OBJECT,NODECK'
//SYSPRINT DD SYSOUT=*
//SYSLIB   DD DISP=SHR,DSN=IMSPSA.IMOB.ADFSMA
//         DD DISP=SHR,DSN=SYS1.MACLIB
//SYSLIN   DD UNIT=3390,DISP=(,PASS,DELETE),SPACE=(CYL,(1,1)),
//         DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
//SYSUT1   DD UNIT=3390,DISP=(,DELETE,DELETE),SPACE=(CYL,(1,1))
//SYSUT2   DD UNIT=3390,DISP=(,DELETE,DELETE),SPACE=(CYL,(1,1))
//SYSUT3   DD UNIT=(3390,SEP=(SYSLIB,SYSUT1,SYSUT2)),
//         DISP=(,DELETE,DELETE),SPACE=(CYL,(1,1))
/*
//LKED     EXEC PGM=IEWL,COND=(0,LT,ASM),
//         PARM='NCAL,LET,LIST,XREF'
//SYSPRINT DD SYSOUT=*
//SYSLIN   DD DISP=(OLD,DELETE,DELETE),
//         DSN=*.ASM.SYSLIN,VOL=REF=*.ASM.SYSLIN
//SYSLMOD  DD DISP=SHR,
//         DSN=IMSPSA.IMOB.SDFSRESL(&MBR)
//SYSUT1   DD UNIT=(3390,SEP=(SYSLMOD,SYSLIN)),
//         DISP=(,DELETE,DELETE),SPACE=(CYL,(1,1))
//         PEND
/*
//DFSPZPIV EXEC PROC=ASMDRA,MBR=DFSIM1B0
//ASM.SYSIN DD *
            TITLE 'DATABASE RESOURCE ADAPTER STARTUP PARAMETER TABLE'
DFSIM1B0 CSECT
EJECT
DFSFRP DSECT=NO,
        FUNCLV=1,          X          CCTL FUNCTION LEVEL          X
        DDNAME=DFSDB2SP,   XXXXXXXX DDN FOR CCTL RESLIB DYNALOC X
        DSNAME=IMSPSA.IMOB.SDFSRESL,
        DBCTLID=IM1B,      NAME OF DBCTL REGION                    X
        USERID=,           XXXXXXXX NAME OF USER REGION          X
        MINTHRD=001,       XXX          MINIMUM THREADS            X
        MAXTHRD=005,       XXX          MAXIMUM THREADS            X
        TIMER=60,          XX          IDENTIFY TIMER VALUE - SECS X
        FPBUF=005,         XXX          FP FIXED BFRS PER THREAD    X
        FPBOF=003,         XXX          FP OVFLW BFRS PER THREAD    X
        SOD=A,             X          SNAP DUMP CLASS              X
        TIMEOUT=060,       XXX          DRATERM TIMEOUT IN SECONDS X
        CNBA=028,          XXX          TOTAL FP NBA BFRS FOR CCTL  X
        AGN=               XXXXXXXX APPLICATION GROUP NAME
        END
/*

```

20.4.2 Step 2: Setting up the DB2 stored procedure address space for ODBA

We use a DB2 stored procedure called DB8PODBA for this sample. See Example 20-4 for a sample of the JCL we use for this procedure.

Example 20-4 Sample stored procedure PROCLIB member

```

//DB8PODBA PROC RGN=OK,APPLENV=DB8PODBA,DB2SSN=DB8P,NUMTCB=8
//IEFPROC EXEC PGM=DSNX9WLM,REGION=&RGN,TIME=NOLIMIT,
//         PARM='&DB2SSN,&NUMTCB,&APPLENV'
//STEPLIB DD DISP=SHR,DSN=DB8PU.RUNLIB.LOAD

```

```
//      DD  DISP=SHR,DSN=DB8P8.SDSNEXIT
//      DD  DISP=SHR,DSN=DB8P8.SDSNLOAD
//      DD  DISP=SHR,DSN=CEE.SCEERUN
//      DD  DISP=SHR,DSN=IMSPSA.IMOB.SDFSRESL
//DFSRESLB DD  DISP=SHR,DSN=IMSPSA.IMOB.SDFSRESL
//CEEDUMP DD  SYSOUT=*
//SYSMDUMP DD  SYSOUT=*
```

We add the following DD statements to the JCL:

- ▶ To the STEPLIB DD statement, add the library where the DRA startup table resides, in our case, the SDFSRESL.
- ▶ The WLM-established DB2 stored procedure address space needs a DFSRESLB DD statements that includes the DRA startup table and DRA modules DFSCDLI0, DFSAERM0, DFSAERA0, and DFSAERG0 in order to establish an ODBA environment during startup. These modules need to be APF authorized. Every data set in this DD statement needs to be APF authorized.
- ▶ In addition, add the CEEDUMP DD for debugging purposes. Add any additional data sets required for the stored procedure. For example, a PL/I stored procedure might require to SYSPRINT in order to help for debugging.

20.4.3 Step 3: Creating the WLM application environment

Note: We do not go into the details of how to set up the WLM. We assume that the WLM is set up for your systems with the use of RRS. For more information about WLM, see Chapter 4, “Setting up and managing Workload Manager,” in the IBM Redbook *DB2 for z/OS Stored Procedures: Through the CALL and Beyond*, SG24-7083.

Use the ISPF application “WLM” to define the application environment. Use Option 9 (Application Environments) with an action code of 1(create) or 2 (copy) to input the application information. Example 20-5 shows information inputted for this sample.

Example 20-5 WLM Application Environment definition for DB2 ODBA stored procedures

Application-Environment	Notes	Options	Help

Create an Application Environment			
Command ==> _____			
Application Environment . . .	DB8PODBA		Required
Description	SP for DB8P ODBA on IM1B _____		
Subsystem Type	DB2_	Required	
Procedure Name	DB8PODBA		
Start Parameters	DB2SSN=DB8P,NUMTCB=8 _____		

Limit on starting server address spaces for a subsystem instance:			
1	1. No limit		
	2. Single address space per system		
	3. Single address space per sysplex		

After you enter this data and it is saved, select **Utilities** from the menu bar, and select **Install Definition** to install and select **Activate service policy** to make it effective.

20.4.4 Step 4: Building the stored procedure by DSNTEJ61

The DSNTEJ61 job is supplied in the SDSNSAMP library of your DB2 system libraries. This job needs to be modified to allocate the correct DB2 libraries and point to the correct DB2 subsystems and programs. The following sections list the steps of each job and what task they perform.

Step 1 PH061S01: Dropping the stored procedure

This step drops the stored procedure for cleaning up your previous executions. At the first execution, you can comment out this step. See Example 20-6.

Example 20-6 Drop the sample ODBA stored procedure

```
DROP PROCEDURE DSN8.DSN8EC1 RESTRICT;
```

Step 2 PH061S02: Creating the stored procedure

Make sure that the WLM environment name is correct in this definition. For our sample, we use DB8PODBA. See Example 20-7.

Example 20-7 Create procedure statements

```
CREATE PROCEDURE
  DSN8.DSN8EC1(
    IN CHAR(8)      CCSID EBCDIC,
    INOUT CHAR(8)   CCSID EBCDIC,
    INOUT CHAR(10)  CCSID EBCDIC,
    INOUT CHAR(10)  CCSID EBCDIC,
    INOUT CHAR(10)  CCSID EBCDIC,
    INOUT CHAR(7)   CCSID EBCDIC,
    OUT INT,
    OUT INT,
    OUT CHAR(4)     CCSID EBCDIC )
  FENCED
  RESULT SETS 0
  EXTERNAL NAME DSN8EC1
  LANGUAGE COBOL
  PARAMETER STYLE GENERAL
  NOT DETERMINISTIC
  NO SQL
  NO DBINFO
  NO COLLID
  WLM ENVIRONMENT DB8PODBA
  ASUTIME LIMIT 50
  STAY RESIDENT NO
  PROGRAM TYPE MAIN
  SECURITY DB2
  RUN OPTIONS 'TRAP(OFF),RPTOPTS(OFF),TERMTHDAC((QUIET),NONOVR)'
  COMMIT ON RETURN NO;
```

Step 3 PH061S03: Precompiling, compiling, and link-editing stored procedure

This step precompiles, compiles, and link-edits the stored procedure. You do not have to include DFSCDLI0 in the link-edit control statement, because the WLM stored procedure address space already has an ODBA interface. See Example 20-8.

Example 20-8 Link-editing condition

```
INCLUDE SYSLIB(DSNRLI)
NAME DSN8EC1(R)
```

Step 4 PH061S04: Binding the stored procedure package

This example does not use any SQL; therefore, we do not run this step.

Note: The PC and PLKED steps in the PH061S03 will get a return code of 04. This is acceptable. All other steps should return 00.

20.4.5 Step 5: Defining the IMS environment

IMS ODBA function requires an RRS interface. Therefore, the IMS execution parameter RRS= in the DFSPBxxx PROCLIB member must be set to Y (the default is N). If you execute your ODBA stored procedure with RRS=N, your APSB call will receive AIBRETRN X'0108', AIBREASN X'0544', which means that the RRS is not active at the time that ODBA attempts to establish a connection to IMS or DBCTL.

We need to make sure that the IMS IVP application program (which is used by our stored procedure) will reference is defined to IMS. This includes building the required IMS control blocks (DBD, PSB, and ACB) and MODBLKS modules that are part of stage 1 and stage 2 IMS generation, and copying them into online libraries. We use the DBD IVPDB1, PSB DFSIVP64, and ACB members that come as part of the IMS IVP environment.

You can confirm your IVP application environment for the stored procedure by some IMS commands, as shown in Example 20-9.

Example 20-9 IMS commands sample

```
R 394,/DIS DB IVPDB1 IVPDB1I
IEE600I REPLY TO 394 IS;/DIS DB IVPDB1 IVPDB1I
DFS000I      DATABASE  TYPE  TOTAL UNUSED  TOTAL UNUSED ACC  CONDITIONS
      IM1B
DFS000I      IVPDB1    DL/I                                UP  NOTOPEN,
ALLOCS      IM1B
DFS000I      IVPDB1I   DL/I                                UP  NOTOPEN,
ALLOCS      IM1B
DFS000I      *05160/210423*  IM1B
*395 DFS996I *IMS READY*  IM1B

R 395,/DIS PGM DFSIVP64
IEE600I REPLY TO 630 IS;/DIS PGM DFSIVP64
DFS000I      PROGRAM    TRAN    TYPE  IM1B
DFS000I      DFSIVP64          BMP   IM1B
DFS000I      *2005160/210708*  IM1B
*396 DFS996I *IMS READY*  IM1B
```

20.4.6 Step 6: Running the stored procedure by DSNTEJ62

The DSNTEJ62 job is supplied in the SDSNSAMP library of your DB2 system libraries. This job needs to be modified to allocate the correct DB2 libraries and point to the correct DB2 subsystems and programs. The following sections list the steps of each job and what task they perform.

Step 1 PH062S01: Precompiling, compiling, and link-editing the client program

This step precompiles, compiles, and link-edits the client program. See Example 20-10 on page 394.

Example 20-10 Link-editing condition

```
INCLUDE SYSLIB(DSNELI)
INCLUDE SYSLIB(DSNTIAR)
```

Step 2 PH062S02: Binding the client program package and plan

This step is used for bind the client program package and plan. SYSTEM(DB8P) is the client-side subsystem that will connect to DB8P through DRDA to run the stored procedure against IM1B. See Example 20-11.

Example 20-11 Bind condition

```
DSN SYSTEM(DB8P)
BIND PACKAGE(DSN8) MEMBER(DSN8EC2) -
    ACT(REP) ISO(CS) ENCODING(EBCDIC)
BIND PACKAGE(DB8P.DSN8) -
    MEMBER(DSN8EC2) ACT(REP) ISO(CS) ENCODING(EBCDIC)
BIND PLAN(DSN8EC2) -
    PKLIST(DSN8.DSN8EC2, -
        DB8P.DSN8.DSN8EC2) -
    ACT(REP) ISO(CS) ENCODING(EBCDIC)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA81) -
    LIB('DB8PU.RUNLIB.LOAD0')
//SYSIN DD *
GRANT BIND, EXECUTE ON PLAN DSN8EC2 TO PUBLIC;
//*
```

Step 3 PH062S03: Invoking the client

Invoke the client for the IMS ODBA stored procedure. Modify the PARMS parameter to specify the DB2 server location name and the IMS subsystem ID, in that order and separated by a single blank character. For example, we use the statements shown in Example 20-12.

Example 20-12 invoking the client program

```
DSN SYSTEM(DB8P)
RUN PROGRAM(DSN8EC2) -
PLAN(DSN8EC2) -
PARMS('DB8P IM1B')
END
```

Note: The PLKED step in PH062S01 and PH062S02 will get a return code of 04. This is acceptable. All other steps should return 00.

20.4.7 Step 7: Analyzing the output

In this final step, we analyze the output.

Messages related to the ODBA stored procedure activation

WLM starts the stored procedure when required. On startup of the stored procedure, you should see DSNX991I message shown in Example 20-13.

Example 20-13 DSNX991I message

```
DSNX991I DSNX9WLM IMS ODBA INITIALIZATION COMPLETED
```

This implies that the CIMS call was performed and that the ODBA environment has been set up. If the CIMS call is not successful, you will get the message shown in Example 20-14.

Example 20-14 DSNX992E message

DSNX992E IMS ODBA INITIALIZATION FAILURE, AIB RC =xxxx

The client application outputs

The first time you run the DSNTEJ62 job successfully, you should see output in two places. From the SYSOUT output of job DSNTEJ62 step PH062S03, you see the output shown in Example 20-15.

Example 20-15 Output from DSNTEJ62 SYSOUT

```
*****
* DSN8EC2: Sample Client for IMS/ODBA DB2 stored procedure sample (DSN8.DSN8EC1
*
*   Now connecting to DB8P
*   for access to IMS node IM1B
*
*   Entry for:
*     - Last Name ..... DOE
*     - First Name ..... JOHN
*     - Extension Number ... 9-876-5432
*     - Internal Zip Code .. 98765
*     added successfully to database DFSIVD1.
*
*   Entry for:
*     - Last Name ..... LAST1
*     - First Name ..... FIRST1
*     - Extension Number ... 8-111-1111
*     - Internal Zip Code .. D01/R01
*     retrieved successfully from DFSIVD1.
*
*****
```

From the SYSOUT output of the DB2 stored procedure DB8PODBA, you see the output shown in Example 20-16.

Example 20-16 Output from DB8PODBA SYSOUT

```
TE>DB2TEMP-IOCMD=ADD
AFTER DPSB PREP, DPCBNME=TELEPCB1
DPSB PREP AIBRETRN=000000000
DPSB PREP AIBREASN=000000000
DPSB PREP AIBRSNM1=DFSIVP64
DPSB PREP AIBRSNM2=IM1B
DPSB PREP AIBRESA1=0000207040
DPSB PREP AIBRESA2=0000000000
DPSB PREP AIBRESA3=0000000000

TE>DB2TEMP-IOCMD=DIS
TE>SSA-KEY=LAST1
AFTER DPSB PREP, DPCBNME=TELEPCB1
DPSB PREP AIBRETRN=000000000
DPSB PREP AIBREASN=000000000
DPSB PREP AIBRSNM1=DFSIVP64
DPSB PREP AIBRSNM2=IM1B
DPSB PREP AIBRESA1=0000207040
DPSB PREP AIBRESA2=0000000000
DPSB PREP AIBRESA3=0000000000
```

If you try to run the DSNTEJ62 job again, you should see that the DSNTEJ62 output looks the same as in Example 20-15 on page 395, but the output from the DB8PODBA is different. Look at Example 20-17. You will notice that the ADD received an AIBRETRN code of 2304, which translates to a X'900'. In the DL/I return and reason codes section of *IMS Version 9: Messages and Codes Volume 1*, GC18-7827, we see that a X'900' is a PCB status code analysis required error. There are not enough displays in this program to show what the PCB status code is, but if you added the necessary displays, you see an *//* status code because you are trying to insert a segment that already exists on the database.

Example 20-17 An example of output from the second run of DB8PODBA

```

TE>DB2TEMP-IOCMD=ADD
ISRT AIBRETRN=000002304
ISRT AIBREASN=000000000
ISRT AIBRESA1=0000207040
ISRT AIBRESA2=0000000000
ISRT AIBRESA3=0574081672
AFTER DPSB PREP, DPCBNME=TELEPCB1
DPSB PREP AIBRETRN=000000000
DPSB PREP AIBREASN=000000000
DPSB PREP AIBRSNM1=DFSIVP64
DPSB PREP AIBRSNM2=IM1B
DPSB PREP AIBRESA1=0000207040
DPSB PREP AIBRESA2=0000000000
DPSB PREP AIBRESA3=0000000000

TE>DB2TEMP-IOCMD=DIS
TE>SSA-KEY=LAST1
AFTER DPSB PREP, DPCBNME=TELEPCB1
DPSB PREP AIBRETRN=000000000
DPSB PREP AIBREASN=000000000
DPSB PREP AIBRSNM1=DFSIVP64
DPSB PREP AIBRSNM2=IM1B
DPSB PREP AIBRESA1=0000207040
DPSB PREP AIBRESA2=0000000000
DPSB PREP AIBRESA3=0000000000

```

20.5 Commands for ODBA DB2 stored procedure environment

In this section, we show some useful commands for monitoring and controlling your ODBA and DB2 stored procedures environment.

20.5.1 IMS commands

Let us take a look at the IMS commands first.

/DISPLAY ACTIVE REGION

The **/DISPLAY ACTIVE REGION** command shows ODBA thread connections with the current status. ODBA threads appear in the TYPE DBT section, the same as CCTL threads. In Example 20-18 on page 397, the stored procedure DB8PODBA has one ACTIVE thread with this IMS, and the thread is working with PSB DFSIVP64. The thread also has REGID 1 (PST number 1).

Example 20-18 /DISPLAY ACTIVE REGION command

```
/DIS ACT REG.
DFS4445I CMD FROM MCS/E-MCS CONSOLE USERID=JOUK02: DIS ACT REG IM1B
DFS4444I DISPLAY FROM ID=IM1B 929
      REGID  JOBNAME    TYPE  TRAN/STEP PROGRAM  STATUS          CLASS
      JMRGN   JMP       NONE
      MSGRGN  TP        NONE
      JBPRGN  JBP       NONE
      BATCHREG BMP      NONE
      FPRGN   FP        NONE
      1 DB8PODBA DBT     DB8PODBA DFSIVP64 ACTIVE
      IM1DBRC DBRC
      IM1DLS  DLS
```

/DISPLAY UOR

The /DISPLAY UOR command displays status information about IMS units of recovery (UORs) for protected resources on the RRS. In Example 20-19, IMS has the one active UOR task that has the unit of recovery identifier (URID) for RRS and the recovery token for IMS. These tokens are written in log records, as shown in Example 20-20. In some case of failure, IMS uses this information for the recovery operation.

Example 20-19 /DISPLAY UOR command

```
/DIS UOR
DFS4445I CMD FROM MCS/E-MCS CONSOLE USERID=JOUK02: DIS UOR IM1B
DFS4444I DISPLAY FROM ID=IM1B 926
      ST P-TOKEN  PSBNAME  RRS-URID                      IMS-RECTOKN
+      A          DFSIVP64 BD236FF47E6F282C0000002801010000 ODBA006C000
      0001100000000
      LUWID=1E908048AER TASK FOR ODBA
      *2005160/223231*
```

Example 20-20 X'5616' log records sample (which means START OF PROTECTED UOW)

```
01000000 56160000 C9D4F1C2 40404040 00000001 00000000 00000000 00000000
00000000 00000000 00000000 D6C4C2C1 F0F0F6C3 00000011 00000000 00000000 IMS-RECTOKN
00000000 00000000 00000000 BD236FF4 7E6F282C 00000028 01010000 00490000 RRS-URID
00010015 00000000 1E908048 C1C5D940 E3C1E2D2 40C6D6D9 40D6C4C2 C1000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 BD237436 738B4B46 00000000 00004F70
```

Tip: X'5616' log (START OF PROTECTED UOW) and X'5611' log (END OF PHASE ONE SYNCPOINT) have the URID field.

20.5.2 DB2 commands

You might find the following DB2 commands useful.

DISPLAY PROCEDURE

The DISPLAY PROCEDURE command displays statistics about stored procedures that are accessed by DB2 applications. This command displays one line of output for each stored procedure that a DB2 application has accessed. In Example 20-21 on page 398, you can check the current status of the DB8PODBA stored procedure.

Example 20-21 *DISPLAY PROCEDURE* command

```
-DB8P DIS PROCEDURE(DSN8.DSN8EC1)
DSNX940I  -DB8P DSNX9DIS DISPLAY PROCEDURE REPORT FOLLOWS - 041

----- SCHEMA=DSN8
PROCEDURE      STATUS ACTIVE QUED MAXQ TIMEOUT FAIL WLM_ENV
DSN8EC1
          STARTED    0    0    1    0    0 DB8PODBA
DSNX9DIS DISPLAY PROCEDURE REPORT COMPLETE
DSN9022I  -DB8P DSNX9COM '-DISPLAY PROC' NORMAL COMPLETION
```

START PROCEDURE

For both DB2-established and WLM-established stored procedure address spaces, the **START PROCEDURE** command activates the definition of a stored procedure that is stopped or refreshes one that is stored in the cache. You do not need to issue **START PROCEDURE** when you define a new stored procedure to DB2. DB2 automatically activates the new definition when it first receives an SQL CALL statement for the new procedure. See Example 20-22.

Example 20-22 *START PROCEDURE* command

```
-DB8P START PROCEDURE(DSN8.DSN8EC1)
DSNX946I  -DB8P DSNX9ST2 START PROCEDURE SUCCESSFUL FOR DSN8.DSN8EC1
DSN9022I  -DB8P DSNX9COM '-START PROC' NORMAL COMPLETION
```

STOP PROCEDURE

The **STOP PROCEDURE** command prevents DB2 from accepting SQL CALL statements for one or more stored procedures. This command does not prevent CALL statements from running if they have already been queued or scheduled by DB2. But in case of abends, DB2 implicitly issues the command **STOP PROCEDURE ACTION(REJECT)** for any stored procedure that exceeds the maximum abend count, and then you might have to issue the **START PROCEDURE** command for reuse. See Example 20-23.

Example 20-23 *STOP PROCEDURE* command

```
-DB8P STOP PROCEDURE(DSN8.DSN8EC1)
DSNX947I  -DB8P DSNX9SP2 STOP PROCEDURE SUCCESSFUL FOR DSN8.DSN8EC1
DSN9022I  -DB8P DSNX9COM '-STOP PROC' NORMAL COMPLETION
```

20.5.3 z/OS Workload Manager commands

You can also use the following z/OS commands for DB2 stored procedures.

DISPLAY WLM,APPLENV=xxxxx

After activating a service policy, you can confirm the application environment by using the z/OS **DISPLAY WLM** command shown in Example 20-24 on page 399. The application environment DB8PODBA has an AVAILABLE status now. An application environment initially enters the AVAILABLE state when the service policy that contains its definition is activated. AVAILABLE means the application environment is available for use, and servers are allowed to be started for it.

Example 20-24 z/OS DISPLAY WLM command

```
D WLM,APPLENV=DB8PODBA
IWM029I 14.54.44 WLM DISPLAY 364
APPLICATION ENVIRONMENT NAME      STATE      STATE DATA
DB8PODBA                          AVAILABLE
ATTRIBUTES: PROC=DB8PODBA SUBSYSTEM TYPE: DB2
```

VARY WLM,APPLENV=xxxxx,QUIESCE

The z/OS VARY WLM QUIESCE option causes workload manager to request the termination of server address spaces for the application environment upon completion of any active requests. Additional work requests are not handled by the servers, although work requests can continue to be queued, waiting for a server. If you do not want work queued, use the DB2 STOP PROCEDURE command with the ACTION(REJECT) option to stop the queuing. You can issue a quiesce action for an application environment that is in the AVAILABLE state. When a quiesce action is issued for an application environment, it first enters the QUIESCING state until all servers have been requested to terminate. It then enters the QUIESCED state. See Example 20-25.

Example 20-25 z/OS vary WLM command (QUIESCE option)

```
V WLM,APPLENV=DB8PODBA,QUIESCE
*IWM031I VARY QUIESCE FOR DB8PODBA IN PROGRESS
IWM032I VARY QUIESCE FOR DB8PODBA COMPLETED
-                                     --TIMINGS (MINS.)--
-      ----PAGING COUNTS----
-JOBNAME STEPNAME PROCSTEP RC EXCP CPU SRB CLOCK SERV
PG PAGE SWAP VIO SWAPS
-DB8PODBA IEFPROC DB8PODBA 00 642 .00 .00 53.2 6247
0 0 0 0 0
IEF404I DB8PODBA - ENDED - TIME=22.54.55 - ASID=006C - SC42
-DB8PODBA ENDED. NAME- TOTAL CPU TIME= .00
TOTAL ELAPSED TIME= 53.2
$HASP395 DB8PODBA ENDED
```

VARY WLM,APPLENV=xxxxx,RESUME

The z/OS VARY WLM RESUME option restarts an application environment that was previously quiesced and is in the QUIESCED state. It indicates to workload manager that server address spaces can once again be started for this application environment. The new servers process any queued requests and all new requests. When a resume action is issued for an application environment, it first enters the RESUMING state until all systems in the sysplex have accepted the action. It then enters the AVAILABLE state. See Example 20-26.

Example 20-26 z/OS vary WLM command (RESUME option)

```
V WLM,APPLENV=DB8PODBA,RESUME
IWM034I PROCEDURE DB8PODBA STARTED FOR SUBSYSTEM DB8P 057
APPLICATION ENVIRONMENT DB8PODBA
PARAMETERS DB2SSN=DB8P,NUMTCB=8,APPLENV=DB8PODBA
IWM032I VARY RESUME FOR DB8PODBA COMPLETED
$HASP100 DB8PODBA ON STCINRDR
$HASP373 DB8PODBA STARTED
IEF403I DB8PODBA - STARTED - TIME=22.55.20 - ASID=006C - SC42
DSNX991I DSNX9WLM IMS ODBA INITIALIZATION COMPLETED
```

VARY WLM,APPLENV=xxxxx,REFRESH

The z/OS VARY WLM REFRESH option requests the termination of existing server address spaces and starts new ones in their place. Existing servers finish their current work requests and end. The new servers process any queued requests and all new requests. You can issue a refresh action for an application environment that is in the AVAILABLE state. When a refresh action is issued for an application environment, it first enters the REFRESHING state until all servers have been requested to terminate. It then enters the AVAILABLE state. See Example 20-27.

Example 20-27 z/OS vary WLM command (REFRESH option)

```
V WLM,APPLENV=DB8PODBA,REFRESH
IWM034I PROCEDURE DB8PODBA STARTED FOR SUBSYSTEM DB8P 671
APPLICATION ENVIRONMENT DB8PODBA
PARAMETERS DB2SSN=DB8P,NUMTCB=8,APPLENV=DB8PODBA
*IWM031I VARY REFRESH FOR DB8PODBA IN PROGRESS
IWM032I VARY REFRESH FOR DB8PODBA COMPLETED
$HASP100 DB8PODBA ON STCINRDR
$HASP373 DB8PODBA STARTED
IEF403I DB8PODBA - STARTED - TIME=17.32.23 - ASID=0068 - SC42
DSNX991I DSNX9WLM IMS ODBA INITIALIZATION COMPLETED
-
--TIMINGS (MINS.)--
----PAGING COUNTS----
-JOBNAME STEPNAME PROCSTEP RC EXCP CPU SRB CLOCK SERV
PG PAGE SWAP VIO SWAPS
-DB8PODBA IEFPROC DB8PODBA 00 127 .00 .00 .6 1267
0 0 0 0 0
IEF404I DB8PODBA - ENDED - TIME=17.32.24 - ASID=0069 - SC42
-DB8PODBA ENDED. NAME- TOTAL CPU TIME= .00
TOTAL ELAPSED TIME= .6
$HASP395 DB8PODBA ENDED
```

20.5.4 RRS panel utility

Note: We do provide details about how to set up the RRS panel utility. We assume that RRS and the panel utility are set up for your systems. For more information about the RRS panel utility, see Appendix A, “Using RRS Panels,” in *z/OS V1R6.0 MVS Programming: Resource Recovery, SA22-7616*.

RRS provides the ISPF panels utility to enable installation people to work with RRS. When you use the panels, you can view the following information:

- ▶ RRS logs
- ▶ UR information
- ▶ Resource manager information

Through the panels, you can also take the following actions:

- ▶ Determine where a resource manager can restart after a system failure
- ▶ Resolve an in-doubt state for a UR to in-commit or in-backout
- ▶ Remove a resource manager's interest in a UR

Therefore, the panels provide a way for you to troubleshoot resource recovery. You might use them, for example, if an application is hung and you suspect that resource recovery might be the cause of the problem.

RRS-related resource manager information

The Resource Manager List shows the name of the resource manager and its state, as known on the local system. In Example 20-28, we can see that our DB2 server (DB8P) and IMS control region (IM1B) registered to RRS are in the Run status on the SC42 system.

Example 20-28 RRS Resource Manager List panel

RRS Resource Manager List					Row 1 to 6 of 6
Command ==>					Scroll ==> PAGE
Commands: v-View Details u-View URs r-Remove Interest					
S	RM Name	State	System	Logging Group	
	CSQ.RRSATF.IBM.MQCB	Run	SC42	WTSCPLX1	
	DSN.RRSATF.IBM.DB7B	Reset	SC42	WTSCPLX1	
	DSN.RRSATF.IBM.DB8P	Run	SC42	WTSCPLX1	
	DSN.RRSPAS.IBM.DB7B	Reset	SC42	WTSCPLX1	
	DSN.RRSPAS.IBM.DB8P	Run	SC42	WTSCPLX1	
	IMS.IM1B___V091.STL.SANJOSE.IBM	Run	SC42	WTSCPLX1	

RRS unit of recovery information

The Unit Of Recovery Details panel shows detailed information about the UOR work. In Example 20-29, we can see the ODBA (IM1B)-related UOR work detail, such as the UR identifier value, UR state, work manager name, and expressions of interest (what resource manager is related to this work).

Note that Example 20-19 on page 397 (/DIS UOR output) and Example 20-29 show same RRS UR identifier value, because both of them represent the same stored procedure request via ODBA interface.

Example 20-29 RRS Unit of Recovery Details panel

```

RRS Unit of Recovery Details
Row 1 to 2 of 2
Command ==> Scroll ==> PAGE

Commands r-Remove Interest v-View URI Details

UR identifier : BD236FF47E6F282C0000002801010000
Create time : 2005/06/10 02:13:16.966007      Comments :
UR state : InFlight      UR type : Prot
System : SC42      Logging Group : WTSCPLX1
SURID : N/A
Work Manager Name : DSN.RRSPAS.IBM.DB8P
    Display Work IDs      Display IDs formatted
    Luwid . : Not Present
    Eid . . : Not Present
    Xid . . : Not Present
Expressions of Interest:
S  RM Name      Type  Role
   DSN.RRSPAS.IBM.DB8P  Unpr  Participant
   IMS.IM1B___V091.STL.SANJOSE.IBM  Prot  Participant

```

20.6 Sample Java client application for ODBA stored procedure

In this section we show how to implement a Java client application to invoke a ODBA stored procedure. We assume that there is already a DB2 UBB for z/OS Version 8 DRDA

environment with ODBA sample stored procedure DSN8EC1, as discussed in 20.4, “Step-by-step instructions for using the sample” on page 388.

For the purpose of this sample, we use the DB2 configuration shown in Table 20-4.

Table 20-4 DB2 configuration for DRDA access

DB2 configuration	Value
Server location	wtsc42.itso.ibm.com
Port number	38000
Database name	DB8P
User ID	JOUK02
Password	DUMMY

We create a simple Java application, AccessODBADB2SP.java, to invoke the stored procedure. We use Rational Application Developer V6.0 for running the application. This application uses the Java Universal Driver (JCC Driver), which comes with DB2 Connect™ V8. You should apply the appropriate Java build path to your application to include the JCC Driver JAR files.

Example 20-30 shows the sample Java application to invoke the stored procedure.

Example 20-30 AccessODBADB2SP.java

```
package accessDB8P;

import java.sql.*;           /*[1]*/
import java.io.*;

public class AccessODBADB2SP {

    static
    {
        try
        {
            Class.forName("com.ibm.db2.jcc.DB2Driver").newInstance(); /*[2]*/
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {

        String url = "jdbc:db2://wtsc42.itso.ibm.com:38000/DB8P"; /*[3]*/
        String usrdb2 = "JOUK02";
        String passdb2 = "DUMMY";

        String dbctlid = "IM1B";
        String command = "DISPLAY";
        String last = "LAST1";
        String first = "";
        String extension = "";
        String zip = "";
        int aibReturn;
```

```

int aibReason;
String error = "";

try
{

Connection con = DriverManager.getConnection(url,usrdb2,passdb2);/*[4]*/

con.setAutoCommit(false);    /*[5]*/

CallableStatement cstmt =
    con.prepareCall("CALL DSN8.DSN8EC1(?,?,?,?,?,?,?,?)"); /*[6]*/

cstmt.setString(1,dbctlid);    /*[7]*/
cstmt.setString(2,command);
cstmt.setString(3,last);
cstmt.setString(4,first);
cstmt.setString(5,extension);
cstmt.setString(6,zip);

cstmt.registerOutParameter(2,Types.CHAR);    /*[8]*/
cstmt.registerOutParameter(3,Types.CHAR);
cstmt.registerOutParameter(4,Types.CHAR);
cstmt.registerOutParameter(5,Types.CHAR);
cstmt.registerOutParameter(6,Types.CHAR);
cstmt.registerOutParameter(7,Types.INTEGER);
cstmt.registerOutParameter(8,Types.INTEGER);
cstmt.registerOutParameter(9,Types.CHAR);

cstmt.execute();    /*[9]*/

command= cstmt.getString(2);    /*[10]*/
last  = cstmt.getString(3);
first = cstmt.getString(4);
extension= cstmt.getString(5);
zip    = cstmt.getString(6);
aibReturn= cstmt.getInt(7);
aibReason= cstmt.getInt(8);
error = cstmt.getString(9);

con.commit();    /*[11]*/

System.out.println(    /*[12]*/
    "Stored Procedure Results are.....\n"
    + "\nCommand      : "
    + command
    + "\nLast Name    : "
    + last
    + "\nFisrt Name    : "
    + first
    + "\nExtension      : "
    + extension
    + "\nZip Code Name : "
    + zip
    + "\nAIBRETRN(Dec) : "
    + aibReturn
    + "\nAIBREASN(Dec) : "
    + aibReason
    + "\nMessage       : "
    + error);

```

```

    }
    catch (Exception e)
    {
        System.out.println("\nCaught exception is: " + e);
    }
}
}

```

Note the following points to note about this code (listed in Example 20-30 on page 402):

1. Import the required standard Java classes for JDBC calls.
2. The application uses the Java Universal Driver to connect to the database on z/OS. The following statement loads the classes for the Java Universal Driver:

```
Class.forName("com.ibm.db2.jcc.DB2Driver")
```

3. Declare variables with the desired values.
4. After the classes for the Universal Driver are loaded, the format of the connection string decides the type of connection. You can choose a Type 2 connection or Type 4 connection. In our example, we use a Type 4 connection by using the format:

```
jdbc:db2://server_or_ip:port/location_name
```
5. Disable auto commit.
6. Create a callable statement object to invoke the stored procedure.
7. The `setString` methods set the defined values to the callable statement object.
8. Register output parameters.
9. Execute the callable statement.
10. The `getString` methods and `getInt` methods retrieve the values from the callable statement object as a consequence of the stored procedure execution.

Note: In steps 6 to 10, set the number of parameters, the field type, and the input and output attribute as the same as the stored procedure definition shown in Example 20-7 on page 392.

11. Commit the work.
12. Display the results.

If you run the Java application successfully, you see the output shown in Example 20-31.

Example 20-31 Sample Java client output

Stored Procedure Results are.....

