IBM Tivoli NetView for z/OS
Version 6 Release 1

Resource Object Data Manager and GMFHS Programmer's Guide

SC27-2862-00

IBM Tivoli NetView for z/OS
Version 6 Release 1

Resource Object Data Manager and GMFHS Programmer's Guide

SC27-2862-00

Note

Before using this information and the product it supports, read the information in “Notices” on page 671.

This edition applies to version 6, release 1 of IBM Tivoli NetView for z/OS (product number 5697-NV6) and to all subsequent versions, releases, and modifications until otherwise indicated in new editions.

Contents

Figures xi

About this publication xv

Intended audience xv
Publications xv
  IBM Tivoli NetView for z/OS library xv
  Related publications xvii
  Accessing terminology online xvii
  Using NetView for z/OS online help xviii
  Using LookAt to look up message explanations xviii
  Accessing publications online xix
  Ordering publications. xix
Accessibility xix
Tivoli technical training xix
Tivoli user groups xx
Downloads xx
Support information xx
Conventions used in this publication xxi
  Typeface conventions xxi
  Operating system-dependent variables and paths xxi
  Syntax diagrams xxii

Part 1. Learning About RODM 1

Chapter 1. Overview 3

Managing SNA Resources with NetView 3
Defining Non-SNA Resources to NetView 3
  Resource Definition Task 4
  Resources Supported by GMFHS 5
  Saving RODM Data 5
RODM in Network Automation 5
  Automation Concepts 6
  Automation Example 7
  For More Information 7
RODM Programming Tasks 7
  RODM Transactions 7
  RODM Functions 8
  Programming Languages 9
  RODM Notification Process 9
  RODM Load Function 9
Additional RODM Documentation. 10
Tools for RODM 11
RODM Samples and Macros 11

Part 2. Defining Resources to NetView 13

Chapter 2. Defining Your Network to GMFHS 17

Manual Network Definition Overview 17
Sample Network. 18
  SNA Components of the Sample Network 18
  Non-SNA Components of the Sample Network 19
Identifying Network Elements to Define. 22
  Identifying Management Objects 22
  Identifying Managed Objects 23

Identifying Connectivity Relationships . . . 25
Identifying Views . . . 28
Defining Your Configuration to RODM . . . 33
Defining Management Objects . . . 33
Defining Managed Objects . . . 36
Defining Connectivity Relationships Between Objects . . . 40
Defining Views . . . 41
Defining Layout Parameters . . . 46
Putting It All Together . . . 54

Chapter 3. Loading the GMFHS Data Model . . . 57
Loading the Data Models and Network Definitions . . . 57
Changing Network Definitions When GMFHS Is Running . . . 57
Selecting the Required GMFHS CONFIG Command . . . 58
Adding NMGs and Domains When GMFHS Is Active . . . 60

Chapter 4. Communicating with Network Management Gateways . . . 61
Defining Non-SNA Presentation Protocol . . . 62
DOMP010 Presentation Protocol . . . 62
DOMP020 Presentation Protocol . . . 63
PASSTHRU Presentation Protocol . . . 64
NONE Presentation Protocol. . . . 64
Output Formatting For All Presentation Protocols . . . 64
DOMP010 Formatting Rules . . . 65
Command Formatting and Protocol Examples . . . 72
Timing Considerations. . . . 74
Defining Non-SNA Session Protocols . . . 75
DOMS010 . . . 75
PASSTHRU . . . 76
NONE . . . 76
Session Establishment for DOMS010 . . . 76
Session Establishment for DOMS010 Using CNMS4406 . . . 76
GMFHS-Initiated Session Establishment . . . 77
INIT Generic Alert for Session Establishment . . . 78
Session Termination . . . 80
Defining Non-SNA Transport Protocols . . . 81
COS Gateway Support. . . . 81
Program-to-Program Interface Gateway . . . 82
OST/PPT Gateway . . . 82
Monitoring Non-Network Devices . . . 83
Types of NMGs . . . 83

Chapter 5. How GMFHS Uses RODM . . . 87
GMFHS Initialization . . . 87
Aggregation Warm Start . . . 87
Resource Status Warm Start . . . 87
GMFHS Initialization Process Overview . . . 88
Monitoring Topology Managers . . . 89
Building Views . . . 89
Object Discovery Process . . . 89
Object Connectivity Process . . . 100
Defining Exception View Objects and Criteria . . . 100
Locate Resource Function . . . 113
Restricting Recursive Views . . . 113
Refreshing Open Views . . . 114
Applying Span-of-Control to Views . . . 114
Views . . . 115
Resources. . . . 117
Helpful Hints . . . 120
Applying Span-of-Control to Set and Clear Operator Status . . . 121

Applying Policy to Views . . . 122
  Representing Policy Definitions in RODM . . . 122
  Resources Belonging to Multiple Policies . . . 124
  Resources Suspended from Aggregation Due to Policy . . . 127
  Suspending Aggregation Using an Aggregate. . . . 128
  System Status Updates No Longer Sent to Resources Due to Policy . . . 129
  Additional Information . . . 129
Aggregation Concepts . . . 130
  Aggregation Overview . . . 130
  Creating an Aggregation Hierarchy . . . 131
  Building the Aggregation Hierarchy in RODM . . . 132
  Updating Status . . . 133
  Status Groups . . . 142
  Using Status Groups . . . 142
  Examples of Customizing Aggregate DisplayStatus . . . 143
Using the Collection Definition Objects . . . 143
  Collection Definition Objects . . . 144
  Using Collection Specifications . . . 145
  Examples of Collection Definition Objects . . . 155
Using NetView Resource Manager . . . 159
  NetView Resource Manager Views . . . 159
  Modifying DUIFSMT for NetView Resource Manager . . . 164
  Using DUIFVINS with NetView Resource Manager. . . . 164
  NetView Resource Manager Sample Loader Files . . . 164

Chapter 6. Customizing GMFHS to Process and Receive Alerts and Resolutions . . . 167

Receiving and Monitoring Alerts or Resolutions . . . 167
  What GMFHS Receives from the Hardware Monitor . . . 167
  Objects in RODM Representing SNA Resources . . . 168
  Objects in RODM Representing NMGs . . . 169
  Objects in RODM Representing Non-SNA Domains . . . 169
  Objects in RODM Representing Non-SNA Resources . . . 171
DUIFEDEF Alert Processing . . . 172
  Parameters . . . 172
  Alert Translation Tables . . . 176

Part 3. Using RODM for Network Automation . . . 179

Chapter 7. Writing Automation Code . . . 181

Advantages of Using the Supplied Data Models for Automation . . . 181
Notifying Your Application about Changes in GMFHS Fields . . . 181
Accessing and Changing GMFHS-Defined Fields . . . 182
Using GMFHS Methods . . . 183
  DUIFCCAN: Clear All Notes . . . 183
  DUIFCATC: Aggregation Threshold Change . . . 183
  DUIFCLRT: Link Resource Type . . . 183
  DUIFCUAP: Update Aggregation Path . . . 183
  DUIFCUUS: Update User Status . . . 184
  DUIFECDS: Change Display Status . . . 184
  DUIFFAWS: Aggregation Warm Start . . . 184
  DUIFFIRS: Set Initial Resource Status . . . 184
  DUIFFRAS: Recalculate Aggregate Status . . . 184
  DUIFFSUS: Set Unknown Status . . . 184
  DUIFRFDS: Refresh DisplayStatus Change Method DUIFCRDC . . . 185
  DUIFVCFT: Change Exception State . . . 185
  DUIFVINS: Install View Notification Granularity Method . . . 185
  GMFHS Methods That Cannot Be Used . . . 185
GMFHS Automation Example . . . 185
  Sample Automation Application and Method. . . . 186

Chapter 8. Using the RODM Automation Platform 189

RODM Automation Platform Services 189
  Sample Automation Code 190

Part 4. Application Programming Using RODM 191

Chapter 9. Understanding RODM Concepts 195

RODM Classes 195
  Class Names 195
  System-Defined Classes 196
RODM Objects 208
  Object Names 208
  Object Identifiers 210
RODM Fields 210
  Field Names. 210
  Field Identifiers. 211
  System-Defined Fields 211
RODM Subfields 213
  Data Types for Subfields. 215
Multivalued Fields and Links between Objects 216
  Link and Unlink Action Functions 218
  Subfields Associated with Fields 219
Indexed Fields 220
Object and Class Locking in RODM 220
Using the Application Program Interfaces 220
  User Application Program Interface (API) 220
  Method Application Program Interface (API) 221
RODM Abstract Data Types 221
  Null Values of Data Type 222
  Data Type Identifiers 222
  Types of Data in Fields 222
  Abstract Data Type Reference 223

Chapter 10. Using the RODM Load Function 239

Considerations When Designing a Data Model 239
Introduction to the RODM Load Function 240
  Load Function Statements 240
  Load Function Operations 240
  Loading the RODM Data Cache 241
Using Load Function Statements 241
  High-Level Load Function Statements 242
  Load Function Primitive Statements 242
  When to Use High-Level or Primitive Load Function Statements 243
Process for Loading the RODM Data Cache 244
  Identifying the Methods to Install 245
  Creating the Class Structure and Object Definitions. 245
  Deciding on the Type of Load 246
  Running the RODM Load Function 248
  Checking the Output Listings 253
Load Function Reference 258
  Understanding the Verify Operation. 258
  Using CLASSID and OBJECTID Data Types 259
  Null Values for RODM Load Function Data Types 260
  Control Table—EKGCTABL. 260
  Method Name Table 261
  Parameter Mapping Table 262
  RODM Data Definition (DD) Statements 264
  z/OS Linkage Conventions. 266
  RODM Load Function Parameter Syntax 269
  Coding RODM High-Level Load Function Statements. 273

Coding RODM Load Function Primitive Statements . . . 282
Common Syntactic Elements . . . 291

Chapter 11. Writing Applications that Use RODM . . . 301

Tasks Best Performed with User Applications. . . . 301
Using the User Application Program Interface . . . 302
Register Conventions . . . 302
Usage Notes. . . . 302
Compiling and Link-Editing . . . 303
Using Control Blocks . . . 304
Access Block . . . 305
Transaction Information Block . . . 307
Function Block . . . 308
Entity Access Information Block . . . 309
Field Access Information Block . . . 312
Response Block. . . . 314
Error Conditions in Transactions . . . 317
RODM Notification Process . . . 318
Setup . . . 319
Wait . . . 321
Notification . . . 324
Clean Up . . . 325
Asynchronous Error Notification . . . 325
Object Deletion Notification . . . 326
Setup for Object-Deletion Notification . . . 326
Wait for Object-Deletion Notification . . . 327
Notification for Object-Deletion Notification . . . 327
Cleanup for Object-Deletion Notification . . . 327
Connecting to RODM . . . 327
Disconnecting from RODM . . . 328

Chapter 12. Topology Object Correlation. . . . 329

Enabling the Correlation Function . . . 329
Enabling SNA Topology Manager Object Correlation . . . 329
Enabling GMFHS Object Correlation . . . 330
Correlation Concepts . . . 330
Correlation Methods . . . 330
Objects Enabled for Correlation . . . 331
Types of Correlation . . . 331
Correlated Aggregate Object Classes and Names . . . 333
Correlated Object Relationships . . . 333
Correlated Aggregate Object Display Labels . . . 333
Correlated Aggregate Object Field Values . . . 334
Using Correlation for Objects You Create . . . 335
Extending Correlation of Objects Created by MultiSystem Manager and SNA Topology Manager . . . 335
How to Determine Object Names. . . . 336
Correlating MultiSystem Manager Objects . . . 336
Correlating SNA Topology Manager Objects . . . 336
Customizing the Correlation Function . . . 336
Changing the Display Name Priority . . . 337
Disabling Correlation for Specific Resources . . . 338

Chapter 13. Writing RODM Methods. . . . 339

Tasks Best Performed with Methods . . . 339
Types of Methods . . . 340
Object-Independent Methods . . . 340
Object-Specific Methods . . . 342
Null Method . . . 352
Deciding Which Method Type to Use . . . 352
When to Use an Object-Independent Method . . . 352

When to Use an Object-Specific Method . . . 352
Using the Method API . . . 353
Register Conventions . . . 354
Usage Notes . . . 355
Method Parameters . . . 355
Installing and Freeing Methods . . . 356
Synchronous and Asynchronous Execution of Functions . . . 357
Method Anchor Service . . . 357
Coding Your RODM Method . . . 358
Installation Written Methods . . . 358
NetView Methods . . . 358
Programming Language Specific Preprocessor Statements . . . 359
Restrictions on Methods . . . 360
RODM Method Services . . . 363
Services Available to both Object-Specific and Object-Independent Methods . . . 363
Other Services Available to Object-Independent Methods . . . 364
Other Services Available to Object-Specific Methods . . . 364
Services Available to the Initialization Method . . . 364
RODM Method Library . . . 365

Chapter 14. Application Programming Reference . . . 367

Summarizing RODM Functions . . . 367
Access Functions . . . 367
Control Functions . . . 367
Administrative Functions . . . 367
Action Functions . . . 368
Query Functions . . . 369
RODM User API Services . . . 370
RODM Method API Services . . . 370
Function Reference . . . 371
Function Reference Format . . . 371
EKG_AddNotifySubscription — Add Notification Subscription . . . 373
EKG_AddObjDelSubs — Add Object Deletion Subscription . . . 374
EKG_ChangeField — Change a Field . . . 376
EKG_ChangeMultipleFields — Change Multiple Fields . . . 377
EKG_ChangeSubfield — Change a Subfield . . . 378
EKG_Checkpoint — Checkpoint RODM to DASD . . . 380
EKG_Connect — Connect to RODM . . . 383
EKG_ConnectLong — Connect to RODM . . . 384
EKG_CreateClass — Create a Class . . . 386
EKG_CreateField — Create a Field . . . 387
EKG_CreateObject — Create an Object . . . 388
EKG_CreateSubfield — Create a Subfield . . . 390
EKG_DeleteClass — Delete a Class . . . 391
EKG_DeleteField — Delete a Field . . . 392
EKG_DeleteNotifySubscription — Delete Notification Subscription . . . 393
EKG_DeleteObject — Delete an Object . . . 395
EKG_DeleteSubfield — Delete a Subfield . . . 396
EKG_DelObjDelSubs — Delete Object Deletion Subscription . . . 397
EKG_Disconnect — Disconnect from RODM . . . 399
EKG_ExecuteFunctionList — Execute a List of Functions . . . 400
EKG_LinkNoTrigger, EKG_LinkTrigger — Link Two Objects . . . 402
EKG_Locate—Locate Objects Using Public Indexed Field . . . 404
EKG_LockObjectList — Lock List of Objects . . . 405
EKG_MessageTriggeredAction — Trigger an Action by a Message . . . 406
EKG_OutputToLog — Output to Log . . . 408
EKG_QueryEntityStructure — Query Structure of an Entity . . . 409
EKG_QueryField — Query a Field . . . 411
EKG_QueryFieldID — Query Field Identifier . . . 412
EKG_QueryFieldName — Query a Field Name . . . 413
EKG_QueryFieldStructure — Query Structure of a Field . . . 415