```

Command       : DISPLAY
Last Name     : LAST1
Fisrt Name    : FIRST1
Extension     : 8-111-1111
Zip Code Name : D01/R01
AIBRETRN(Dec) : 0
AIBREASN(Dec) : 0
Message       :

```

IMS Remote Database Services

IBM IMS provides the capability to allow Java programs to access IMS resources. In this chapter, we focus on IMS Java Remote Database Services (RDS), which is introduced on IMS Version 9. With IMS RDS, you can develop and deploy applications that run on WebSphere Application Server on non-z/OS platforms and access IMS databases remotely through an IBM-provided EJB on WebSphere Application Server for z/OS. In this chapter, we provide the information about the infrastructure of IMS RDS, how to work with the DL/I model utility, considerations about making SQL calls, and the step-by-step procedure for creating the Web application to access the IMS database through IMS RDS.

21.1 The big picture of the IMS Java environment

Since IMS Version 7, IMS has provided the capability to allow Java programs to access IMS resources. In the environment, access to the IMS database from a Java application is provided by SQL statements through the Java Database Connectivity (JDBC) interface. There are two types of IMS dependent regions that are supported to execute IMS Java applications.

21.1.1 IMS dependent regions

Two IMS-dependent regions provide a Java Virtual Machine (JVM) environment for Java applications:

- ▶ **Java Message Processing (JMP) region**

JMP regions enable the scheduling of only Java message-processing applications. A JMP application is started when there is a message in the queue for the JMP application and IMS schedules the message to be processed (similar to MPP applications). It can access the following resources:

- IMS message queue for input and output messages
- IMS databases
- DB2 for z/OS databases

- ▶ **Java Batch Processing (JBP) region**

JBP regions run the Java applications that perform batch-type processing online and can access the IMS message queues for output (similar to non-message-driven BMP applications). JBP applications are started by submitting a job with JCL or from the Time Sharing Option (TSO). It can access the following resources:

- IMS message queue for output messages
- IMS databases
- DB2 for z/OS databases

These Java-dependent regions are attached to an IMS control region directly.

21.1.2 IBM products on the z/OS environment

Now, many IBM products on the z/OS environment can support running Java applications. Some of them, such as CICS Transaction Server for z/OS, DB2 UDB for z/OS, and WebSphere Application Server for z/OS, also can run the IMS Java applications for IMS database access. This capability provides us with the following advantages:

- ▶ **Increased portability of IMS database access logic**

After you developed your IMS database access logic with Java code, you can run it on any IBM products supported by IMS Java.

- ▶ **Use of the Web connectivity functions of each IBM product for IMS database access**

Each product has its native connectivity functions in the Web environment. This means that if you deploy your Java-based business logic (which contains IMS DB access) to the products, you can gain full connectivity from them, such as the CICS External Call Interface (ECI) and External Presentation Interface (EPI), CICS Web service support, DB2 DRDA function, WebSphere Application Server HTTP access, EJB access, and Web services.

DB2 UDB for z/OS and WebSphere Application Server for z/OS are attached to an IMS control region through ODBA interface. The CICS Java environment is attached through a DRA interface.

Figure 21-1 shows the big picture of the IMS Java environment.

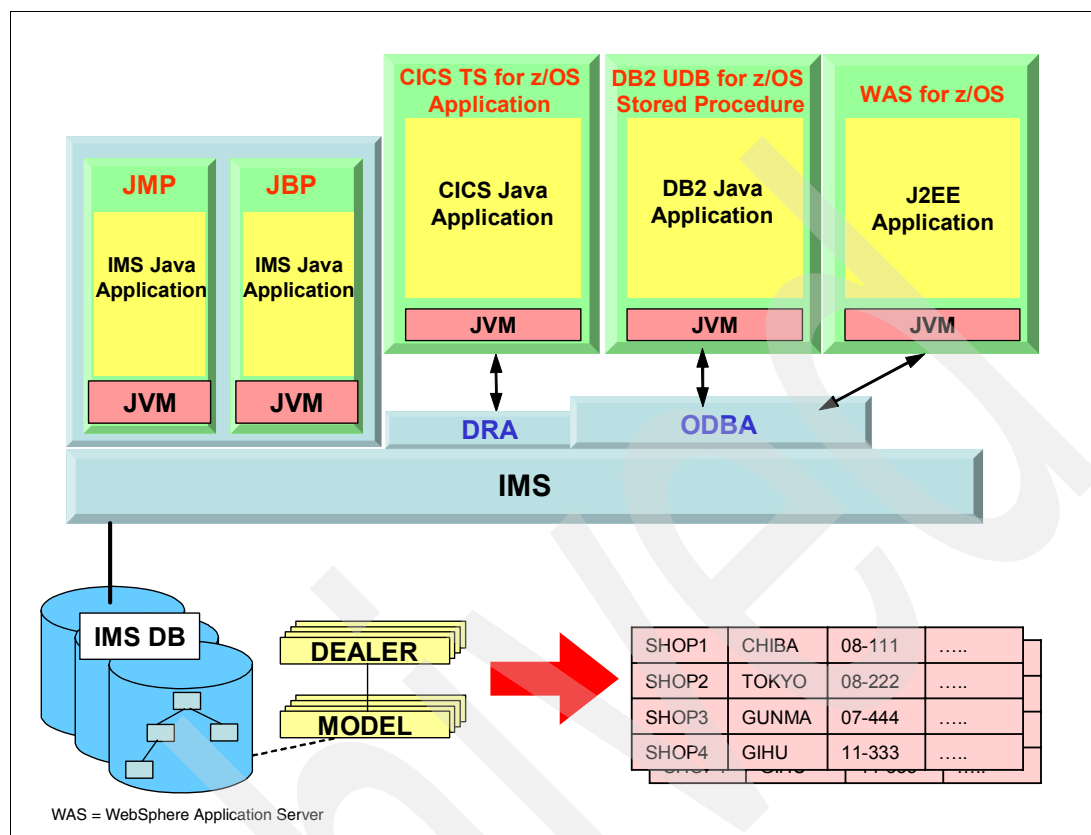


Figure 21-1 The big picture of IMS Java environment

21.2 IMS JDBC interface

IMS Java implements the JDBC 2.1 API, which is SQL-based standard interface for data access in the Java 2 Platform, Standard Edition (J2SE) and Enterprise Edition (J2EE). However, IMS has a *hierarchical* database structure made up of segments. Therefore, the IMS Java's implementation of JDBC/SQL supports a selected subset of the full facilities of the JDBC 2.1 API and standard SQL grammar. To access a hierarchical database as a *relational* database through the IMS JDBC interface, you must know some of the dialect of IMS Java SQL grammar and the restrictions of IMS JDBC API, which is derived from the logical conversion of the IMS hierarchical structure to its original relational model. In this section, we introduce the composition and characteristics of the IMS JDBC interface.

21.2.1 The layered set of IMS Java class libraries

IMS Java is delivered in a layered set of class libraries. These class libraries are shipped with the IMS product. This concept allows for the high-level classes to focus on ease-of-use, and the lower-level classes to provide access to IMS services and serve as the implementation for the higher-level classes. The diagram in Figure 21-2 on page 408 shows the layered set of class libraries.

segments have their own hierarchical relationship. Therefore, your relational tables that come from the occurrences involve these logical relationships and provide access from a parent segment table to a child segments table with the hierarchical path of segments.

With the IMS JDBC interface, the hierarchical relationship of IMS databases is represented by the *implicit join* with a parent segments table and a child segments table. Figure 21-4 shows an example of the relational representation of the hierarchy.

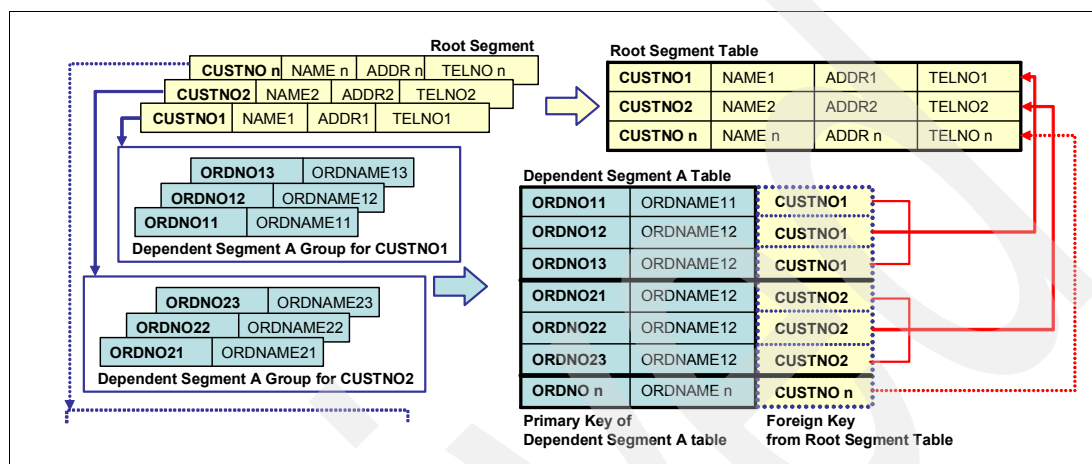


Figure 21-4 Relational representation of the hierarchy

This join concept of IMS JDBC is similar to the relational database model. A child segment table is joined with its parent segment table, which is all along the same hierarchical path. The relationship between tables is captured by *foreign* and *primary* keys. A child segment table has its own primary key field and the generated foreign key field that comes from parent's primary key field. In the example, the ORDER table (dependent segment A table) has the primary key field ORDNO and the foreign key field CUSTNO from the CUSTOMER table (root segment table). Therefore, if you specify both the primary and foreign key value in your WHERE clause of the SQL statement, you can access specific rows in the ORDER table as close as specifying the Segment Search Argument (SSA) of the root segment and the SSA of the dependent segment in traditional DL/I call statement. Additionally, the important difference between IMS Java and a relational database is that the join is operated *implicitly* by the IMS JDBC interface. In a relational database, when you join two tables, you specify the both table names in the FROM clause of SQL statement. But in IMS Java, you only specify the lowest-level target table in the hierarchical path in the FROM clause, because all the data in segments along the hierarchical path from the root segment to the target segment are included implicitly with use of the DL/I path call generated internally by the IMS JDBC interface.

21.2.3 Comparison of DL/I access and IMS JDBC SQL access

Figure 21-5 on page 410 shows a comparison sample of DL/I access and IMS JDBC SQL access. In this sample, the query requirement is the same for both access methods, which is: "Show us the all *order names* where the *customer number* is 2, and the *order number* is equal or greater than 21." In spite of the same query conditions, the following differences are shown in the sample:

- ▶ DL/I access
 - To retrieve all segments corresponding the query conditions, you need some application logic to deal with it:
 - The first time, you can issue a GU call to find the segment.

- Then, repeat the GN call to retrieve all segments with this conditions, until your GN call receives the status GE or GB.
- The query conditions and its hierarchy are represented in the SSA specifications.
- To retrieve the specific field in the segment, you also need some application logic for parsing the segment structure, because IMS will return whole segment image to the application I/O area.
- IMS JDBC SQL access
 - You do not have to code your application logic to retrieve all segments, because with this SQL query, you will receive all required data corresponding the query conditions.
 - The query conditions are represented in the FROM and WHERE clauses:
 - In the FROM clause, the table name is specified.
 - In the WHERE clause, the query conditions (the primary and foreign key values) are specified. You might not have to mind the hierarchy of the database, but it is better to know the hierarchy of the database for efficient access.
 - You do not have to code your application logic to retrieve the specific column in the table, because the IMS JDBC interface will return the column data that is specified in the SELECT statement only.

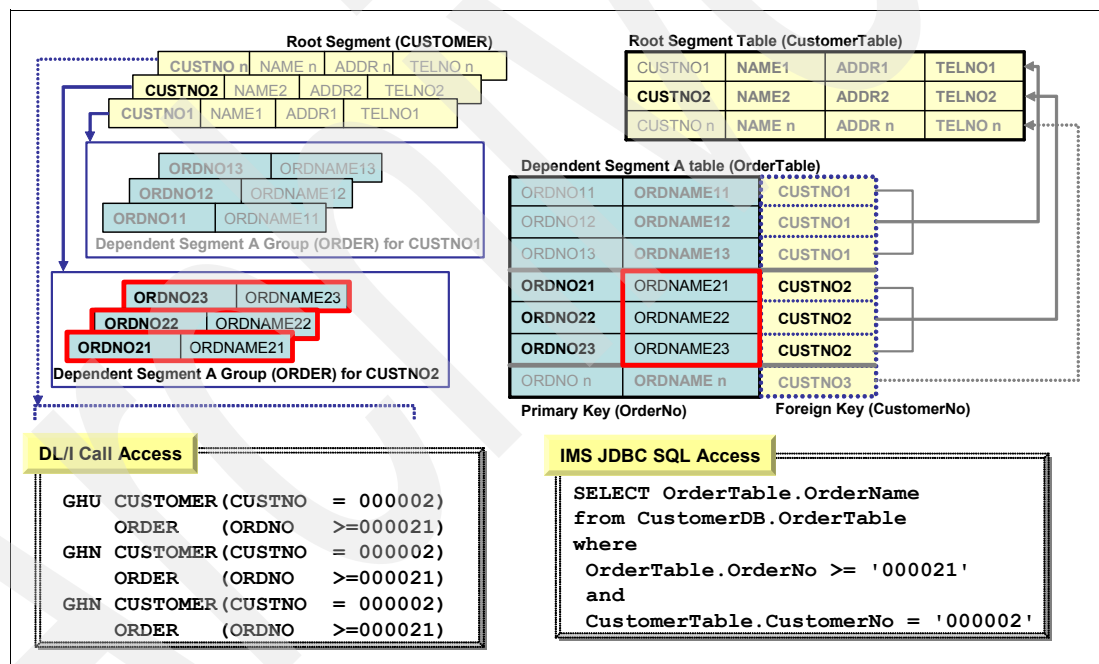


Figure 21-5 Comparison of DL/I access and IMS JDBC SQL access

Important: The figures in this section are only to help you understand how to use JDBC calls in a hierarchical environment. The IMS JDBC interface does not change the physical structure of IMS database in any way.

21.2.4 Supported SQL keywords

The IMS JDBC interface currently supports following portable SQL keywords. We summarize IMS-specific usage for important keywords in the following section. None of the keywords are

case-sensitive. These keywords are a subset of all SQL keywords. IMS JDBC interface supports the following functions:

- ▶ ALL
- ▶ AND
- ▶ AS
- ▶ ASC
- ▶ AVG
- ▶ COUNT
- ▶ DELETE
- ▶ DESC
- ▶ DISTINCT
- ▶ FROM
- ▶ GROUP BY
- ▶ INSERT
- ▶ INTO
- ▶ MAX
- ▶ MIN
- ▶ OR
- ▶ ORDER BY
- ▶ SELECT
- ▶ SET
- ▶ SUM
- ▶ UPDATE
- ▶ VALUES
- ▶ WHERE

Important: You cannot use any SQL keywords as Java aliases for PCBs, fields, or segments. For example, if you have the segment name UNION in the DBD statement, you have to define explicitly a different Java alias such as UnionTable by using DL/I model utility function. Note that the SQL UNION function is currently not supported by the IMS JDBC interface, but any use of SQL keywords for Java aliases are banned in the IMS Java environment. For a complete list of SQL keywords, see *IMS Version 9: IMS Java Guide and Reference*, SC18-7821.

21.2.5 IMS Java SQL usage

In this section, we focus on the difference between IMS Java SQL grammar and standard SQL grammar. We summarize the important SQL usage of the IMS JDBC interface and provide some SQL samples to help you understand how to use it. For more information about SQL usage, see *IMS Version 9: IMS Java Guide and Reference*, SC18-7821.

Sample tables for this section

Figure 21-6 on page 412 shows the sample tables for this section. We assume that SQL samples are issued against the CustomerTable and OrderTable shown in the figure. The CustomerTable is made up of the root segments and has the primary key field CustomerNo. The OrderTable is made up of the dependent segments of the root and has the primary key field OrderNo and foreign key field, which comes from the root segment. All segment (table) names and field (column) names are defined as different Java aliases from the original DBD definitions.

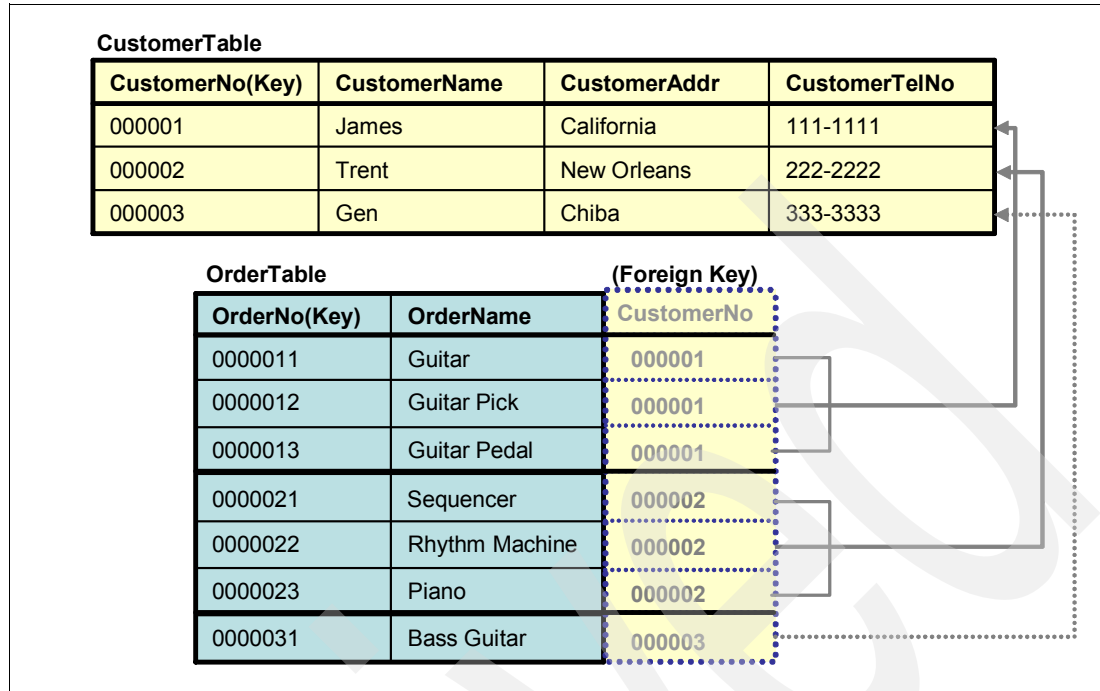


Figure 21-6 The example table for this section

Qualification rules

In an IMS Java SQL environment, the qualification rules of the column or table are very important for clarity and performance, and especially to avoid from handling the wrong table. In this section, we discuss the qualification rules of IMS Java SQL.

Segment qualification

For example, SQL dictates that whenever a field is common between two tables in an SQL query, the desired field must be table qualified to resolve the ambiguity. Similarly, whenever a field name is common in any two segments along a hierarchical path, the field might have to be segment qualified. For example, if a PCB has two tables (segments), CustomerTable and OrderTable, and both possess a column (field) named ID, any query referencing to the ID field must be segment qualified (remember, the IMS JDBC interface always joins with these tables implicitly).

Example 21-1 is incorrect because the ID field is not segment qualified.

Example 21-1 Incorrect example without segment qualification

```
SELECT ID
FROM CustomerDB.ORDER
WHERE ID='10'
```

Example 21-2 is correct because the ID field is segment qualified.

Example 21-2 Correct example with segment qualification

```
SELECT OrderTable.ID
FROM CustomerDB.OrderTable
WHERE CustomerTable.ID ='10'
```

PCB qualification

In IMS Java, connections are made to PSBs. Therefore, there must be a way to specify which PCB (using its alias) in a PSB to use when executing an SQL query on the `java.sql.Connection` object. To specify which PCB to use, you might have to qualify tables (segments) that are referenced by the SQL statement. For example, if a PSB has two database definitions (PCB), `CustomerDB` and `CustomerHistoryDB`, and both contain the table (segment) named `CustomerTable`, the `CustomerTable` in the `FROM` clause must be PCB qualified.

Example 21-3 is incorrect because `CustomerTable` is not PCB qualified.

Example 21-3 Incorrect example without PCB qualification

```
SELECT CustomerTable.CustomerName
FROM CustomerTable
WHERE CustomerTable.CustomerNo ='000001'
```

Example 21-4 is correct because `CustomerTable` is PCB qualified.

Example 21-4 Correct example with PCB qualification

```
SELECT CustomerTable.CustomerName
FROM CustomerHistoryDB.CustomerTable
WHERE CustomerTable.CustomerNo ='000001'
```

Qualification rules summary

Table 21-1 shows the summary of qualification rules. Each statement and clause is qualified with its table (segment) name or database (PCB) name.

Table 21-1 Qualification rules summary

SQL statements or clauses	Qualification rule
SELECT statement	Segment qualified
INSERT statement	PCB qualified
DELETE statement	PCB qualified
UPDATE statement	PCB qualified
FROM clause	PCB qualified
WHERE clause	Segment qualified

Important: For clarity and performance reasons, we strongly recommend that you always qualify columns or tables with its qualification rules.

SELECT statement

A `SELECT` statement is a query used as a top-level SQL statement. A `SELECT` statement can be executed against a `Statement` or `PreparedStatement` object, which returns the results as a `ResultSet` object.

Example 21-5 on page 414 shows an example of `SELECT` statement that queries the specific row/column of `CustomerTable` and `OrderTable`.

Example 21-5 SELECT statement

```
SELECT CustomerTable.CustomerName, CustomerTable.CustomerTelNo, OrderTable.OrderName
FROM   CustomerDB.OrderTable
WHERE  CustomerTable.CustomerNo = '000001' AND OrderTable.OrderNo = '0000011'
```

Example 21-6 shows an example of SELECT statement where all of the fields from both CustomerTable and OrderTable are retrieved.

Example 21-6 SELECT statement where all of the fields are retrieved

```
SELECT CustomerTable.*, OrderTable.*
FROM   CustomerDB.OrderTable
```

Note the following considerations for the SELECT statement:

- ▶ Do not join tables in the FROM clause. Only specify the table that comes from the lowest-level segment. All data in segments along the hierarchical path from the root segment to the target segment is included implicitly with the use of the DL/I path call generated internally by IMS JDBC interface. In the example, the OrderTable, which comes from the lowest-level segment in the FROM clause, is equivalent of the relational join with CustomerTable and OrderTable.
- ▶ When you code a SELECT list, generally try to supply predicates in the WHERE clause for all levels down the hierarchy to your target segment. If you supply a predicate in the WHERE clause for a target segment somewhere down the hierarchy and omit predicates for its parents, IMS must scan all candidate segments at the parent levels in an attempt to match the predicate that you supplied. In the example, if you are retrieving the OrderTable and you supply a predicate for that OrderNo, but do not supply CustomerNo for CustomerTable, IMS might perform a full database scan, testing every OrderTable (which consists of second-level segment) under every CustomerTable rows (which consist of every root) against the predicate. This has performance implications, particularly at the root level, and also might result in unexpected segments being retrieved.

Important: This implicit inclusion of segments is called a path call. For a path call to be made, the PROCOPT parameter in the PCB or SENSEG statement of the PSB source must include P. If P is not included in the PROCOPT parameter and you issue a query that requires a path call to be made, you receive the following SQLException generated from DL/I status code AM:

```
com.ibm.ims.db.DLISQLException: AM: IMCOMPATIBLE_CALL_FUNCTION
```

INSERT statement

An INSERT statement inserts a row (segment instance) with the specified data under any number of parent tables (segments) that match the criteria specified in the WHERE clause.

Example 21-7 shows an example of an INSERT statement that inserts a row to OrderTable.

Example 21-7 INSERT statement

```
INSERT INTO CustomerDB.OrderTable (OrderNo, OrderName)
VALUES ('0000032', 'Bass Pedal')
WHERE  CustomerTable.CustomerNo = '000003'
```

Note the following considerations for the INSERT statement:

- ▶ All column names must be specified in the statement, unless you set a default value in the IMS Java metadata class with the DLIModel utility control statements.

- One difference between JDBC queries to relational databases and to IMS is that standard SQL does not have a WHERE clause in an INSERT statement because tuples are being inserted into the table that is specified by the INTO keyword. In an IMS database, you are actually inserting a new instance of the specified segment, so you need to know where in the database this segment occurrence should be placed. With an INSERT statement, the WHERE clause is always necessary, unless you are inserting a root segment. Therefore, when you insert a new segment, generally try to supply predicates in the WHERE clause for all levels down the hierarchy to your target new segment. If you omit a predicate for any level down to the insert target segment, IMS chooses the first occurrence of a segment at that level that allows it to satisfy remaining predicates and performs the insert in that path. This might not be what you intended. In the example, if you insert a OrderTable row and do not supply a predicate for CustomerTable, your new row will always be inserted under the first root segment (which contains 000001 of CustomerNo).

DELETE statement

A DELETE statement can delete any number of rows (segment occurrences) that match the criteria specified in the WHERE clause.

Example 21-8 shows an example of a DELETE statement that deletes a row from CustomerTable.

Example 21-8 DELETE statement

```
DELETE FROM CustomerDB.CustomerTable
WHERE CustomerTable.CustomerNo = '000002'
```

Note the following considerations for the DELETE statement:

- If you delete a row that is not a bottom-level (leaf) segment in its hierarchy, you also delete the remaining segments in that hierarchical subtree. The entire family of segments of all types that are located hierarchically below your target deleted segment are also usually deleted. In the example, the rows from OrderNo = '0000021' to OrderNo = '0000023' in OrderTable are also deleted implicitly.
- If no WHERE clause is specified, all of the segment occurrences of that type are deleted, as are all of their child segment occurrences.

UPDATE statement

An UPDATE statement modifies the value of the fields in any number of segment occurrences.

Example 21-9 shows an example of an UPDATE statement that updates a row in OrderTable.

Example 21-9 UPDATE statement

```
UPDATE CustomerDB.OrderTable
SET OrderName = 'Synthesizer'
WHERE CustomerTable.CustomerNo = '000002' AND OrderTable.OrderNo = '0000023'
```

Note the following considerations for the UPDATE statement:

- If the UPDATE statement does not have a WHERE clause, the SET operation is applied to all rows of the specified table.
- A SET clause contains at least one assignment. In each assignment, the values to the right of the equal sign are computed and assigned to columns to the left of the equal sign.

- ▶ Before the updates are made, the IMS JDBC interface retrieves the rows that meet the query conditions internally. Therefore, the same considerations as with the SELECT statement need to be applied for the WHERE clause specification of the UPDATE statement.

FROM clause

When using the FROM clause in SQL calls to IMS databases, apply the following considerations:

- ▶ Do not join tables in the FROM clause; list only one table, which comes from the lowest-level segment.
- ▶ Qualify the table in the FROM clause by using the PCB alias.

WHERE clause

IMS Java converts the WHERE clause in an SQL query to an SSA list when querying a database. SSA rules restrict the type of conditions you can specify in the WHERE clause. This section describes how you must form your WHERE clause so that it can be converted into SSA lists. Note the following considerations:

- ▶ The WHERE clause can contain columns only from the table in the FROM clause or tables that are higher in the hierarchy. The columns in the WHERE clause must be DBD-defined fields. These fields that are in the DBD are marked in the DLIModel IMS Java report as being either primary key fields or search fields.
- ▶ You cannot use parentheses in the WHERE clause because SSAs do not support parentheses.
- ▶ Columns in the WHERE clause can be compared only to values, not to other columns. For example, the following statement fails because the WHERE clause is comparing two columns:

```
WHERE OrderTable.OrderNo = CustomerTable.CustomerNo
```

- ▶ You can use the following operators between column names and values in the individual qualification statements:
 - <
 - <=
 - =
 - =<
 - >
 - !=
- ▶ You can combine multiple qualification statements with AND and OR operators, but you must follow these special rules:
 - Because separate SSAs are created for each segment, list all qualification statements for a segment together and combine qualification statements for different segments with an AND operator. Qualification statements that are combined with an AND operator make up a qualification set. For a qualification set to be satisfied (true), all qualification statements in the set must be satisfied. For the WHERE clause (and therefore, the SSA qualification) to be satisfied, at least one qualification set must be satisfied.
 - The OR operator can be used only between qualification statements that contain columns from the same table. Because of the way SSA lists are created, you cannot use the OR operator across tables. For example, the following WHERE clause fails because the CustomerNo and OrderNo columns are in different tables:

```
WHERE CustomerTable.CustomerNo='000001' OR OrderTable.OrderNo='0000011'
```

However, the following WHERE clause is valid because the OR operator is between two qualification statements for the same tables:

```
WHERE OrderTable.OrderNo='0000011' OR OrderTable.OrderNo='0000012'
```

Non-DBD-defined fields in the WHERE clause

Using non-DBD-defined (search) fields in your WHERE clause, you can use fields that are defined by a COBOL copybook or the DLIModel utility, as long as the fields are a subset of a field defined in a DBD. This function is useful when you have broken a large field that is defined in the DBD into smaller subfields. IMS supports all type conversions for the individual subfields.

Note: The function is for *dividing* a large searching field in the DBD definition into the subset, not for *adding* search fields, which is not defined in the DBD definitions. Create the *non-DBD-defined-fields* for the WHERE clause from an existing large search field in the DBD definition, and all subfields must be provided consecutively in the query.

For example, if you have the segment definition in the DBD shown in Example 21-10, and you have the COBOL copybook definition for this segment shown in Example 21-11, you can divide the large search field 'ADDR' and create new search fields of the subset by coding the DL/I model utility control statement shown in Example 21-12.

Example 21-10 Segment definition with a large search field

```
SEGM  NAME=CUSTOMER,PARENT=0,BYTES=60
FIELD  NAME=(CUSTNO,SEQ,U),BYTES=6,START=00001
FIELD  NAME=CUSTNAME,BYTES=20,START=7
FIELD  NAME=ADDR,BYTES=40,START=27
FIELD  NAME=TELNO,BYTES=10,START=67
```

Example 21-11 COBOL copybook for the segment

```
01 CUSTOMER.
  02 CUSTOMER_NO      PIC X(06).
  02 CUSTOMER_NAME    PIC X(20).
  02 CUSTOMER_ADDR.
    03 CUSTOMER_ADDR_STREET  PIX X(10).
    03 CUSTOMER_ADDR_CITY   PIC X(10).
    03 CUSTOMER_ADDR_STATE  PIC X(10).
    03 CUSTOMER_ADDR_ZIPCODE PIC X(10).
  02 CUSTOMER_TELNO PIC X(10).
```

Example 21-12 DL/I model utility control statements for non-DBD-defined fields

```
SEGM DBDName=CUSTDB SegmentName=CUSTOMER Javaname=CustomerTable
FIELD NAME=CUSTNO      Start=1  Bytes=6  JavaType=CHAR  JavaName=CustomerNo
FIELD NAME=CUSTNAME    Start=7  Bytes=20 JavaType=CHAR  JavaName=CustomerName
FIELD NAME=ADDR       Start=27  Bytes=40 JavaType=CHAR  JavaName=CustomerAddr
FIELD                Start=27  Bytes=10 JavaType=CHAR  JavaName=CustomerStreet
FIELD                Start=37  Bytes=10 JavaType=CHAR  JavaName=CustomerCity
FIELD                Start=47  Bytes=10 JavaType=CHAR  JavaName=CustomerState
FIELD                Start=57  Bytes=10 JavaType=CHAR  JavaName=CustomerZipCode
FIELD NAME=TELNO       Start=67  Bytes=10 JavaType=CHAR  JavaName=CustomerTelNo
```

With the metadata class created by these definitions, you can treat the fields CustomerStreet, CustomerCity, CustomerState, and CustomerZipCode as the subfields.

The following rules apply when you use subfields in an SQL WHERE clause:

- ▶ The set of subfields that make up a DBD-defined field must account for all of the bytes in the DBD-defined field.
- ▶ All subfields in a set that make up a DBD-defined field must be listed together (similarly to how all fields in a segment must be listed together), but these subfields can be listed in any order.
- ▶ The only comparison operator allowed for subfields is =.
- ▶ The subfields in a set that make up a DBD-defined field must be separated by the AND operator. OR operators are not allowed to connect subfields in a set together. OR operators can be used to connect two sets of subfields.

Example 21-13 shows the usage of the subfields for a SELECT statement.

Example 21-13 Sample usage of the non-DBD-defined fields

```
SELECT CustomerTable.CustomerName
FROM   CustomerDB.CustomerTable
WHERE  CustomerStreet = 'SanIgnacio' AND CustomerCity   = 'San Jose' AND
       CustomerState  = 'California' AND CustomerZipCode = '00000095119'
```

This function is quite useful when your large search field consists of multiple type of data fields, because you can simply use the second query above and the IMS Java library will handle all of the type conversion on behalf of the applications.

Supported JDBC interfaces

In this section, we summarize the required interface by JDBC 2.1 implemented in IMS JDBC interface and describe its the functions and restrictions. For more information about the IMS JDBC interface, see *IMS Version 9: IMS Java Guide and Reference*, SC18-7821. See also the following Web page for the IMS JDBC API reference:

http://www.ibm.com/software/data/ims/imsjava/api9_1/index.html

java.sql.Connection

`java.sql.Connection` is an object that represents the connection to the database. A Connection reference is retrieved from the `DriverManager` object that is implemented in the `java.sql` package. The `DriverManager` object obtains a Connection reference by querying its list of registered Driver instances until it finds one that supports the Universal Resource Locator (URL) that is passed to the `DriverManager.getConnection` method.

There is an IMS Java restriction of the `java.sql.Connection` interface: IMS does not support the local, connection-based commit scope that is defined in the JDBC model. Therefore, the IMS Java implementation of the methods `Connection.commit`, `Connection.rollback`, and `Connection.setAutoCommit` result in an SQL exception when these methods are called.

java.sql.DatabaseMetaData

The `DatabaseMetaData` interface defines a set of methods for querying information about the database, including capabilities the database might or might not support. The class is provided for tool developers and is normally not used in client programs. Much of the functionality is specific to relational databases and is not implemented for DL/I databases.

java.sql.Driver

The Driver interface itself is not usually used in client applications, although an application must dynamically load a particular Driver implementation by name.

java.sql.Statement

A Statement interface is returned from the Connection.createStatement method. The Statement class and its subclass, PreparedStatement, define the interfaces that accept SQL statements and return tables as ResultSet objects.

IMS Java restrictions of the java.sql.Statement interface include:

- ▶ Named cursors. Therefore, the method Statement.setCursorName throws an SQL exception.
- ▶ Aborting a DL/I operation. Therefore, the method Statement.cancel throws an SQL exception.
- ▶ Setting a timeout for DL/I operations. Therefore, the methods Statement.setQueryTimeout and Statement.getQueryTimeout throw SQL exceptions.

java.sql.ResultSet

The ResultSet interface defines an iteration mechanism to retrieve the data in the rows of a table, and to convert the data from the type defined in the database to the type required in the application.

Important: Rather than building a complete set of results after a query is run, the IMS Java implementation of ResultSet interface retrieves a new segment occurrence each time the method ResultSet.next is called.

IMS Java restrictions of the java.sql.ResultSet interface include:

- ▶ Returning data as an ASCII stream. Therefore, the method ResultSet.getAsciiStream throws an SQL exception.
- ▶ Named cursors. Therefore, the method ResultSet.setCursorName throws an SQL exception.
- ▶ The method ResultSet.getUnicodeStream, which is deprecated in JDBC 2.1.

java.sql.ResultSetMetaData

The java.sql.ResultSetMetaData interface defines methods to provide information about the types and properties in a ResultSet object. It includes methods such as getColumnCount, isSigned, getPrecision, and getColumnName.

java.sql.PreparedStatement

The PreparedStatement interface extends the Statement interface, adding support for precompiling an SQL statement (the SQL statement is provided at construction instead of execution) and for substituting values in the SQL statement.

JDBC prepared statements for SQL

To improve the performance of your IMS Java application, use JDBC prepared statements for the SQL. The PreparedStatement class completes the initial steps in preparing queries only once so that you need to provide the parameters only before each repeated database call.

The PreparedStatement object performs the following actions only once before repeated database calls are made:

1. Parses the SQL.
2. Cross-references the SQL with the IMS Java DLIDatabaseView object.
3. Builds SQL into SSAs before a database call is made.

Example 21-14 shows an example of SQL INSERT for a prepared statement.

Example 21-14 SQL INSERT statement for prepared statement

```
INSERT INTO CustomerDB.OrderTable(OrderNo, OrderName)
VALUES (?,?)
WHERE CustomerTable.CustomerNo = ?
```

Supported JDBC data types

IMS Java supports the JDBC data types listed in Table 21-2. The table also lists the supported Java data types and how COBOL copybook types are mapped to it for JDBC type.

You can use several different Java methods to handle Java data types. For mapping descriptions, see *IMS Version 9: IMS Java Guide and Reference*, SC18-7821.

Table 21-2 Supported JDBC data types

JDBC data type	Java data type	COBOL format
CHAR	string	PIC X
CLOB	CLOB (supported only for IMS XML database)	N/A
VARCHAR	string	PIC X
BIT	Boolean	N/A
TINYINT	byte	N/A
SMALLINT	short	From PIC 9(1) BINARY To PIC 9(4) BINARY
INTEGER	int	From PIC 9(5) BINARY To PIC 9(9) BINARY
BIGINT	long	From PIC 9(10) BINARY To PIC 9(18) BINARY
FLOAT	float	COMP-1
DOUBLE	double	COMP-2
BINARY	byte[]	N/A
PACKEDDECIMAL	java.math.BigDecimal	PIC 9 COMP-3
ZONEDDECIMAL	java.math.BigDecimal	PIC 9 DISPLAY
DATE	java.sql.Date	N/A
TIME	java.sql.Time	N/A
TIMESTAMP	java.sql.Timestamp	N/A

21.3 DLIModel utility

In order for a Java application to access an IMS database, it needs information about the database. This information is contained in the PSBs and DBDs, but you must first convert this information into a form that you can use in the Java application: a subclass of the `com.ibm.ims.db.DLIDatabaseView` class called the *IMS Java metadata class*. The DLIModel utility generates this metadata from the IMS PSBs, DBDs, COBOL copybooks, and other

input specified by utility control statements. In addition to creating the metadata class, this utility also generates XML schemas of IMS databases. These schemas are used when retrieving XML data from or storing XML data in IMS databases. In this section, we focus on the metadata creation function of the DLIModel utility and provide a sample operation procedure. For complete information about the DLIModel utility, see *IMS Version 9: Utilities Reference: System*, SC18-7834.

Figure 21-7 shows the inputs and outputs from the DLIModel utility.

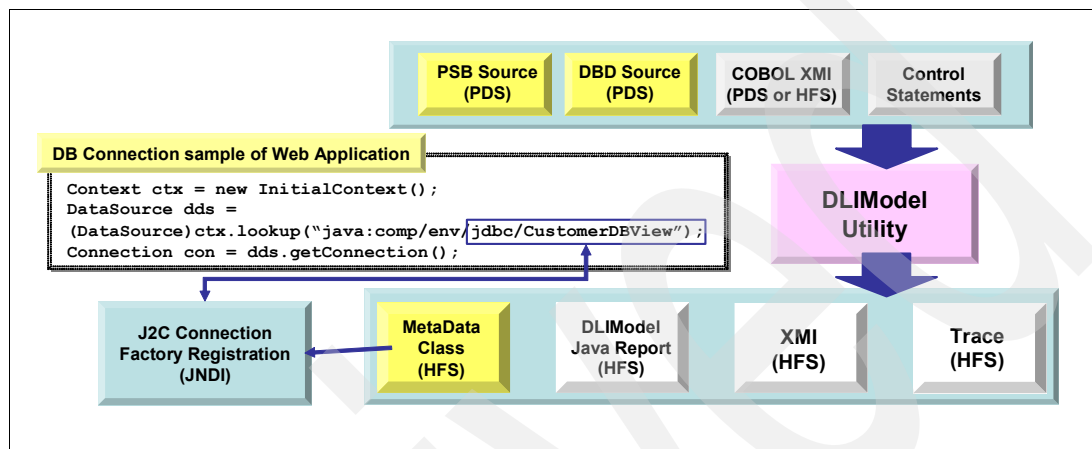


Figure 21-7 DLIModel utility inputs and outputs

The actions of the DLIModel utility are directed by control statements that you supply. The control statements can specify:

- ▶ Which PSB to process during a run
- ▶ Aliases for the PSB, PCBs, segments, and fields
- ▶ Data types and format masks for fields
- ▶ XMI files that contain XMI descriptions of COBOL copybook members for segments
- ▶ Additional field definitions for fields that are not defined in the DBD or the COBOL copybook XMI file
- ▶ Information that overrides PSB, DBD, and COBOL copybook XMI information
- ▶ Default values for newly inserted segments

The DLIModel utility reads the PSB and DBD source members from the partition data set (PDS) or partition data set extended (PDSE) and parses them to build an in-memory model of the database structure and the PSB view of that structure. The utility then generates the outputs that were requested through control statements. You can specify an XMI description of the entire in-memory model in which one description covers the PSB and all DBDs processed in the run. You can also request a detailed trace file of the DLIModel utility execution if such a trace is necessary for problem resolution.

After you create a metadata class from a PSB, you can install the metadata class as a Datasource in WebSphere Application Server environment. The relationship between metadata class and Datasource is mapped by Java Naming and Directory (JNDI) API. A Web application in WebSphere Application Server for z/OS or distributed platforms will use JNDI to make the specific ODBA connection to the IMS database manager.

21.3.1 Example of using the DLIModel utility

In this section, we show an example of DLIModel utility execution. We create the metadata class from the IMS IVP application (DFSIVP37) for later use of our RDS sample application. Example 21-15 is the PSB source of DFSIVP37, and Example 21-16 is the DBD source of IVPDB2, which is referenced by the DFSIVP37 application. Both of them are in the IMS.SDFSISRC library. Note that the member name of IVPDB2 is DFSIVD2.

Example 21-15 DFSIVP37 PSB source