EKG_QueryFunctionBlockContents — Query Function Block Contents 416
EKG_QueryMultipleSubfields — Query Multiple Value Subfields 418
EKG_QueryNotifyQueue — Query Notification Queue 421
EKG_QueryObjectName — Query Object Name 423
EKG_QueryResponseBlockOverflow — Query for Response Block Overflow 424
EKG_QuerySubfield — Query a Subfield 426
EKG_ResponseBlock — Output to Response Block 428
EKG_RevertToInherited — Revert to Inherited Value 429
EKG_SendNotification — Send a Notification 431
EKG_SetReturnCode — Set Return and Reason Codes. 432
EKG_Stop — Stop RODM 434
EKG_SwapField — Swap a Field 435
EKG_SwapSubfield — Swap a Subfield. 437
EKG_TriggerNamedMethod — Trigger a Named Method. 438
EKG_TriggerOIMethod — Trigger an Object-Independent Method. 440
EKG_UnlinkNoTrigger, EKG_UnlinkTrigger — Unlink Two Objects 441
EKG_UnlockAll — Unlock All Held Entities 443
EKG_WhereAmI — Where Am I 444
Function Parameter Descriptions 445
RODM Return and Reason Codes 452
Reason Codes for Return Code 0 453
Reason Codes for Return Code 4 453
Reason Codes for Return Code 8 457
Reason Codes for Return Code 12 467
List of Reason Codes for Each Function 470
List of Functions for Each Reason Code 472
List of Function Names by Function ID. 478
List of Reason Codes from Supplied Methods 479
Maximizing RODM Performance 480
Data Model Structure and Size 480
Method Design 480
User Application Design. 480
Customization Parameters and System Fields. 480
Indexed Fields 480
Supplied Methods 480
RODM Notification Methods 481
RODM Change Methods 484
RODM Named Methods. 485
RODM Object-Independent Methods 485
GMFHS Methods 488

Part 5. Appendixes 501

Appendix A. RODM Tools 503
RODMView 503
Navigating within RODMView 503
RODMView Restrictions. 504
Starting RODMView 505
Access and Control Function 506
Simple Query Function 508
Compound Query Function 514
Locate Objects Function 521
Link/Unlink Function 524
Change Field Function 527
Subfield Actions Function 530
Create Actions Function 531
Delete Actions Function 533
Method Actions Function 535
RODM Unload Function. 537
Starting the RODM Unload Function 538

Customizing the RODM Unload Function . . . 538
Running the RODM Unload Function . . . 540
FLCARODM . . . 541
Overview. . . . 542
Stem Building Subroutines . . . 542
About the Examples . . . 548
FLCARODM Command . . . 548
FLCARODM Functions . . . 553
Putting It All Together . . . 565
Result Stem . . . 571
Return Codes . . . 579
Object Data Stream Detail . . . 581
RODM Collection Manager . . . 585
Visual BLDVIEWS . . . 586
BLDVIEWS . . . 586
Before You Begin . . . 587
BLDVIEWS Processing . . . 587
BLDVIEWS Control Statements . . . 588
Running BLDVIEWS . . . 649
BLDVIEWS Control Statement Examples . . . 652
Deleting Views . . . 654
DELVIEWS . . . 655

Appendix B. View Layout Facility . . . 657
View Layout Examples . . . 657
Choosing a View Layout Type . . . 660
GMFHS Fields Used By the View Layout Facility . . . 661
Layout Type Descriptions . . . 661
Radial Layout View by Link Type . . . 662
Radial Layout View by Cluster ID . . . 662
Local Area Network Layout View . . . 663
Token-Ring Network Layout View Interface . . . 664
Bus Network Layout View Interface . . . 664
Hierarchical Graph Layout View . . . 665
Elliptical Layout View . . . 665
Connectivity Tree Layout View . . . 666
Grid Layout . . . 667
Grid Layout Notes . . . 668

Notices . . . 671
Programming Interfaces . . . 672
Trademarks . . . 672

Index . . . 675

Figures

1. Using RODM to Support the NetView management console 4
2. Sample Network 18
3. Sysplex Network 20
4. Ethernet Network 20
5. Token-Ring LAN 21
6. ITNM Network 22
7. Exception View Example 29
8. High-Level View BIGPIC 30
9. Management View SAMPNET 31
10. Peer View of ETHERNET Network. 32
11. Exception View of a Network 42
12. Network View of the Sysplex Network 43
13. Peer View of Token-Ring Network TRLANNET 45
14. Defining Layout Parameters for More Detail Views 51
15. Defining Layout Parameters for Objects in More Detail Views 54
16. Single-Response Protocol 72
17. Multiple-Response Protocol 73
18. Session Establishment at the Request of the NMG 76
19. Session Establishment at the Request of GMFHS 78
20. Session Termination 80
21. Technique for Linking Display_Resource_Type_Class Objects Prior to NetView Version 3 90
22. Technique for Linking Display_Resource_Type_Class Objects Now . . 91
23. View_Information_Object_Class Object Determination Technique One 92
24. View_Information_Object_Class Object Determination Technique Two 93
25. Sample Table DUIFSMT 102
26. Macro DUIFSMTE Syntax 105
27. Customizing a Resource 109
28. Example of a MYNAME and RESOURCE Keyword in the Same DUIFSMTE Entry . . 110
29. DisplayStatus Mapping Table Coding Example 1 110
30. DisplayStatus Mapping Table Coding Example 2. 111
31. Aggregation Example Using Real (R) and Aggregate (A) Objects. 131
32. Links Between AggregationChild and AggregationParent Fields 133
33. Example DUIFSMTE Statements in Table DUIFSMT. 135
34. Example of Customizing Aggregate Display Status 143
35. Resources Properties Notebook 160
36. Other Data Field 161
37. RODM System-Defined Classes 196
38. Examples of Links between Objects in RODM 217
39. RODM System Structure (z/OS) 221
40. Format of BER Data 224
41. Identifier Byte in Short Form 225
42. Identifier Byte in Long Form 225
43. Length Byte in Short Form 225
44. Length Byte in Long Form 226
45. Example IndexList Field 230
46. SelfDefining Data Type Syntax 234
47. Example SelfDefining Field 235
48. Adding Objects and Classes. 239
49. Data Set Concatenation for EKGIN1 248
50. Data Set Concatenation for EKGIN3 248
51. Object Load Batch Job Using EKGLLOAD Sample. 250
52. Class and Method Load Batch Job Using EKGLLOAD Sample 251
53. Example of PARSE Operation Output to EKGPRINT 255
54. Example of Structure Load Output to EKGPRINT 256
55. Example of Object Load Output to EKGPRINT 257
56. Sample Control Table EKGCTABL with Column Scale 260
57. Relationship between EKGCTABL, EKGINMTB, EKGPTENU and JCL 261
58. Method Name Table Format with Column Scale 262
59. Sample Control Table EKGCTABL with Column Scale 262
60. Sample Parameter Table EKGPTENU with Column Scale 264
61. z/OS Linkage Conventions Required for Module Call to EKGLJOB 267
62. Calling the RODM Load Function from a PL/I Program 268
63. Hierarchical Pseudo-Structure for Examples 275
64. High-Level Input Statements for Pseudo-Structure 276
65. Create Object Example 279
66. Delete Object Example 280
67. Set Value of Fields in an Object Example 281
68. Typical User API Invocation in C and PL/I 302
69. API Query Function Control Block Example 305
70. PL/I Coding Example 322
71. C Coding Example. 323
72. Correlate Objects on Multiple Free-Form Values 332
73. Aggregate Resource Symbol 333
74. Default Display Name Priority. 337
75. Customized Display Name Priority 338
76. Object-Independent Method Procedure Interface for PL/I 341
77. Object-Independent Method Procedure Interface for C 341
78. Change Method Procedure Interface for PL/I 344

79. Change Method Procedure Interface for C 344
80. Query Method Procedure Interface for PL/I 346
81. Query Method Procedure Interface for C 346
82. Notification Method Procedure Interface for PL/I 348
83. Notification Method Procedure Interface for C 348
84. Named Method Procedure Interface for PL/I 350
85. Named Method Procedure Interface for C 350
86. Method API Interface Declaration and Invocation Example 354
87. Method API Query Field Control Block Sample. 355
88. Example RODM Load Function Primitive Statement to Invoke EKGSPPI 488
89. RODM Load Function Primitive Statement Invoking DUIFCLRT 490
90. RODM load function primitive statement invoking DUIFCUAP 492
91. RODMView NetView Command Line Call 505
92. RODMView Main Menu — EKGVMMNI 506
93. RODMView Access and Control Panel — EKGVACTI 506
94. RODMView Message for a Successful Connection 507
95. RODMView Query Panel — EKGVQUEI 508
96. RODMView Querying Your User ID 509
97. RODMView Query Output Panel 509
98. RODMView Simple Query Specifying SystemView Class and Field Names 511
99. RODMView Simple Query-Translated SystemView Textual Class and Field Names . 512
100. RODMView Query for Fields That Contain the Word Log 513
101. RODMView Query Output for Fields Containing 'Log' 513
102. RODMView Excessively Large Query Output 514
103. RODMView Compound Query Panel 1 — EKGVQA1I 515
104. RODMView Query Criteria Panel 2 — EKGVQA2I 515
105. RODMView Query Traversed Criteria Panel 3 — EKGVQA3I 516
106. RODMView Query Field Selection Panel 4 — EKGVQA4I 516
107. Starting a Compound Query on the GMFHS_Aggregate_Objects_Class 517
108. Selecting Only Those Entities that Have Nonsatisfactory DisplayStatus 517
109. Selecting Only the DisplayResourceName Field to be Displayed 518
110. Compound Query Example 1 Output 518
111. Starting a Compound Query on the GMFHS_Aggregate_Objects_Class 519
112. Selecting Only Those Entities Having a Satisfactory DisplayStatus 519
113. Traversing Across the ComposedOfPhysical Link Field and Adding DisplayStatus Criteria. 520
114. Selecting Only the DisplayResourceName Field to be Displayed 520
115. Query Output Example 2 521
116. Locate Objects Panel 521
117. Locating Objects with an Indexed CharVar Field 522
118. Locate Objects Output 523
119. Locating Objects with Number of Objects and No Object Detail 523
120. Locate Objects Output, No Object Detail 524
121. RODMView Link Objects Panel — EKGVLNKI 524
122. RODMView Linking Two Objects 525
123. RODMView Linking a GMFHS Aggregate Object To Its Resource Type. 526
124. Updating the Aggregation Path Between NETVIEW.T46A and ITNM 527
125. RODMView Change Field Panel — EKGVCHGI 527
126. RODMView Changing a Field 528
127. Adding Multiple Values to an IndexList Field in Character Format 529
128. RODMView Subfield Actions Panel — EKGVSUBI 530
129. RODMView Creating a Notify Subfield 531
130. RODMView Create Actions Panel — EKGVCREI 532
131. RODMView Creating an Object 533
132. RODMView Creating a Field 533
133. RODMView Delete Actions Panel — EKGVDELI 534
134. RODMView Deleting a Field from a Class 535
135. RODMView Method Actions Panel — EKGVMETI 536
136. RODMView Triggering a Named Method 536
137. RODMView Return and Reason Codes From a Triggered Method 537
138. Sample JCL for EKGIN1 538
139. Sample SYSIN DD file of the JCL.. 539
140. EKGKUJCL SYSIN Parameters to Unload RODM Completely 540
141. EKGKUJCL SYSIN Parameters to Unload Network Monitorable Objects 541
142. EKGKUJCL SYSIN Parameters to Unload an Object When Class is Unknown 541
143. EKGKUJCL SYSIN Parameters to Unload an Object When Class is Known 541
144. EKGKUJCL SYSIN Parameters to Determine Object Definitions for Two Classes 541
145. Issuing the FLCARODM Command 548
146. Sample FLCSX6. 566
147. Sample FLCSX7. 566
148. Sample FLCSX14 567
149. Sample FLCSX15 567
150. Sample FLCSX16 568
151. Sample FLCSX17 568
152. Sample FLCSX18 568
153. Sample FLCSXL02 569
154. Sample FLCSXF1 569
155. Sample FLCSX10 570
156. Sample FLCSX9. 570
157. Sample FLCSX19 570
158. Sample FLCSX11 571
159. Sample FLCSX8. 571
160. Sample FLCSX22 571

161. Radial Layout Example 657
162. Token-Ring Layout Example 658
163. LAN Net Layout Example 658
164. LAN Bus Layout Example 658
165. Ellipse Layout Example 659
166. Hierarchical Graph Layout Example 659
167. Connectivity Tree Layout Example 659
168. Grid Layout Example 660

About this publication

The IBM® Tivoli® NetView® for z/OS® product provides advanced capabilities that you can use to maintain the highest degree of availability of your complex, multi-platform, multi-vendor networks and systems from a single point of control. This publication, the *IBM Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer's Guide*, describes the NetView Resource Object Data Manager (RODM). It describes how to define your non-SNA network to RODM and manage your network (non-SNA, SNA resources, or both) using the NetView management console. This publication also describes how to implement network automation using RODM. Finally, this publication describes the use of RODM for application programming.

Intended audience

This publication is for system programmers who need to define their non-SNA networks to RODM; create or modify RODM applications, methods, and data models; and plan how to automate their networks using RODM.

Publications

This section lists publications in the IBM Tivoli NetView for z/OS library and related documents. It also describes how to access Tivoli publications online and how to order Tivoli publications.