```
PHONEAP PCB TYPE=DB,DBDNAME=IVPDB2,PROCOPT=A,KEYLEN=10
        SENSEG NAME=A1111111,PARENT=0,PROCOPT=AP
        PSBGEN LANG=JAVA,PSBNAME=DFSIVP37
        END
```

Example 21-16 IVPDB2 DBD source

```
DBD     NAME=IVPDB2,ACCESS=HDAM,RMNAME=(DFSHDC40,40,100)
DATASET DD1=DFSIVD2,DEVICE=3380,SIZE=2048
SEGM    NAME=A1111111,PARENT=0,BYTES=40,RULES=(LLL, LAST)
FIELD   NAME=(A1111111,SEQ,U),BYTES=010,START=00001,TYPE=C
DBDGEN
FINISH
END
```

Table definition of IVPDB2

Figure 21-8 shows the table definition of the IVPDB2 database. We change its segment name and field name into new Java aliases and add some fields that are not defined in the SEGM statement.

Table Name : Person (Segment Name in DBD: A1111111)			
LastName (Field Name in DBD : A1111111)	FirstName (No DBD Definition)	Extension (No DBD Definition)	ZipCode (No DBD Definition)
LAST1	FIRST1	8-111-1111	D01/R01
LAST2	FISRT2	8-222-2222	D02/R02
LAST n...	FIRST n...	8-nnn-nnnn...	D0n/R0n

Figure 21-8 Person table definition from IVPDB2 database

DLIModel utility execution

You can run the DLIModel utility in two ways:

- ▶ Standard z/OS batch job (BPXBATCH)
- ▶ Execution from command prompt of UNIX System Services environment

In this example, we execute the DLIModel utility in a UNIX System Services environment.

Table 21-3 on page 423 shows the directories and PDS locations of our execution.

Table 21-3 DLIModel utility execution environment

Resource	Location
PSB source	JOUKO2.SRCLIB.IMSRDS(DFSIVP37)
DBD source	JOUKO2.SRCLIB.IMSRDS(IVPDB2)
DLIModel utility control statement	/u/jouko2/imsrds/@dlictl
DLIModel utility outputs	/u/jouko2/imsrds/
DLIModel utility execution file	/SC53/imsv9/imsjava91/dlimodel/go

Example 21-17 shows our control statement for the DLIModel utility.

Example 21-17 Control statement for the DLIModel utility

```

OPTIONS PSBds=JOUKO2.SRCLIB.IMSRDS <1>
        DBDds=JOUKO2.SRCLIB.IMSRDS
        GenJavaSource <2>
        OutPath=/u/jouko2/imsrds <3>
        Package=redbookRds <4>
        GenTrace <5>

PSB PSBName=DFSIVP37 JavaName=DFSIVP37DatabaseView <6>

PCB PCBName=PHONEAP JavaName=PhoneBook <7>

SEGM DBDName=IVPDB2 SegmentName=A1111111 JavaName=Person <8>
FIELD Name=A1111111 Start=1 Bytes=10 JavaName=LastName JavaType=CHAR <9>
FIELD Start=11 Bytes=10 JavaName=FirstName JavaType=CHAR <10>
FIELD Start=21 Bytes=10 JavaName=Extension JavaType=CHAR
FIELD Start=31 Bytes=7 JavaName=ZipCode JavaType=CHAR

```

Note the following points in this statement.

For the OPTION statement:

1. Specify the data set name of PSB/DBD source data set.
2. Specify to generate the metadata class Java source file.
3. Specify the HFS directory where the DLIModel utility writes the output files.
4. Specify the package name for which the IMS Java classes are generated.
5. Specify to generate a trace file named dlimodeltrace of the utility run.
6. For the PSB statement: Specify the name of the PSB that is used by the DLIModel utility and its Java alias name.
7. For the PCB statement: Specify the name of the PCB and its Java alias name.
8. For the SEGM statement: Specify the name of the DBD (with segment name) and its Java alias name.
9. For the FIELD statement: Specify the name of the FIELD in the SEGM statement and its Java alias name/data type.
10. For the FIELD statement: Add the three fields that are not specified in the SEGM statement.

Note: You cannot use the FirstName, Extension, or Zipcode fields in a SQL WHERE clause, because they are not specified as the search field in the source DBD definition.

Example 21-18 shows the execution command and the response message of the DLIModel utility in the UNIX System Services environment. We execute the command file **go** with the control statement on the execution directory shown in Table 21-3 on page 423.

Example 21-18 DLIModel utility execution in UNIX System Services environment

```
JOUK02 @ SC53:/SC53/imsv9/imsjava91/dlimodel>go /u/jouko2/imsrds/@d1ict1
DLIModel completed successfully.
```

If the DLIModel utility completed successfully, you can see the generated files on your location, which is specified in the control statement.

Example 21-19 shows the metadata class Java source file for DFSIVP37. You can access specific PSB resource and IMS database as the relational table by using the metadata class.

Example 21-19 DFSIVP37DatabaseView.java

```
package imsrds;

import com.ibm.ims.db.*;
import com.ibm.ims.base.*;

public class DFSIVP37DatabaseView extends DLIDatabaseView {

    // This class describes the data view of PSB: DFSIVP37
    // PSB DFSIVP37 has database PCBs with 8-char PCBNAME or label:
    //     PHONEAP

    // The following describes Segment: A1111111 ("Person") in PCB: PHONEAP ("PhoneBook")
    static DLTypeInfo[] PHONEAPA111111Array= {
        new DLTypeInfo("LastName", DLTypeInfo.CHAR, 1, 10, "A1111111",
            DLTypeInfo.UNIQUE_KEY),
        new DLTypeInfo("FirstName", DLTypeInfo.CHAR, 11, 10),
        new DLTypeInfo("Extension", DLTypeInfo.CHAR, 21, 10),
        new DLTypeInfo("ZipCode", DLTypeInfo.CHAR, 31, 7)
    };
    static DLISegment PHONEAPA111111Segment= new DLISegment
        ("Person","A1111111",PHONEAPA111111Array,40);

    // An array of DLISegmentInfo objects follows to describe the view for PCB: PHONEAP
    ("PhoneBook")
    static DLISegmentInfo[] PHONEAPArray = {
        new DLISegmentInfo(PHONEAPA111111Segment,DLIDatabaseView.ROOT)
    };

    // Constructor
    public DFSIVP37DatabaseView() {
        super("2.0","DFSIVP37", "PhoneBook", "PHONEAP", PHONEAPArray);
    } // end DFSIVP37DatabaseView constructor

} // end DFSIVP37DatabaseView class definition
```

Example 21-20 on page 425 shows the generated DLIModel IMS Java report. The report describes the generated metadata class definition, so you can understand what table is created by your PSB/DBD source and the control statement.

Example 21-20 DLIModel IMS Java report

```
DLIModel IMS Java Report
=====
Class: DFSIVP37DatabaseView in package: imsrds generated for PSB: DFSIVP37

=====
PCB: PhoneBook
=====
Segment: Person
Field: LastName   Type=CHAR Length=10 ++ Primary Key Field ++
Field: FirstName  Type=CHAR Length=10
Field: Extension  Type=CHAR Length=10
Field: ZipCode     Type=CHAR Length=7
```

21.3.2 DLIModel utility plug-in

IBM provides a DLIModel utility plug-in technology preview, which enables you to generate and modify metadata in an Eclipse application development environment.

You can download the DLIModel utility plug-in from the following Web site:

<http://www.ibm.com/software/data/ims/toolkit/dlimodelutility/>

DLIModel utility plug-in features

The DLIModel utility plug-in technology preview contains two components: a wizard and an editor. The wizard guides you through the tasks involved in creating metadata. The editor enables you to:

- ▶ Generate XML schemas of IMS databases, which are used to retrieve XML data from or store XML data in IMS databases.
- ▶ Incorporate additional field information from XMI input files that describe COBOL copybooks.
- ▶ Incorporate additional PCB, segment, and field information, or override existing information.
- ▶ Generate a DLIModel IMS Java report, which is designed to assist Java application programmers.
- ▶ Generate a DLIModel trace log.

Software requirements

The DLIModel utility plug-in technology preview requires one of the following software packages:

- ▶ Eclipse Version 2.1.X only, which is available for download from the Eclipse Web site. This plug-in will not work with Eclipse 3.0.X.
- ▶ WebSphere Studio Application Developer Version 5.1.
- ▶ WebSphere Studio Application Developer Integration Edition Version 5.1.

You also need Java Runtime Environment version 1.3.1 or later.

Installation instructions

To install the plug-in:

1. Close Eclipse, WebSphere Studio Application Developer, or WebSphere Studio Application Developer Integration Edition, if open.
2. Download and unzip dlmodel.zip into your Eclipse, WebSphere Studio Application Developer, or WebSphere Studio Application Developer Integration Edition program folder. A subdirectory named com.ibm.ims.dliutility.gui_1.5.0 is created in both the plug-ins and features directories. Default program folder locations include:
 - For WebSphere Studio Application Developer:
C:\Program Files\IBM\WebSphere Studio\Application Developer\v5.1\
 - For WebSphere Studio Application Developer Integration Edition:
C:\Program Files\IBM\WebSphere Studio\Application DeveloperIE\v5.1\
3. Launch Eclipse, WebSphere Studio Application Developer, or WebSphere Studio Application Developer Integration Edition.
4. Launch the DLIModel utility wizard by clicking **File** → **New** → **Other**.

For additional information about how to run the DLIModel utility plug-in technology preview, including information about requirements and restrictions, see Eclipse or WebSphere Studio after you have installed the plug-in. To access this information in Eclipse or WebSphere Studio, click **Help** → **Help Contents** on the top toolbar. When the Help - Eclipse Platform window opens, select **DLIModel Utility Plug-in Guide** from Contents pane.

21.3.3 Example of using the DLIModel utility plug-in

In this section, we show an example of a DLIModel utility plug-in execution. We create the metadata class from the same IVP application as in 21.3.1, “Example of using the DLIModel utility” on page 422. We assume that you have already installed the plug-in by following the installation instructions we provided earlier. We use WebSphere Studio Application Developer- Integration Edition Version 5.1 for this execution.

Note: Before executing the sample, you must download the PSB and DBD source files. Additionally, your PSB and DBD sources need the file extensions .psb and .dbd for the plug-in execution.

Perform the following steps:

1. In the Java perspective, click **File** → **New** → **Other**. Then, select the **DLIModel Utility** plug-in, as shown in Figure 21-9.

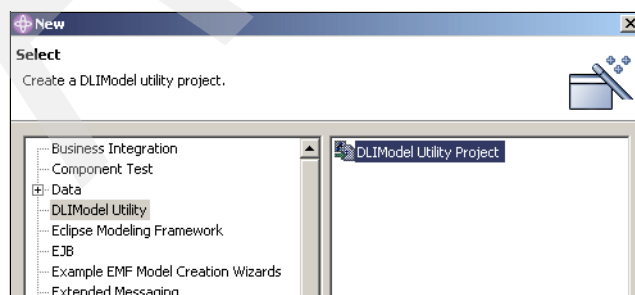


Figure 21-9 DLIModel Utility plug-in

2. To create a new DLIModel utility project, enter the Project name and Java package name, as shown in Figure 21-10. Click **Next**.

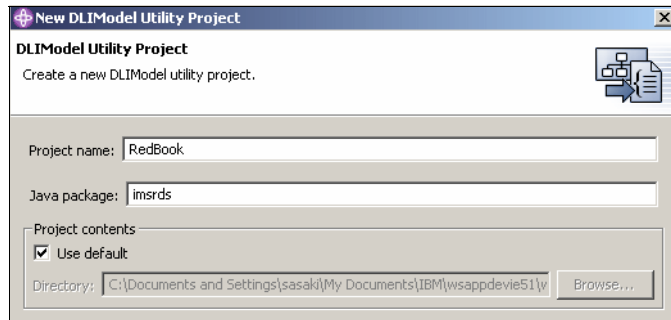


Figure 21-10 DLIModel Utility Project

3. To generate DLI metadata, add the source files from existing projects or import the source files from local file systems, as shown in Figure 21-11. Click **Import** and specify the PSB source file and DBD source file on your computer. Then, click **Finish**.

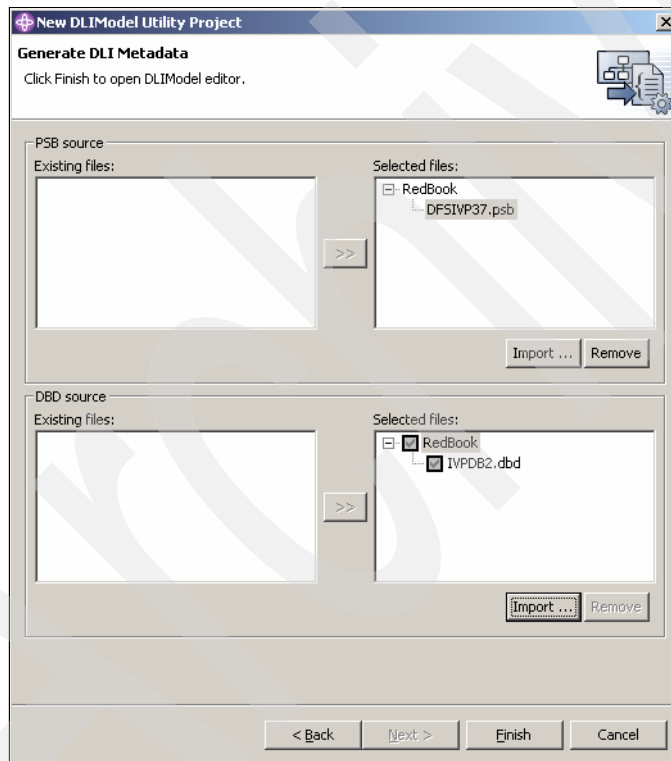


Figure 21-11 Generate DLI Metadata

4. After the metadata is generated to your project, the DLIModel editor, as shown in Figure 21-12, opens. You can use the editor to modify a field or segment.

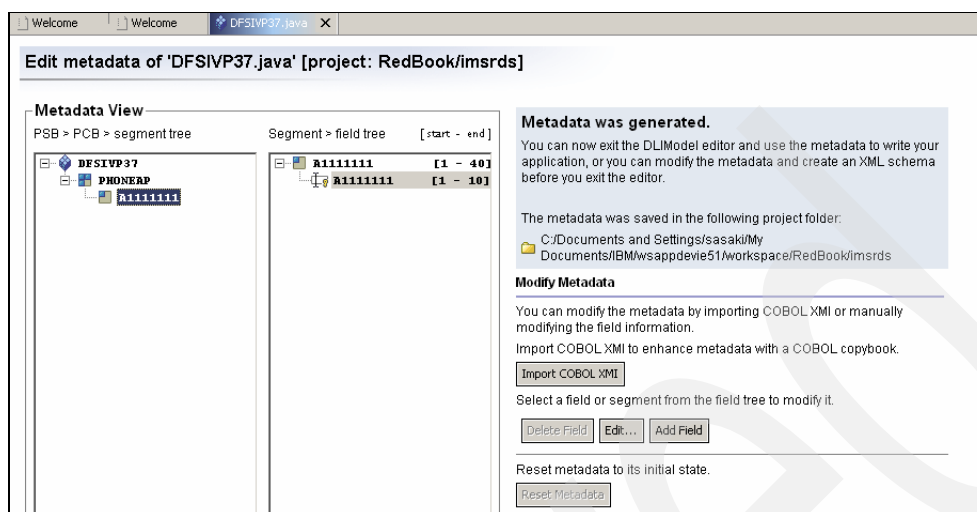


Figure 21-12 DLIModel editor

5. In this example, we change the segment name A1111111 to the Java alias Person. Right-click the table, and select **Edit**.
6. In the Edit Metadata Segment window, as shown in Figure 21-13, enter a new Alias name for the segment, and then click **Finish**. You also can edit, add, and delete the segments and fields using the editor.

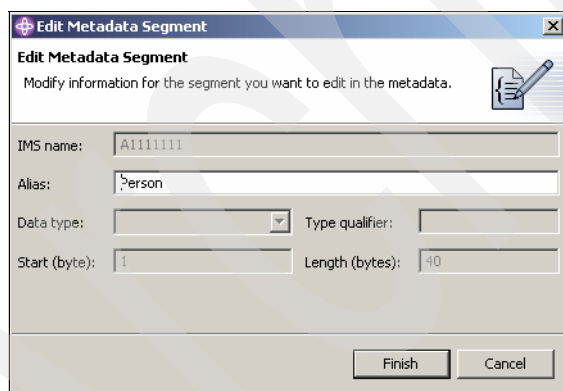


Figure 21-13 Edit Metadata Segment

21.4 Remote Database Services

IMS Version 9 provides IMS Java Remote Database Services, in which you can develop and deploy applications that run on WebSphere Application Server on a non-z/OS platform and access IMS databases remotely through an IBM-provided EJB on WebSphere Application Server for z/OS. Through several IMS versions, you have been able to write applications that run on WebSphere Application Server for z/OS and access IMS databases. See Figure 21-14 on page 429.

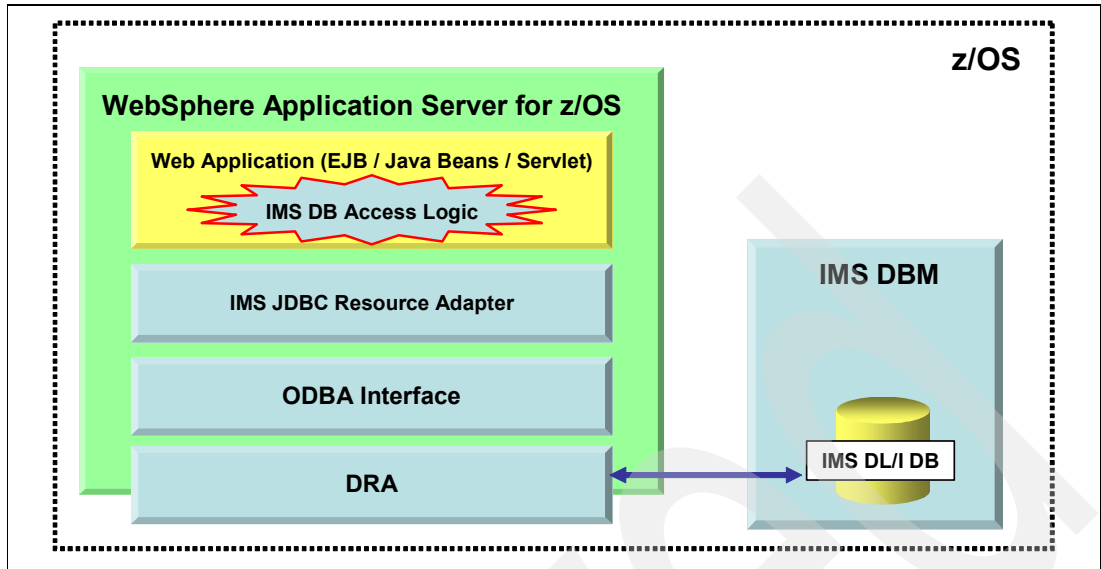


Figure 21-14 Web application on WebSphere Application Server for z/OS using IMS Java

This shows a Web application on WebSphere Application Server for z/OS accessing IMS data. IMS JDBC calls are passed to the IMS JDBC interface layer, which converts the calls to DL/I calls. The IMS JDBC interface passes these calls to ODBA, which uses the DRA to access the DL/I region in IMS. Note that you still need to develop a z/OS application to access IMS data, that is, DB access logic is still on z/OS.

Figure 21-15 shows the Remote Database Services provided from IMS Version 9. You can deploy your IMS DB access logic on a non-z/OS platform.

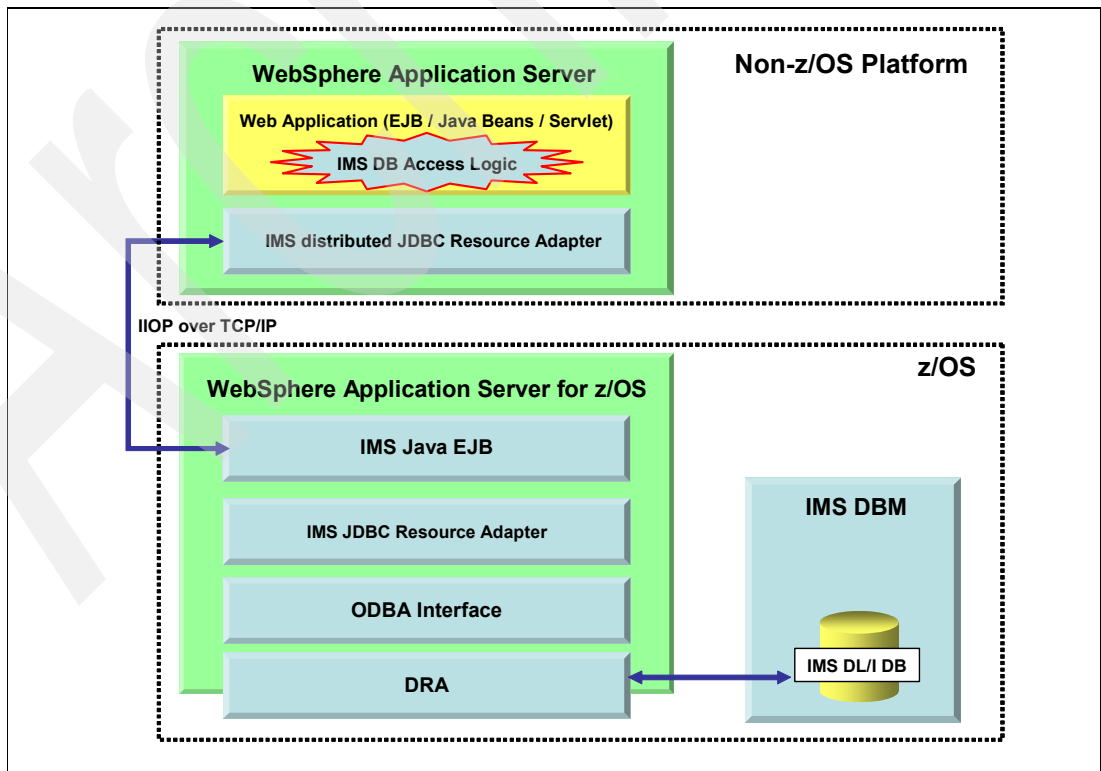


Figure 21-15 Web application on distributed platform using IMS Java

The Web application that contains your business logic, and is deployed on WebSphere Application Server on a non-z/OS platform, passes database access requests to the IMS distributed JDBC resource adapter provided by IMS Version 9. This resource adapter contains a type 3 JDBC driver. Then, Internet Inter-ORB Protocol (IIOP) is used between WebSphere Application Server for z/OS and WebSphere Application Server running on another platform. IIOP allows the servers to exchange data, which is securely transferred across the Internet using Secure Sockets Layer (SSL). The IMS Java-supplied EJB on the z/OS side receives the data through IIOP and then passes request information to the IMS JDBC resource adapter that is deployed on the z/OS platform. The IMS JDBC resource adapter passes data to ODBA, which uses the DRA.

21.4.1 Remote Database Services components

IMS Java Remote Database Services is a set of J2EE components that provide remote access to IMS data through IMS ODBA. The client-side and server-side components support retrieval, update, delete, and insert activity to the IMS databases. These requests are sent (transparently to the application) across the network and processed in IMS. This support provides an architected solution that enables Web applications deployed on distributed WebSphere Application Server to issue JDBC calls to access IMS Databases.

Client-side component

For the client side of the connection, the IMS Java RDS provides an IMS distributed JDBC resource adapter. To condition the distributed WebSphere Application Server for JDBC access to IMS, the IMS JDBC resource adapter must first be installed. This component contains a type 3 JDBC driver that interprets the JDBC request to access and manipulates the IMS data. A type 3 JDBC driver is defined as a standard net-protocol, fully Java technology-enabled driver that translates JDBC API calls into a DBMS-independent net protocol that is then translated to a DBMS protocol by a server.

After it is installed, a J2C connection factory instance can be deployed. This is a DataSource object deployed in Java Naming and Directory Interface (JNDI) that can be used to obtain a JDBC connection. The DataSource object defines properties that pertain to the actual target data source for the connection. See Figure 21-16.

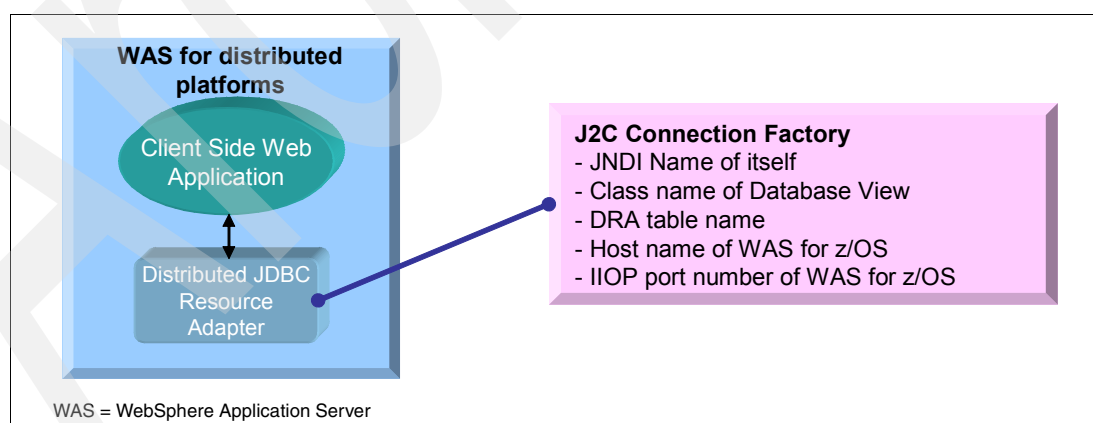


Figure 21-16 RDS client-side components

Table 21-4 on page 431 shows the important datasource properties and descriptions. For more information about the J2C connection factory, check your WebSphere Application Server Information Center.

Table 21-4 J2C connection factory property of IMS distributed JDBC resource adapter

Property name	Description
JNDI name	The JNDI name of this resource that connects to WebSphere Application Server for z/OS.
DRAName	The 4 byte DRA name (DFSxxxx0) of the IMS to connect.
DatabaseViewName	Full qualified name of the metadata class. The metadata class needs to be on z/OS side, and IMS JDBC resource adapter on z/OS needs to know the classpath for it.
HostName	The host name or IP address of WebSphere Application Server for z/OS.
PortNumber	The IIOF port number of WebSphere Application Server for z/OS.

After the connection is obtained, a customer-written application in that distributed server can use the JDBC API to access IMS DB data. To the application, JDBC access appears local, that is, the application does not have to be written with any knowledge that network traffic is involved.

Server-side component

For the server side (z/OS side) of the connection, a server EJB is delivered as part of the IMS Java RDS support. This EJB must reside on a WebSphere Application Server for z/OS environment because it uses the IMS JDBC resource adapter. Here we describe the reasons for the WebSphere Application Server for z/OS requirement:

- ▶ The IMS ODBA interface requires thread connections to its requester in the same z/OS LPAR. WebSphere Application Server for z/OS provides this capability.
- ▶ WebSphere Application Server for z/OS provides a mechanism to invoke a custom service when it is brought up and down. The IMS JDBC resource adapter can use this function to build an ODBA environment.
- ▶ WebSphere Application Server for z/OS supports an EJB container environment, so it can handle the transaction semantics of the client request.

When the server is brought up, the IMS adapter initializes the ODBA environment and correspondingly terminates it when the server is brought down. Therefore, after the server is brought up, every application running in the server can use an already initialized ODBA environment. In an ODBA environment, and for WebSphere Application Server for z/OS, RRS is used, so the RRS component requires the use of the RRS=Y in your IMS startup parameters. See Figure 21-17.

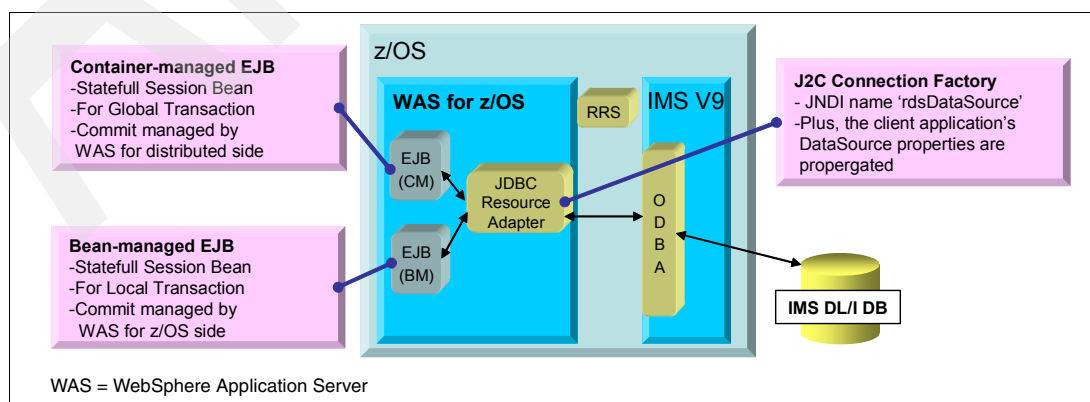


Figure 21-17 RDS server-side components

The IMS JDBC resource adapter should have the specific J2C connection factory configuration for IMS RDS environment. The only requirement is it should have the JNDI name of `rdsDataSource`, because the server EJB will look up and build a connection to the ODBA server using this JNDI name. At run time, the client application's data source properties are propagated to an instance of this data source.

The server EJB that is provided is a stateful session bean that enables it to act as a server-side extension of the client's IMS distributed JDBC resource adapter. The primary responsibility of the server EJB is to maintain the state of a client application with respect to IMS and forward all database requests to the IMS JDBC resource adapter. The server EJB is invoked as one of two different types: a container-managed EJB (CM EJB) and a bean-managed EJB (BM EJB).

When the server EJB runs as a container-managed EJB, it supports a client EJB application that has requested global transaction semantics. This means that the client application operates under one transaction context, or unit of work, for all its database activity. In this case, the distributed J2EE application server has the responsibility for being the coordinator of the global transaction.

However, when the server EJB runs as a bean-managed EJB, it supports a client EJB application that uses local transaction semantics. Each database connection operates under its own transaction context. When a commit is issued against work done on one connection (database), this does not affect work on another connection. In this scenario, WebSphere Application Server for z/OS becomes the coordinator of the transaction. This is because ODBA requires a transaction context to be present. What really happens is that the BM EJB starts a global transaction context on the server side to communicate with IMS while maintaining the local transaction aspect with respect to the client application's connection.

21.4.2 Client/server interaction

Figure 21-18 shows a diagram of the client/server interaction components.

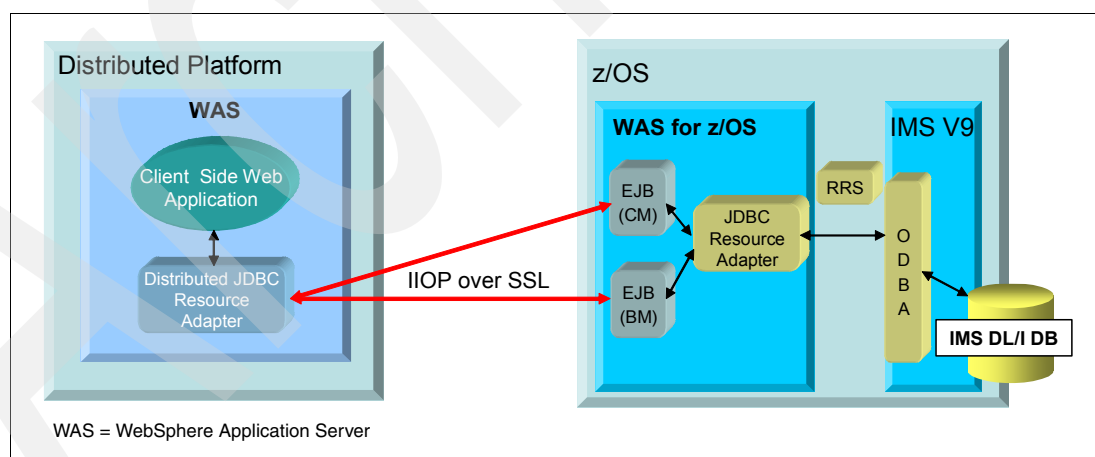


Figure 21-18 Client/server interaction

This figure shows the flow of a request from a client-side Web application issuing a JDBC call all the way to where it is serviced in IMS. The IMS distributed JDBC resource adapter hides the remote access interaction from the client-side Web application, takes the request, and establishes communication with the server EJB on the z/OS platform. All activity is kept within the distributed resource adapter until a JDBC connection statement is executed. At this point, an IIOP connection is established with the server-side EJB. IIOP is a transport mechanism standard that supports seamless interoperability between J2EE distributed objects.

The IMS distributed JDBC resource adapter chooses an EJB type (CM or BM) based on whether or not a transaction context exists. A transaction context exists and a CM EJB will be created if the client-side Web application previously issued a `getTransaction()` method requesting the use of global transaction semantics, or the EJB (on the client side) deployment descriptor represents the needs of a transaction context. Otherwise, local transaction semantics are used, and a BM EJB will be created. Information from the client-side DataSource (DRA name, metadata class name) is also propagated to the EJB. After the instance of the server-side EJB (CM or BM) is created, a connection to ODBA is requested. This results in the loading of the metadata classes and the allocation of the PSB. After the connection is established, an indication of success or failure is sent back to the client.

Note: Store the database metadata classes produced by the DLIModel utility on z/OS for use by the IMS JDBC resource adapter. Also, the IMS JDBC resource adapter on z/OS needs the appropriate classpath to them. They do not need to be moved to the distributed platforms.

The IMS distributed JDBC resource adapter, on a successful connection indication, creates the Statement object. After this object is handed to the client EJB, a JDBC query can be executed.

After the Web application receives the results and issues the commit request, the commit work occurs under the cooperation of WebSphere Application Server for z/OS (or WebSphere Application Server for distributed platforms, or both) and RRS and IMS ODBA, depending on the transaction semantics.

21.4.3 Security

There are several resources that need to be properly secured in an IMS Java Remote Database Services environment.

Access to client-side business logic

In a J2EE environment, the security context (or the identity) on the execution thread is determined by a deployment property of the application component known as “run-as.” The run-as value can be one of the following properties:

- ▶ System Identity, which specifies that the application component runs as the identity of the server region ID.
- ▶ Caller Identity, which specifies that the application component runs as the identity of its caller.
- ▶ Role, which specifies that the application component runs as a particular identity that is defined by the administrator in the J2EE server.

Note that the run-as values of Caller Identity and Role are not supported in this implementation. For information about run-as options and other security issues, see the WebSphere Application Server Information Center.

Network security

Secure Sockets Layer (SSL) is used to protect the communication between WebSphere Application Server on distributed platforms and WebSphere Application Server on z/OS. The client application that uses the IMS distributed JDBC resource adapter must be deployed with a run-as identity of system. This is currently a requirement for interoperation between a distributed application server and WebSphere Application Server for z/OS. As such, the client

application should be protected with restricted access so that only authorized users can access it on the distributed platform.

There is no support for passing the caller identity of the client application. At this point, the use of SSL and IIOP requires the identity to be switched to “system identity” when communicating between distributed and z/OS application servers.

Security on WebSphere Application Server for z/OS

Security regarding WebSphere Application Server for z/OS can be separated into two topics, server-side EJB and between IMS.

Server-side EJB

The distributed application server propagates the run-as property of the client-side EJB, and WebSphere Application Server for z/OS places an appropriate security identity on the thread that will be used to access IMS. The server-side EJB defaults to the run-as identity value of the system, which is the server region ID of WebSphere Application Server for z/OS. You can change the run-as property in the deployment descriptor of the server-side EJB before installing it.

Between IMS

The ODBA environment requires a previously verified Access Control Environment Element (ACEE), which WebSphere Application Server for z/OS places on the execution thread. ACEE is a control block that is built when a call to RACF or an equivalent security product is issued. In the WebSphere Application Server for z/OS environment, the IMS JDBC resource adapter uses sync-to-thread processing to ensure that a security context is placed on the thread during execution to access an IMS database. This places an ACEE on the execution thread, based on the run-as property of the server-side EJB.

21.5 Sample IMS RDS access

The following sections describe how to implement IMS Remote Database Services to access an IMS system through the ODBA. We demonstrate the environment setup for IMS, WebSphere Application Server for z/OS, and WebSphere Application Server for distributed platforms. Then, we provide a sample Web application to query the IMS IVP database with local transaction semantics.

We assume that there is already a WebSphere Application Server for z/OS Version 6 and IMS Version 9 system up and running. We use Rational Application Developer Version 6 for the Microsoft Windows platform to develop and test our Web client application.

We divide this task into the following areas:

1. Setting up ODBA for the IMS subsystem: This includes customizing the DRA startup table. It enables WebSphere Application Server for z/OS to use ODBA for this IMS subsystem.
2. Setting up the WebSphere Application Server for z/OS subsystem: This includes concatenating the IMS library to the WebSphere Application Server servant region, installing the IMS JDBC resource adapter for IMS Java EJB, installing the custom service, and installing the EAR file including the IMS Java EJB.
3. Creating and installing the metadata class for the sample application: This includes executing DLIModel utility and adding a classpath to the IMS JDBC resource adapter.
4. Setting up the WebSphere Application Server for distributed platforms subsystem: This includes installing the IMS distributed JDBC resource adapter and configuring the J2C connection properties for our sample application.

5. Developing and deploying the sample application: This includes developing an IMS JDBC application to query the IMS IVP database and deploying the sample application to WebSphere Application Server for distributed platforms.
6. Defining the IMS environment: This includes setting the RRS= parameter and confirming the IVP applications that will be used by this sample.
7. Executing the sample application and then analyzing the output.

For the purpose of this sample, we use the parameters shown in Table 21-5.

Table 21-5 Parameters

Parameter	Value
IMSID	IMSG
WebSphere Application Server for z/OS server location	wtsc53.itso.ibm.com
WebSphere Application Server for z/OS bootstrap address	12809

21.5.1 Step 1: Creating the IMS DRA startup table

We customized the IMS IVP member IV_E308J for this purpose. For more information about the creation of the DRA startup table, see 20.4.1, “Step 1: Creating an IMS DRA startup table” on page 389. Note that we change the DBCTLID and the DRA startup module name from previous sample, because the current IMS system is different from the previous environment.

21.5.2 Step 2: Setting up WebSphere Application Server for z/OS subsystem

In this section, we introduce how we set up the WebSphere Application Server for z/OS environment.

Step 2-1: Concatenating IMS libraries to servant regions

We place our DRA module into IMS910G.SDFSRESL. The WebSphere Application Server for z/OS servant region STEPLIB should have the load library that contains the DRA startup table, the ODBA runtime code, and the SDFSJLIB data set. We add both data sets to our WebSphere Application Server for z/OS servant region JCL, as shown in Example 21-21.

Example 21-21 The servant region JCL for WebSphere Application Server for z/OS