IBM Tivoli NetView for z/OS library

The following documents are available in the IBM Tivoli NetView for z/OS library:

- *Administration Reference*, SC27-2869, describes the NetView program definition statements required for system administration.
- *Application Programmer's Guide*, SC27-2870, describes the NetView program-to-program interface (PPI) and how to use the NetView application programming interfaces (APIs).
- *Automation Guide*, SC27-2846, describes how to use automated operations to improve system and network efficiency and operator productivity.
- *Command Reference Volume 1 (A-N)*, SC27-2847, and *Command Reference Volume 2 (O-Z)*, SC27-2848, describe the NetView commands, which can be used for network and system operation and in command lists and command procedures.
- *Customization Guide*, SC27-2849, describes how to customize the NetView product and points to sources of related information.
- *Data Model Reference*, SC27-2850, provides information about the Graphic Monitor Facility host subsystem (GMFHS), SNA topology manager, and MultiSystem Manager data models.
- *Installation: Configuring Additional Components*, GC27-2851, describes how to configure NetView functions beyond the base functions.
- *Installation: Configuring Graphical Components*, GC27-2852, describes how to install and configure the NetView graphics components.
- *Installation: Configuring the GDPS Active/Active Continuous Availability Solution*, SC14-7477, describes how to configure the NetView functions that are used with the GDPS Active/Active Continuous Availability solution.

- *Installation: Configuring the NetView Enterprise Management Agent*, GC27-2853, describes how to install and configure the NetView for z/OS Enterprise Management Agent.
- *Installation: Getting Started*, GI11-9443, describes how to install and configure the base NetView functions.
- *Installation: Migration Guide*, GC27-2854, describes the new functions that are provided by the current release of the NetView product and the migration of the base functions from a previous release.
- *IP Management*, SC27-2855, describes how to use the NetView product to manage IP networks.
- *Messages and Codes Volume 1 (AAU-DSI)*, GC27-2856, and *Messages and Codes Volume 2 (DUI-IHS)*, GC27-2857, describe the messages for the NetView product, the NetView abend codes, the sense codes that are included in NetView messages, and generic alert code points.
- *Programming: Assembler*, SC27-2858, describes how to write exit routines, command processors, and subtasks for the NetView product using assembler language.
- *Programming: Pipes*, SC27-2859, describes how to use the NetView pipelines to customize a NetView installation.
- *Programming: PL/I and C*, SC27-2860, describes how to write command processors and installation exit routines for the NetView product using PL/I or C.
- *Programming: REXX and the NetView Command List Language*, SC27-2861, describes how to write command lists for the NetView product using the Restructured Extended Executor language (REXX) or the NetView command list language.
- *Resource Object Data Manager and GMFHS Programmer's Guide*, SC27-2862, describes the NetView Resource Object Data Manager (RODM), including how to define your non-SNA network to RODM and use RODM for network automation and for application programming.
- *Security Reference*, SC27-2863, describes how to implement authorization checking for the NetView environment.
- *SNA Topology Manager Implementation Guide*, SC27-2864, describes planning for and implementing the NetView SNA topology manager, which can be used to manage subarea, Advanced Peer-to-Peer Networking, and TN3270 resources.
- *Troubleshooting Guide*, GC27-2865, provides information about documenting, diagnosing, and solving problems that occur in the NetView product.
- *Tuning Guide*, SC27-2874, provides tuning information to help achieve certain performance goals for the NetView product and the network environment.
- *User's Guide: Automated Operations Network*, SC27-2866, describes how to use the NetView Automated Operations Network (AON) component, which provides event-driven network automation, to improve system and network efficiency. It also describes how to tailor and extend the automated operations capabilities of the AON component.
- *User's Guide: NetView*, SC27-2867, describes how to use the NetView product to manage complex, multivendor networks and systems from a single point.
- *User's Guide: NetView Enterprise Management Agent*, SC27-2876, describes how to use the NetView Enterprise Management Agent.
- *User's Guide: NetView Management Console*, SC27-2868, provides information about the NetView management console interface of the NetView product.
- *Licensed Program Specifications*, GC31-8848, provides the license information for the NetView product.

- *Program Directory for IBM Tivoli NetView for z/OS US English*, GI11-9444, contains information about the material and procedures that are associated with installing the IBM Tivoli NetView for z/OS product.
- *Program Directory for IBM Tivoli NetView for z/OS Japanese*, GI11-9445, contains information about the material and procedures that are associated with installing the IBM Tivoli NetView for z/OS product.
- *Program Directory for IBM Tivoli NetView for z/OS Enterprise Management Agent*, GI11-9446, contains information about the material and procedures that are associated with installing the IBM Tivoli NetView for z/OS Enterprise Management Agent.
- *IBM Tivoli NetView for z/OS V6R1 Online Library*, LCD7-4913, contains the publications that are in the NetView for z/OS library. The publications are available in PDF, HTML, and BookManager® formats.

Related publications

You can find additional product information on the NetView for z/OS web site at http://www.ibm.com/software/tivoli/products/netview-zos/.

For information about the NetView Bridge function, see *Tivoli NetView for OS/390 Bridge Implementation*, SC31-8238-03 (available only in the V1R4 library).

Accessing terminology online

The IBM Terminology web site consolidates the terminology from IBM product libraries in one convenient location. You can access the Terminology web site at http://www.ibm.com/software/globalization/terminology/.

For NetView for z/OS terms and definitions, see the IBM Terminology web site. The following terms are used in this library:

NetView
For the following products:
- Tivoli NetView for z/OS version 6 release 1
- Tivoli NetView for z/OS version 5 release 4
- Tivoli NetView for z/OS version 5 release 3
- Tivoli NetView for z/OS version 5 release 2
- Tivoli NetView for z/OS version 5 release 1
- Tivoli NetView for OS/390® version 1 release 4

CNMCMD
For the CNMCMD member and the members that are included in it using the %INCLUDE statement

CNMSTYLE
For the CNMSTYLE member and the members that are included in it using the %INCLUDE statement

PARMLIB
For SYS1.PARMLIB and other data sets in the concatenation sequence

MVS™ For z/OS operating systems

MVS element
For the base control program (BCP) element of the z/OS operating system

VTAM®
For Communications Server - SNA Services

IBM Tivoli Network Manager

For either of these products:
- IBM Tivoli Network Manager
- IBM Tivoli OMNIbus and Network Manager

IBM Tivoli Netcool/OMNIbus

For either of these products:
- IBM Tivoli Netcool/OMNIbus
- IBM Tivoli OMNIbus and Network Manager

Unless otherwise indicated, references to programs indicate the latest version and release of the programs. If only a version is indicated, the reference is to all releases within that version.

When a reference is made about using a personal computer or workstation, any programmable workstation can be used.

Using NetView for z/OS online help

The following types of NetView for z/OS mainframe online help are available, depending on your installation and configuration:
- General help and component information
- Command help
- Message help
- Sense code information
- Recommended actions

Using LookAt to look up message explanations

LookAt is an online facility that you can use to look up explanations for most of the IBM messages you encounter, and for some system abends and codes. Using LookAt to find information is faster than a conventional search because, in most cases, LookAt goes directly to the message explanation.