```
//WS6531S PROC ENV=,
// SET ROOT='/WebSphereJJ/V6R0/BS01'
//BBOSR EXEC PGM=BBOSR,REGION=OM,TIME=NOLIMIT,
// PARM='TRAP(ON,NOSPIE),ENVAR("_EDC_UMASK_DFLT=007") /'
//BBOENV DD PATH='&ROOT/&ENV/was.env'
//*
/* Output DDs
/*
//CEEDUMP DD SYSOUT=*,SPIN=UNALLOC,FREE=CLOSE
//SYSOUT DD SYSOUT=*,SPIN=UNALLOC,FREE=CLOSE
//SYSPRINT DD SYSOUT=*,SPIN=UNALLOC,FREE=CLOSE
/*
/*Steplib Setup
/*
//STEPLIB DD DISP=SHR,DSN=BB06053.SBBOLD2
// DD DISP=SHR,DSN=BB06053.SBBLOAD
// DD DISP=SHR,DSN=IMS910G.SDFSRESL
// DD DISP=SHR,DSN=IMS910G.SDFSJLIB
```

Step 2-2: Installing the IMS JDBC resource adapter

We install the IMS JDBC resource adapter by performing the following steps:

1. From the WebSphere Application Server for z/OS administrative console, click **Resources**, and then click **Resource Adapters**. This displays a list of resource adapters, as shown in Figure 21-19.

Resource adapters

A resource adapter is an implementation of the J2EE Connector Architecture Specification that provides access for applications to resources outside of the server or provides access for an Enterprise Information System (EIS) to applications on the server. It can provide application access to resources such as DB2, CICS, SAP and PeopleSoft. It can provide an EIS with the ability to communicate with message driven beans that are configured on the server. Some resource adapters are provided by IBM; however, third party vendors can provide their own resource adapters. A resource adapter implementation is provided in a resource adapter archive file; this file has an extension of .rar. A resource adapter can be provided as a standalone adapter or as part of an application, in which case it is referred to as an embedded adapter. Use this panel to install a standalone resource adapter archive file. Embedded adapters are installed as part of the application install.

Scope: Cell=**cel6531**, Node=**nd6531sc53**

☐ Cell : cel6531 Scope specifies the level at which the resource definition is visible. For detailed information on what scope is and how it works, [see the scope settings help](#)

☒ Node : nd6531sc53

☐ Server : ws6531sc53

Apply

Preferences

Install RAR **New** **Delete**

Select	Name
<input type="checkbox"/>	SIB JMS Resource Adapter
<input type="checkbox"/>	WebSphere Relational Resource Adapter

Total 2

Figure 21-19 List of resource adapters

2. Click **Install RAR**. A panel for installing the resource adapter opens, as shown in Figure 21-20. Select **Server path** and type the path to the imsjava91.rar file. In our case, the location of the RAR file is:

/SC53/imsv9/imsjava91/imsjava91.rar

Click **Next**.

Resource adapters Close page

Install RAR File

RAR files can be installed using two methods. You can choose to upload a RAR file from local file system or you can specify an existing RAR file on a server.

Path

☐ Local path:
Specify path **Browse...**

☒ Server path:
Specify path

Scope
Node

Next **Cancel**

Figure 21-20 Install RAR File panel

3. A configuration panel opens, as shown in Figure 21-21. Enter the following information:
 - Name: A name for resource adapter, in our case, IMS JDBC Resource Adapter.
 - Class path: The path to the imsjava.jar file, including the file name, in our case, /SC53/imsv9/imsjava91/imsjava.jar.

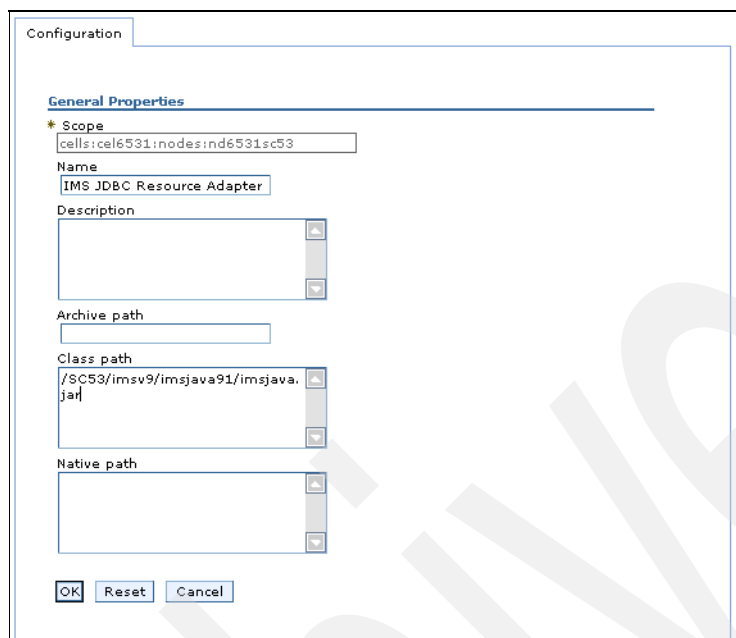


Figure 21-21 Configuration panel

4. Click **OK**. The IMS JDBC resource adapter is listed, as shown in Figure 21-22.

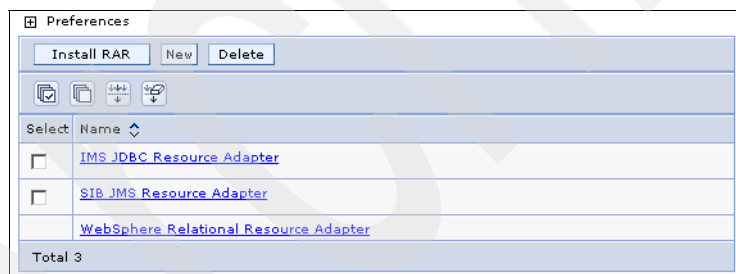


Figure 21-22 List of resource adapters with IMS JDBC resource adapter

5. Click **Save**. The Save page opens. Under Save to Master Configuration, click **Save** to ensure that the changes have been made.

Step 2-3: Installing the custom service

When WebSphere Application Server for z/OS is started, the custom service initializes the ODBA environment. When the server is stopped, the custom service terminates the ODBA environment. After a server is started, every application that is running in the server uses the initialized ODBA environment.

We install the IMS JDBC resource adapter by performing the following steps:

1. Modify the WebSphere Application Server for z/OS server.policy file, which is in the properties directory of the WebSphere Application Server installation directory, by adding the code shown in Example 21-22 on page 438.

Example 21-22 Permission code for WebSphere Application Server for z/OS server.policy file

```
grant codeBase "file:/SC53/imsv9/imsjava91/-" {  
    permission java.util.PropertyPermission "*", "read, write";  
    permission java.lang.RuntimePermission "loadLibrary.JavTDLI";  
};
```

2. In the left frame of the WebSphere Application Server for z/OS administrative console, click **Servers**, and then click **Application Servers**. This opens a list of application servers, as shown in Figure 21-23. Click the name of the server on which you want to deploy your enterprise application.

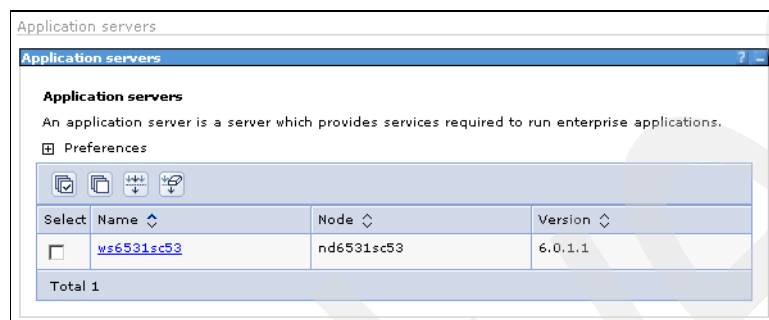


Figure 21-23 List of application servers

3. Under Server Infrastructure in the right panel, click **Administration** → **Custom Services**. This displays a list of custom services, as shown in Figure 21-24.

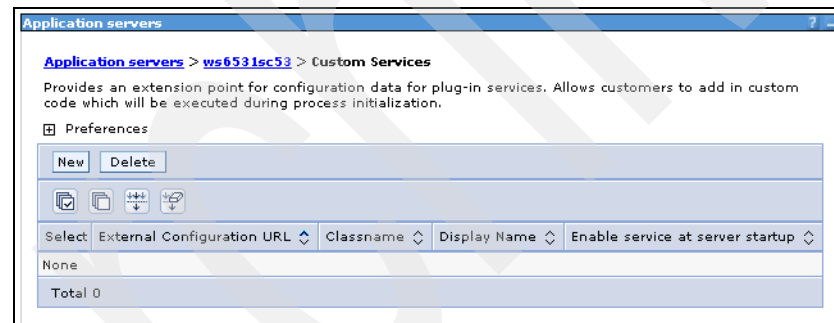


Figure 21-24 List of custom services

4. Click **New**. A configuration panel opens, as shown in Figure 21-25 on page 439. Configure the custom service. Select the **Enable service at server** option. If you do not select this option, the custom service is not invoked when you start the server.

Enter the following information:

- Classname: com.ibm.connector2.ims.db.IMSJdbcCustomService
- Display Name: A name for the custom service, in our case, IMS ODBA Custom Service
- Classpath: The path to the directory that contains imsjava.jar and libJavTDLI.so, in our case, /SC53/imsv9/imsjava91

Click **OK**.

Note: You might have to add the execute authority to the libJavTDLI.so file on UNIX System Services.

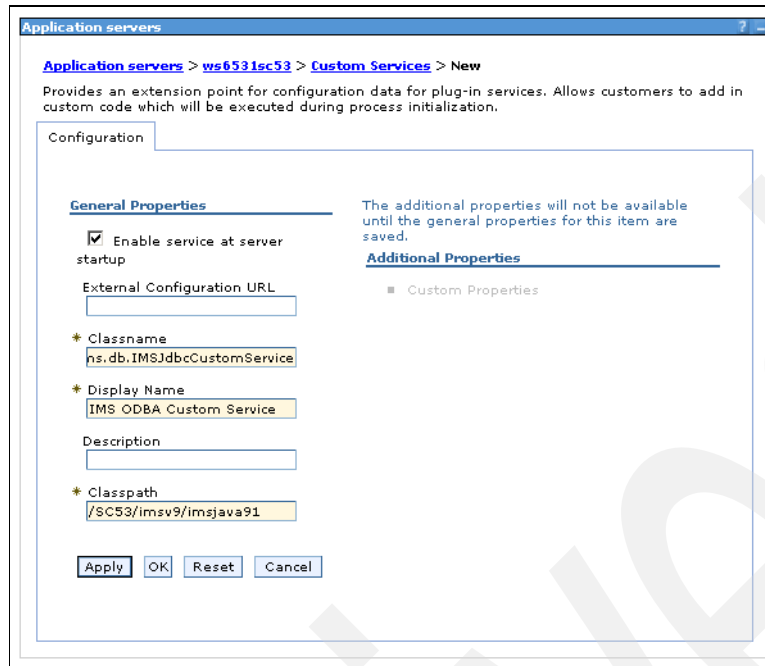


Figure 21-25 Configuration panel for the custom service

This lists the custom service, as shown in Figure 21-26.

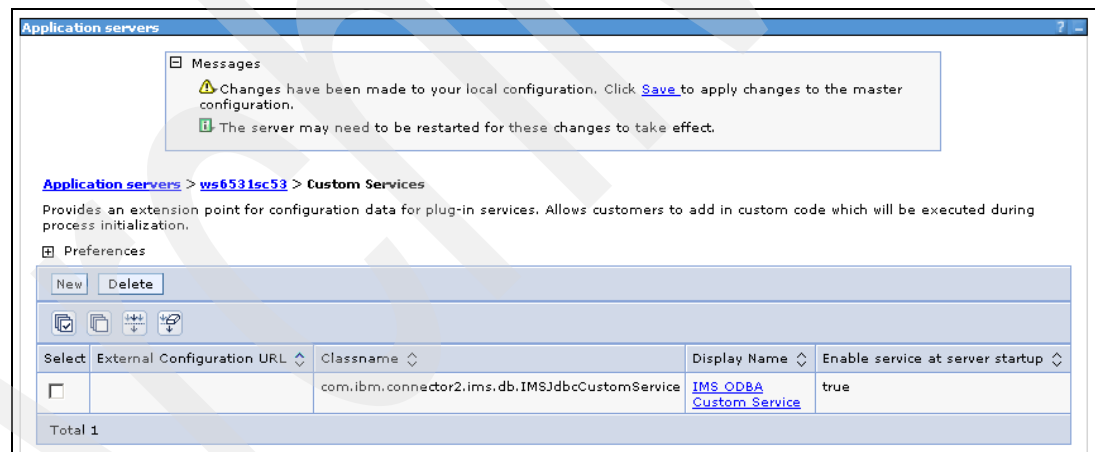


Figure 21-26 List of custom services with IMS ODBA Custom Service

5. Click **Save**. The Save page opens. Under Save to Master Configuration, click **Save** to ensure that the changes have been made.

Step 2-4: Installing the data source for IMS Java EJB on z/OS

Unlike the data source for z/OS applications, this data source does not have values for the IMS-specific properties. At run time, the client application's data source properties are propagated to an instance of this data source. The only requirement is that you should specify

the JNDI name as `imsRDSDataSource`. We install the data source for IMS Java EJB by performing the following steps:

1. In the left frame of the WebSphere Application Server for z/OS administrative console, click **Resources**, and then click **Resource Adapters**. This opens a list of resource adapters, as shown in Figure 21-27.

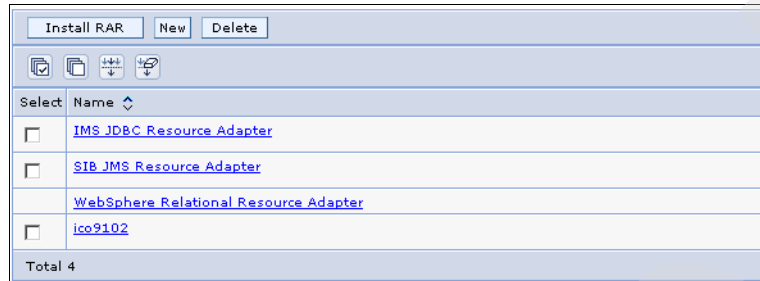


Figure 21-27 List of resource adapters

2. Click the name of IMS JDBC resource adapter that you chose when you installed the adapter. A configuration opens, as shown in Figure 21-28. Under Additional Properties, click **J2C connection factories**.

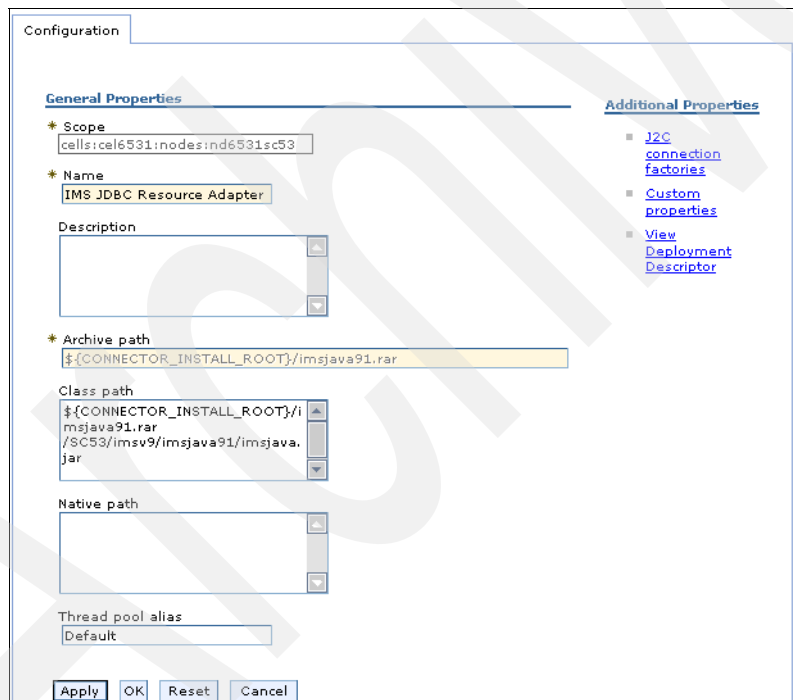


Figure 21-28 Configuration panel for resource adapters

3. In the J2C connection factories panel, click **New**. A configuration panel opens, as shown in Figure 21-29 on page 441. Enter the following information:
 - Name: The name for the data source, in our case, `IMS RDS DataSource`
 - JNDI name: `rdsDataSource`
 Click **OK**.

Configuration

General Properties

* Scope
cells:cel6531:nodes:nd6531sc53

* Name
IMS RDS DataSource

JNDI name
rdsDataSource

Description

* Connection factory interface
javax.sql.DataSource

Category

Component-managed authentication alias
Component-managed authentication alias
(none)

Container-managed authentication
Container-managed authentication alias
(deprecated in V6.0, use resource reference authentication settings instead)
(none)

The additional properties will not be available until the general properties for this item are saved.

Additional Properties

- Connection pool properties
- Advanced connection factory properties
- Custom properties

Related Items

Figure 21-29 Configuration panel for J2C connection factory

This lists the data source in the J2C Connection Factories, as shown in Figure 21-30.

Preferences

New Delete

Select	Name	JNDI name	Description	Connection factory interface	Category
<input type="checkbox"/>	IMS RDS DataSource	rdsDataSource		javax.sql.DataSource	

Total 1

Figure 21-30 List of J2C connection factories with IMS RDS data source

- Click **Save to Master Configuration**. The Save to Master Configuration opens. Click **Save**.

Step 2-5: Installing the EAR file including IMS Java EJB

The EAR file contains the two IMS Java-provided EJBs. These stateful session beans act as server-side extensions of the IMS distributed JDBC resource adapter. We install the EAR file, including the IMS Java EJB, by performing the following steps:

Note: Before running this step, you must download the required resources for RDS from the IMS Web site. For more information about the download, see the following Web site:

<http://www.ibm.com/software/data/ims/imsjava/rds.html>

- From the WebSphere Application Server for z/OS administrative console, click **Applications**, and then click **Install New Application**. A panel for installing a new application opens, as shown in Figure 21-31 on page 442. Type the path to the EAR file named `imsjavaRDS.ear`. In our case, the path is `/u/jouko2/imsjavaRDS.ear`.

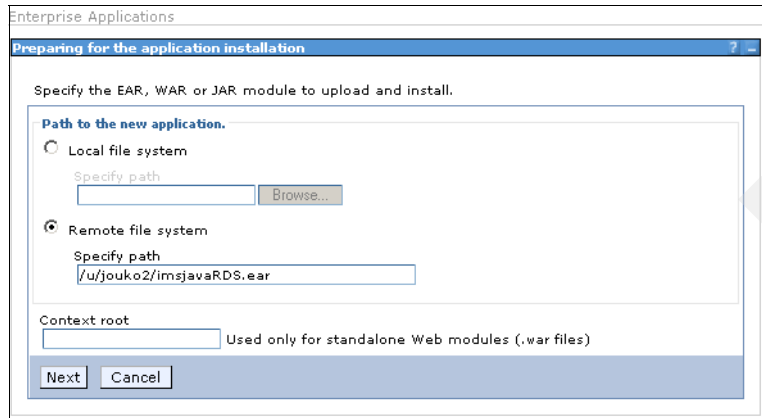


Figure 21-31 Installing a new application panel

2. Click **Next**. A panel for preparing for application installation opens, as shown in Figure 21-32. Accept the defaults. Click **Next**.

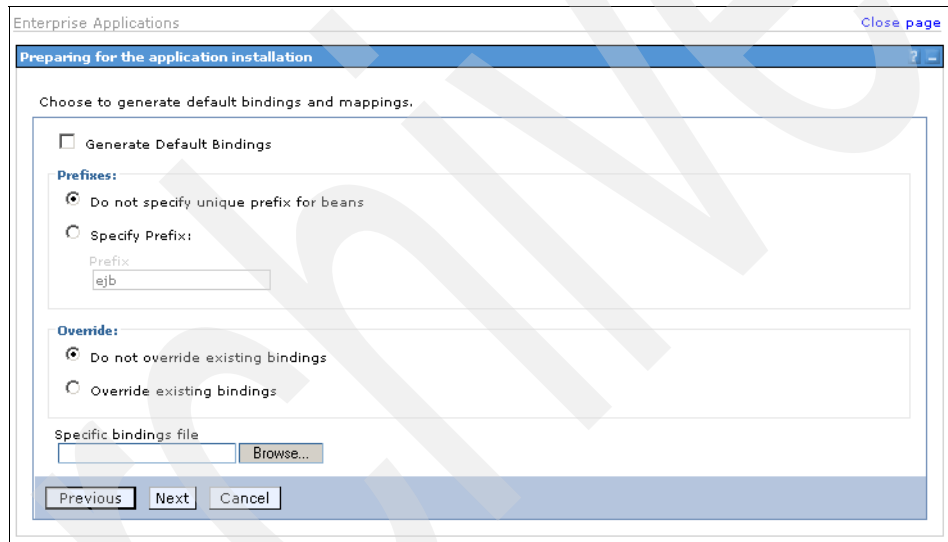


Figure 21-32 Preparing for the application installation panel

- The Install New Application wizard starts. The “Step 1: Select installation options” panel opens, as shown in Figure 21-33. Clear the **Create MBeans for resources** check box. Click **Next**.

Specify options for installing enterprise applications and modules.

Step 1: Select installation options

Specify the various options that are available to prepare and install your application.

☐ Pre-compile JSP

Directory to install application:

☒ Distribute application

☐ Use Binary Configuration

☐ Deploy enterprise beans

Application name:

☐ Create MBeans for resources

☐ Enable class reloading

Reload interval in seconds:

☐ Deploy Web services

Validate Input off/warn/fail:

☐ Process embedded configuration

Figure 21-33 Select installation options

- The “Step 2: Map modules to servers” panel opens, as shown in Figure 21-34. Accept the defaults. Click **Next**.

Specify options for installing enterprise applications and modules.

Step 2: Map modules to servers

Specify targets such as application servers or clusters of application servers where you want to install the modules contained in your application. Modules can be installed on the same application server or dispersed among several application servers. Also, specify the Web servers as targets that will serve as routers for requests to this application. The plug-in configuration file (plugin-cfg.xml) for each Web server is generated based on the applications which are routed through it.

Clusters and Servers:

Select	Module	URI	Server
<input checked="" type="checkbox"/>	IMS Java Remote Database Services EJB	imsjavaRDSHost.jar,META-INF/ejb-jar.xml	WebSphere: cell=cel6531,node=nd6531sc53,server=ws6531sc53

Figure 21-34 Map modules to servers

- The “Step 3: Provide JNDI Names for Beans” panel opens, as shown in Figure 21-35. In the JNDI name field, verify that the JNDI names are as follows:

```
ejb/com/ibm/ims/rds/host/HostBeanManagedSessionHome
ejb/com/ibm/ims/rds/host/HostContainerManagedSessionHome
```

Click **Next**.

Specify options for installing enterprise applications and modules.

Step 3: Provide JNDI Names for Beans

Each non-message-driven enterprise bean in your application or module must be bound to a Java Naming and Directory Interface (JNDI) name.

EJB module	EJB	URI	JNDI name
IMS Java Remote Database Services EJB	HostBeanManagedSession	imsjavaRDSHost.jar,META-INF/ejb-jar.xml	ejb/com/ibm/ims/rds/ho
IMS Java Remote Database Services EJB	HostContainerManagedSession	imsjavaRDSHost.jar,META-INF/ejb-jar.xml	ejb/com/ibm/ims/rds/ho

Previous Next Cancel

Figure 21-35 Provide JNDI Names for Beans

- The “Step 4: Map resource references to resources” panel opens, as shown in Figure 21-36. Verify that the JNDI name of the resource reference for both EJBs of the IMS Java Remote Database Services EJB module is rdsDataSource. Click **Next**.

Select	Module	EJB	URI	Reference binding	JNDI name	Login configuration
<input type="checkbox"/>	IMS Java Remote Database Services EJB	HostBeanManagedSession	imsjavaRDSHost.jar,META-INF/ejb-jar.xml	rdsDataSource	rdsDataSource	Resource authorization: Container Authentication method: none
<input type="checkbox"/>	IMS Java Remote Database Services EJB	HostContainerManagedSession	imsjavaRDSHost.jar,META-INF/ejb-jar.xml	rdsDataSource	rdsDataSource	Resource authorization: Container Authentication method: none

Figure 21-36 Map resource references to resources

Note: After this panel, you might receive an Application Resource Warning with ADMA8019E message, which can be ignored because this simply indicates you are installing the EAR file with a dataSource that applies to a J2C resource adapter. If you see the message, click **Continue**.

- The “Step 5: Correct use of System Identity” panel opens, as shown in Figure 21-37. Verify that no role has been selected. Click **Next**.

Figure 21-37 Correct use of System Identity

- The “Step 6: Ensure all unprotected 2.x methods have the correct level of protection” panel opens, as shown in Figure 21-38. Make any necessary changes for your security requirement. In this case, we select the **Uncheck** option. Click **Next**.

Figure 21-38 Ensure all unprotected 2.x methods have the correct level of protection

- The options that you specified are displayed in the “Step 7: Summary of the Install New Application wizard” panel, as shown in Figure 21-39. Verify that the options are correct, and then click **Finish**. A message is displayed that indicates first that the imsjavaRDS application is being installed, and then that the installation was successful.

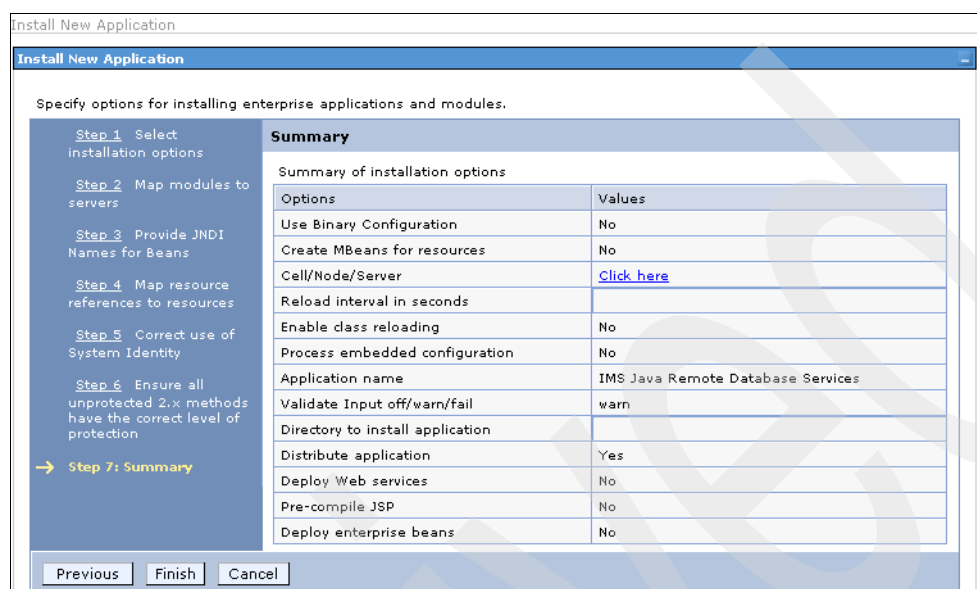


Figure 21-39 Summary of the Install New Application wizard

- Click **Save to Master Configuration**. The Save page opens. Under Save to Master Configuration, click **Save**.

21.5.3 Step 3: Installing the metadata class for the sample application

We use IMS IVP application (DFSIVP37) for our sample Web application; therefore, we need the metadata class from the DFSIVP37 PSB source and IVPDB2 DBD source. We already discussed the sample metadata class creation in 21.3.1, “Example of using the DLIModel utility” on page 422 with these PSB/DBD sources. In this section, we introduce how to set the WebSphere Application Server for z/OS classpath to the location of your metadata classes. Before adding the classpath, we compile the metadata source file and create the JAR file by using the command shown in Example 21-23 on our UNIX System Services environment.

Example 21-23 Compile the metadata source and create the JAR file

```
JOUK02 @ SC53:/u/jouko2>javac imsrds/*.java

JOUK02 @ SC53:/u/jouko2>jar -cvf imsrds.jar ./imsrds/*
added manifest
adding: imsrds/@dictl(in = 601) (out= 274)(deflated 54%)
adding: imsrds/DFSIVP37DatabaseView.class(in = 1208) (out= 605)(deflated 49%)
adding: imsrds/DFSIVP37DatabaseView.java(in = 1279) (out= 517)(deflated 59%)
adding: imsrds/DFSIVP37DatabaseViewJavaReport.txt(in = 437) (out= 212)(deflated 51%)
adding: imsrds/META-INF/(in = 0) (out= 0)(stored 0%)
adding: imsrds/META-INF/MANIFEST.MF(in = 62) (out= 61)(deflated 1%)
adding: imsrds/imsrds.jar(in = 2495) (out= 2055)(deflated 17%)
```

One way to set the classpath is to add the metadata class to the IMS JDBC resource adapter classpath by performing the following steps:

1. From the WebSphere Application Server for z/OS administrative console, click **Resources**, and then click **Resource Adapters**. This displays a list of resource adapters, as shown in Figure 21-40.



Figure 21-40 List of resource adapters

2. Click the name of the IMS JDBC resource adapter. A configuration panel opens, as shown in Figure 21-41. In the Class path field, add your location of metadata class. In our case, the location is `/u/jouko2/imsrds/imsrds.jar`.

Do not delete `imsjava.jar`. Click **OK**.

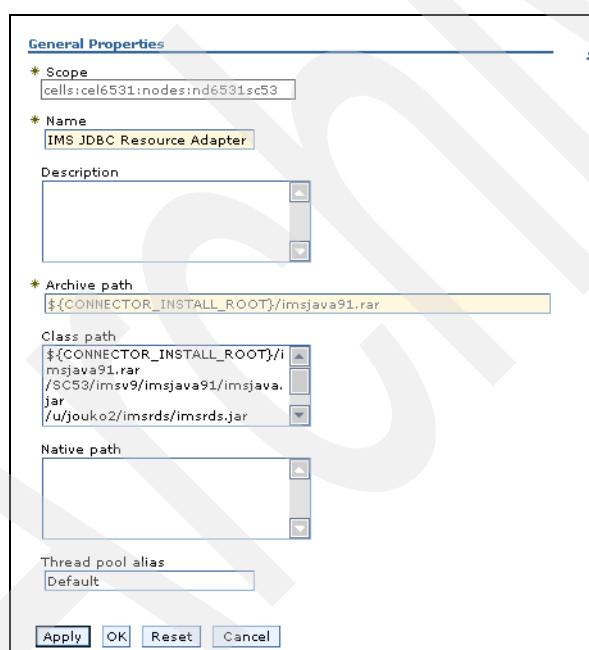


Figure 21-41 Configuration panel for resource adapter

3. Click **Save**. The Save page opens. Under Save to Master Configuration, click **Save** to ensure that the changes have been made.

After completing this step, the configuration for then WebSphere Application Server for z/OS side is completed. Then, you have to recycle the WebSphere Application Server for z/OS region to reflect your new configurations.

21.5.4 Step 4: Setting up application server for distributed platforms environment

In this section, we introduce how we set up the WebSphere Application Server for distributed platforms environment.

Step 4-1: Installing the IMS distributed JDBC resource adapter

Before deploying applications, you must first set up WebSphere Application Server on the non-z/OS client side by installing the IMS distributed JDBC resource adapter. WebSphere Application Server on the client side requires only the IMS distributed JDBC resource adapter, *imsjavaRDS.rar*.

We install the IMS distributed JDBC resource adapter by performing the following steps:

1. From the client-side WebSphere Application Server administrative console, click **Resources**, and then click **Resource Adapters**. This displays a list of resource adapters, as shown in Figure 21-42.

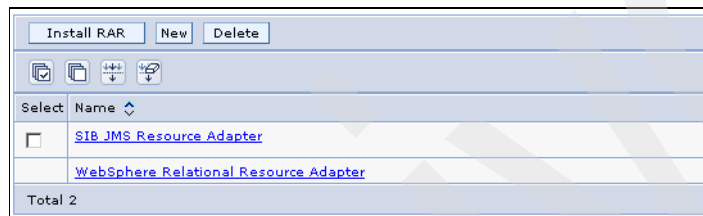


Figure 21-42 List of resource adapters in WebSphere Application Server for distributed platforms

2. Click **Install RAR**. A panel for installing the resource adapter opens, as shown in Figure 21-43. Then, enter the Local path to the *imsjavaRDS.rar* file. Click **Next**.



Figure 21-43 Install RAR File panel

3. A configuration panel opens, as shown in Figure 21-44. Click **Next**.

Configuration

General Properties

* Scope
cells:dhcp39116Node01.Cell:nodes:dhcp39116Node01

Name
IMS Distributed JDBC RA

Description

Archive path

Class path

Native path

OK Reset Cancel

Figure 21-44 Configuration panel

4. The IMS distributed JDBC resource adapter is listed, as shown in Figure 21-45.

Install RAR New Delete

Select Name

<input type="checkbox"/>	IMS Distributed JDBC RA
<input type="checkbox"/>	SIB JMS Resource Adapter
<input type="checkbox"/>	WebSphere Relational Resource Adapter

Total 3

Figure 21-45 List of resource adapters with IMS distributed JDBC resource adapter

5. Click **Save to Master Configuration**. The Save page opens. Under Save to Master Configuration, click **Save**.

Step 4-2: Installing the data source for the client application

Our client application will use the data source on the WebSphere Application Server for distributed side to connect the WebSphere Application Server for z/OS and make the specific ODBA connection with the PSB. The J2C properties on the WebSphere Application Server for distributed platforms will be propagated to an instance of the IMS RDS data source definition on the WebSphere Application Server for z/OS side. We install the data source for our client application by performing the following steps:

1. In the left frame of the client-side WebSphere Application Server administrative console, click **Resources**, and then click **Resource Adapters**. This displays a list of resource adapters.

2. Click **IMS Distributed JDBC RA**. A configuration panel opens, as shown in Figure 21-46. Under **Additional Properties**, click **J2C connection factories**.

Configuration

General Properties

* Scope
cells:dhcp39116Node01Cell:nodes:dhcp39116Node01

* Name
IMS Distributed JDBC RA

Description

Additional Properties

- [J2C connection factories](#)
- [Custom properties](#)
- [View Deployment Descriptor](#)

Figure 21-46 Configuration panel

3. Click **New**. A configuration panel opens, as shown in Figure 21-47. Enter the following information:
 - Name: Display name of the J2C connection factory, in our case, `imsjavaRDSRedBook`
 - JNDI name: JNDI name of the J2C connection factory, in our case, `imsjavaRDSRedBook`Click **OK**.

Tip: To avoid the messages J2CA0107I and J2CA0114W, both of which can be ignored, set the default values for the component-managed authentication alias and container-managed authentication alias.

Configuration

General Properties

* Scope
cells:dhcp39116Node01Cell:nodes:dhcp39116Node01

* Name
imsjavaRDSRedBook

JNDI name
imsjavaRDSRedBook

Description

* Connection factory interface
javax.sql.DataSource

Category

The additional properties will not be available until the general properties for this item are saved.

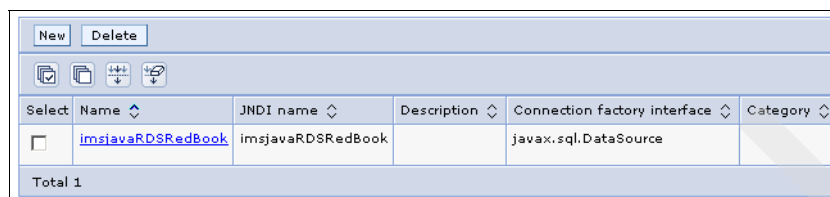
Additional Properties

- [Connection pool properties](#)
- [Advanced connection factory properties](#)
- [Custom properties](#)

Related Items

Figure 21-47 Configuration panel for the J2C connection factory

- The data source is listed in the J2C connection factories, as shown Figure 21-48. Click the name of the data source that you installed in step 3.



Select	Name	JNDI name	Description	Connection factory interface	Category
<input type="checkbox"/>	imsjavaRDSRedBook	imsjavaRDSRedBook		javax.sql.DataSource	

Total 1

Figure 21-48 List of data source with the J2C connection factories

- Under **Additional Properties**, click **Custom Properties**. A table lists six properties. We specified the property values in Table 21-6.

Table 21-6 Custom properties

Custom property name	Description	Our value
DatabaseViewName	Fully qualified name of the metadata class	imsrds.DFSIIP37DatabaseView
DRAName	The DRA name of the IMS to which to connect	IMSG
HostName	The host name (or IP address) of WebSphere Application Server for z/OS	wtsc53.itso.ibm.com
PortNumber	The IIOP port number of WebSphere Application Server for z/OS	12809
TraceLevel	Trace level for J2EE tracing	0
TransactionResource Registration	Type of transaction resource registration (enlistment)	This value must be "dynamic" (deferred) for this resource adapter.

- Click **Save**. The Save page opens. Under Save to Master Configuration, click **Save** to ensure that the changes have been made.

After completing this step, the configuration for the WebSphere Application Server for distributed platforms side is completed. Then, you have to recycle WebSphere Application Server for distributed platforms to reflect your new configurations.

21.5.5 Step 5: Developing the sample application

In this section, we discuss how to develop your enterprise application that accesses the IMS database from the WebSphere Application Server for distributed platforms environment. For this purpose, we make simple Java servlets to access the IMS database on the environment configured in the previous sections. You can download the ZIP file, which contains two Rational Application Developer Version 6 projects (the dynamic Web project and the EAR project), from the following IBM Redbook Web site (see Appendix C, "Additional material" on page 507):

<ftp://www.redbooks.ibm.com/redbooks/SG246794/IMSRDSSampleProjects.zip>

You can add the two projects to your Rational Application Developer workspace from the ZIP file using Rational Application Developer import function. To import the projects, execute the following steps:

1. Click **File** and then click **Import**. A import wizard opens, as shown in Figure 21-49. Select **Project Interchange**. Click **Next**.

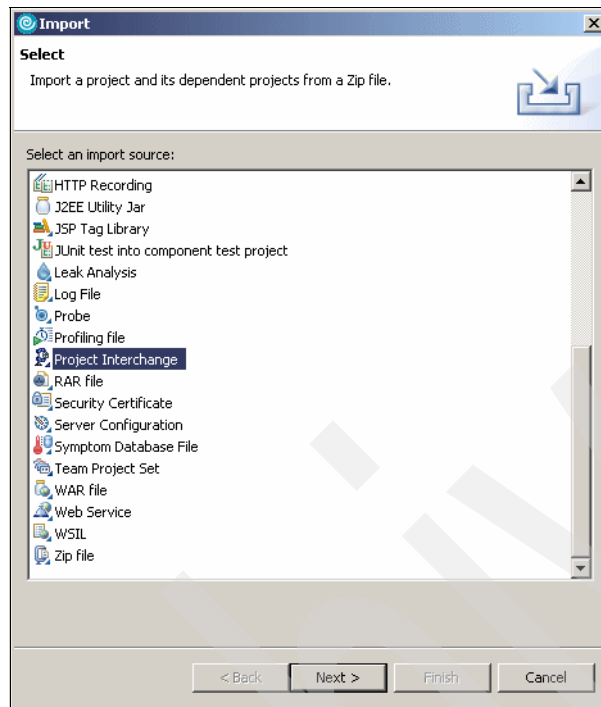


Figure 21-49 Import wizard

2. In the Import Project Interchange Contents panel shown in Figure 21-50 on page 453, specify your downloaded Zip file location in the From zip file field. Then, select the following projects:
 - **RedBookIMSRDS**
 - **RedBookIMSRDSEAR**

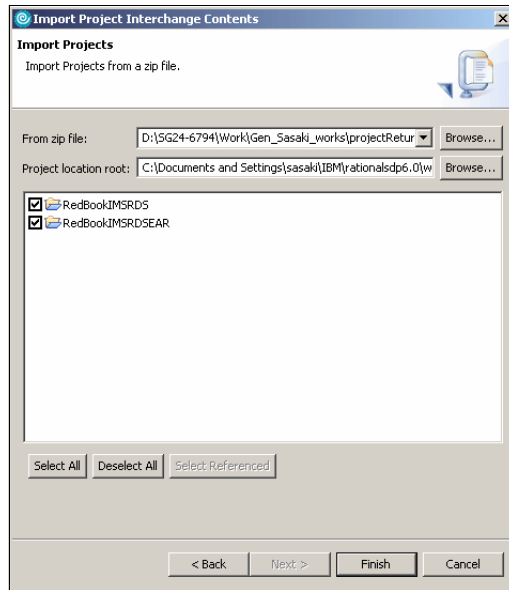


Figure 21-50 Import Project Interchange Contents

3. Click **Finish**. Two projects are added to your workspace, as shown in Figure 21-51.



Figure 21-51 Project Explorer on workspace

The RedBookIMSRDS project contains the resources shown in Table 21-7.

Table 21-7 RedBookIMSRDS project contents

Resource	Description
ImsRdsSampleGlobal.java	Servlet for IMS JDBC access with global transaction semantics
ImsRdsSample.java	Servlet for IMS JDBC access with local transaction semantics
GlobalInput.html	HTML document that invokes the servlet for global transaction
LocalInput.html	HTML document that invokes the servlet for local transaction
Output.jsp	JSP™ file for display the results

Servlet for IMS JDBC access with global transaction semantics

ImsRdsSampleGlobal.java is the servlet that contains the IMS JDBC access logic and transaction handling logic for global transaction execution in the Web container. In this section, we explain how this servlet works. Appendix B, “IMS RDS application example” on page 499 provides the complete servlet code.

Step 1: Looking up the datasource and user transaction

Example 21-24 shows the code for looking up dataSource and the userTransaction. The dataSource is registered with the naming service based on JNDI. This dataSource object has properties that pertain to the actual data source that the application needs to access. The sample looks up the dataSource, which is configured in “Step 4-2: Installing the data source for the client application” on page 449.

Example 21-24 Look up datasource and user transaction