You can use LookAt from the following locations to find IBM message explanations for z/OS elements and features, z/VM®, VSE/ESA, and Clusters for AIX® and Linux systems:
- The Internet. You can access IBM message explanations directly from the LookAt web site at http://www.ibm.com/systems/z/os/zos/bkserv/lookat/.
- Your z/OS TSO/E host system. You can install code on your z/OS or z/OS.e system to access IBM message explanations, using LookAt from a TSO/E command line (for example, TSO/E prompt, ISPF, or z/OS UNIX System Services running OMVS).
- Your Microsoft Windows workstation. You can install LookAt directly from the *z/OS Collection* (SK3T-4269) or the *z/OS and Software Products DVD Collection* (SK3T-4271) and use it from the resulting Windows graphical user interface (GUI). The command prompt (also known as the DOS command line) version can still be used from the directory in which you install the Windows version of LookAt.
- Your wireless handheld device. You can use the LookAt Mobile Edition from http://www.ibm.com/systems/z/os/zos/bkserv/lookat/lookatm.html with a handheld device that has wireless access and an Internet browser.

You can obtain code to install LookAt on your host system or Microsoft Windows workstation from the following locations:
- A CD in the *z/OS Collection* (SK3T-4269).

- The *z/OS and Software Products DVD Collection* (SK3T-4271).
- The LookAt web site. Click **Download** and then select the platform, release, collection, and location that you want. More information is available in the LOOKAT.ME files that is available during the download process.

Accessing publications online

The documentation DVD, *IBM Tivoli NetView for z/OS V6R1 Online Library*, SK2T-6175, contains the publications that are in the product library. The publications are available in PDF, HTML, and BookManager formats. Refer to the readme file on the DVD for instructions on how to access the documentation.

IBM posts publications for this and all other Tivoli products, as they become available and whenever they are updated, to the Tivoli Information Center web site at http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/index.jsp.

Note: If you print PDF documents on other than letter-sized paper, set the option in the **File → Print** window that enables Adobe Reader to print letter-sized pages on your local paper.

Ordering publications

You can order many Tivoli publications online at http://www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss

You can also order by telephone by calling one of these numbers:
- In the United States: 800-879-2755
- In Canada: 800-426-4968

In other countries, contact your software account representative to order Tivoli publications. To locate the telephone number of your local representative, perform the following steps:

1. Go to http://www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss.
2. Select your country from the list and click **Go**.
3. Click **About this site** to see an information page that includes the telephone number of your local representative.

Accessibility

Accessibility features help users with a physical disability, such as restricted mobility or limited vision, to use software products successfully. Standard shortcut and accelerator keys are used by the product and are documented by the operating system. Refer to the documentation provided by your operating system for more information.

For additional information, see the Accessibility appendix in the *User's Guide: NetView*.

Tivoli technical training

For Tivoli technical training information, refer to the following IBM Tivoli Education web site at http://www.ibm.com/software/tivoli/education.

Tivoli user groups

Tivoli user groups are independent, user-run membership organizations that provide Tivoli users with information to assist them in the implementation of Tivoli Software solutions. Through these groups, members can share information and learn from the knowledge and experience of other Tivoli users.

Downloads

Clients and agents, NetView product demonstrations, and several free NetView applications can be downloaded from the NetView for z/OS support web site:

http://www.ibm.com/software/sysmgmt/products/support/IBMTivoliNetViewforzOS.html

In the "IBM Tivoli for NetView for z/OS support" pane, click **Download** to go to a page where you can search for or select downloads.

These applications can help with the following tasks:

- Migrating customization parameters and initialization statements from earlier releases to the CNMSTUSR member and command definitions from earlier releases to the CNMCMDU member.
- Getting statistics for your automation table and merging the statistics with a listing of the automation table
- Displaying the status of a job entry subsystem (JES) job or canceling a specified JES job
- Sending alerts to the NetView program using the program-to-program interface (PPI)
- Sending and receiving MVS commands using the PPI
- Sending Time Sharing Option (TSO) commands and receiving responses

Support information

If you have a problem with your IBM software, you want to resolve it quickly. IBM provides the following ways for you to obtain the support you need:

Online

Access the Tivoli Software Support site at http://www.ibm.com/software/sysmgmt/products/support/index.html?ibmprd=tivman. Access the IBM Software Support site at http://www.ibm.com/software/support/probsub.html.

IBM Support Assistant

The IBM Support Assistant is a free local software serviceability workbench that helps you resolve questions and problems with IBM software products. The Support Assistant provides quick access to support-related information and serviceability tools for problem determination. To install the Support Assistant software, go to http://www.ibm.com/software/support/isa/.

Troubleshooting information

For more information about resolving problems with the NetView for z/OS product, see the *IBM Tivoli NetView for z/OS Troubleshooting Guide*. Additional support for the NetView for z/OS product is available through the NetView user group on Yahoo at http://groups.yahoo.com/group/NetView/. This support is for NetView for z/OS customers only, and registration is required. This forum is monitored by NetView developers who answer questions and provide guidance. When a problem with the code is found, you are asked to open an official problem management record (PMR) to obtain resolution.

Conventions used in this publication

This publication uses several conventions for special terms and actions, operating system-dependent commands and paths, and command syntax.

Typeface conventions

This publication uses the following typeface conventions:

Bold

- Lowercase commands and mixed case commands that are otherwise difficult to distinguish from surrounding text
- Interface controls (check boxes, push buttons, radio buttons, spin buttons, fields, folders, icons, list boxes, items inside list boxes, multicolumn lists, containers, menu choices, menu names, tabs, property sheets), labels (such as **Tip:**, and **Operating system considerations**:)
- Keywords and parameters in text

Italic

- Citations (examples: titles of publications, diskettes, and CDs
- Words defined in text (example: a nonswitched line is called a *point-to-point line*)
- Emphasis of words and letters (words as words example: "Use the word *that* to introduce a restrictive clause."; letters as letters example: "The LUN address must start with the letter *L*.")
- New terms in text (except in a definition list): a *view* is a frame in a workspace that contains data.
- Variables and values you must provide: ... where *myname* represents...

`Monospace`

- Examples and code examples
- File names, programming keywords, and other elements that are difficult to distinguish from surrounding text
- Message text and prompts addressed to the user
- Text that the user must type
- Values for arguments or command options

Operating system-dependent variables and paths

For workstation components, this publication uses the UNIX convention for specifying environment variables and for directory notation.

When using the Windows command line, replace $*variable* with %*variable*% for environment variables and replace each forward slash (/) with a backslash (\) in directory paths. The names of environment variables are not always the same in the Windows and UNIX environments. For example, %TEMP% in Windows environments is equivalent to $TMPDIR in UNIX environments.

Note: If you are using the bash shell on a Windows system, you can use the UNIX conventions.

Syntax diagrams

This section describes how syntax elements are shown in syntax diagrams. Read syntax diagrams from left-to-right, top-to-bottom, following the horizontal line (the main path).

Symbols

The following symbols are used in syntax diagrams:

►►	Marks the beginning of the command syntax.
►	Indicates that the command syntax is continued.
|	Marks the beginning and end of a fragment or part of the command syntax.
►◄	Marks the end of the command syntax.

Parameters

The following types of parameters are used in syntax diagrams:

Required	Required parameters are shown on the main path.
Optional	Optional parameters are shown below the main path.
Default	Default parameters are shown above the main path. In parameter descriptions, default parameters are underlined.