```
.....
//Obtain the initial JNDI Naming context for JDBC Connection
Context initialContext = new InitialContext();
dataSource = (DataSource)initialContext.lookup
    ("java:comp/env/imsjavaRDSRedBook");
System.out.println("IMS RDS Servlet : Success Create DataSource");

//Obtain the initial JNDI Naming context for User Transaction
Context initctx2 = new InitialContext();
userTransaction = (UserTransaction)initctx2.lookup("java:comp/UserTransaction");
System.out.println("IMS RDS Servlet : Success Create UserTransaction");
.....
```

To look up the specific data source in the WebSphere environment, the Web application must have a resource-ref element in the deployment descriptor. The resource-ref element describes external resources. In the resource-ref element, you must have the following elements:

```
<res-type>javax.sql.DataSource</res-type>
<res-sharing-scope>Unshareable</res-sharing-scope>
```

The <res-type> tag element specifies the type of data source. The <res-sharing-scope> tag element specifies that the connections are *not* shareable.

We describe the resource reference in the Web application deployment descriptor. Because the servlet has JDBC access logic in the servlet itself, not the EJB, the Web container will refer the resource reference in the Web application deployment descriptor. Example 21-25 shows the resource reference description of this sample application.

Example 21-25 Resource reference description in the Web deployment descriptor

```
.....
<resource-ref id="ResourceRef_1119506677212">
  <res-ref-name>imsjavaRDSRedBook</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Application</res-auth>
  <res-sharing-scope>Unshareable</res-sharing-scope>
</resource-ref>
.....
```

The sample code also looks up the userTransaction object. The servlet resides outside of the EJB container and cannot use an EJBContext object. The initial context requires an additional JNDI lookup to locate and instantiate the UserTransaction interface.

Step 2: Beginning the user transaction

Example 21-26 on page 455 shows the code for the begin of the user transaction. You can define the scope of the transaction begin by using begin method. After issuing this method, the application logic to access resources are included in the transaction demarcation until the commit request is issued.

Example 21-26 Begin user transaction

```
.....
        //Start User Transaction
        userTransaction.begin();
        System.out.println("IMS RDS Servlet : Success Start UserTransaction");
.....
```

Step 3: Getting the connection

Example 21-27 shows the code for getting the connection. When the `getConnection` method is issued, the database connection builds (in this case, the ORB connection from the WebSphere Application Server for distributed platform to the WebSphere Application Server for z/OS).

Example 21-27 Get connection

```
.....
        Connection connection = null;
.....
        //Get JDBC Connection
        connection = dataSource.getConnection();
        System.out.println("IMS RDS Servlet : Success Get Connection");
.....
```

Step 4: Issuing an SQL statement

Example 21-28 shows the code for the issuing SQL. The `Statement` class defines the interfaces that accept SQL statements by the `createStatement` method. Then, the `executeQuery` method of the `Statement` class returns tables as the `ResultSet` object. In this sample, the results object stores the query results from the IVPDB2 database defined as the `Person` table in the metadata class. After receiving the SQL query result, the `getString` method, which is implemented in the `ResultSet` interface, retrieves the row of the table and converts the data type defined in the metadata class.

Example 21-28 Issue SQL statement

```
.....
        Statement statement = null;
        ResultSet results = null;
.....
        //Issue SQL
        String queryString = "SELECT * FROM PhoneBook.Person "
            + "WHERE Person.LastName = '" + keyvalue + "'";
        statement = connection.createStatement();
        results = statement.executeQuery(queryString);
.....
        //Get Output
        while(results.next()) {
            LastName = results.getString("Person.LastName");
            FirstName = results.getString("Person.FirstName");
            Extension = results.getString("Person.Extension");
            ZipCode = results.getString("Person.ZipCode");
        }
        System.out.println("IMS RDS Servlet : Success Get Result");
.....
```

Step 5: Committing or rolling back a user transaction

Example 21-29 on page 456 shows the code of a commit/rollback user transaction. After issuing the commit or rollback method, the resource update in the transaction demarcation

will be committed or aborted. In this case, WebSphere Application Server for distributed platforms controls the two-phase commit process with the cooperation of WebSphere Application Server for z/OS, RRS, and IMS database manager.

Example 21-29 Commit/rollback user transaction

```
..... try{
.....     userTransaction.begin();
.....     //Commit UserTransaciton
.....     userTransaction.commit();
.....     System.out.println("IMS RDS Servlet : Success Commit UserTransaction");
..... } catch (Exception e) {
.....     //RollBack UserTransaciton
.....     userTransaction.rollback();
..... }
```

Step 6: Closing the statement and connection

Example 21-30 shows the code for closing the statement and connection. After the `connection.close` method is issued, the connection is returned to the connection pool on WebSphere Application Server for distributed platforms and can be reused for another connection request.

Example 21-30 Close statement and connection

```
..... //Close JDBC Resources
..... if (statement != null) {statement.close();}
..... if (connection != null) {connection.close();}
.....
```

Servlet for IMS JDBC access with local transaction semantics

`ImsRdsSample.java` is the servlet that contains the IMS JDBC access logic and transaction handling logic for local transaction execution in the Web container. Appendix B, “IMS RDS application example” on page 499 shows the complete servlet code. With local transaction semantics, you can commit or roll back a transaction that is started by creating a data source connection. The IMS Java EJB that is on the z/OS server side automatically starts a transaction if a global transaction context does not exist when a connection is created. Note the following differences between the global transaction servlet and the local transaction servlet:

- ▶ The local transaction servlet does not have any codes related with the `javax.transaction.UserTransaction` interface. Therefore, the IMS Java JDBC request is executed outside the global transaction demarcation.
- ▶ The local transaction servlet issues a commit/rollback request against the connection object.

Example 21-31 on page 457 is a commit/rollback request against the connection object.

Example 21-31 Commit/rollback request against the connection object

```
..... try{
.....     connection = dataSource.getConnection();
.....     //Commit LocalTransaciton
.....     connection.commit();
.....     System.out.println("IMS RDS Servlet : Success Commit LocalTransaction");
..... } catch (Exception e) {
.....     //RollBack LocalTransaction
.....     connection.rollback();
..... }
```

Important: The local transaction programming model for IMS Java applies only to applications that run on WebSphere Application Server on a *non-z/OS platform* and that use the *Remote Database Services* of IMS Java. Any other IMS Java applications that run on WebSphere Application Server for z/OS, CICS Transaction Server, DB2 stored procedure, or JMP/JBP require the global transaction semantics, which is provided with the Java Transaction API JB container service or the native transactional API, depending on the environment.

21.5.6 Step 6: Defining the IMS environment

The IMS ODBA function requires the RRS interface. Therefore, the IMS execution parameter RRS= in the DFSPBxxx PROCLIB member must be set to Y (the default is N). If you execute your Web application with RRS=N, you receive `SQLException` with `AIBRETRN X'0108'`, `AIBREASN X'0544'`, which means that RRS is not active at the time that ODBA attempts to establish a connection to IMS or DBCTL.

We need to make sure that the IMS IVP that the application program (which is used by our stored procedure) will reference is defined to IMS. This includes building the required IMS control blocks, DBD, PSB, and ACB, and the MODBLKS modules, which are part of stage 1 and stage 2 IMS generation, and copying them from the staging libraries to the online libraries. We use the DBD(IVPDB2), PSB(DFSIVP37), and ACB members that come as part of the IMS IVP environment.

You can confirm your IVP application environment for the sample execution using the IMS commands shown in Example 21-32.

Example 21-32 IMS commands sample

```
R 731,/DIS DB IVPDB2
DFS000I  DATABASE TYPE TOTAL UNUSED TOTAL UNUSED ACC CONDITIONS IMSG
DFS000I  IVPDB2 DL/I UP ALLOCS IMSG
DFS000I  *05178/144754* IMSG
*732 DFS996I *IMS READY* IMSG

R 732,/DIS PGM DFSIVP37
DFS000I  PROGRAM TRAN TYPE IMSG
DFS000I  DFSIVP37 IVTCM JMP IMSG
DFS000I  *05178/144901* IMSG
*733 DFS996I *IMS READY* IMSG
```

21.5.7 Step 7: Running a Web application

Important: Before running the sample Web application, ensure that you establish a *two-way* TCP/IP communication between that WebSphere Application Server for distributed platforms environment and WebSphere Application Server for z/OS environment. Note that two-way means not only the communication path from the distributed environment to z/OS environment, but also the communication path *from z/OS environment to distributed environment* for ORB transportation. This can include giving the appropriate information about domain name resolution to z/OS and considering an adequate network path through the firewall.

Step 7-1: Adding the sample to the application server in Rational Application Developer workspace

We add the sample Web application to WebSphere Application Server in the Rational Application Developer workspace environment by performing the following steps:

1. In the Web perspective, right-click the server instance, and then select **Add and remove projects**.
2. The Add and Remove Projects wizard opens, as shown in Figure 21-52. Click **RedBookIMSRDSEAR**, and then click **Add**. Click **Finish**.

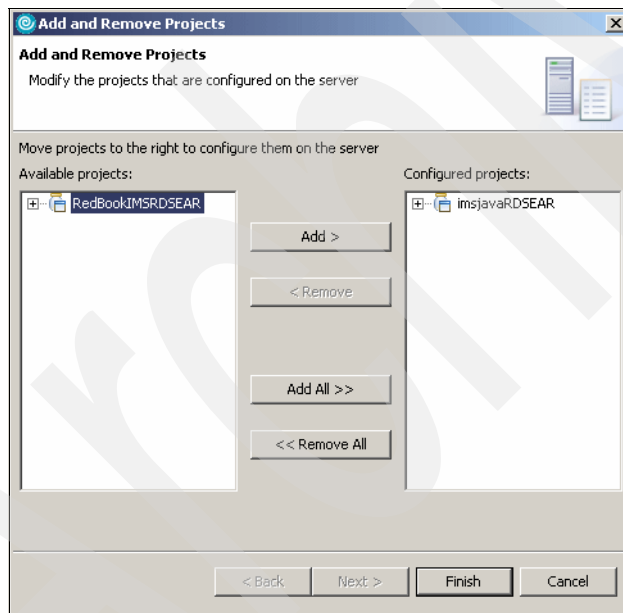


Figure 21-52 Add and Remove Projects wizard

Step 7-2: Executing the sample application and analyzing the output

You can invoke the servlet for the global transaction and the servlet for the local transaction from different Web pages:

- ▶ To invoke the servlet for a global transaction, enter the following URL:
`http://your_was_distributed_name:portnumber/RedBookIMSRDS/GlobalInput.html`
- ▶ To invoke the servlet for a local transaction, enter the following URL:
`http://your_was_distributed_name:portnumber/RedBookIMSRDS/LocalInput.html`

In this section, we show an example execution of the servlet for global transaction.

Enter the URL of the Web page for the global transaction. Figure 21-53 is displayed. Enter the key value (for example, LAST1) that you want to query, and then click **Execute**.

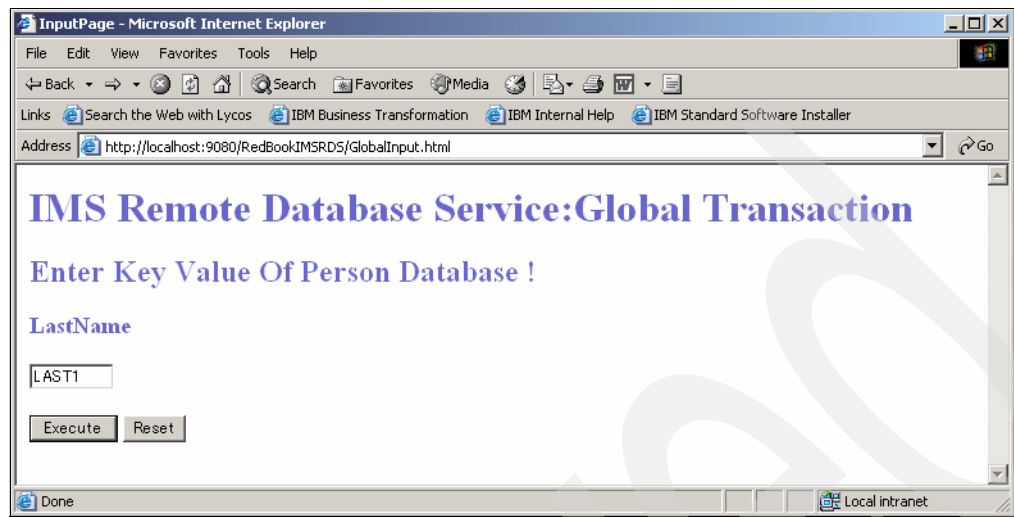


Figure 21-53 IMS RDS with Global Transaction execution page

The result page opens, as shown in Figure 21-54. In this case, the information about the person named LAST1 is displayed.



Figure 21-54 Result page for the success of query

At the first execution of the servlet, you will see the messages shown in Example 21-33 on page 460 on the console log of WebSphere Application Server for distributed platforms.

Example 21-33 Console log of WebSphere Application Server for distributed platforms (at first execution of the servlet)

```
[6/27/05 15:59:27:051 PDT] 00000036 SystemOut      0 IMS RDS Servlet : Start
[6/27/05 15:59:27:061 PDT] 00000036 SystemOut      0 IMS RDS Servlet : Success Create DataSource
[6/27/05 15:59:27:061 PDT] 00000036 SystemOut      0 IMS RDS Servlet : Success Create UserTransaction
[6/27/05 15:59:27:061 PDT] 00000036 ServletWrapper A  SRVE0242I: [ImsRdsSampleGlobal]: Initialization
successful.
[6/27/05 15:59:27:071 PDT] 00000036 SystemOut      0 IMS RDS Servlet : Success Start UserTransaction
[6/27/05 15:59:27:071 PDT] 00000036 SystemOut      0 IMS RDS Servlet : Success Get Connection
[6/27/05 15:59:28:243 PDT] 00000036 SystemOut      0 IMS RDS Servlet : Success Get Result
[6/27/05 15:59:28:513 PDT] 00000036 SystemOut      0 IMS RDS Servlet : Success Commit UserTransaction
[6/27/05 15:59:28:754 PDT] 00000036 ServletWrapper A  SRVE0242I: [/Output.jsp]: Initialization successful.
[6/27/05 15:59:28:784 PDT] 00000036 SystemOut      0 IMS RDS Servlet : Success Dispatch Result
[6/27/05 15:59:28:784 PDT] 00000036 SystemOut      0 IMS RDS Servlet : End
```

If you enter a key value of a segment that does not exist in the IVPDB2 database (for example, SASAKI), you will see the result page shown in Figure 21-55.

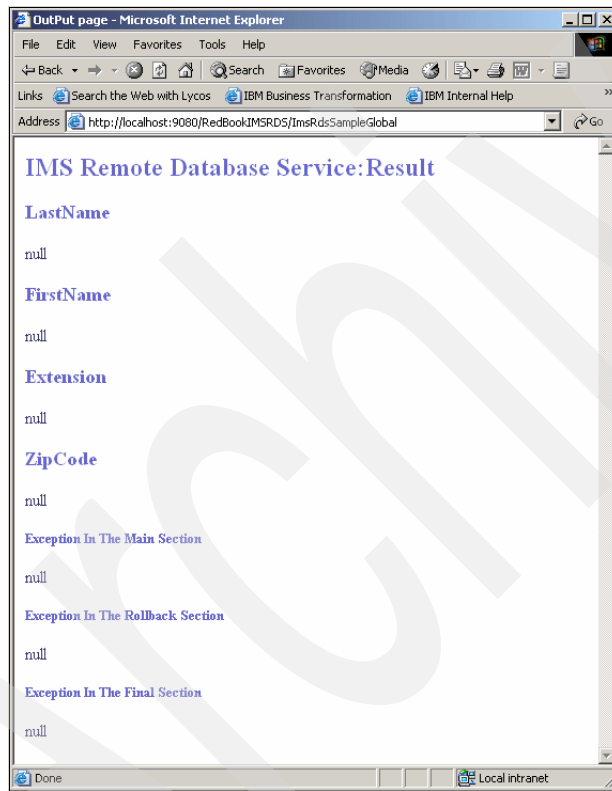


Figure 21-55 Result page for the success of query (person does not exist)

The resultset object shows null values with no exceptions. Because you are using the SQL language, you never see the status corresponding the DL/I status code "GE/GB". When the resultset.next method is executed, it invokes an internal DL/I call issue from the SQL query condition in the IMS Java EJB on the z/OS side. If the IMS Java EJB receives the AIB status code, which means status GE or GB, it simply returns the control to the WebSphere Application Server for distributed platforms side, even if no segment is matched with the query condition.

If you have an exception (for example, PSB DFSIVP37 is in stopped status), you will see the result page as shown in Figure 21-56 on page 461.

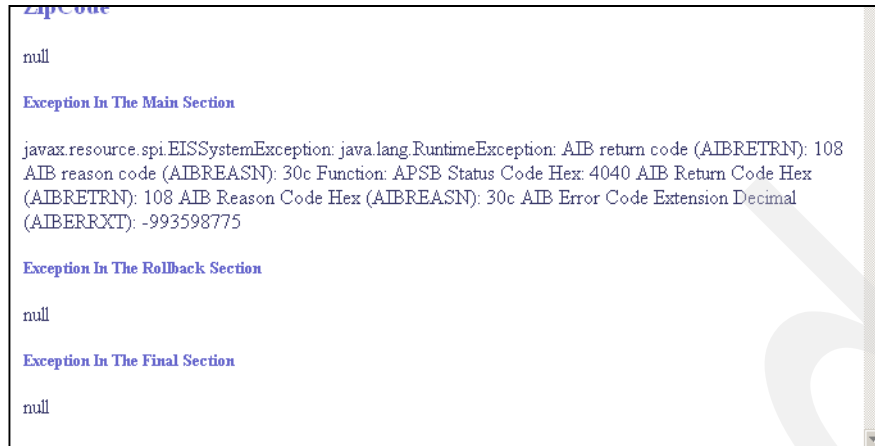


Figure 21-56 Result page for the failure with the exception (DFSIVP37 PSB is stopped status)

Also, you will see the following messages shown in Example 21-34 on the console log of WebSphere Application Server for distributed platforms.

Example 21-34 Console log of WebSphere Application Server for distributed platforms (with exception)

```
[6/27/05 18:02:13:504 PDT] 0000003f SystemOut      0 IMS RDS Servlet : Success Start UserTransaction
[6/27/05 18:02:13:504 PDT] 0000003f SystemOut      0 IMS RDS Servlet : Success Get Connection
[6/27/05 18:02:14:415 PDT] 0000003f SystemOut      0 IMS RDS Servlet : Success Rollback UserTransaction
[6/27/05 18:02:14:505 PDT] 0000003f SystemOut      0 IMS RDS Servlet : Caught exception in main section is:
java.sql.SQLException: javax.resource.spi.EISSystemException: java.lang.RuntimeException: AIB return code
(AIBRETRN): 108 AIB reason code (AIBREASN): 30c
    Function: APSB
    Status Code Hex: 4040
    AIB Return Code Hex (AIBRETRN): 108
    AIB Reason Code Hex (AIBREASN): 30c
    AIB Error Code Extension Decimal (AIBERRXT): -993598775
[6/27/05 18:02:14:505 PDT] 0000003f SystemOut      0 IMS RDS Servlet : Success Dispatch Result
[6/27/05 18:02:14:515 PDT] 0000003f SystemOut      0 IMS RDS Servlet : End
```

This information represents the following situation:

- ▶ The servlet received java.sql.SQLException when it executed the SQL query, and then it roll backed the user transaction.
- ▶ The exception information delivered from the IMS Java EJB contains the AIB return code X'108' and the AIB reason code X'30C', which means: "The program attempted to allocate a PSB that is marked permanently bad. IMS cannot allocate the PSB."

21.5.8 Problem determination for Remote Database Services

The Remote Database Services consists of a lot of components and applications to deliver your SQL request to the IMS ODBA interface. This means that whenever your Web application gets an exception, all of the components will be candidates for your problem determination. Some of the information that can be used for this purpose includes:

- ▶ java.sql.SQLException information that you receive
- ▶ WebSphere Application Server for distributed platforms console logs
- ▶ IMS distributed JDBC resource adapter traces
- ▶ WebSphere Application Server for z/OS console logs
- ▶ IMS JDBC resource adapter traces
- ▶ Joblogs of WebSphere Application Server for z/OS servant regions and control region

- IMS system messages
- IMS logs and traces, dumps (usually in case of a defect)

In this section, we introduce some techniques to determine a problem concerning the Remote Database Services. For the basic concepts of problem determination for the ODBA environment, see 19.6.1, “Finding the problem” on page 375.

Tracing DL/I calls with image capture

The SQL requests from your Web application will change into DL/I calls in the IMS Java engine on the WebSphere Application Server for z/OS side. Therefore, if you have a performance problem with executing a SQL, or receive an unexpected result for the query condition, it is a good idea to evaluate how your SQL is described in the DL/I call format intently. For this purpose, you can use the tracing facility for DL/I calls with image captures by using the IMS command.

To take a DL/I call image capture (for example, for the PSB DFSIVP37), use the following sequence of events for tracing:

1. Turn on trace with the following command:
/TRACE SET ON PSB DFSIVP37 COMP
2. Run the Web application.
3. Turn off trace with the following command:
/TRACE SET OFF PSB DFSIVP37
4. Switch the online log data sets (OLDSs). This also causes the contents to be archived to the secondary log data sets (SLDSs) with the following command:
/SWITCH OLDS
5. Execute the print utility DFSERA10, specifying the latest SLDS as input in SYSUT1 DD. The DFSERA10 input control statements to retrieve the information from the log look similar to the following statements (in a format that can be used as input to DFSDDLTO):

```
OPTION    PRINT OFFSET=5,VALUE=5F,COND=M
OPTION    PRINT EXITR=DFSERA50,OFFSET=25,FLDTYP=C
          VALUE=DFSIVP37,FLDLLEN=8,DDNAME=OUTDDN,COND=E
```

For example, we executed the SQL statement shown in Example 21-35 against the PERSON table through the metadata DFSIVP37DatabaseView with the DL/I call image capture trace.

Example 21-35 SQL example

```
SELECT * FROM PhoneBook.Person
```

After formatting the IMS log that contains trace data (X'5F' log), we see the trace result shown in Example 21-36.

Example 21-36 DL/I call image capture result

```
U      DATE=2005/180  TIME=15.04.09  DFSIVP37
S 1 1 1 1 1
S 1 1 1 1 1      000011
L      GHU  A1111111
E      DATA LAST6  FIRST6      8-111-6666D06/R06
E 01  A1111111 0010LAST6
L      GHN  A1111111
E      DATA LAST1  FIRST1      8-111-1111D01/R01
E 01  A1111111 0010LAST1
L      GHN  A1111111
```

```

E      DATA LAST2      FIRST2      8-111-2222D02/R02
E 01    A1111111 0010LAST2
L      GHN  A1111111
E      DATA LAST3      FIRST3      8-111-3333D03/R03
E 01    A1111111 0010LAST3
L      GHN  A1111111
E      DATA LAST5      FIRST5      8-111-5555D05/R05
E 01    A1111111 0010LAST5
L      GHN  A1111111
E      DATA LAST4      FIRST4      8-111-4444D04/R04
E 01    A1111111 0010LAST4
L      GHN  A1111111
E 00 GB          0000

```

The trace result shows the following information:

- ▶ The table name Person in the SQL statement is changed into the segment name A1111111 in the SSA, followed by the definition in the metadata class.
- ▶ At the first execution of the DL/I call, the IMS Java engine issued the get hold unique (GHU) call with segment-qualified SSA that has no key value (because, in the SQL statement, we did not specify the WHERE clause), and then got the first segment from IVPDB2.
- ▶ Next, the IMS Java engine issued the GHN call with the same SSA condition until receiving the status GB.
- ▶ The segments from IVPDB2 are not ordered by key value, because IVPDB2 has an HDAM organization.

J2EE information tracing in a distributed environment

You can trace the IMS library classes using the WebSphere Application Server tracing service for both the WebSphere Application Server for z/OS side and WebSphere Application Server for distributed platforms side. You can choose several tracing levels by specifying the custom property value in the J2C connection factory of the resource adapters. In this section, we show the example trace data in the WebSphere Application Server for distributed platforms environment. The following information summarizes how to enable the WebSphere Application Server (for distributed platforms) tracing service.

Specifying the level of tracing

To use the WebSphere Application Server tracing service, you must first specify the level of tracing. The level of tracing is defined as one of the custom property values in the J2C connection factory. Table 21-8 shows the TraceLevel value and its description. You can change the value from the WebSphere Application Server administrative console.

Table 21-8 Trace level in the J2C connection factory

Value	Description
0	No library trace
1	Library exceptions
2	High-level constructors
3	High-level methods
4	High-level parms and return values
5	Middle-level constructors

Value	Description
6	Middle-level methods
7	Middle-level parms and return values
8	Low-level constructors
9	Low-level methods
10	Low-level parms and return values

Specifying the application server and the package to trace

After you specify the level of tracing, you can specify the application server and package to trace. First, you must select the **Enable log** option from the Diagnostic Trace Service menu in the administrative console. Then, you must add the following description in the Change Log Detail Levels menu:

```
com.ibm.ims.rds.*=all=enabled
```

If you add these configurations in the Configuration tab, you must restating your server instance. If you add it in the Runtime tab, you are not required to restart your server instance, but the configuration is not persistent. For more information about enabling the trace, see *IMS Java Guide and Reference*, SC18-7821

An example of a WebSphere trace for IMS RDS

Figure 21-37 shows an example of the WebSphere trace for the IMS Remote Database Services with TraceLevel value 5 (middle-level constructors). You can see the internal logic and communication interaction of the IMS distributed JDBC resource adapter on the WebSphere Application Server for distributed platform side. You can also check some values in the J2C connection factory, your SQL statement, and the returned value.

Example 21-37 WebSphere trace for IMS Remote Database Services with TraceLevel value 5

```
[6/30/05 11:44:54:719 PDT] 00000045 ManagerAdmin I TRAS0018I: The trace state has changed. The new trace state is *=info:com.ibm.ims.rds.*=all.
[6/30/05 11:45:59:963 PDT] 00000045 SystemOut 0 IMS RDS Servlet : Start
[6/30/05 11:46:00:513 PDT] 00000045 ClientManaged 3 -> [ClientManagedConnectionFactory.createConnectionFactory(ConnectionManager)]
[6/30/05 11:46:00:533 PDT] 00000045 ClientManaged 3 <-> [ClientDataSource()]
[6/30/05 11:46:00:533 PDT] 00000045 ClientManaged 3 -- [DataSource: com.ibm.ims.rds.ClientDataSource@1489d997]
[6/30/05 11:46:00:533 PDT] 00000045 ClientManaged 3 <- [ClientManagedConnectionFactory.createConnectionFactory(ConnectionManager)]
[6/30/05 11:46:00:533 PDT] 00000045 SystemOut 0 IMS RDS Servlet : Success Create DataSource
[6/30/05 11:46:00:543 PDT] 00000045 SystemOut 0 IMS RDS Servlet : Success Create UserTransaction
[6/30/05 11:46:00:543 PDT] 00000045 ServletWrapper A SRVE0242I: [ImsRdsSampleGlobal]: Initialization successful.
[6/30/05 11:46:00:624 PDT] 00000045 SystemOut 0 IMS RDS Servlet : Success Start UserTransaction
[6/30/05 11:46:00:624 PDT] 00000045 ClientManaged 3 -> [ClientDataSource.getConnection()]
[6/30/05 11:46:00:634 PDT] 00000045 ClientManaged 3 -> [ClientConnectionRequestInfo(String, String, String, Integer, PrintWriter, int)]
[6/30/05 11:46:00:634 PDT] 00000045 ClientManaged 3 <- [ClientConnectionRequestInfo(String, String, String, Integer, PrintWriter, int)]
[6/30/05 11:46:00:634 PDT] 00000045 ClientManaged 3 -> [ClientManagedConnectionFactory.createManagedConnection(Subject, ConnectionRequestInfo)]
[6/30/05 11:46:00:644 PDT] 00000045 ClientManaged 3 -> [ClientManagedConnectionFactory.createManagedConnection(Subject, ConnectionRequestInfo)]
[6/30/05 11:46:00:644 PDT] 00000045 ClientManaged 3 -- [databaseViewName: imsrds.DFSIVP37DatabaseView]
[6/30/05 11:46:00:644 PDT] 00000045 ClientManaged 3 -- [managedConnectionFactory: com.ibm.ims.rds.ClientManagedConnectionFactory@c2176b9d]
[6/30/05 11:46:00:644 PDT] 00000045 ClientManaged 3 -- [draName: IMSG]
[6/30/05 11:46:00:644 PDT] 00000045 ClientManaged 3 -- [isManagedServer: true]
[6/30/05 11:46:00:644 PDT] 00000045 ClientManaged 3 <- [ClientManagedConnection(String, Subject, ClientManagedConnectionFactory, String, boolean)]
[6/30/05 11:46:00:644 PDT] 00000045 ClientManaged 3 -- [ManagedConnection: com.ibm.ims.rds.ClientManagedConnection@4bd05995]
[6/30/05 11:46:00:644 PDT] 00000045 ClientManaged 3 <- [ClientManagedConnectionFactory.createManagedConnection(Subject, ConnectionRequestInfo)]
[6/30/05 11:46:00:644 PDT] 00000045 ClientManaged 3 -> [ClientManagedConnectionFactory.matchManagedConnection(Set, Subject, ConnectionRequestInfo)]
[6/30/05 11:46:00:644 PDT] 00000045 ClientManaged 3 -- [Matched ManagedConnection: com.ibm.ims.rds.ClientManagedConnection@4bd05995]
[6/30/05 11:46:00:644 PDT] 00000045 ClientManaged 3 <- [ClientManagedConnectionFactory.matchManagedConnection(Set, Subject, ConnectionRequestInfo)]
[6/30/05 11:46:00:684 PDT] 00000045 ClientManaged 3 -> [ClientManagedConnection.getConnection(Subject, ConnectionRequestInfo)]
[6/30/05 11:46:00:704 PDT] 00000045 ClientManaged 3 -> [ClientConnection(ClientManagedConnection, boolean)]
[6/30/05 11:46:00:704 PDT] 00000045 ClientManaged 3 -- [isManagedServer: true]
[6/30/05 11:46:00:704 PDT] 00000045 ClientManaged 3 <- [ClientConnection(ClientManagedConnection, boolean)]
[6/30/05 11:46:00:704 PDT] 00000045 ClientManaged 3 <- [ClientManagedConnection.getConnection(Subject, ConnectionRequestInfo)]
[6/30/05 11:46:00:714 PDT] 00000045 ClientManaged 3 -- [Connection: com.ibm.ims.rds.ClientConnection@17991995]
[6/30/05 11:46:00:714 PDT] 00000045 ClientManaged 3 <- [ClientDataSource.getConnection()]
[6/30/05 11:46:00:714 PDT] 00000045 SystemOut 0 IMS RDS Servlet : Success Get Connection
[6/30/05 11:46:00:714 PDT] 00000045 ClientManaged 3 -> [ClientConnection.createStatement()]
[6/30/05 11:46:00:714 PDT] 00000045 ClientManaged 3 -> [ClientConnection.createStatement(int, int)]
[6/30/05 11:46:00:714 PDT] 00000045 ClientManaged 3 -- [resultSetType: 1003]
[6/30/05 11:46:00:714 PDT] 00000045 ClientManaged 3 -- [resultSetConcurrency: 1007]
[6/30/05 11:46:02:977 PDT] 00000045 ClientManaged 3 -> [ClientStatement(ClientConnection, int, int, PrintWriter, int)]
[6/30/05 11:46:02:977 PDT] 00000045 ClientManaged 3 -- [ClientConnection: com.ibm.ims.rds.ClientConnection@17991995]
[6/30/05 11:46:02:977 PDT] 00000045 ClientManaged 3 -- [resultSetType: 1003]
[6/30/05 11:46:02:977 PDT] 00000045 ClientManaged 3 -- [resultSetConcurrency: 1007]
```

21.5.9 Summary of the IMS RDS implementation

1. Define the data requirement of your business logic to the IMS database, and select the DBD.

2. Create a new PSB, or choose an existing PSB that meets the requirement. Note that:
 - A JDBC connection is made to a PSB. Your PSB needs to have enough DBPCBs for your data requirement to avoid creating a lot of connections in the Web application.
 - All DBPCBs need the PCB label for AIB access.
 - PROCOPT=xP is required for the internal DL/I path call.
3. Execute ACB generation, MODBLKS generation, and add them to the online libraries if needed.
4. Design the relational table structure from the IMS database. You can redefine the IMS PSB/DBD definition much like human-familiar structures, but consider the IMS JDBC characteristics of the logical conversion from a hierarchical database.
5. Implement the relational table design using the DLIModel utility.
6. Prepare the IMS environment. Define the DRA startup table parameter values for your requirement, and create the DRA module. For example, the DRA MAXTHRD parameter should have the following number:

The number of MAXTHRD >= The number of the J2C connection from WebSphere Application Server for z/OS >= The number of the J2C connection from WebSphere Application Server for distributed

In addition, consider the FP buffer definitions in the DRA startup table carefully, because your SQL request might require a lot of FP resources that depend on your query conditions. Set the IMS execution parameter RRS= to Y.

7. Configure the WebSphere Application Server for z/OS environment by:
 - Concatenating the IMS libraries to the servant regions
 - Installing the IMS JDBC resource adapter
 - Installing the custom service
 - Installing the data source for IMS Java EJB by defining the J2C connection factory
 - Adding the classpath to the metadata class
 - Installing the IMS Java EJB
8. Configure the WebSphere Application Server for distributed platforms environment by:
 - Installing the IMS distributed JDBC resource adapter
 - Installing the data source for the target IMS JDBC connection by defining the J2C connection factory
9. Design and develop your Web application. Consider the following information:
 - Transaction semantics

You should understand your transactional requirement (include your XA resource updates within the single transaction boundary or not) and how the transaction semantics affects commit and rollbacks operations. Generally, the global transaction has larger system costs than the local transaction, and it might affect Web application performance and IMS resource occupancy. This also might be the trade-off between the transactional requirement and the performance.
 - Security semantics

There are three areas to consider for security:

 - Access to the client-side Web application: Deploy the client-side Web application with the run-as deployment property set to system. Restrict access to the client-side Web application.
 - Network security: You can use identity assertion or SSL to secure the network communication between the two application servers.

- Security between WebSphere Application Server for z/OS and IMS: ODBA requires a preverified Access Control Environment Element (ACEE), which WebSphere Application Server for z/OS places on the execution thread.

– IMS Java JDBC/SQL implementation

JDBC programmers should know the characteristics of the IMS table that comes from the metadata definition and consider the special requirement of the IMS Java SQL statement and JDBC interface. Be careful that the SQL API provides for the querying data contained in a relational model, but, in fact, you are accessing the IMS database by using DL/I API implicitly. For example:

- If you supply a predicate in the WHERE clause for a target segment somewhere down the hierarchy and omit predicates for its parents, IMS must scan all candidate segments at the parent levels in an attempt to match the predicate that you supplied. Depending on the query condition, it can easily cause a full-database scan.
- The Resultset.next method on the client side will invoke the next single DL/I call (GHN) request in the IMS Java EJB on z/OS side, until either a GE or GB status code is returned in the AIB, indicating there are no more segments to process. In the IMS RDS environment, this means that if your query condition meets 10000 segments in the IMS database, the interaction, including network transportation between the IMS Java EJB on the z/OS side and your Web application on the distributed side, will occur 10,000 times within one SQL process.

For better performance, the IMS JDBC API has been enhanced to provide support for additional methods that address the access issues from distributed environments. Specifically, the setFetchSize methods on both the Statement and ResultSet APIs are supported. The fetch size indicates the amount of rows to fetch from the IMS database for a call to ResultSet.next(). For example, if you set the fetch size to 50, the first call to next() retrieves the first 50 rows satisfying the query and stores them in the WebSphere Application Server for distributed platforms address space. This way, the next 49 calls to next() will not go across the network because the data is on the client side. After the client requests the fifty-first result, the call then goes across the network and returns the next 50 rows in one network call.

Additionally, in the case of scroll-insensitive result sets (or queries that use aggregate functions), all of the data is gathered from the IMS database on the first request and stored on the WebSphere Application Server for z/OS side, up to the MAX_ROWS limit specified by the client application. Then, the same thing happens as described previously; for each call to next(), the specified number of rows are returned from the server side, but no further DL/I calls are made to the IMS database.

10. Deploy and test your Web application in the test environment. We recommend that you check not only the expected process of your Web application, but also monitor the resource usage and performance of WebSphere Application Server for distributed platforms, WebSphere Application Server for z/OS, RRS, and, of course, IMS database manager.
11. Release your Web application to the production environment. Keep monitoring the performance and resource usage in the environment.

Archived

Sample code: Non-IMS Connector for Java client code

This appendix contains the complete sample programs that we use throughout this book. You can download the source code from the following FTP address:

<ftp://www.redbooks.ibm.com/redbooks/SG246794>

Note: The code examples here are provided “AS-IS” and there are certain conditions for using them. Before using the samples, read the information in “Notices” on page xi.

We provide an example of a basic client written in the C and Java programming languages. Both versions are functionally equivalent and can be used as a base to implement more complex features. In 14.5, “Detailed code examples” on page 282, we cover the use of this code.

C sample source code

Example A-1 provides the C sample source code.

Example: A-1 Sample C code for an IMS Connect client

```
/* **** */
/* sample.c - IMS Connector basic client written in C */
/* ----- */
/* See the redbook IMS V9 Connectivity Update - SG24-6794 for details */
/* and terms & conditions of use. */
/* */
/* Copyright (c) - International Business Machines, Inc. 2005,2001 */
/* */
/* Author: Jordi Guillaumes Pons and others */
/* */
/* **** */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <netdb.h>
#include <getopt.h>
#include <limits.h>
#include <netinet/in.h>
#include <sys/socket.h>

#define PREFIX_LENGTH 96
#define MAX_SEGMENTS 100

#define BLANK8 " "

/*
 * Symbolic constants
 */
#define CM0 0x40
#define CM1 0x20
#define SL_NONE 0x00
#define SL_CONFIRM 0x01
#define SL_SYNCPT 0x02
#define SL_PURGE 0x04
#define SL_REROUTE 0x08
#define SL_PURGE_NOT_DELIVERABLE 0x04

/*
 * Structure to hold all the data about the interaction
 * we are going to have with our host.
 */
struct sampletran {
    /* IMS information */
    char * tranCode;           /* Transaction code */
    char * tranText;           /* Transaction data */
    char * dataStoreID;        /* Datastore (IMS) id */
    char * ltermName;          /* LTERM override - optional */
    char ** response;          /* Response segments */
    /* RACF security information */
    char * racfUserID;         /* RACF User Id - optional */
    char * racfGroupName;     /* RACF Group name - optional */
    char * password;           /* RACF Password - optional */
};
```

```

/* IMS Connect information */
char * hostName;          /* Host name or IP address in dot format */
int  portNumber;          /* Port number where IMS Connect listens */
char * clientID;          /* Unique Client-Id - optional */
char * exitID;            /* Exit Id - Set to *SAMPLE* */
char syncLevel;           /* Sync Level (encoded) */
char commitMode;          /* Commit Mode (encoded) */
int  sendOnly;            /* 1: Use a send-only interaction */
int  resumeTpipe;         /* 1: Issue a RESUME TPIPE */
int  ackRequired;         /* ACK required by IMS Connect (output field) */
int  nakRequired;         /* Send a NAK if possible and required */
char timer;               /* Value for IRM_TIMER */
char reroute;             /* Reroute not deliverable */
char purge;               /* Purge not deliverable */
char *rerouteName;        /* Optional reroute destination name */
};

/*****
/* Utility functions */
*****/

/*
 * Copy size bytes into bufPtr and return the first address available after copied data
 *
 * Parameters:
 * bufPtr  Destination buffer address.
 * src     Source string address.
 * size    Number of characters to copy.
 *
 * Returns:
 * Address of the next available byte in the buffer.
 *
 * Warning: this function DOES NOT check, nor has any way to do it, if the destination
 * buffer has enough space available. DO NOT USE THIS CODE IN PRODUCTION.
 * There is a buffer-overflow risk, which could lead to a security problem.
 */
char *addBuffer(void *bufPtr, void *src, size_t size) {
    memcpy(bufPtr, src, size);
    bufPtr += size;
    return bufPtr;
}

/*
 * Copy a string from src to dst, up to size chars.
 * If the length of src is less than size, then pad dst with the
 * specifies padding char.
 *
 * Parameters:
 * dst     Destination buffer address.
 * src     Source string address.
 * size    Destination buffer size.
 * padding Padding character (as int).
 */
void padCopy(char *dst, char *src, size_t size, int padding) {
    int i=0;
    int len = strlen(src);
    if (len > size) len = size;

    for(i=0;i<len;i++) dst[i] = src[i];
    for(;i<size;i++) dst[i] = padding;
}

```

```

}

void freeResponse(struct sampletran *sample) {
    int i=0;

    /* Free the response segments */
    if (sample->response != NULL) {
        for(i=0; i<MAX_SEGMENTS && sample->response[i] != NULL; i++) {
            free(sample->response[i]);
        }
        free(sample->response);
    }
}

/*
 * Free the dynamically allocated memory corresponding to
 * one instance of sampletran.
 *
 * Parameters:
 * sample    Address of the sampletran structure to be freed.
 */
void freeSampleTran(struct sampletran *sample) {

    /* Free the response segments */
    freeResponse(sample);

    /* Free the sample parameters */
    free(sample->datastoreID);
    free(sample->racfUserID);
    free(sample->racfGroupName);
    free(sample->password);
    free(sample->clientID);
    free(sample->ltermName);
    free(sample->tranCode);
    free(sample->tranText);
    free(sample->rerouteName);
}

/*****
 * Communications setup functions
 *****/

/*
 * Connects to the host and returns a socket descriptor if successful
 *
 * Parameters:
 * sample    Address of the sampletran structure with the communication parameters.
 *
 * Returns:
 *          A connected socket descriptor.
 */
int sample_connect(struct sampletran *sample) {
    int sockfd;
    struct hostent * host;
    struct sockaddr_in socketAddress;

    /* get host info */
    if ((host = gethostbyname(sample->hostName)) == NULL) {
        perror("gethostbyname");
    }
}

```

```

        exit(1);
    }

    /* initialize the socket descriptor */
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    /* set some socket address values */
    socketAddress.sin_family = AF_INET;
    socketAddress.sin_port = htons(sample->portNumber); /* network byte order */
    socketAddress.sin_addr = *((struct in_addr *)host->h_addr);
    memset(&(socketAddress.sin_zero), 0, 8); /* zero out the rest of the struct */

    /* connect the socket */
    if (connect(sockfd, (struct sockaddr *)&socketAddress, sizeof(struct sockaddr)) == -1)
    {
        perror("connect");
        exit(1);
    }

    /* return the socket descriptor */
    return sockfd;
}

/*
 * Closes the socket
 *
 * Parameters:
 * sockfd    Socket descriptor to close.
 */
void sample_disconnect(int sockfd) {
    /* close the socket */
    close(sockfd);
}

/*****
 * User messages
 *****/

/*
 * Shows the correct syntax and prints an error message
 */
void syntaxError(char *message) {
    fprintf(stderr, "%s\n", message);

    fprintf(stderr, "Syntax:\n");
    fprintf(stderr, "SEND          : sample -h hostName -p portNumber [-d datastoreName]\n");
    fprintf(stderr, "                [-c clientId] [-t ltermName]\n");
    fprintf(stderr, "                [-u userId [-g groupName] [-w password]]\n");
    fprintf(stderr, "                [-x] [-y [rerouteName]]\n");
    fprintf(stderr, "                [-m {0|1}] [-l {N|C|S}] [-s] [-n] truncate
trandata...\n");
    fprintf(stderr, "RESUME TPIPE: sample -h hostName -p portNumber [-d datastoreName]\n");
    fprintf(stderr, "                -c clientId -r [-n]\n");
    fprintf(stderr, "                [-u userId [-g groupName] [-w password]] \n");
    fprintf(stderr, "The -m flag sets the commit mode to 0 or 1 on SEND interactions.\n");
    fprintf(stderr, "The -x flag enables the purge not deliverable feature.\n");
}

```

```

        fprintf(stderr, "The -y flag enables the reroute not deliverable feature. You can
specify an optional\n");
        fprintf(stderr, "        reroute destination name.\n");
        fprintf(stderr, "The -l flag sets the Sync Level to N(one), C(onfirm) or S(ynch). on
SEND interactions.\n");
        fprintf(stderr, "The -n flag forces a NAK, if the Sync Level is not NONE.\n");
        fprintf(stderr, "The -s flag sets the interaction as SEND ONLY\n");
        fprintf(stderr, "The Datastore Id (-d flag) should be specified unless your IMS Connect
exits take care of setting it.\n");
        exit(8);
    }

/*
 * Prints the content of the sampletran structure to stdout
 */
void listSampleTran(struct sampletran *sample) {
    printf("IMS Connect information: \n");
    printf("\tClient ID : \t%s\n", sample->clientID);
    printf("\textitID   : \t%s\n", sample->exitID);
    printf("\tsyncLevel : \t");
    switch(sample->syncLevel) {
        case SL_NONE:
            printf("NONE\n");
            break;
        case SL_CONFIRM:
            printf("CONFIRM\n");
            break;
        case SL_SYNCPT:
            printf("SYNCPOINT\n");
            break;
        default:
            printf("UNKNOWN\n");
            break;
    }
    printf("\tcommitMode: \t");
    switch(sample->commitMode) {
        case CM0:
            printf("0 - COMMIT THEN SEND\n");
            break;
        case CM1:
            printf("1 - SEND THEN COMMIT\n");
            break;
        default:
            printf("UNKNOWN!\n");
            break;
    }

    if (sample->purge == 1) {
        printf("\tPURGE NOT DELIVERABLE\n");
    }

    if (sample->reroute == 1) {
        printf("\tREROUTE NOT DELIVERABLE\n");
        printf("\t\t%s\n", sample->rerouteName);
    }

    if (sample->sendOnly == 1) {
        printf("\tSEND ONLY\n");
    } else {
        printf("\tSEND-RECEIVE\n");
    }
}

```

```

        if (sample->resumeTpipe == 1) {
            printf("\tRESUME TPIPE\n");
        }
        printf("IMS information:\n");
        printf("\tDatastore : \t%s\n", sample->datastoreID);
        printf("\tLTerm      : \t%s\n", sample->ltermName);
        if (sample->resumeTpipe == 0) {
            printf("Transaction information:\n");
            printf("\tCode       : \t%s\n", sample->tranCode);
            printf("\tText        : \t%s\n", sample->tranText);
        }
    }

    /*****
    /* Send and receive functions
    *****/

    /*
    * Sends the input data to the host
    *
    * Parameters:
    * sockfd    Connected socket descriptor
    * sample    Address of a sampletran structure with the interaction data
    * msgType   Value of IRM_F4 (' ', 'A', 'R')
    *
    * Notice: This code has been modified to send all the data to IMS Connect in a
    * single write to enhance performance. The old IMS Connect sample does a write for
    * each field. YOU SHOULD NOT DO THAT. Build first the whole message in memory and
    * send it in one call.
    */

void sample_send(int sockfd, struct sampletran *sample, char msgType) {
    int totalLength, totalLengthBE;
    short segmentLength;
    short prefixLength = PREFIX_LENGTH;
    char irm_f3 = (char) 0;
    char irm_arch = (char) 0x01; /* Arch level 1 to use Reroute Name */
    char *message = NULL;
    char *currPtr = NULL;
    int zero = 0; /* need this so we can pass a pointer to zero */

    /* +4 for first LL, ZZ and final LL, ZZ */
    totalLength = 4 + PREFIX_LENGTH + 4;

    /* add in segment length, if segment is defined */
    if (sample->tranText != NULL) {
        totalLength += strlen(sample->tranText) + 12; /* +12 for LL, ZZ, tranCode */
    }

    /* Compute the IRM_F3 value */
    irm_f3 = sample->syncLevel;
    if (sample->purge) {
        irm_f3 |= SL_PURGE;
    }
    if (sample->reroute) {
        irm_f3 |= SL_REROUTE;
    }

    message = malloc(totalLength);
    if (message == NULL) {

```

```

        perror("Could not allocate memory for the whole message.");
        exit(32);
    }
    memset(message, 0, totalLength); /* Clean up the new allocated space */
    currPtr = message; /* Current write pointer set to beginning of message space */

    /* convert lengths to big endian */
    totalLengthBE = htonl(totalLength);
    prefixLength = htons(prefixLength);

    /* Build the message structure in the allocated buffer */
    /* Build the IRM prefix first */
    currPtr = addBuffer(currPtr, &totalLengthBE, 4); /* Total message length */
    currPtr = addBuffer(currPtr, &prefixLength, 2); /* IRM_LL */
    currPtr = addBuffer(currPtr, &irm_arch, 1); /* IRM_ARCH */
    currPtr = addBuffer(currPtr, &zero, 1); /* IRM_F0 */
    currPtr = addBuffer(currPtr, sample->exitID, 8); /* IRM_ID */
    currPtr = addBuffer(currPtr, &zero, 4); /* IRM_RES */
    currPtr = addBuffer(currPtr, &zero, 1); /* IRM_F5 - No option flow */
    currPtr = addBuffer(currPtr, &sample->timer, 1); /* IRM_TIMER */
    currPtr = addBuffer(currPtr, &zero, 1); /* IRM_SOCT - Transaction socket
*/
    currPtr = addBuffer(currPtr, &zero, 1); /* IRM_ES */
    currPtr = addBuffer(currPtr, sample->clientID, 8); /* IRM_CLIENTID */
    currPtr = addBuffer(currPtr, &zero, 1); /* IRM_F1 - No MODNAME request */
    currPtr = addBuffer(currPtr, &sample->commitMode, 1); /* IRM_F2 - Set commit mode */
    currPtr = addBuffer(currPtr, &irm_f3, 1); /* IRM_F3 - Set sync level... */
    currPtr = addBuffer(currPtr, &msgType, 1); /* IRM_F4 - Set message type */
    currPtr = addBuffer(currPtr, sample->tranCode, 8); /* IRM_TRNCOD */
    currPtr = addBuffer(currPtr, sample->datastoreID, 8); /* IRM_IMSDESTID */
    currPtr = addBuffer(currPtr, sample->ltermName, 8); /* IRM_LTERM */
    currPtr = addBuffer(currPtr, sample->racfUserID, 8); /* IRM_RACF_USERID */
    currPtr = addBuffer(currPtr, sample->racfGroupName, 8); /* IRM_RACF_GRPNAME */
    currPtr = addBuffer(currPtr, sample->password, 8); /* IRM_RACF_PW */
    currPtr = addBuffer(currPtr, BLANK8, 8); /* IRM_APPL_NM */
    currPtr = addBuffer(currPtr, sample->rerouteName, 8); /* IRM_REROUT_NM */

    /* Add the transaction segment (trancode + tranText) just for IRM_F4 = ' ' or 'S' */
    if (msgType == ' ' || msgType == 'S') {
        /* + 12 for LL and ZZ and tranCode */
        segmentLength = (short) (strlen(sample->tranText) + 12);
        /* convert to big endian */
        segmentLength = htons(segmentLength);
        currPtr = addBuffer(currPtr, &segmentLength, sizeof(short)); /* Transaction LL */
        currPtr = addBuffer(currPtr, &zero, sizeof(short)); /* Transaction ZZ */
        currPtr = addBuffer(currPtr, sample->tranCode, 8); /* Transaction code
*/
        currPtr = addBuffer(currPtr, sample->tranText, strlen(sample->tranText)); /* data
*/
    }

    /* send final LL ZZ to signal no more data to IMS Connect */
    segmentLength = 4;
    /* convert to big endian */
    segmentLength = htons(segmentLength);
    currPtr = addBuffer(currPtr, &segmentLength, sizeof(short)); /* End of message LL */
    currPtr = addBuffer(currPtr, &zero, sizeof(short)); /* End of message zeros
*/

```

```

        /* Send the built message to IMS Connect and free the malloc() */
        send(sockfd, message, totalLength, 0);
        free(message);
    }

    /*
     * Receive the data from the host, and parses it into segment strings
     * We are using HWSSMPL0, so we don't have a total-length prefix.
     * If we were using HWSSMPL1 we should take that into account.
     *
     * Parameters:
     * sockfd    Connected socket descriptor
     * sample    Address of a sampletran structure with the interaction data
     *
     * Warning: if you are adapting this code to be used in a production
     * program, please notice that this function DOES NOT free the heap
     * memory it allocates, namely the array sample->response and its
     * components.
     */
    void sample_receive(int sockfd, struct sampletran *sample) {
        int totalLength, recordLength, segmentIndex, returnCode, reasonCode;
        char xsm_flg;
        char reserved;
        char * * segments;
        char * identifier;
        char * modName;
        char * segment;
        int done = 0;

        sample->nakRequired = 0;
        sample->ackRequired = 0;
        /*
         * malloc() up the segment array
         */
        segments = (char * *) malloc(sizeof(char *) * MAX_SEGMENTS);
        for (segmentIndex = 0; segmentIndex < MAX_SEGMENTS; segmentIndex++)
            segments[segmentIndex] = NULL;
        segmentIndex = 0;

        /* read total length */
        recv(sockfd, &totalLength, sizeof(short), 0);
        /* read flags byte */
        recv(sockfd, &xsm_flg, sizeof(char), 0);
        /* read reserved byte */
        recv(sockfd, &reserved, sizeof(char), 0);

        /* convert total length from big endian */
        totalLength = ntohs(totalLength);

        /* read identifier (exitID) */
        if (totalLength < 12) {
            identifier = (char *) malloc(sizeof(char) * (totalLength - 4) + 1);
            memset(identifier, 0, sizeof(char) * (totalLength - 4) + 1);
            recv(sockfd, identifier, totalLength - 4, 0);
        } else {
            identifier = (char *) malloc(sizeof(char) * 8 + 1);
            memset(identifier, 0, sizeof(char) * 8 + 1);
            recv(sockfd, identifier, 8, 0);
        }
    }

```

```

/* check first segment for possible errors / alerts */
if (strcmp(identifier, "**REQMOD**") == 0) {
    /* read mod name */
    modName = (char *) malloc(sizeof(char) * 8);
    recv(sockfd, modName, 8, 0);
    /* add mod name to segment array */
    segments[segmentIndex++] = modName;
} else if (strcmp(identifier, "**REQSTS**") == 0) {
    /* read return code and reason code */
    recv(sockfd, &returnCode, sizeof(int), 0);
    recv(sockfd, &reasonCode, sizeof(int), 0);
    /* convert them from big endian */
    returnCode = ntohl(returnCode);
    reasonCode = ntohl(reasonCode);
    /* add them to the segment array */
    segments[segmentIndex] = (char *) malloc(sizeof(char) * 20);
    sprintf(segments[segmentIndex++], "RETURN CODE: %i", returnCode);
    segments[segmentIndex] = (char *) malloc(sizeof(char) * 20);
    sprintf(segments[segmentIndex++], "REASON CODE: %i", reasonCode);
    done = 1;
    /* return since there should be no more data */
    /* return segments; */
} else {
    if (totalLength <= 12)
        segments[segmentIndex++] = identifier;
    else {
        /* read in the rest of the segment data */
        segment = (char *) malloc(sizeof(char) * (totalLength - 12) + 1);
        /* + 1 for trailing zero */
        memset(segment, 0, sizeof(char) * (totalLength - 12) + 1); /* Clear buffer */
        recv(sockfd, segment, sizeof(char) * (totalLength - 12), 0);
        segments[segmentIndex] = (char *) malloc(sizeof(char) * (totalLength - 4));
        sprintf(segments[segmentIndex++], "%s%s", identifier, segment);
    }
}

/* continue trying to read in data till we come across *CSMOKY* */
if (done == 0) {
    while ((strcmp(segment, "**CSMOKY**") != 0) && (done == 0)) {
        /* read next segment */
        /* read LL */
        recv(sockfd, &recordLength, sizeof(short), 0);
        /* read ZZ */

        /* read flags byte */
        recv(sockfd, &xsm_flg, sizeof(char), 0);
        /* read reserved byte */
        recv(sockfd, &reserved, sizeof(char), 0);

        /* convert record length from big endian */
        recordLength = ntohs(recordLength);
        /* read in segment data */
        segment = (char *) malloc(sizeof(char) * (recordLength - 4) + 1);
        /* + 1 for trailing zero */
        memset(segment, 0, sizeof(char) * (recordLength - 4) + 1); /* Clear buffer */
        recv(sockfd, segment, sizeof(char) * (recordLength - 4), 0);
        /* add it to the segment vector */
        segments[segmentIndex++] = segment;
        if (segmentIndex >= MAX_SEGMENTS) {
            fprintf(stderr, "Maximum number of segments exceeded.\n");
        }
    }
}

```

```

        done = 1;
        sample->nakRequired = 1;
    }
}
}
if (sample->nakRequired == 0) {
    /* Now we have reached the CSM or the RSM */
    /* Check if ACK required */
    if (xsm_flg & 0x20) {
        sample->ackRequired = 1;
    } else {
        sample->ackRequired = 0;
    }
}
sample->response = segments;
free(identifier);
}

/*****
/* main function */
*****/
/*
 * We use the getopt() function to parse the command-line arguments. This
 * function is part of the POSIX.2 specification, and should be present
 * in any reasonably compatible UNIX system. It's on Linux, on *BSDs, on
 * Mac OS X, in AIX and in USS. In case your system does not have
 * a getopt(), it should be easy to overcome the limitation, since
 * the command-parsing is quite trivial.
 */

int main(int argc, char **argv) {
    int sockfd, i;
    int seglen = 0;
    int forceNAK = 0;
    int ackNakSent = 0;
    struct sampletran sample;
    char * segment = NULL;
    int ch;
    unsigned int timerInt=0;

    /* Initialize the sample structure to zeroes */
    memset(&sample, 0, sizeof(sample));
    sample.commitMode = CM1; /* Default: commit mode 1 */
    sample.syncLevel = SL_CONFIRM; /* Default: Sync Level CONFIRM */
    sample.resumeTpipe = 0; /* Default: no RESUME TPIPE */
    sample.sendOnly = 0; /* Default: no SEND ONLY */
    sample.exitID = "**SAMPLE**"; /* Use sample EXIT */
    sample.timer = 0; /* User ICONN default value */

    /* Allocate storage for the sample structure items */
    /* In a real-world application we should check if the malloc() succeeded! */
    sample.datastoreID = malloc(9); strcpy(sample.datastoreID, BLANK8);
    sample.racfUserID = malloc(9); strcpy(sample.racfUserID, BLANK8);
    sample.racfGroupName = malloc(9); strcpy(sample.racfGroupName, BLANK8);
    sample.password = malloc(9); strcpy(sample.password, BLANK8);
    sample.clientID = malloc(9); strcpy(sample.clientID, BLANK8);
    sample.ltermName = malloc(9); strcpy(sample.ltermName, BLANK8);
    sample.tranCode = malloc(9); strcpy(sample.tranCode, BLANK8);
    sample.rerouteName = malloc(9); strcpy(sample.rerouteName, BLANK8);

```

```

/* Loop through the command line arguments, and fill up the sample structure and
some other variables */
while((ch=getopt(argc,argv,"h:p:d:u:g:w:c:t:l:m:o:rsnxy:")) != -1) {
switch(ch) {
case 'h':
    sample.hostName = optarg;
    break;
case 'p':
    sample.portNumber = atoi(optarg);
    break;
case 'd':
    padCopy(sample.datastoreID, optarg, 8, ' ');
    break;
case 'u':
    padCopy(sample.racfUserID, optarg, 8, ' ');
    break;
case 'g':
    padCopy(sample.racfGroupName, optarg, 8, ' ');
    break;
case 'w':
    padCopy(sample.password, optarg, 8, ' ');
    break;
case 'c':
    padCopy(sample.clientID, optarg, 8, ' ');
    break;
case 't':
    padCopy(sample.ltermName, optarg, 8, ' ');
    break;
case 'l':
    if (strncmp(optarg,"N",1) == 0) {
        sample.syncLevel = SL_NONE;
    } else if (strncmp(optarg,"C",1) == 0) {
        sample.syncLevel = SL_CONFIRM;
    } else if (strncmp(optarg,"S",1) == 0) {
        sample.syncLevel = SL_SYNCPT;
    } else {
        syntaxError("Sync level must be N, C or S. The default value is C.");
    }
    break;
case 'm':
    if (strncmp(optarg,"0",1) == 0) {
        sample.commitMode = CM0;
    } else if (strncmp(optarg,"1",1) == 0) {
        sample.commitMode = CM1;
    } else {
        syntaxError("Commit mode must be 0 or 1. The default value is 1.");
    }
    break;
case 'o':
    i = sscanf(optarg,"%x",&timerInt);
    if (timerInt > UCHAR_MAX || i == 0) {
        syntaxError("The timer value must be a hexadecimal byte.");
    } else {
        sample.timer = (unsigned char) timerInt;
    }
    break;
case 'r':
    sample.resumeTpipe = 1;
    sample.commitMode = CM0;
}
}

```

```

        break;
    case 's':
        sample.sendOnly = 1;
        sample.commitMode = CM0;
        break;
    case 'n':
        forceNAK = 1;
        break;
    case 'x':
        sample.purge = 1;
        break;
    case 'y':
        sample.reroute = 1;
        if (optarg[0] == '-') {
            optind--; // Back up one pos in arg list
        } else {
            padCopy(sample.rerouteName, optarg, 8, ' ');
        }
        break;
    default:
        break;
}
}
argc -= optind;
argv += optind;

/*
    At this point argc contains the number of non-parsed arguments, and
    argv points to the first one of those arguments.

    The first one should be the transaction code, and the rest the transaction text.
*/

/* We can have a trancode, or --resumetpipe, but not both */
if (argc < 1 && sample.resumeTpipe == 0)
    syntaxError("Should specify at least the transaction code.");
else if (argc > 0 && sample.resumeTpipe == 1)
    syntaxError("Transaction data not allowed with -r.");

/* Check the coherence of the command line args */
if (memcmp(sample.hostName, BLANK8, 8) == 0 ||
    sample.portNumber == 0) {
    syntaxError("You must specify host name and port number.");
}

if (sample.resumeTpipe == 1) {
    if (memcmp(sample.ltermName, BLANK8, 8) != 0 ||
        sample.sendOnly != 0 ||
        sample.commitMode != CM0) {
        syntaxError("Lterm, commit mode and SEND ONLY are not allowed with -r.");
    }
    if (sample.sendOnly == 1) {
        if (sample.commitMode != CM0) {
            syntaxError("-s (SEND ONLY) requires commit mode 0.");
        }
    }
}

/*
    * Check the coherence between SYNC LEVEL and COMMIT MODE.

```

```

    */
    if (sample.commitMode == CMO && sample.syncLevel != SL_CONFIRM) {
        syntaxError("Commit mode 0 requires Synclevel CONFIRM.");
    }

    /*
    * Check if NAK can be forced
    */
    if (forceNAK != 0 && sample.syncLevel == SL_NONE) {
        fprintf(stderr, "The -n flag (force NAK) will be ignored (sync level = NONE).\n");
    }

    /* Let's parse the rest of the command line arguments */
    if (argc > 0) {
        if (strlen(argv[0]) > 8) {
            syntaxError("The transaction code is longer than 8 characters.");
        } else {
            padCopy(sample.tranCode, argv[0], 8, ' ');
        }
        /* compute the maximum segment length we need to send our transaction */
        seglen = 0;
        for (i=1; i<argc; i++) {
            seglen += strlen(argv[i]); /* Add next argument size */
            seglen++; /* Add space for one blank or trailing zero*/
        }
        /* Allocate memory for our segment, and zero-init it */
        segment = malloc(seglen);
        memset(segment, 0, seglen);

        /* Concatenate the arguments after trancode into our segment */
        for (i=1; i<(argc-1); i++) {
            strlcat(segment, argv[i], seglen);
            strlcat(segment, " ", seglen);
        }
        strlcat(segment, argv[argc-1], seglen); /* Add last argument */

        /* Now we have our transaction text, and we are ready to proceed */
        sample.tranText = segment;
    }

    listSampleTran(&sample);

    printf("Connecting to the host...\n");
    sockfd = sample_connect(&sample);

    if (sample.resumeTpipe == 1) {
        printf("Sending RESUME TPIPE...\n");
        sample_send(sockfd, &sample, 'R');
    } else {
        printf("Sending input data...\n");
        if (sample.sendOnly == 0) {
            sample_send(sockfd, &sample, ' ');
        } else {
            sample_send(sockfd, &sample, 'S');
        }
    }

    if (sample.sendOnly == 0) {
        printf("Receiving response...\n");
    }

```

```

        sample_receive(sockfd, &sample);

        /* print out the segment results */
        i = 0;
        while (i < MAX_SEGMENTS && sample.response[i] != NULL) {
            printf("Segment %i: %s\n", i, sample.response[i]);
            i++;
        }
    }

    /* Send NAK, if required */
    if (sample.syncLevel != SL_NONE) {
        if (sample.nakRequired != 0 || forceNAK != 0) {
            printf("Sending NAK...\n");
            sample.timer = 0xE9;
            sample_send(sockfd, &sample, 'N');
            ackNakSent = 1;
        }
    }

    /* Send ACK, if required */
    if (sample.ackRequired != 0 && forceNAK == 0) {
        printf("Sending ACK...\n");
        sample.timer = 0xE9;
        sample_send(sockfd, &sample, 'A');
        ackNakSent = 1;
    }

    if (ackNakSent == 1) {
        printf("Waiting for ACK/NAK response...\n");
        freeResponse(&sample); /* Free current response segments */
        sample_receive(sockfd, &sample);
        /* print out the segment results */
        i = 0;
        while (i < MAX_SEGMENTS && sample.response[i] != NULL) {
            printf("Segment %i: %s\n", i, sample.response[i]);
            i++;
        }
    }

    printf("Disconnecting from the host...\n");
    sample_disconnect(sockfd);

    freeSampleTran(&sample);

    return(0);
}

```

Java sample source code

Example A-2 provides the Java sample source code.

Example: A-2 Sample Java code for an IMS Connect client (non-IMS Connector for Java)

```
import java.io.ByteArrayOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.PrintWriter;
import java.io.StringWriter;
import java.net.Socket;
import java.util.Collection;
import java.util.Iterator;
import java.util.Vector;

/**
 * IMS Connector sample client written in Java
 * This class implements a basic IMS Connect client using the TCP/IP
 * sockets interface.
 *
 * @author Jordi Guillaumes Pons and others
 */
public class Sample {

    private Socket socket = null;

    // connection information
    private String hostName;
    private int portNumber;

    // IMS information
    private String tranCode;
    private String tranText;
    private String dataStoreID;
    private String ltermName;
    private Collection response;

    // RACF security information
    private String racfUserID;
    private String racfGroupName;
    private String password;

    // IMS Connect information
    private String clientID;
    private String exitID;
    private byte syncLevel;
    private byte commitMode;
    private boolean sendOnly;
    private boolean resumeTpipe;
    private boolean ackRequired;
    private boolean nakRequired;
    private int prefixLength = 96; // With IRM_ARCH = 0x01
    private byte timer;
    private boolean reRoute;
    private boolean purgeAsync;
    private String reRouteName;
    private boolean RSM = false; // Response is RSM
    private int returnCode = 0;
```

```

private int reasonCode = 0;

public static final byte CM0 = (byte) 0x40;
public static final byte CM1 = (byte) 0x20;
public static final byte SL_NONE = (byte) 0;
public static final byte SL_CONFIRM = (byte) 0x01;
public static final byte SL_SYNCPT = (byte) 0x02;
public static final byte SL_PURGE = (byte) 0x04;
public static final byte SL_REROUTE = (byte) 0x08;
public static final byte IRM_F5_NONE = (byte) 0x00;
public static final byte IRM_F5_SINGLE = (byte) 0x01;
public static final byte IRM_F5_AUTO = (byte) 0x02;
public static final byte IRM_F5_NOAUTO = (byte) 0x04;

//////////////////////////////////////
// Constructor                                     //
//////////////////////////////////////

/**
 * Constructor for a Sample object.
 *
 * @param hostName Host name or IP address in dot format
 * @param portNumber Port number which IMS Connect listens
 * @param dataStoreID Name of the IMS System we want to send transactions
 * @param ltermName LTERM override
 * @param tranCode Transaction code
 * @param tranText Transaction text
 * @param clientID Unique client identification
 * @param racfUserID RACF User id
 * @param racfGroupName RACF Group name
 * @param password RACF password
 * @param syncLevel Sync Level coded value
 * @param commitMode Commit Mode coded value
 * @param sendOnly This interaction will be send only
 * @param resumeTpipe This interaction is a RESUME TPIPE
 */
public Sample(
    String hostName,
    int portNumber,
    String dataStoreID,
    String ltermName,
    String tranCode,
    String tranText,
    String clientID,
    String racfUserID,
    String racfGroupName,
    String password,
    byte syncLevel,
    byte commitMode,
    boolean sendOnly,
    boolean resumeTpipe,
    byte timer,
    boolean purge,
    boolean reroute,
    String rerouteName) {
    // set the corresponding transaction data, making all strings 8
    // characters long
    this.hostName = hostName;
    this.portNumber = portNumber;
    this.dataStoreID = stringPad(dataStoreID, ' ', 8);

```

```

        this.ltermName = stringPad(ltermName, ' ', 8);
        this.tranCode = stringPad(tranCode, ' ', 8);
        this.tranText = tranText;
        this.clientID = stringPad(clientID, ' ', 8);
        this.racfUserID = stringPad(racfUserID, ' ', 8);
        this.racfGroupName = stringPad(racfGroupName, ' ', 8);
        this.password = stringPad(password, ' ', 8);
        this.reRouteName = stringPad(rerouteName, ' ', 8);

        // Set the interaction parameters
        this.syncLevel = syncLevel;
        this.commitMode = commitMode;
        this.sendOnly = sendOnly;
        this.resumeTpipe = resumeTpipe;
        this.timer = timer;
        this.purgeAsync = purge;
        this.reRoute = reroute;

        // We will use HWSSMPL0
        this.exitID = "*SAMPLE*";
    }

    //////////////////////////////////////
    // Communications setup methods                                     //
    //////////////////////////////////////

    /**
     * Establish a socket connection with the IMS Connect host
     */
    public void connect() {
        try {
            // open a socket for the transaction
            socket = new Socket(hostName, portNumber);
        } catch (Exception e) {
            System.err.println(e);
            System.exit(1);
        }
    }

    /**
     * Drop the socket connection.
     */
    public void disconnect() {
        // verify socket open before attempting to disconnect
        if (socket != null) {
            try {
                socket.close();
                socket = null;
            } catch (Exception e) {
                System.err.println(e);
                System.exit(1);
            }
        }
    }

    //////////////////////////////////////
    // Send and receive methods                                     //
    //////////////////////////////////////

    /**

```

```

* Build and send a message to IMS Connect
* This method sends the message using a single write() call
* to improve efficiency and performance.
* The z/OS TCP/IP stack can be configured in such way that
* there is a 200ms delay for every ack message (TCP ack, not
* IMS Connect ACK). If such is the case, then you should expect
* a severe performance penalty if you use multiple write() calls.
* The old IMS Connect sample used a write for each field. DO
* NOT DO THAT.
* @param msgType Value for the IRM_F4 field (' ','R','A','N')
*/
public void send(char msgType) {
    int totalLength;
    String segment = null;
    response = null;
    byte irm_f3 = 0;

    if (msgType == ' ')
        segment = this.tranText;

    // Compute the total length of the message.
    // In java we don't care about byte order, since the write() methods
    // of DataOutputStream always work in network order.