Syntax diagrams do not rely on highlighting, brackets, or braces. In syntax diagrams, the position of the elements relative to the main syntax line indicates whether an element is required, optional, or the default value.

Parameters are classified as keywords or variables. Keywords are shown in uppercase letters. Variables, which represent names or values that you supply, are shown in lowercase letters and are either italicized or, in NetView help and BookManager publications, displayed in a differentiating color.

In the following example, the USER command is a keyword, the *user_id* parameter is a required variable, and the *password* parameter is an optional variable.

```
►►── USER ─ user_id ──┬────────────┬────────────────────────────────────►◄
                      └─ password ─┘
```

Punctuation and parentheses

You must include all punctuation that is shown in the syntax diagram, such as colons, semicolons, commas, minus signs, and both single and double quotation marks.

When an operand can have more than one value, the values are typically enclosed in parentheses and separated by commas. For a single value, the parentheses typically can be omitted. For more information, see "Multiple operands or values" on page xxiv.

If a command requires positional commas to separate keywords and variables, the commas are shown before the keywords or variables.

When examples of commands are shown, commas are also used to indicate the absence of a positional operand. For example, the second comma indicates that an optional operand is not being used:

`COMMAND_NAME opt_variable_1,,opt_variable_3`

You do not need to specify the trailing positional commas. Trailing positional and non-positional commas either are ignored or cause a command to be rejected. Restrictions for each command state whether trailing commas cause the command to be rejected.

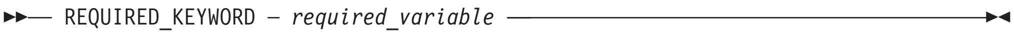
Abbreviations

Command and keyword abbreviations are listed in synonym tables after each command description.

Syntax examples

This section show examples for the different uses of syntax elements.

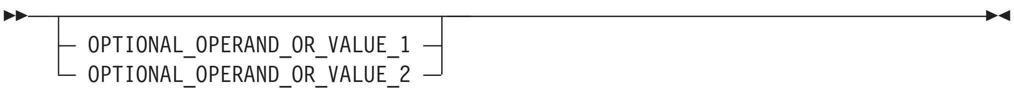
Required syntax elements: Required keywords and variables are shown on the main syntax line. You must code required keywords and variables.



A required choice (two or more items) is shown in a vertical stack on the main path. The items are shown in alphanumeric order.

Optional syntax elements: Optional keywords and variables are shown below the main syntax line. You can choose not to code optional keywords and variables.

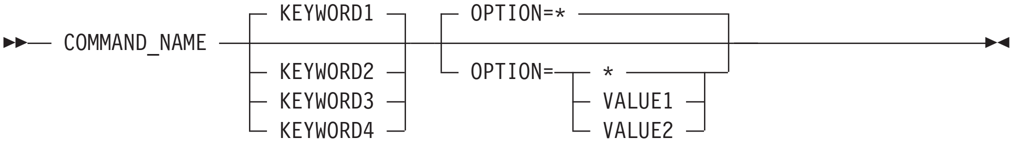
A required choice (two or more items) is shown in a vertical stack below the main path. The items are shown in alphanumeric order.



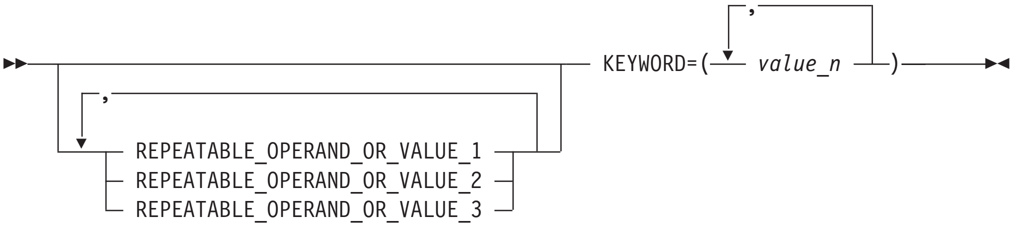
Default keywords and values: Default keywords and values are shown above the main syntax line in one of the following ways:

- A default keyword is shown only above the main syntax line. You can specify this keyword or allow it to default. The following syntax example shows the default keyword KEYWORD1 above the main syntax line and the rest of the optional keywords below the main syntax line.
- If an operand has a default value, the operand is shown both above and below the main syntax line. A value below the main syntax line indicates that if you specify the operand, you must also specify either the default value or another value shown. If you do not specify the operand, the default value above the

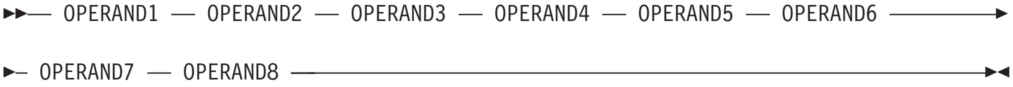
main syntax line is used. The following syntax example shows the default values for operand OPTION=* above and below the main syntax line.



Multiple operands or values: An arrow returning to the left above a group of operands or values indicates that more than one can be selected or that a single one can be repeated.



Syntax that is longer than one line: If a diagram is longer than one line, each line that is to be continued ends with a single arrowhead and the following line begins with a single arrowhead.



Syntax fragments: Some syntax diagrams contain syntax fragments, which are used for lengthy, complex, or repeated sections of syntax. Syntax fragments follow the main diagram. Each syntax fragment name is mixed case and is shown in the heading of the fragment. The following syntax example shows a syntax diagram with two fragments that are identified as Fragment1 and Fragment2.

Fragment1

Fragment2

Part 1. Learning About RODM

Chapter 1. Overview 3
Managing SNA Resources with NetView 3
Defining Non-SNA Resources to NetView 3
  Resource Definition Task 4
  Resources Supported by GMFHS 5
  Saving RODM Data 5
RODM in Network Automation 5
  Automation Concepts 6
  Automation Example 7
  For More Information 7
RODM Programming Tasks 7
  RODM Transactions 7
  RODM Functions 8
  Programming Languages 9
  RODM Notification Process 9
  RODM Load Function 9
Additional RODM Documentation. 10
Tools for RODM 11
RODM Samples and Macros 11

Chapter 1. Overview

This document describes IBM Tivoli NetView for z/OS Resource Object Data Manager (RODM), which runs under the z/OS operating system. This document describes how to:

- Manually define your network resources to RODM so that you can manage these resources using NetView management console.
- Automate network operations based on the status of resources stored in RODM.
- Write programs that use the services of RODM.

RODM is an object-oriented data cache. Objects in RODM represent resources in your network. The data cache is located entirely in the memory of the host processor resulting in fast access to data and high transaction rates. Many applications can interact with a single RODM, and more than one RODM can run on a host processor. You can use RODM for many tasks. RODM provides application programming interfaces (APIs) that can be used by any application running in the host processor.

The Graphic Monitor Facility host subsystem (GMFHS) is the host program that works with RODM and the NetView program running on the host processor, and NetView management console to manage resources.

GMFHS works with the SNA topology manager and NetView management console to manage SNA resources. For more information, see the *IBM Tivoli NetView for z/OS SNA Topology Manager Implementation Guide*, SC27-2864.

GMFHS also works with MultiSystem Manager and NetView management console to manage non-SNA resources. For more information, see the *IBM Tivoli NetView for z/OS User's Guide: NetView Management Console*.