    // +4 for first LL, ZZ and final LL, ZZ
    totalLength = 4 + prefixLength + 4;

    // add in segment length, if segment is defined
    if ((segment != null) && (segment.length() > 0)) {
        totalLength += segment.length() + 12; // +12 for LL, ZZ, tranCode
    }

    try {
        // Allocate a conveniently sized byte stream.
        ByteArrayOutputStream messageBuffer =
            new ByteArrayOutputStream(totalLength);
        // Associate a DataOutputStream to this newly created byte stream.
        DataOutputStream out = new DataOutputStream(messageBuffer);
        // Prepare another DataOutputStream to send the real message
        // to our IP socket.
        DataOutputStream outsocket =
            new DataOutputStream(socket.getOutputStream());

        // Compute the IRM_F3 byte
        irm_f3 = syncLevel;
        if (purgeAsync) {
            irm_f3 |= SL_PURGE;
        }
        if (reRoute) {
            irm_f3 |= SL_REROUTE;
        }

        // Build the IRM prefix
        out.writeInt(totalLength); // total message length
        out.writeShort(prefixLength); // IRM_LL
        out.writeByte(0x01); // IRM_ARCH
        out.writeByte(0x00); // IRM_F0
        // out.writeShort((short) 0); // IRM_RSV
        out.writeBytes(exitID); // IRM_ID
        out.writeInt(0); // IRM_RES
    }
}

```

```

        out.writeByte(IRM_F5_SINGLE); // IRM_F5
        out.writeByte(timer); // IRM_TIMER
        out.writeByte(0x40); // IRM_SOCKET - Transaction socket
        out.writeByte(0); // IRM_ES
        out.writeBytes(clientID); // IRM_CLIENTID
        out.writeByte(0); // IRM_F1 - No MODNAME request
        out.write(commitMode); // IRM_F2 - Set commit mode
        out.write(irm_f3); // IRM_F3 - Set sync level et al
        out.writeByte(msgType); // IRM_F4 - Set message type
        out.writeBytes(tranCode); // IRM_TRNCOD
        out.writeBytes(datastoreID); // IRM_IMSDESTID
        out.writeBytes(ltermName); // IRM_LTERM
        out.writeBytes(racfUserID); // IRM_RACF_USERID
        out.writeBytes(racfGroupName); // IRM_RACF_GRPNAME
        out.writeBytes(password); // IRM_RACF_PW
        out.writeBytes(" "); // IRM_APPL_NM
        out.writeBytes(reRouteName); // IRM_REROUT_NM

        // Add the transaction segment (trancode + tranText) if required
        if (msgType == ' ' || msgType == 'S') {
            // 12 for LL and ZZ and tranCode
            short recordLength = (short) (segment.length() + 12);
            out.writeShort(recordLength); // Transaction LL
            out.writeShort((short) 0); // Transaction ZZ
            out.writeBytes(tranCode); // Transaction code
            out.writeBytes(segment); // Transaction data
        }

        // send final LL ZZ to signal no more data to IMS Connect
        out.writeShort((short) 4); // send LL
        out.writeShort((short) 0); // send ZZ

        // Send the built message to IMS Connect
        out.flush();
        outsocket.write(messageBuffer.toByteArray());
        outsocket.flush();
    } catch (Exception e) {
        System.err.println(e);
        System.exit(1);
    }
}

/**
 * Read and parse the response from IMS Connect
 * We are using HWSSMPL0, so we won't have a total-length
 * prefix. If we were using HWSSMPL1 instead, we should have
 * taken that into consideration.
 */
public void receive() {
    byte xsm_flg = 0; // Flag byte
    boolean done = false; // End of loop indication

    // initialize segment vector
    Vector segments = new Vector();

    RSM = false;
    try {
        DataInputStream in = new DataInputStream(socket.getInputStream());

        // read total length. We don't care about byte order

```

```

int totalLength = (int) in.readShort();
// read flags
xsm_flg = in.readByte();
// read and ignore reserved byte
in.readByte();

// read identifier
byte[] identifierBytes = null;
if (totalLength < 12) {
    identifierBytes = new byte[totalLength - 4];
} else {
    identifierBytes = new byte[8];
}
in.readFully(identifierBytes);
String identifier = new String(identifierBytes);

// check first segment for possible errors / alerts
if (identifier.equals("*REQMOD*")) { // Parse RMM
    // read mod name
    byte[] modBytes = new byte[8];
    in.readFully(modBytes);
    String modName = new String(modBytes);
    // add mod name to segment vector
    segments.add("MOD NAME: " + modName);
} else if (identifier.equals("*REQSTS*")) { // Parse RSM
    // read return code and reason code
    RSM = true;
    returnCode = in.readInt();
    reasonCode = in.readInt();
    done = true; // No more data after RSM
} else {
    if (totalLength <= 12)
        segments.add(identifier);
    else {
        // read in the rest of the segment data
        byte[] segmentBytes = new byte[totalLength - 12];
        // identifier was actually
        in.readFully(segmentBytes);
        // part of the data. So we add it
        segments.add(identifier + new String(segmentBytes));
    }
}

String segmentData = "";
// continue trying to read in data till we come across *CSMOKY*
while (!segmentData.equals("*CSMOKY*") && !done) {
    // read next segment
    // read LL, XSM_FLG and reserved byte
    short recordLength = in.readShort();
    xsm_flg = in.readByte();
    in.readByte();
    // read in segment data
    byte[] segmentBytes = new byte[recordLength - 4];
    in.readFully(segmentBytes);
    segmentData = new String(segmentBytes);
    // add it to the segment vector
    segments.add(segmentData);
}

// At this point, we have just read an RSM or a CSM.

```

```

        // Check if an ACK will be required

        if ((xsm_flg & 0x20) == 0x20)
            ackRequired = true;

    } catch (Exception e) {
        System.err.println(e);
        System.exit(1);
    }

    // return segment vector
    this.response = segments;
}

//////////////////////////////////////
// Utility methods
//////////////////////////////////////

/**
 * Pads or truncates a string to the specified length
 * @param String String to pad or truncate
 * @param padChar Padding character
 * @param padLength Length to pad or truncate to
 * @return The padded or truncated string
 */
private String stringPad(String string, char padChar, int padLength) {
    String input = string;
    // check if string is null, use a space in that case
    if (input == null)
        input = " ";
    // construct a stringBuffer for padding efficiency
    StringBuffer stringBuffer = new StringBuffer(input);
    // pad the stringBuffer if string.length() is less than padLength
    for (int i = 0; i < (padLength - input.length()); i++) {
        stringBuffer.append(padChar);
    }
    // if truncation was necessary, substring will take care of that
    return stringBuffer.substring(0, padLength);
}

/**
 * Print an error message and show the correct command-line syntax
 * @param message Message to print
 */
private static void syntaxError(String message) {
    System.err.println(message);

    System.err.println("Syntax:");
    System.err.println(
        "SEND          : java Sample -h hostName -p portNumber -d datastoreName";
    System.err.println(
        "                                [-c clientId] [-t ltermName]");
    System.err.println(
        "                                [-u userId [-g groupName] [-w password]]");
    System.err.println(
        "                                [-x] [-y [rerouteName]]");
    System.err.println(
        "                                -m {0|1} [-l {N|C|S}] [-s] [-n] truncate
trandata...");
    System.err.println(
        "RESUME TPIPE: java Sample -h hostName -p portNumber -d datastoreName");
}

```

```

        System.err.println("                [-c clientid] -r [-n]");
        System.err.println("                [-u userid [-g groupName] [-w password]]");
        System.err.println("                [-m flag sets the commit mode to 0 or 1 on SEND interactions.]);
        System.err.println("                [-x flag enables the purge not deliverable feature.]);
        System.err.println("                [-y flag enables the reroute not deliverable feature. You can specify an optional");
        System.err.println("                reroute destination name.]);
        System.err.println("                [-l flag sets the Sync Level to N(one), C(onfirm) or S(ynch) on SEND interactions.]);
        System.err.println("                [-n flag forces a NAK if the Sync Level is not NONE.]);
        System.err.println("                [-s flag sets the interaction as SEND ONLY.]);
        System.err.println("                [-d flag] must be specified unless your IMS Connect exits take care of setting it.]);
        System.exit(8);
    }

    /**
     * Return a printable String representation of an instance of Sample
     */
    public String toString() {
        // Use a PrintWriter based on a StringWriter
        // to build the String representation
        StringWriter sw = new StringWriter();
        PrintWriter pw = new PrintWriter(sw);

        pw.println("IMS Connect information:");
        pw.println("\tHostname   :\t" + hostName);
        pw.println("\tPort       :\t" + portNumber);
        pw.println("\tClient ID  :\t" + clientID);
        pw.println("\texitID     :\t" + exitID);
        pw.print("\tsyncLevel :\t");
        switch (syncLevel) {
            case SL_NONE :
                pw.print("NONE\n");
                break;
            case SL_CONFIRM :
                pw.print("CONFIRM\n");
                break;
            case SL_SYNCPT :
                pw.print("SYNCPPOINT\n");
                break;
            default :
                pw.print("UNKNOWN\n");
                break;
        }
        pw.print("\tcommitMode:\t");
        switch (commitMode) {
            case CM0 :
                pw.print("0 - COMMIT THEN SEND\n");
                break;
            case CM1 :
                pw.print("1 - SEND THEN COMMIT\n");
                break;
        }
    }
}

```

```

        default :
            pw.print("UNKNOWN!\n");
            break;
    }
    pw.println("\tSend Only:\t" + sendOnly);
    pw.println("\tPurge      :\t" + purgeAsync);
    pw.println("\tReRoute    :\t" + reRoute);
    if (reRoute) {
        pw.println("\t      Name:\t" + reRouteName);
    }

    pw.print("IMS information:\n");
    pw.println("\tDatastore :\t" + datastoreID);
    pw.println("\tLTerm      :\t" + ltermName);
    if (resumeTpipe) {
        pw.println("RESUME TPIPE");
    } else {
        pw.println("Transaction information:");
        pw.println("\tCode      :\t" + tranCode);
        pw.println("\tText      :\t" + tranText);
    }
    // Flush the Stream and return the built String
    pw.flush();
    return sw.getBuffer().toString();
}

//////////////////////////////////////
// Main method                                     //
//////////////////////////////////////

/**
 * Main static procedure - execute one interaction.
 */
public static void main(String[] args) {
    int i = 0;
    String hostName = null;
    String portNumberString = null;
    int portNumber = 0;
    String datastoreID = null;
    String racfUserID = null;
    String racfGroupName = null;
    String password = null;
    String clientID = null;
    String ltermName = null;
    String commitModeString = null;
    String timeoutString = null;
    byte timeout = (byte) 0;
    byte commitMode = CM1;
    String syncLevelString = "C";
    byte syncLevel = SL_CONFIRM;
    boolean resumeTpipe = false;
    String tranCode = null;
    String tranText = null;
    boolean sendOnly = false;
    boolean forceNAK = false;
    boolean purge = false;
    boolean reroute = false;
    String rerouteName = null;
    String segment = null;
    boolean endProcess = false;

```

```

// Parse the command line parameters.

boolean endParseable = false;
for (i = 0; i < args.length && !endParseable; i++) {
    if (args[i].equals("-h"))
        hostName = args[++i];
    else if (args[i].equals("-p"))
        portNumberString = args[++i];
    else if (args[i].equals("-d"))
        dataStoreID = args[++i];
    else if (args[i].equals("-u"))
        racfUserID = args[++i];
    else if (args[i].equals("-g"))
        racfGroupName = args[++i];
    else if (args[i].equals("-w"))
        password = args[++i];
    else if (args[i].equals("-c"))
        clientID = args[++i];
    else if (args[i].equals("-t"))
        ltermName = args[++i];
    else if (args[i].equals("-m"))
        commitModeString = args[++i];
    else if (args[i].equals("-l"))
        syncLevelString = args[++i];
    else if (args[i].equals("-r")) {
        resumeTpipe = true;
        commitMode = CMO;
    } else if (args[i].equals("-s")) {
        sendOnly = true;
        commitMode = CMO;
    } else if (args[i].equals("-n")) {
        forceNAK = true;
    } else if (args[i].equals("-o")) {
        timeoutString = args[++i];
    } else if (args[i].equals("-x")) {
        purge = true;
    } else if (args[i].equals("-y")) {
        reroute = true;
        if (!args[i + 1].startsWith("-")) {
            rerouteName = args[++i];
        }
    } else {
        endParseable = true;
        i--; // Adjust the index so we not jump over the parameter.
    }
}

// At this point, the variable i contains the index of the first
// non-parsed parameter. We assume it's the transaction code we want
// to send, followed by the transaction text.

// let's do some sanity test

// Host and port are mandatory
if (hostName == null || portNumberString == null) {
    syntaxError("You must specify the host name and the port number.");
}

// Port number must be numeric

```

```

try {
    portNumber = Integer.parseInt(portNumberString);
} catch (NumberFormatException nfe) {
    syntaxError("The port number should be an integer.");
}

// IRM_TIMER value should be an hex byte
if (timeoutString != null) {
    try {
        timeout = (byte) Byte.parseByte(timeoutString, 16);
    } catch (NumberFormatException nfe) {
        syntaxError("The timer value should be an hexadecimal byte.");
    }
}

// Check commit mode.
if (commitModeString != null) {
    if (commitModeString.equals("0")) {
        commitMode = CMO;
    } else if (commitModeString.equals("1")) {
        commitMode = CM1;
    } else {
        syntaxError("The commit mode should be 0 or 1. The default value es 1.");
    }
}

// Check sync level
if (syncLevelString.equals("N")) {
    syncLevel = SL_NONE;
} else if (syncLevelString.equals("C")) {
    syncLevel = SL_CONFIRM;
} else if (syncLevelString.equals("S")) {
    syncLevel = SL_SYNCPT;
} else {
    syntaxError("The sync level should be N, C or S. The default is C (Confirm).");
}

// No trancode+data allowed in RESUME TPIPE
if (resumeTpipe) {
    if (i < args.length) {
        syntaxError("Transaction data not allowed with -r (RESUME TPIPE).");
    } else {
        // No LTERM nor commit mode allowed in RESUME TPIPE
        if (ltermName != null
            || commitMode != CMO
            || purge
            || reroute) {
            syntaxError("Lterm, commit mode, purge and reroute are not allowed with -r (RESUME TPIPE).");
        }
    }
} else {
    // Transaction code is mandatory in SEND interactions
    if (i == args.length) {
        syntaxError("Should specify at least a transaction code.");
    }
    // SEND ONLY not compatible with CM1
    if (sendOnly && commitMode != CMO) {
        syntaxError("-s (SEND ONLY) requires commit mode 0.");
    }
}

```

```

    }

    // CM1 requires sync level CONFIRM
    if (commitMode == CMO && syncLevel != SL_CONFIRM) {
        syntaxError("Commit mode 0 requires sync level CONFIRM.");
    }

    // In Commit Mode 0 and SL_NONE, we will not send a NAK even if
    // it was requested.
    if (forceNAK && syncLevel == SL_NONE) {
        System.err.println(
            "The -n flag (force NAK) will be ignored (sync level = NONE).");
    }

    // Parse the transaction text
    if (i < args.length) {
        tranCode = args[i++];

        // Now i points to the first token of the transaction text. We'll
        // concatenate all those tokens into a String.

        StringBuffer textbuf = new StringBuffer(1024); // Initial size: 1K

        for (; i < (args.length - 1); i++) {
            textbuf.append(args[i]);
            textbuf.append(" ");
        }
        textbuf.append(args[args.length - 1]);

        // set input text from StringBuffer
        segment = new String(textbuf);
    }

    // create a new sample object
    Sample sample =
        new Sample(
            hostName,
            portNumber,
            datastoreId,
            ltermName,
            tranCode,
            segment,
            clientID,
            racfUserID,
            racfGroupName,
            password,
            syncLevel,
            commitMode,
            sendOnly,
            resumeTpipe,
            timeout,
            purge,
            reroute,
            rerouteName);

    // Display the sample object (it will use the toString() method)
    System.out.println(sample);

    // connect to the host
    System.out.println("Connecting to the host...");

```

```

sample.connect();

// We will deal separately for SEND_RECEIVE, SEND_ONLY and RESUME TPIPE

if (sample.isResumeTpipe()) {
    System.out.println("Sending RESUME TPIPE...");
    sample.send('R');
    System.out.println("Receiving response...");
    sample.receive();
    if (!sample.isRSM()) {
        // print out the response

        Iterator it = sample.getResponse().iterator();
        i = 0;
        while (it.hasNext()) {
            System.out.println("Segment " + i++ + ": " + it.next());
        }
    } else {
        System.out.print("Status code: " + sample.getReturnCode());
        System.out.println("(X'" + Integer.toHexString(sample.getReturnCode()) +
"')");
        System.out.print("Reason code: " + sample.getReasonCode());
        System.out.println("(X'" + Integer.toHexString(sample.getReasonCode()) +
"')");
    }
    System.out.flush();
    // If requested and posible, send NAK
    if (sample.isAckRequired() && (sample.isNakRequired() || forceNAK)) {
        System.out.println("Sending NAK...");
        sample.setTimer((byte) 0xe9);
        sample.send('N');
    } else {
        // If necessary, send ACK
        if (sample.isAckRequired()) {
            System.out.println("Sending ACK...");
            sample.setTimer((byte) 0xe9);
            sample.send('A');
        }
    }
} else if (sample.isSendOnly()) {
    System.out.println("Sending SEND ONLY transaction ...");
    sample.send(' ');
} else {
    System.out.println("Sending transaction ...");
    sample.send(' ');
    System.out.println("Receiving response...");
    sample.receive();
    // print out the response
    Iterator it = sample.getResponse().iterator();
    i = 0;
    while (it.hasNext()) {
        System.out.println("Segment " + i++ + ": " + it.next());
    }
    System.out.flush();
    while (!sample.isRSM()) {
        // If requested and posible, send NAK
        if (sample.isAckRequired()
            && (sample.isNakRequired() || forceNAK)) {
            System.out.println("Sending NAK...");

```

```

        sample.setTimer((byte) 0xe9);
        sample.send('N');
    } else {
        // If necessary, send ACK
        if (sample.isAckRequired()) {
            System.out.println("Sending ACK...");
            sample.setTimer((byte) 0xe9);
            sample.send('A');
        }
    }
    System.out.println("Receiving ACK/NAK response...");
    sample.receive();
    // print out the response
    it = sample.getResponse().iterator();
    i = 0;
    while (it.hasNext()) {
        System.out.println("Segment " + i++ + ": " + it.next());
    }
    System.out.flush();
}
System.out.print("Status code: " + sample.getReturnCode());
System.out.println("(X'" + Integer.toHexString(sample.getReturnCode()) + "'");
System.out.print("Reason code: " + sample.getReasonCode());
System.out.println("(X'" + Integer.toHexString(sample.getReasonCode()) + "'");
}
// disconnect from the host
System.out.println("Disconnecting from the host...");
sample.disconnect();
}

////////////////////////////////////
// Some getter methods follow
////////////////////////////////////

/**
 * @return Returns the response.
 */
public Collection getResponse() {
    return response;
}

/**
 * @return Returns the resumeTpipe.
 */
public boolean isResumeTpipe() {
    return resumeTpipe;
}

/**
 * @return Returns the sendOnly.
 */
public boolean isSendOnly() {
    return sendOnly;
}

/**
 * @return Returns the syncLevel.
 */
public byte getSyncLevel() {
    return syncLevel;
}

```

```

    }

    /**
     * @return Returns the ackRequired.
     */
    public boolean isAckRequired() {
        return ackRequired;
    }

    /**
     * @return Returns the nakRequired.
     */
    public boolean isNakRequired() {
        return nakRequired;
    }

    /**
     * @return Returns the timer.
     */
    public byte getTimer() {
        return timer;
    }

    /**
     * @param timer The timer to set.
     */
    public void setTimer(byte timer) {
        this.timer = timer;
    }

    /**
     * @return
     */
    public boolean isRSM() {
        return RSM;
    }

    /**
     * @return
     */
    public int getReasonCode() {
        return reasonCode;
    }

    /**
     * @return
     */
    public int getReturnCode() {
        return returnCode;
    }
}

```

IMS RDS application example

The following samples illustrate simple IMS database access (query only) through the IMS Remote Database Services (RDS). We use a Web application that consists of HTML, servlets, and JSP for this purpose. The servlets have the JDBC access code for the IMS database access. We developed these materials in the Rational Application Developer for Windows product. You can download the materials in the Rational Application Developer project interchange format from the following IBM Redbook Web site:

<ftp://www.redbooks.ibm.com/redbooks/SG246794/IMSRDSSampleProjects.zip>

Table B-1 lists the sample source components.

Table B-1 IMS RDS sample source

Resource	Description
ImsRdsSampleGlobal.java	Servlet for IMS JDBC access with global transaction semantics
ImsRdsSample.java	Servlet for IMS JDBC access with local transaction semantics
GlobalInput.html	HTML document that invokes the servlet for global transaction
LocalInput.html	HTML document that invokes the servlet for local transaction
Output.jsp	JSP file for display the results

ImsRdsSampleGlobal.java

ImsRdsSampleGlobal.java is the servlet for the IMS JDBC access with the global transaction semantics, as shown in Example B-1.

Example: B-1 ImsRdsSampleGlobal.java

```
package imsRds;

import java.io.IOException;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.servlet.RequestDispatcher;
import javax.servlet.Servlet;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import java.sql.*;
import javax.transaction.UserTransaction;

public class ImsRdsSampleGlobal extends HttpServlet implements Servlet {

    DataSource dataSource;
    UserTransaction userTransaction;

    public void init() throws ServletException {
        try {
            //Obtain the initial JNDI Naming context for JDBC Connection
            System.out.println("IMS RDS Servlet : Start ");
            Context initialContext = new InitialContext();
            dataSource = (DataSource)initialContext.lookup("java:comp/env/imsjavaRDSRedBook");
            System.out.println("IMS RDS Servlet : Success Create DataSource");

            //Obtain the initial JNDI Naming context for User Transaction
            Context initctx2 = new InitialContext();
            userTransaction = (UserTransaction)initctx2.lookup("java:comp/UserTransaction");
            System.out.println("IMS RDS Servlet : Success Create UserTransaction");
        } catch (Exception e){
            e.printStackTrace();
        }
    }

    protected void doPost(HttpServletRequest arg0, HttpServletResponse arg1)
        throws ServletException, IOException {

        Connection connection = null;
        Statement statement = null;
        ResultSet results = null;

        String LastName = null;
        String FirstName = null;
        String Extension = null;
        String ZipCode = null;

        try{
            //Start User Transaction
```

```

userTransaction.begin();
System.out.println("IMS RDS Servlet : Success Start UserTransaction");

//Get JDBC Connection
connection = dataSource.getConnection();
System.out.println("IMS RDS Servlet : Success Get Connection");

//Get Key Value from Input Page
String keyValue = arg0.getParameter("keyValue").trim();

//Issue SQL
String queryString = "SELECT * FROM PhoneBook.Person "
    + "WHERE Person.LastName = '" + keyValue + "'";
statement = connection.createStatement();
results = statement.executeQuery(queryString);

//Get Output
while(results.next()) {
    LastName = results.getString("Person.LastName");
    FirstName = results.getString("Person.FirstName");
    Extension = results.getString("Person.Extension");
    ZipCode = results.getString("Person.ZipCode");
}
System.out.println("IMS RDS Servlet : Success Get Result");

//Commit UserTransaciton
userTransaction.commit();
System.out.println("IMS RDS Servlet : Success Commit UserTransaction");

//Set the result to the JSP
arg0.setAttribute("OUT__LastName", LastName);
arg0.setAttribute("OUT__FirstName", FirstName);
arg0.setAttribute("OUT__Extension", Extension);
arg0.setAttribute("OUT__ZipCode", ZipCode);

} catch (Exception e) {
    try{
        //RollBack UserTransaciton
        userTransaction.rollback();
        System.out.println("IMS RDS Servlet : Success Rollback UserTransaciton");

        //Close JDBC Resources
        if (statement != null) {statement.close();}
        if (connection != null) {connection.close();}
        System.out.println("IMS RDS Servlet : Caught exception in main section is: " + e);
        arg0.setAttribute("e",e.getMessage());
    } catch (Exception s){
        System.out.println("IMS RDS Servlet : Caught exception in rollback section is: " + s);
        arg0.setAttribute("s",s.getMessage());
    }
} finally {
    try{
        //Close JDBC Resources
        if (statement != null) {statement.close();}
        if (connection != null) {connection.close();}
        RequestDispatcher disp = arg0.getRequestDispatcher("Output.jsp");
        disp.forward(arg0,arg1);
        System.out.println("IMS RDS Servlet : Success Dispatch Result");
        System.out.println("IMS RDS Servlet : End" );
    } catch (Exception t){

```

```

        System.out.println("IMS RDS Servlet : Caught exception in final section is: " + t);
        arg0.setAttribute("t",t.getMessage());
        RequestDispatcher disp = arg0.getRequestDispatcher("Output.jsp");
        System.out.println("IMS RDS Servlet : Success Dispatch Result");
        System.out.println("IMS RDS Servlet : End" );
        disp.forward(arg0,arg1);
    }
}
}
}
}

```

ImsJavaRdsSample.java

ImsJavaRdsSample.java is the servlet for the IMS JDBC access with the local transaction semantics, as shown in Example B-2.

Example: B-2 ImsJavaRdsSample.java

```

package imsRds;

import java.io.IOException;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.servlet.RequestDispatcher;
import javax.servlet.Servlet;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import java.sql.*;
import javax.sql.*;

public class ImsRdsSample extends HttpServlet implements Servlet {

    DataSource dataSource;

    public void init() throws ServletException {
        try {
            //Obtain the initial JNDI Naming context for JDBC Connection
            System.out.println("IMS RDS Servlet : Start ");
            Context initialContext = new InitialContext();
            dataSource = (DataSource)initialContext.lookup("java:comp/env/imsjavaRDSRedBook");
            System.out.println("IMS RDS Servlet : Success Create DataSource");
        } catch (Exception e){
            e.printStackTrace();
        }
    }

    protected void doPost(HttpServletRequest arg0, HttpServletResponse arg1)
        throws ServletException, IOException {

        Connection connection = null;
        Statement statement = null;
        ResultSet results = null;

        String LastName = null;
    }
}

```

```

String FirstName = null;
String Extension = null;
String ZipCode = null;

try{
    //Get JDBC Connection
    connection = dataSource.getConnection();
    System.out.println("IMS RDS Servlet : Success Get Connection");

    //Get Key Value from Input Page
    String keyValue = arg0.getParameter("keyValue").trim();

    //Issue SQL
    String queryString = "SELECT * FROM PhoneBook.Person "
        + "WHERE Person.LastName = '" + keyValue + "'";
    statement = connection.createStatement();
    results = statement.executeQuery(queryString);

    //Get Output
    while(results.next()) {
        LastName = results.getString("Person.LastName");
        FirstName = results.getString("Person.FirstName");
        Extension = results.getString("Person.Extension");
        ZipCode = results.getString("Person.ZipCode");
    }
    System.out.println("IMS RDS Servlet : Success Get Result");

    //Commit LocalTransaction
    connection.commit();
    System.out.println("IMS RDS Servlet : Success Commit LocalTransaction");

    //Set the result to the JSP
    arg0.setAttribute("OUT__LastName", LastName);
    arg0.setAttribute("OUT__FirstName", FirstName);
    arg0.setAttribute("OUT__Extension", Extension);
    arg0.setAttribute("OUT__ZipCode", ZipCode);
} catch (Exception e) {
    try{
        //RollBack LocalTransacion
        connection.rollback();
        System.out.println("IMS RDS Servlet : Success Rollback LocalTransaction");

        //Close JDBC Resources
        if (statement != null) {statement.close();}
        if (connection != null) {connection.close();}
        System.out.println("IMS RDS Servlet : Caught exception in main section is: " + e);
        arg0.setAttribute("e",e.getMessage());
    } catch (Exception s){
        System.out.println("IMS RDS Servlet : Caught exception in rollback section is: " + s);
        arg0.setAttribute("s",s.getMessage());
    }
} finally {
    try{
        //Close JDBC Resources
        if (statement != null) {statement.close();}
        if (connection != null) {connection.close();}
        RequestDispatcher disp = arg0.getRequestDispatcher("Output.jsp");
        disp.forward(arg0,arg1);
        System.out.println("IMS RDS Servlet : Success Dispatch Result");
    }
}

```

```

        System.out.println("IMS RDS Servlet : End" );
    } catch (Exception t){
        System.out.println("IMS RDS Servlet : Caught exception in final section is: " + t);
        arg0.setAttribute("t",t.getMessage());
        RequestDispatcher disp = arg0.getRequestDispatcher("Output.jsp");
        System.out.println("IMS RDS Servlet : Success Dispatch Result");
        System.out.println("IMS RDS Servlet : End" );
        disp.forward(arg0,arg1);
    }
}
}
}
}
}

```

GlobalInput.html

GlobalInput.html is the HTML document that invokes the servlet for the global transaction, as shown in Example B-3.

Example: B-3 GlobalInput.html

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<META http-equiv="Content-Type" content="text/html; charset=Shift_JIS">
<META name="GENERATOR" content="IBM Software Development Platform">
<META http-equiv="Content-Style-Type" content="text/css">
<LINK href="theme/Master.css" rel="stylesheet"
      type="text/css">
<TITLE>InputPage</TITLE>
</HEAD>
<BODY>
<H1>IMS Remote Database Service:Global Transaction</H1>
<H2>Enter Key Value of Person Database !</H2>
<FORM METHOD="post" ACTION="ImsRdsSampleGlobal">
<H3>LastName</H3>
<P>
    <INPUT TYPE="text" NAME="keyValue" ID="keyValue" SIZE="10" MAXLENGTH="10" >
</P>
<P>
    <INPUT TYPE="submit" NAME="Execute" ID="Execute" VALUE="Execute">
    <INPUT TYPE="reset" NAME="Reset" ID="Reset" VALUE="Reset">
</P>
</FORM>

</BODY>
</HTML>

```

LocalInput.html

LocalInput.html is HTML document that invokes the servlet for the local transaction, as shown in Example B-4.

Example: B-4 LocalInput.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<META http-equiv="Content-Type" content="text/html; charset=Shift_JIS">
<META name="GENERATOR" content="IBM Software Development Platform">
<META http-equiv="Content-Style-Type" content="text/css">
<LINK href="theme/Master.css" rel="stylesheet"
      type="text/css">
<TITLE>InputPage</TITLE>
</HEAD>
<BODY>
<H1>IMS Remote Database Service:Local Transaction</H1>
<H2>Enter Key Value of Person Database !</H2>
<FORM METHOD="post" ACTION="ImsRdsSample">
<H3>LastName</H3>
<P>
  <INPUT TYPE="text" NAME="keyValue" ID="keyValue" SIZE="10" MAXLENGTH="10" >
</P>
<P>
  <INPUT TYPE="submit" NAME="Execute" ID="Execute" VALUE="Execute">
  <INPUT TYPE="reset" NAME="Reset" ID="Reset" VALUE="Reset">
</P>
</FORM>

</BODY>
</HTML>
```

Output.jsp

Output.jsp is the JSP file to display the results, as shown in Example B-5.

Example: B-5 Output.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<HTML>
<HEAD>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
      pageEncoding="ISO-8859-1"%>
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<META name="GENERATOR" content="IBM Software Development Platform">
<META http-equiv="Content-Style-Type" content="text/css">
<LINK href="theme/Master.css" rel="stylesheet"
      type="text/css">
<TITLE>OutPut page</TITLE>
<BODY>
</HEAD>
<H2>IMS Remote Database Service:Result</H2>
<H3>LastName</H3>
<%= request.getAttribute("OUT__LastName") %>
<H3>FirstName</H3>
```

```
<%= request.getAttribute("OUT_FirstName") %>
<H3>Extension</H3>
<%= request.getAttribute("OUT_Extension") %>
<H3>ZipCode</H3>
<%= request.getAttribute("OUT_ZipCode") %>

<H5>Exception in the Main Section</H5>
<%= request.getAttribute("e") %>
<H5>Exception in the Rollback Section</H5>
<%= request.getAttribute("s") %>
<H5>Exception in the Final Section</H5>
<%= request.getAttribute("t") %>

</BODY>
</HTML>
```

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described in this appendix.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/SG246794>

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select **Additional materials** and open the directory that corresponds with the redbook form number, SG246794.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

<i>File name</i>	<i>Description</i>
JAVA SAMPLE IVP.zip	Java sample program for testing IMS Connect
IMSRDSSampleProjects.zip	Zipped RDS sample projects
sample.zip	Sample C and Java client programs

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material ZIP file into this folder.

Abbreviations and acronyms

ACEE	access control environment element	J2EE	Java 2 Platform, Enterprise Edition
AGN	Application Group Name	JCA	J2EE Connector architecture
AIB	application interface block	JCL	job control language
APF	authorized program facility	JDBC	Java Database Connectivity
APPC	Advanced Program-to-Program Communication	JDK™	Java Development Kit
BPE	Base Primitive Environment	JRE	Java Runtime Environment
CCF	Common Connector Framework	JSP	JavaServer Pages
CGI	Common Gateway Interface	JVM	Java Virtual Machine
CICS	Customer Information Control System	LAN	local area network
CSM	Complete Status Message	LPAR	logical partition
CTG	CICS Transaction Gateway	LTERM	logical terminal
DBCTL	Database Control	LU	logical unit
DBD	database description	LU2	logical unit 2
DRA	database resource adapter	MCI	message control information
DVIPA	dynamic virtual IP address	MFS	Message Format Service
EAB	Enterprise Access Builder	MOD	message output descriptor
ECB	Event Control Block	MPP	message processing program
EMH	Expedited Message Handler	MSC	Multiple Systems Coupling
FTP	File Transfer Protocol	MVS	Multiple Virtual System
GUI	graphical user interface	ODBA	Open Database Access
HTML	Hypertext Markup Language	OO	object-oriented
HTTP	Hypertext Transfer Protocol	OTMA	Open Transaction Manager Access
IBM	International Business Machines Corporation	OTMA C/I	OTMA callable interface
IMS	Information Management System	PC	personal computer
IMS TOC	IMS TCP/IP OTMA Connection	PCB	program communication block
IPCS	Interactive Problem Control System	PPT	program properties table
IPL	initial program load	PSB	program specification block
IRM	IMS Request Message	PST	partition specification table
ISC	Intersystem Communication	RACF	Resource Access Control Facility
ISPF	Interactive Systems Productivity Facility	RAR	resource archive
ITOC	IMS TCP/IP OTMA Connection	RMM	Request MOD Message
ITSO	International Technical Support Organization	RRS/MVS	Resource Recovery Services/MVS
ITSO	International Technical Support Organization	RSM	request status message
IVP	installation verification program	SGML	Standard Generalized Markup Language
J2C	J2EE Connector architecture	SMB	scheduler message block
		SMP/E	System Modification Program/Extended
		SNA	Systems Network Architecture
		SOA	service-oriented architecture
		SOAP	Simple Object Access Protocol

<i>STSN</i>	Set and Test Sequence Numbers
<i>SVL</i>	Silicon Valley Laboratories
<i>TCB</i>	task control block
<i>TCP/IP</i>	Transmission Control Protocol/Internet Protocol
<i>Tpipe/TPIPE</i>	transaction pipe
<i>UOR</i>	unit of recovery
<i>W3C</i>	World Wide Web Consortium
<i>WAN</i>	wide area network
<i>VIPA</i>	virtual IP address
<i>WLM</i>	workload manager
<i>WSDL</i>	Web Service Definition Language
<i>WWW</i>	World Wide Web
<i>XCF</i>	cross-system coupling facility
<i>XML</i>	Extensible Markup Language

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 513. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *IMS e-business Connectors: A Guide to IMS Connectivity*, SG24-6514
- ▶ *Ensuring IMS Data Integrity Using IMS Tools*, SG24-6533
- ▶ *IMS Installation and Maintenance Processes*, SG24-6574
- ▶ *IMS Version 8 Implementation Guide - A Technical Introduction of the New Features*, SG24-6594
- ▶ *IMS DataPropagator Implementation Guide*, SG24-6838
- ▶ *Using IMS Data Management Tools for Fast Path Databases*, SG24-6866
- ▶ *IMS in the Parallel Sysplex, Volume I: Reviewing the IMSplex Technology*, SG24-6908
- ▶ *IMS in the Parallel Sysplex, Volume II: Planning the IMSplex*, SG24-6928
- ▶ *IMS in the Parallel Sysplex, Volume III: Operations and Implementation*, SG24-6929
- ▶ *The Complete IMS HALDB Guide, All You Need to Know to Manage HALDBs*, SG24-6945
- ▶ *Reorganizing Databases Using IMS Tools - A Detailed Look at the IBM IMS High Performance Tools*, SG24-6074
- ▶ *IMS Version 9 Implementation Guide - A Technical Overview*, SG24-6398
- ▶ *DB2 for z/OS Stored Procedures: Through the CALL and Beyond*, SG24-7083
- ▶ *WebSphere Application Server V6 Scalability and Performance Handbook*, SG24-6392
- ▶ *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 5: Availability, Scalability, and Performance*, SG24-6517

Other publications

These publications are also relevant as further information sources:

- ▶ *IMS Version 9: Administration Guide: Database Manager*, SC18-7806
- ▶ *IMS Version 9: Administration Guide: System*, SC18-7807
- ▶ *IMS Version 9: Customization Guide*, SC18-7817
- ▶ *IMS Version 9: IMS Java Guide and Reference*, SC18-7821
- ▶ *IMS Version 9: Messages and Codes Volume 1*, GC18-7827
- ▶ *IMS Version 9: Utilities Reference: System*, SC18-7834
- ▶ *IMS Version 9: Open Transaction Manager Access Guide and Reference*, SC18-7829
- ▶ *IMS Version 9: IMS Connect Guide and Reference*, SC18-9287

- ▶ *DB2 UDB for z/OS Version 8 Application Programming and SQL Guide*, SC18-7415
- ▶ *DB2 UDB for z/OS Version 8 Installation Guide*, GC18-7418
- ▶ *z/OS V1R6.0 MVS Initialization and Tuning Reference*, SA22-7592
- ▶ *z/OS V1R6.0 MVS Programming: Resource Recovery*, SA22-7616
- ▶ *z/OS V1R6.0 MVS System Commands*, SA22-7627
- ▶ *IBM WebSphere Application Server for z/OS, Version 6.0.1: Securing applications and their environment*, SA22-7961
- ▶ *IMS Connect Extensions for z/OS V1.1 User's Guide*, SC18-7255
- ▶ *IMS Message Format Services Reversal Utilities for z/OS User's Guide*, SC27-0823
- ▶ *IBM IMS Performance Analyzer for z/OS User's Guide*, SC27-0912
- ▶ *IBM IMS Performance Analyzer for z/OS Report Analysis*, SC27-0913
- ▶ *z/OS V1R6.0 CS: IP Configuration Reference*, SC31-8776
- ▶ *WebSphere MQ for z/OS System Setup Guide, Version 5 Release 3.1*, SC34-6052
- ▶ *Program Directory for IBM IMS Connect Extensions for z/OS*, GI10-8504
- ▶ *Program Directory for IBM IMS Connect for z/OS*, GI10-8506
- ▶ *Program Directory for IBM Information Management System Transaction and Database Servers*, GI10-8594
- ▶ *Concepts, Planning, and Installation for Edge Components Version 6.0*, GC31-6855

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ IMS product page
<http://www.ibm.com/ims/>
- ▶ IMS Connect for z/OS
<http://www.ibm.com/software/data/db2imstools/imstools/imsconnect.html>
- ▶ IMS Connector for Java library
<http://www.ibm.com/software/data/db2imstools/imstools-library.html#imsconjav-lib>
- ▶ Information Management Software for z/OS Solutions Information Center
<http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp>
- ▶ WebSphere Application Server - Edge Component Information Center
<http://www.ibm.com/software/webservers/appserv/ecinfocenter.html>
- ▶ WebSphere Studio Application Developer Integration Edition Information Center
<http://publib.boulder.ibm.com/infocenter/adiehelp/index.jsp>
- ▶ WebSphere MQ product page
<http://www.ibm.com/software/integration/wmq/>
- ▶ WebSphere Application Server - Edge Component Information Center
<http://www.ibm.com/software/webservers/appserv/ecinfocenter.html>
- ▶ Networking Technologies: VIPA
<http://www.ibm.com/servers/eserver/zseries/networking/vipa.html>

- ▶ ARMWRAP code
<ftp://www.redbooks.ibm.com/redbooks/REDP0173/>
- ▶ Updates to *IMS Connect Extensions for z/OS V1.1 User's Guide*, SC18-7255
http://www.ibm.com/support/docview.wss?rs=434&context=SSZJXP&dc=DA400&uid=swg27005885&loc=en_US&cs=utf-8&lang=en
- ▶ IBM developerWorks article "Introduction to the J2EE Connector Architecture"
<http://www.ibm.com/developerworks/java/edu/j-dw-javajca-i.html>
- ▶ IBM developerWorks article "JCA 1.5, Part 1: Optimizations and life-cycle management"
<http://www.ibm.com/developerworks/java/library/j-jca1/>
- ▶ IMS Family examples
<http://www.ibm.com/software/data/ims/examples/exHome.html>
- ▶ IMS Integration Solutions Suite, IMS MFS Web Support
<http://www.ibm.com/software/data/ims/toolkit/mfswebsupport/index.html>
- ▶ IMS SOAP Gateway
<http://www.ibm.com/software/data/ims/soap/>
- ▶ IMS Java 9.1 API Specification
http://www.ibm.com/software/data/ims/imsjava/api9_1/index.html
- ▶ DLIModel Utility Plug-in
<http://www.ibm.com/software/data/ims/toolkit/dlimodelutility/>
- ▶ Networking technologies
<http://www.ibm.com/servers/eserver/zseries/networking/technology.html>
- ▶ Information about Unicode, the Unicode Standard, and the Unicode Consortium
<http://www.unicode.org/>
- ▶ Cygwin
<http://www.cygwin.com>

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Archived

Index

Symbols

/DISPLAY OTMA 21, 26, 74
/DISPLAY TMEMBER 74
/SEC OTMA 21
/SECURE OTMA 20–22, 28, 114
/START OTMA 19

A

ACCEPT 21, 44, 74, 77, 159, 181–182, 239, 342, 367, 419, 442–443, 455
access control environment element (ACEE) 22
ACEE 22, 156, 200, 364–365, 434, 467
ACK 10, 13–14, 38, 49, 60, 93–96, 99, 101–102, 104–105, 128, 134, 169, 176–178, 181–184, 190, 198, 205, 207, 210, 212, 248, 258, 262–264, 266–267, 271, 273–275, 279–282, 284, 289–290, 292, 295–296, 302, 305, 487, 490, 496–497
ACK/NAK 14, 38, 70, 134, 177, 182–183, 205, 207, 210, 212, 267, 497
ACK/NAK required flag 134
ACK/NAK required notification support 38
ADFSMAC 390
AERTDLI 360, 363, 366–368, 370, 372
aggregate functions 467
AGN 362–366, 390
AIB 360, 363, 366–368, 370, 372–373, 375–376, 380, 388, 395, 460–461, 466–467
AIBREASN 369, 376, 393, 395–396, 404, 457, 461
AIBRETRN 369, 376, 393, 395–396, 404, 457, 461
ALTPCB 17–18, 20, 88, 100, 135, 258–259, 262, 281
APF 22, 45–46, 362–363, 391
APIs 360, 467
APPC xiii, 10, 19
applet 58–59
APPLID 20
APPLY xv, 28, 40, 44, 159, 204, 241, 246, 253, 259, 264, 270, 294, 344, 351, 402, 416, 418
APSB call 365, 369–370, 372, 385, 393
APSB security 365–366
ASCII 120, 122, 202, 212, 270, 276–278, 283, 419
asynchronous output 10, 23, 25, 38, 40, 51, 60, 91, 100–102, 105–107, 135, 137, 190, 244–245, 247, 261, 271, 273, 281–282, 303
asynchronous output support 38, 56, 100–101
ATRBAC 371
ATRCMIT 371
Authorized Program Facility (APF) 45
AUTO message control 104–105
autonomic computing 1

B

Base Primitive Environment (BPE) 44, 46
BPE 36–37, 44, 52–53, 63, 73, 130–131, 148–149,

152–153
BPE header 130–131, 144
BPECFG 45, 52, 142, 148, 160
BPECFGxx 44
BPXPRMxx 49, 320

C

cancel timer support 40
CLASSPATH 431, 433–434, 437–438, 446–447, 466
client bid 22, 31, 112, 114
client communication component (CCC) 36
clientID 58, 61, 67, 69, 97, 106, 127, 129, 137, 147, 172, 219, 233–234, 240, 242–244, 247, 249, 257, 260–261, 263, 272–273, 281–282, 285, 293, 299, 303–305, 309, 484–485, 488, 490–493, 495
ClientLauncher 58–59
CLOSEHWS 64–65, 73
CNBA buffers 374
COBOL 28, 145, 221, 355–356, 367, 379, 386–387, 392, 417, 420–421, 425
command component (CMD) 36
commit mode 9–13, 16, 21, 30, 38–40, 91, 93–98, 100, 102, 105–106, 119, 121–122, 129, 145, 168–169, 171, 176, 184, 210, 233–234, 240, 242–243, 246, 248–250, 255, 258, 260, 262, 264, 267, 270, 272–273, 279–282, 286, 293, 300, 304–305, 485, 488, 494–495
commitMode 229, 243–244, 247, 250, 286, 293, 484–486, 488, 491–495
commit-then-send 9–13, 23, 28, 74, 88, 93, 96–102, 129, 137
Complete Status Message (CSM) 92, 101, 133
connection pooling 226, 232–234
ConnectionFactory 224–225, 232–233, 236–237, 311–312
ConnectionManager 464
conversational xiii, 8–9, 17, 19, 22, 30, 39, 93–96, 99–100, 133–134, 168, 190, 204, 232, 244, 272, 274–275, 326, 340
conversations 15
correlator file 355–356
Coupling Facility 7, 35, 78
CREATE PROCEDURE 386–387, 392
CSM, Complete Status Message 133
CSQZPARM 31
CSSLIB 45–46, 142
cursor 337

D

database xiii, 2, 4, 14, 34, 41, 45, 224, 227, 237–238, 254, 359–360, 366, 374, 376, 379–380, 383–384, 386, 388, 390, 393, 395, 402, 404–408, 410, 413–416, 418–419, 421–422, 424, 428–430, 432, 434, 444, 451, 455, 457, 460–461, 464–465, 467, 499, 504–505
DataSource 421, 430, 433, 440, 444, 454, 457, 460, 464,

500–503

datastore 24–25, 34, 36, 44, 47, 50–52, 55, 58, 64–65, 67, 69–70, 73, 87–88, 115, 123–124, 129, 137, 156, 159, 167, 172–173, 177–178, 181, 183, 190–196, 216, 218, 233, 240, 270, 279–281, 333–334, 347–348, 491
datastore communication component (DCC) 36
datastore name 50, 69, 73, 124, 172, 348
DATASTORE statement 50–51
DB/DC 362–363
DB2 3, 13, 29–30, 34, 36, 41, 155, 181, 223–224, 227, 255, 376, 380, 383–387, 389–398, 401–402, 404, 406, 457
DB2 stored procedures 4, 380, 383–386, 389, 396, 398
DBCTL 362, 374, 390, 393, 457
DBRC 397
DCE/RPC 8
DEALLOC 190
DEDB 362, 374
DELETE 18, 41, 54, 73, 159, 199, 201, 203–204, 303, 313, 378, 386, 388, 390, 411, 413, 415, 428, 430, 447
DESC 139, 411
DFS058I 340
DFSCCMD0 114
DFSCTRN0 114
DFSDDLTO 462
DFSERA10 462
DFSISIS0 364–365
DFSIVP37 422–425, 446, 457, 460, 462
DFSMO1 338, 341
DFSMO2 338, 341
DFSMO3 338–339, 341
DFSMO5 338, 341
DFSPBxxx 19–20, 50, 114, 136, 374, 393, 457
DFSRAS00 364–366
DFSYDRU0 20, 25, 51, 135, 258
DFSYPRX0 20, 25, 135–138, 258–259
diagnosing problems 319
DISPLAY ACTIVE 396
DL/I 227, 360, 366, 368–371, 376, 380, 383, 388, 393, 396, 405, 409, 411, 414, 417–419, 429, 457, 460, 462–463, 466–467
DLIModel 414, 416–417, 420–426, 428, 433–434, 466
DLIModel Utility 420–426
DRA 360–364, 366, 370, 372–374, 380, 385–386, 389, 391, 406, 429–431, 433–435, 451, 466
DRA startup table 360–364, 385, 389, 391, 435
driver components 37
DRU 28–29, 51, 135–137
DSN 27, 45–46, 54, 142, 160, 171–172, 174, 176, 390, 394, 401, 435
DSNAIMS 29–30
DUPLEX 10
DVIPA 85–86
DVIPA environment 86
dynamic allocation 362
dynamic VIPA 80–86

E

EBCDIC 120, 122, 202–203, 270, 276–278, 283, 324, 347, 392, 394

ECB 48

EJB 3, 110, 227, 250–252, 255, 311, 321–323, 374, 377, 405–406, 428, 430–434, 439–441, 444, 454, 456–457, 460–461, 466
E-MCS console 397
EMH 9
environment component (EVC) 36
ESS interface 30
exit 20, 24, 28–29, 31, 38–39, 47–56, 58, 64, 74, 87–88, 92, 98, 101, 111, 113, 115–126, 128, 130–138, 142–143, 145, 147, 159–162, 168–169, 171–173, 176–177, 184, 188, 196–197, 201–204, 211, 214–216, 218–219, 258, 268–269, 273–274, 277, 284–285, 290–291, 293, 296, 326, 333, 337–338, 340–341, 344, 364–366, 486, 488, 490–491

F

FACILITY class 22, 27, 53, 113–114
failover 77–78, 81, 87–88
Fast Path 8–9, 362, 374
FMID 34, 45, 71
formatted dump 38, 153–154
full-duplex message 10

G

GENERATE 17, 92, 111, 133, 219, 232, 240, 244, 322–323, 332–333, 336, 346–347, 349, 355–356, 362, 421, 423, 425, 427
GenericCredential 230
GRNAME 19, 21, 50
GSAM 373

H

HMK9900 34, 45
host name 57, 81, 237, 284, 291, 319, 333, 347, 431, 451, 485, 493
HOSTNAME 24, 48, 52, 55, 58, 64, 233, 238, 282–283, 291, 332–335, 348, 357, 431, 451, 484–486, 490–493, 495
HTML xv, 58, 230, 265, 333, 338, 344, 418, 441, 453, 458, 499, 504–506
HTTP 79, 242, 266, 330, 336, 338, 352, 354, 406, 458, 500, 502, 504–505
HTTP session 330, 338
HWS 24, 45–47, 50, 52–53, 55, 64–66, 69, 107, 113, 123, 142, 149–150, 158–159, 196, 262, 274
HWS statement 47
HWSCFGxx 24, 40, 44, 55, 65, 67–71, 262, 300, 303
HWSCSLO0 48, 54–55, 64, 117, 119, 121, 160
HWSCSLO1 48, 54–55, 64, 117, 119, 121, 160
HWS0252W 301
HWSFTRC0 148–150
HWSIMSO0 51, 54, 98–99, 101, 118–120, 122, 128–130, 135
HWSIMSO1 51, 98–99, 118–120, 122, 128–129, 134–135
HWSJAVA0 48, 53–54, 98–99, 117–118, 121–122, 124, 127–128, 131, 144–145, 160, 203

HWSRCORD 46, 142, 160
 HWSSMPL0 52, 55–56, 58, 64, 74, 98–99, 101,
 118–122, 124, 128–130, 160, 172–173, 176, 184, 188,
 268–269, 276, 283, 287, 290, 294, 486, 488
 HWSSMPL1 24, 98–99, 118–122, 124, 128, 134–135,
 268–269, 274, 287, 294, 296, 488
 HWSUINIT 48, 51, 53–54, 117, 119, 123–125
 HWSYDRU0 135, 137–138

I
 IMS xiii–xiv, 1–4, 7–17, 19–31, 33–41, 43–61, 63–74,
 77–79, 81–87, 89–107, 109–138, 141–145, 147–150,
 152–171, 173–198, 200–210, 212–217, 219, 221–227,
 229–232, 235–244, 246–248, 250, 253–262, 264–287,
 290–294, 296–305, 307–312, 314, 316, 319–327,
 329–333, 335–336, 338–341, 343, 345–348, 350,
 353–357, 359–360, 362–376, 378–381, 383, 385–389,
 393–397, 399–401, 405–410, 412–426, 428–435,
 437–441, 444, 446–449, 451, 453, 455–457, 459–467,
 469, 484–488, 491–492, 499–505
 IMS Client for Java 56–61
 IMS command 8–9, 11, 20–21, 28–30, 51, 74, 244, 462
 IMS Connect xiii, 3–4, 8, 20–21, 23–25, 33–41, 43–56,
 58, 60–61, 63–74, 77–78, 81, 83–89, 91–93, 97–107,
 109–113, 115–119, 121–128, 130–135, 137–138,
 141–142, 144–145, 147–153, 155–165, 167–171,
 174–185, 187, 189–197, 200–205, 207–208, 210,
 212–213, 215–217, 219, 221, 224, 229, 231–232, 235,
 237–240, 244–248, 250, 258–259, 261–277, 279–287,
 290–294, 296–305, 309–310, 314, 319–320, 333, 336,
 341, 347, 470, 484, 487–488, 491
 IMS Connect Base Primitive Environment 36–37, 52, 141
 IMS Connect BPE 36–37, 52, 73, 148–149
 IMS Connect commands 64, 319
 IMS Connect Dump Formatter 141, 148–151
 IMS Connect Extensions 4, 41, 87, 155–161, 163, 167,
 170, 175, 177, 181, 184, 195, 200, 203–205, 208, 218,
 298, 301, 306
 IMS Connector for Java xiii, 4, 8, 34–35, 38–39, 41, 48,
 50, 53, 67, 70, 74, 91, 98–100, 105–107, 111, 118, 121,
 127–128, 130–132, 144–145, 147, 221, 224–225,
 229–232, 235, 237–240, 255, 257–259, 261–265, 270,
 291, 297–303, 305–310, 312, 319, 322, 332–333,
 335–336, 338, 341, 348, 350, 484
 IMS Control Center 4, 33, 36, 39, 41, 43, 48, 54–55, 65,
 70, 119, 121
 IMS conversation 99
 IMS conversational transaction 244
 IMS initialization errors 376
 IMS Java 377, 380, 405–407, 409, 411, 416, 418–420,
 424–425, 430, 433–434, 440–441, 456, 461–462, 464,
 466
 IMS JDBC Resource Adapter 377, 430, 432–434, 437
 IMS Monitor Trace 142
 IMS OTMA 24, 26, 28–30, 36–37, 45, 47, 50, 53, 116,
 138, 190, 240, 244, 246, 324
 IMS OTMA Communications 37
 IMS Performance Analyzer 156, 177–178, 181–184
 IMS Problem Investigator 156, 158, 167, 184, 205
 IMS Remote Database Services xiii, 4

IMS Request Message (IRM) 92
 IMS security 22, 109, 114, 365
 IMS SOAP Gateway xiii, 353–357
 IMS TCP/IP OTMA Connection (ITOC) 37
 IMSConnectionFactory 224–225, 232, 237, 240
 IMSConnectionSpec 225, 234, 242–244, 248–249, 259
 IMSID 69, 360, 362, 388, 435
 IMSInteractionSpec 229–230, 242, 259, 261, 299, 301,
 334, 340
 imsjava.jar 437–438, 447
 IMSMON 142
 IMSPLEX 19, 36–37, 41, 46–48, 50–51, 55, 65, 68–71,
 116–117, 119, 121
 IMSplex 19, 39, 51, 55, 65, 68, 70–71, 121
 IMSplex communications component (ICC) 36
 IMSplex driver (IPDC) 36
 IMSPLEX statement 51
 IMSXCF.group.member 22, 53
 INSTALIB 362
 INSTALL 3, 38, 45, 53–54, 56, 58, 61, 126, 149, 342,
 350–351, 391, 421, 426, 436–437, 440–441, 446,
 448–449, 466
 INSTALL/IVP 61
 installing 44, 57, 342, 434, 436–437, 439, 441, 444, 446,
 448–449
 instance servlets 329, 335–336, 338, 349
 integrated IMS Connect function 3, 34, 40, 42, 63, 72,
 217
 InteractionSpec 50, 229, 232, 252, 302, 309, 341
 Internet 3, 39, 48, 77, 90, 109, 216, 329, 343, 430, 507
 IOPCB 9, 11, 16–19, 21, 30, 40, 88, 100, 105–106,
 258–262, 267, 273, 305, 370
 IPCS 141, 148–150, 152
 IPV6 48, 55, 64, 216
 IRM header 70, 93–97, 112, 118–119, 121–122, 124,
 127–129, 147, 175, 202, 276–277, 305
 IRM_TIMER 49, 97, 102–103, 127–128, 169, 269,
 271–272, 279, 281–282, 285, 293, 298, 300–301,
 303–305, 488, 494
 ISC 13
 ISIS 363–365
 ISPF 149, 157–160, 175, 210, 391, 400
 ITOC 37–38, 56, 123, 147, 175
 ITOCRC 142–146
 ITOCSN 142, 144–147
 IVP 61, 145, 330, 345–346, 362, 386–389, 393, 422,
 426, 434–435, 446, 457
 IVTCM 457
 IVTNO 143, 145–147, 175, 178–179, 332–333, 347, 352

J

J2C 38, 225, 232, 236, 241, 298–299, 306–309, 312,
 430, 432, 434, 440–441, 444, 449–451, 463–464, 466
 J2EE xiii, 34, 38–39, 110–111, 221–223, 234–236, 241,
 245, 252, 335, 341, 377, 407, 430, 432–433, 451, 463
 J2EE Connector architecture 38–39, 222
 Java xiii, 2–4, 8, 34–35, 38–39, 41, 43, 48, 50, 53, 56–61,
 67, 70, 74, 91, 98–100, 105–107, 111–112, 118, 121,
 127–132, 144–145, 147, 221–223, 225, 227–233,
 235–240, 242–244, 246, 248, 250–252, 255, 257–262,

264–265, 268, 270, 275–276, 279, 283, 291–292, 294, 297–303, 305, 307–309, 311–312, 316, 319, 321–322, 330, 332–333, 335–336, 338–341, 347–350, 354, 357, 377, 380–381, 401–402, 404–409, 411–414, 416, 418–419, 421–426, 428–431, 433–434, 438–439, 441, 444, 446, 451, 453, 456, 460–461, 463, 466, 469, 484, 487, 490, 499–500, 502, 505

Java 2 Platform, Enterprise Edition 221

Java applications 48, 246, 304, 309, 319, 377, 406, 457

Java classes 223, 306, 310, 330, 404, 423

Java development 41

Java execution 300

Java language 282

Java Native interface

 JNI 223

Java runtime 309, 425

Java Virtual Machine 57, 406

java.io 241, 299, 402, 484, 500, 502

JavaScript 346

JBP 397, 406, 457

JCA 111, 221–224, 226–228, 230–231, 241, 245, 291, 316

JCL 27, 44–47, 50, 52–54, 83–85, 142–143, 160, 165, 170, 174–175, 362, 376, 386–387, 389–390, 406, 435

JDBC 224, 376–378, 380–381, 402, 404, 406–407, 409–410, 412, 414–416, 418–420, 429–434, 436–437, 440–441, 447–450, 453–456, 461, 464, 466–467, 499–503

JMP 397, 406, 457

JVM 282, 406

K

keyring 237–239

keystore 237–238

KSDS 158

L

LANG 53, 148, 251, 311, 422, 438, 461

libJavTDLI.so 438

Line Trace 46, 141, 143

Linux 3, 79, 283, 348

Load Balancer 78–79, 86

load balancing 77–80, 85, 87–89

Local Option 34–38, 46, 49, 66, 113, 224, 236–237, 253

local option communication component (LOCC) 36

local option driver (PCDC) 37

LTERM Name 111, 137, 246, 258, 273

LU2 11

M

map name 9, 11

Max Connections 313, 317, 319

MAXFILEPROC 49, 320

maximum connections 319

MAXSOC 24, 40, 48, 55, 64, 69, 314, 319

MAXSOCKETS 49, 320

MCS 397

Message Control Information 9

metadata 223, 307, 323, 330–333, 335–336, 339, 379, 414, 417, 420–428, 431, 433–434, 446–447, 451, 455, 462, 466

metadata catalog 379

MFS 9, 29, 129, 133–134, 145, 221, 241, 243–244, 246, 268, 270, 275, 321–327, 329–332, 335–341, 344–347, 349–350

MFS MOD name 93, 133–134

MFS Web Enablement 329–330, 336, 344

MFS Web Services 321–324

middleware 1, 115

migration 2, 120, 216

Min Connections 313

MOD 29, 93, 129, 133–134, 144, 246, 268, 270, 274–275, 287–288, 290, 294–296, 323, 330, 332–333, 339, 341, 346, 489

mode 8–13, 16–17, 19, 21, 30, 38–40, 91, 93–98, 100, 102, 105–106, 117, 119, 121, 129, 139, 145, 168–169, 171, 176, 184, 210, 233–234, 240, 242, 244, 246–250, 253–255, 258, 260, 262, 264, 267, 270, 272–273, 279–282, 286, 293, 300, 304–305, 331–332, 346–347, 349–350, 485, 488, 491, 494–495

MODname 9, 11, 172, 286, 288, 293, 295, 339, 341, 488–489

MPP 249–250, 254, 406

MQSeries 30

MSC 9, 19, 87

MVS 27, 320, 332, 367, 375, 400

N

NAK 10, 13–14, 22, 38, 60, 70, 94–96, 101–102, 104–106, 133–134, 177–183, 190, 205, 207, 210, 212, 262–263, 266–267, 271, 273–275, 282, 290, 491, 495–497

namespace 225, 232, 234

Network Dispatcher 77–80

NOAUTO message control 103–104

non-persistent socket 100, 269

O

ODBA 4, 359–372, 374, 376–378, 380, 383–397, 399–401, 406, 421, 429–435, 437–439, 449, 457, 461–462, 467

OLDS 462

OM 36–37, 39, 41, 51, 55, 65, 68, 119

on demand 1–3, 34

on demand strategy 1

online change 386

Open Database Access (ODBA) 4

Open Transaction Manager Access (OTMA) 4, 35

OPENDS 64–65, 73

OPENIP 64–65, 73

OPENPORT 64, 66, 73

operations 3, 21, 34, 57, 63, 321, 326, 419

Operations Manager 36, 51, 55, 119

Operations Manager (OM) 37, 41, 65

OTMA xiii, 4, 7–31, 35–37, 40–41, 44–45, 47–51, 53, 56, 69, 74, 89, 91–93, 97–98, 101–105, 109, 111–117, 120–123, 126–132, 135–138, 142, 144–145, 147, 162,

167–173, 176–184, 190, 206, 211, 216, 240, 244–246,
258–259, 267, 270, 272–273, 275–277, 279, 281, 300,
305, 324
OTMA C/I 8, 26–29
OTMA C/I restrictions 28
OTMA callable interface 8, 26–28
OTMA client 8, 10–11, 13, 16, 20–23, 26, 30, 50, 101,
112, 135–137, 258
OTMA driver (OTDC) 37
OTMA DRU (destination resolution) exit 135
OTMA header 9–10, 98, 101, 112, 114, 120–121, 128,
130–132, 144, 147, 277
OTMA message format 35, 92
OTMA message structure 9
OTMA super member 89
OTMA= 20
otma_alloc 28
otma_close 29
otma_create 28
otma_free 29
otma_open 28
otma_send_async 28
otma_send_receive 28
otma_send_receivex 28
OTMAASY 17–18, 21
OTMAASY=N 18
OTMACON 31
OTMAMD 20, 136–137
OTMANM 20–21, 50, 69–70
OTMANM= 20
OTMASE 20–21, 114
OTMASP 20

P

Parallel 4, 10, 78, 254
Parallel Sysplex 7, 30, 77–78
PasswordCredential 230
PCB 25–26, 29, 101, 135–136, 246, 248–249, 258, 267,
367–369, 371, 375, 396, 412–414, 416, 422–425, 466
performance 2–4, 7, 34, 36, 38, 43, 47–48, 80, 84,
112–113, 155–156, 158, 162, 167, 177–184, 190, 200,
205, 224, 226, 284–285, 292, 294, 305, 314–318,
374–375, 412–414, 419, 462, 466–467, 487
persistent socket 38–39, 56, 98–100, 102, 127, 129,
168–169, 229, 234, 240, 242–244, 246–247, 249, 269,
299
persistent socket support 39
PL/I 28, 367, 391
port 40, 44, 49, 57, 65–71, 73–74, 79, 83–84, 87, 122,
158, 160, 167, 172–173, 175–178, 181–182, 184,
196–198, 205–207, 214, 216, 219, 237–238, 273,
279–282, 284, 298–299, 303, 314–316, 319, 321,
332–333, 347–348, 356–357, 402, 404, 431, 451, 485,
493
PORTID 24, 49, 52, 55, 57, 64, 66–68, 71
POSIX 51, 283
PPT 27, 45–46
PQ62379 41
PQ69527 41
PQ70216 41

PQ90146 49
PROCLIB 19–20, 45, 47, 49, 51–52, 55, 136, 142, 160,
374, 389–390, 393, 457
Program Properties Table (PPT) 45–46
program-to-program switch 21, 246, 281
PROGxx 46
project xiii–xiv, 355, 427, 451–453, 458, 499
PSBGEN 369, 422
PST 372–373, 396
PTKTDATA statement 51
purge not deliverable support 40

Q

QUERY 41, 73, 124, 227, 250, 255, 319, 380, 408–409,
412–414, 416–419, 433–434, 455, 459–461, 466–467,
499
QUERY MEMBER 73

R

RACF 20–22, 27, 45, 47, 49, 51–53, 55, 57–58, 64, 66,
69, 73, 111–114, 122–123, 126, 129, 172, 178, 235,
237–238, 270, 332, 334, 339, 343, 347–349, 364–366,
434, 484–485
RAR 223–224, 350, 436, 448
RATE 178–180
Rational Application Developer 41, 230, 232, 245, 259,
306, 355, 402, 434, 499
RECEIVE 8, 22, 26, 28–29, 40, 44, 50, 58, 60, 65, 68,
77, 88, 94, 97, 101–103, 115, 126, 128, 144, 147, 150,
159, 169, 193, 205, 212, 224, 244, 246, 258, 267,
279–282, 287, 291, 294, 299–300, 303–304, 324, 333,
351–352, 367, 393, 410, 414, 444, 457, 461, 486, 488,
496–497
RECORDER 64, 66, 73, 141–142, 145, 167, 174–175,
298, 305–306
Recoverable Resource Services (RRS) 39
Recovery Resource Services 359
Redbooks Web site 508, 513
Contact us xv
Remote Database Services xiii, 4, 405, 428, 444, 457
Remote Database Services (RDS) 405
Request Mod Message 93, 133, 144, 268, 275
Request Mod Message (RMM) 93, 133
Request Status Message (RSM) 92, 133
res-auth option 236
resource 3, 15–16, 21, 27, 39, 53–54, 58, 88, 110–111,
113, 126, 156, 161, 167, 177–178, 200, 216, 222–224,
226–227, 230–232, 236, 241, 243–244, 248–250,
252–253, 299, 301, 305, 307–311, 317, 333, 335, 341,
350, 359–360, 364–366, 369, 371, 374–375, 377, 379,
390, 400, 418, 423–424, 430–434, 436–437, 440–441,
444, 447–449, 451, 453–454, 461, 463–464, 466–467,
499
Resource Manager 15–16, 221, 223, 227, 386, 400–401
Resource Recovery Services/MVS 15
RESUME TPIPE 23, 25, 60, 88–89, 101–104, 106,
128–129, 178, 182, 190, 261, 267, 271–273, 277,
281–282, 284, 298, 303, 485, 490, 492, 494, 496
Resume Tpipe report 182

RM 401
 RRNAME 50, 107, 262, 274
 RRS 10, 15–16, 19, 29–30, 39, 47, 67, 69, 223–224, 248,
 250, 253–254, 359–360, 371–372, 374–375, 386, 391,
 393, 397, 400–401, 431, 433, 435, 456–457, 466–467
 RRSF 386
 RSR 20
 RunAs 236
 running errors 376
 RUNOPTS 51

S

SAF 22, 113, 159, 178–181, 200, 234
 SCEERUN 45–46, 142, 391
 SCHEDxx 27, 46
 SCI 36–37, 51, 55, 65, 68
 SDFSJLIB 435
 SDFS MAC 9, 54, 127, 131–132
 SDFSRESL 27–28, 45–46, 54–55, 119, 121, 126, 142,
 160, 362–363, 387, 390–391, 435
 SDFS SRC 54
 Secure Sockets Layer (SSL) 49
 security 9, 20–22, 27, 34, 39, 43, 45, 49, 53, 66, 69, 74,
 91, 109–115, 119, 122, 155–156, 159, 178, 197,
 200–202, 221, 223, 230–231, 234–237, 273, 298,
 342–343, 351, 361–366, 379, 392, 433–434, 445, 466,
 484
 security data 9, 22, 112, 114
 SECURITY macro 365
 send-then-commit 9–16, 21, 38, 70, 74–75, 88, 93–96,
 98–99, 119, 121–122, 129, 145, 279
 service-oriented architecture (SOA) 3, 353
 servlet 110, 226, 252, 319, 329, 331–336, 338–341,
 343–344, 347–349, 352, 453–456, 458–459, 461,
 464–465, 499–504
 set and test sequence number 13
 SETRACF 47, 64, 66–67, 73, 113, 235
 SETRRS 64, 67, 73
 shared queues 19, 25–26, 89–90
 Simple Object Access Protocol (SOAP) 353
 SINGLE 4, 12, 34, 36, 39–40, 77, 79–80, 85, 87, 90,
 98–103, 128, 137, 142, 145, 159, 168–169, 182,
 190–191, 193, 195, 197–198, 205, 223, 227, 237,
 244–245, 259, 269, 271, 284–285, 287, 291–292, 294,
 319–320, 341, 361, 372–373, 378, 380, 384, 391, 394,
 466–467, 487
 single point of control (SPOC) 41
 SMB 11, 136
 SMP/E 38, 44, 159
 SMU 364, 366
 SOAP xiii, 3, 315–316, 353–357
 SOAP message 354, 357
 socket call 92
 sockets 33–34, 36–40, 48–49, 77–78, 98–100, 105, 167,
 175, 233–234, 237, 240, 248, 250, 266, 291, 303–304,
 319–320, 338, 430, 484
 SPA 243
 SPOC 41
 SQL 29–30, 376, 378–381, 384–387, 392, 398, 402,
 405–408, 410, 412–413, 415–416, 418–419, 424,

454–455, 460–464, 466, 500–503
 SRRBACK 371
 SRRCMIT 371
 SSL support 39, 46, 51, 110
 SSLENVAR 49, 239
 SSLPORT 49, 239
 STARTDS 64–65, 73
 STARTIP 64–65, 73
 STARTP 66
 STARTPT 64, 73
 state data 9–12, 25, 340, 399
 static VIPA 81
 STGCLASS 31
 STOPCLNT 64, 67, 73, 303
 STOPDS 64, 67–68, 73
 STOIP 64, 68, 73
 STOPPORT 64, 68, 73
 storage 17, 25, 27–28, 31, 36–37, 46, 52, 117, 130, 148,
 200, 259, 290, 296, 360, 367–368
 storage class 31
 stored procedure 29–30, 376, 383–398, 401–404, 457
 STSN 13
 super member 23–26, 89
 sync level=confirm 14–15, 93–96, 99–100, 134, 280
 sync level=none 13–14, 94
 sync point 13–16, 18–19, 39, 88, 248, 252–253, 256,
 359, 386, 397, 491
 synchronization level 9–11, 13–14, 94–97, 250, 267, 279
 SYS1.PARMLIB 27, 45–46, 320
 sysplex 7, 23, 30, 36, 43, 77–80, 82–83, 85–88, 314,
 375, 391, 399
 Sysplex Distributor 23, 77, 85–86
 system definition 20, 158–161, 163, 169–170, 185, 193,
 196, 200, 365
 system generation 19, 386

T

takeover 80–82, 86
 TCP/IP 3, 8, 33–38, 40–41, 44–50, 52, 56, 58, 60, 65–68,
 70, 77, 80–84, 88, 91–92, 101, 109, 113, 115–116,
 118–119, 122, 125, 147, 162, 178, 181, 205, 222, 224,
 226, 229, 237, 239–240, 253, 266, 271, 279, 281, 284,
 292, 298–299, 306, 320, 374, 458, 484, 487
 TCP/IP clients 34–35, 37, 45, 48, 115
 TCP/IP driver (TIDC) 37
 TCP/IP statement 40, 48
 TERMINATE 64, 99, 102, 104–105, 267, 273, 338, 340,
 372, 374, 399–400
 TIMEOUT xiii, 4, 49, 55, 64, 69, 97, 103, 162, 179, 182,
 205, 213–216, 244–245, 249, 254, 257–258, 267, 269,
 271–272, 279–281, 298–305, 312–314, 316, 319,
 332–333, 335, 338, 357, 362, 390, 398, 419, 492,
 494–495
 timeout 49, 97, 103, 182, 213–215, 244–246, 271, 281,
 299–305, 313–314, 319, 332–334, 338, 390
 T MEMBER 14, 24–25, 31, 50–52, 55, 64–65, 71, 74, 88,
 167, 183
 TPIPE 10–14, 20, 23, 25–26, 29, 31, 40, 60, 74, 88–89,
 93, 96–97, 101–104, 106, 128–129, 137, 144, 147,
 172–173, 178, 182–184, 190, 243, 247–250, 257–259,

261–263, 267, 271–274, 277, 281–282, 284, 298, 303, 305, 485, 490, 492, 494, 496
 Tpipe 10–12, 14, 20, 23, 25, 31, 40, 74, 88–90, 94–95, 97, 101–103, 106, 128, 137, 172, 182, 184, 247–249, 257, 259, 261–262, 267, 271–273, 281, 303, 494
 TRANSACT 13, 178–181, 183
 transaction code 11, 31, 61, 70, 129, 147, 177–178, 182, 258, 266, 270, 273, 276–278, 286, 293–294, 299, 337, 340, 344, 485, 488, 493–494
 transaction socket 98–100, 102, 127, 129, 176, 210, 269, 279–280, 285, 291, 293, 488
 TRCLEV 53, 148
 truststore 237–238
 two-phase commit 10, 13, 16, 39, 47, 126, 221, 227, 254, 372, 374, 380, 386, 456

U

Unicode 38, 120, 269, 275–278
 unit of recovery (UOR) 16
 UNIX System Services 319–320, 422
 UOR 16, 72–73, 397, 401
 UPDATE 41, 45, 73, 86, 113, 149, 204, 227, 248, 254–255, 305, 338, 376, 378, 386, 388, 411, 413, 415–416, 430, 455
 URID 71, 397
 user data 9, 28–29, 31, 98, 122, 127–128, 130–132, 134, 147, 278
 user exit 38, 47, 54, 58, 98, 113, 115, 117–118, 125–126, 128–129, 136, 162, 168–169, 202–204, 277, 365
 user initialization exit 47, 53, 123
 user initialization support 38
 user message exit 39, 47–48, 87–88, 98–99, 101, 117, 119–121, 124, 142, 144, 177, 276

V

VIEWDS 64, 69, 73
 VIEWHWS 25, 64, 67, 69, 73, 319
 VIEWIP 64, 70, 73
 VIEWPORT 64, 71, 73, 298
 VIEWUOR 64, 71, 73
 VIPA 77, 80–87
 virtual IP address (VIPA) 77, 80
 virtualization 1
 VisualAge for Java 38
 VTAM 8

W

Web application 330–331, 333, 335, 338, 349, 352, 405, 421, 429–430, 432–434, 446, 454, 457, 461–462, 465–467, 499
 Web Service Definition Language (WSDL) 322
 Web Services 2–4, 34, 321–325, 353–356, 406
 Web Services Invocation Framework (WSIF) 322
 WebSphere xiii, 2–3, 20, 30, 34–35, 38, 41, 79, 110, 225, 231, 236, 241, 250, 253, 259, 298, 306, 312, 333, 338, 342, 347–348, 350–352, 355, 377–379, 426, 430–431, 433–435, 437, 448–449, 451, 454, 458, 461, 463–464, 467

WebSphere Application Server 35, 38, 80, 106, 110–113, 221, 223–224, 230, 234–236, 238, 241–242, 250, 253–254, 261, 298, 306, 308–310, 312–315, 319, 321–322, 329–333, 335–336, 338, 341–343, 348, 350–352, 377–378, 380–381, 405–406, 421, 428–435, 437–438, 440–441, 446–449, 451, 455–461, 463–464, 466–467
 WebSphere Application Server for z/OS 39, 236, 377, 406, 428, 431, 433–435, 438, 449, 457, 461, 467
 WebSphere Application Server SOAP service 315
 WebSphere Edge Components 77–78, 80, 86
 WebSphere Edge Server 79
 WebSphere Information Integrator Classic Federation 378–381
 WebSphere MQ 8, 20, 30–31, 223, 259
 WebSphere MQ API 30
 WebSphere MQ IMS Bridge 30–31
 WebSphere Studio 38, 230, 243–244, 246, 306, 321–322, 325–327, 425–426
 WebSphere Studio Application Developer Integration Edition 232, 241, 261
 WLM 375–376, 380, 384–387, 389, 391–392, 394, 398–399
 WLM application environment 386, 391
 workload balancing 3, 34, 85, 156, 158, 160, 189, 193–194, 218
 Workload Manager 391, 398
 writing ODBA application programs 366
 WSDL file 321–323, 355–356
 WSDL message 322
 WTO 138–139, 298

X

XCF 7–8, 10–12, 19–22, 26–27, 30–31, 35–37, 39, 43–44, 50, 53, 69–70, 74, 82, 86, 89, 92, 183, 240, 250
 XCF group 7–8, 21, 27, 31, 50, 69–70, 74, 113, 183
 XCF member 21, 31, 50, 69–70, 74
 XCFGNAME 31
 XCFMNAME 31
 XIB 47, 123–125
 XIB control block 123
 XIBAREA 47, 55, 64, 123
 XIBD 123
 XIBDS 123–124
 XIBDS control block 124
 XML descriptions 421
 XML 3, 223, 275, 311, 323, 329–336, 338–339, 343, 346–349, 354–355, 420–421, 425
 XML data 3, 330, 336, 338, 354, 421, 425
 XRF 20

Z

z/OS 7–8, 15, 22, 26–27, 29–31, 34–36, 38–40, 44–46, 48, 64, 73, 78, 82, 84, 106, 119, 159–160, 165, 168, 173, 177–178, 184, 205, 217, 224, 230, 236–238, 242, 250, 253, 281, 283–284, 292, 298, 320, 322, 325, 355, 359–364, 366, 368, 370–373, 375–376, 378–380, 384, 386, 391, 398–400, 404–406, 421–422, 428–435, 437–441, 446–449, 451, 455–456, 458, 460–461, 463,

466–467, 487
z/OS UNIX System Services 319–320

Archived



Redbooks

IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity

(1.0" spine)

0.875" <-> 1.498"

460 <-> 788 pages



Redbooks

IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity

Become familiar with IMS OTMA and IMS Connect details and usage

Explore IMS MFS Web Services and IMS SOAP Gateway

Introduce yourself to ODBA, stored procedures, and IMS RDS

IBM Information Management System (IMS) is the IBM premier transaction and hierarchical database management system. Connectivity has always been a priority with IMS. IMS exploits the latest technologies to address customers' requirements for accessing IMS transactions and data. This IBM Redbook is about IMS connectivity.

This book provides a general overview of the IMS Open Transaction Manager Access (OTMA) function and extensive information about IMS Connect and its usage, including a chapter that describes the IMS Connect Extensions product and how you can enhance the IMS Connect operating environment with it.

This book provides a broad understanding of IMS Connector for Java. We cover some special considerations, such as using the conversational transactions, rerouting, and timeout support, as well as programming roll-your-own clients without using IMS Connector for Java.

We also introduce Open Database Access and provide examples of using it with stored procedures and with IMS Remote Database Services. As for future directions, we also include a chapter about the IMS SOAP Gateway. This book updates and adds to the information in the previous IBM Redbook *IMS e-business Connectors: A Guide to IMS Connectivity*, SG24-6514.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-6794-00

ISBN 0738494224