Managing SNA Resources with NetView

Using the SNA topology manager, the NetView program provides subarea and Advanced Peer-to-Peer Networking (APPN) network management from NetView management console. You can display graphic views of resources in the network, and you can issue commands to resources you select from the view. The views contain both status and configuration information about your network. For more information, see the *IBM Tivoli NetView for z/OS SNA Topology Manager Implementation Guide*.

Defining Non-SNA Resources to NetView

Using the MultiSystem manager, the NetView program enables you to dynamically discover and manage non-SNA networks from NetView management console. You can display graphic views of resources in the network, and you can issue commands to resources you select from the view. The views contain both status and configuration information about your network.

You can also manually define your non-SNA resources. You need to provide information about your network to the NetView program so that views can be created and commands can be processed. For SNA networks, NetView gets its information from the VTAM and NCP definitions you create. For non-SNA networks, NetView gets its information from RODM definitions you create. This document describes the RODM definitions that you need to create and how you can create them.

NetView management console communicates with GMFHS. Figure 1 shows that GMFHS runs in its own address space in the host and communicates with RODM, which also runs in its own address space in the host.

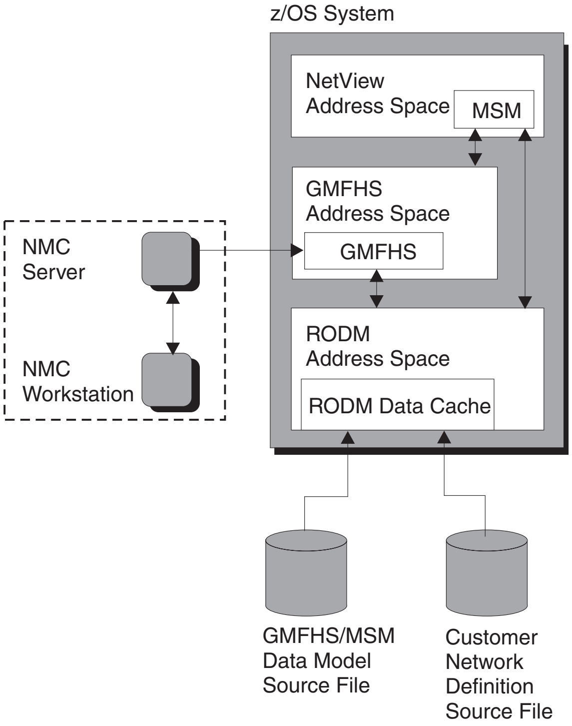


Figure 1. Using RODM to Support the NetView management console

Resource Definition Task

Resources in your non-SNA network are represented by objects in the RODM data cache. You can create three general types of objects:

- Management objects
- Managed objects
- View objects

Management objects represent the programs that control parts of the network and that connect to the NetView program. IBM Tivoli Network Manager is an example of a management object. The programs represented by management objects send alerts to NetView to update the status of resources in the network. These programs receive commands from the NetView program for the network resources they control.

Managed objects represent the network resources that you are managing. Managed objects contain both status and configuration information. A personal computer connected to a token-ring local area network (LAN) and a printer connected to an Ethernet LAN are examples of resources represented by managed objects. Managed objects must have a corresponding management object that sends status to the NetView program and receives commands for the resource.

View objects represent graphic views that can be displayed on NetView management console. Most graphic views are created automatically based on the configuration information contained in RODM. You might also want to define specific views as well. The information about which resources to display and how to display them is contained in the view object.

Network configuration information is represented by links between managed objects. For example, each managed object representing a resource on a token-ring segment has links to each adjacent resource on the segment. You can define both the logical configuration and the physical configuration of your network.

Resources Supported by GMFHS

GMFHS supports resources that can send status updates to the NetView program in a standard format. A *service point* is the program that interfaces the non-SNA network to the SNA network that contains the NetView program. The service point generates alerts that GMFHS converts to the status of objects in RODM.

The alerts sent to the NetView program identify the resource which has changed status. You need to assign names to RODM objects that match the names that are supplied by alerts. For information about how GMFHS uses resource names from alerts, see Chapter 6, "Customizing GMFHS to Process and Receive Alerts and Resolutions," on page 167. It also describes how you can customize GMFHS alert processing to recognize additional alert types.

Saving RODM Data

All of the data in the RODM data cache is stored in memory. If you stop RODM, shut down your processor, or your system fails, all of the data in the data cache is lost. The *checkpoint* function enables you to save a copy of the data cache to DASD. When you restart RODM, you can read in the stored data from DASD. The checkpoint function can be requested by a program, by the z/OS console operator, or by a NetView operator, if the NetView program used by the operator is set up to send commands to z/OS. Because status information stored in RODM is volatile, restoring data from DASD might not be appropriate.

A *warm start* of RODM is when you start RODM and read in checkpoint data. The data cache contains the exact data at the time of the checkpoint. After a warm start, you might need to update some objects in the data cache. If the applications that maintain the status of your resources keep track of updates sent to RODM, the applications can resend any changes since the checkpoint.

A *cold start* means you start RODM without checkpoint data. The data cache contains only the system-defined classes. You then need to load your data model and data.

RODM in Network Automation

Using the SNA topology manager, you can automate the management of your subarea network. For more information, see the *IBM Tivoli NetView for z/OS SNA Topology Manager Implementation Guide*.

You can also automate the management of your non-SNA network resources using RODM. Because GMFHS maintains the status of the non-SNA network resources in the RODM data cache, you can write automation routines using the data in RODM. The following RODM concepts are important to automation.

Automation Concepts

Two types of programs work with RODM, user applications and methods. A RODM *user application* is a program that runs in a different address space than RODM, and that communicates with RODM using an API. The user application must run on the same z/OS host as RODM. User applications can be written in any programming language. Sample control blocks for the API are supplied for use with PL/I and C. Therefore, you might prefer to use one of these two languages.

A *method* is a program that runs in the RODM address space and communicates with RODM using another API. Methods are usually small programs that perform specific tasks on data in the data cache. Running or executing a method is referred to as *triggering* the method. Methods must be written in PL/I or C. They are restricted in the types of functions they can perform. There are six types of methods:

- RODM triggers the *query method* for a field when the value of the field is queried. For example, it can issue a command to a network resource to request its current status. The *query subfield* specifies the query method for a field.
- RODM triggers the *change method* for a field when another method or user application requests to change the value of the field. For example, the change method can issue a command to change the real status of the network resource to match the new status of the object that represents the resource in RODM. The *change subfield* specifies the change method for a field.
- RODM triggers the *notification method* when the value of a field changes. You can define any number of notification methods for a field. It notifies user applications of changes. The notification method is particularly valuable for automation tasks. The *notify subfield* specifies the notification methods for a field.
- RODM triggers a *named method* when another method or user application requests it. A named method is specified by a field of an object or class. Named methods can be used to perform some action for a particular object or class. For example, you can create a named method that contains the commands to activate the object with which the method is associated.
- An *object-independent method* is any method that is not associated with a specific object or class. Object-independent methods can act on many objects and classes. For example, an object-independent method can query the status of all objects that represent the workstations on a specified LAN.
- The *initialization method* is a special type of object-independent method. The initialization method, if specified, is automatically triggered when RODM is started.

The query method, change method, notification method, and named method are known as *object-specific* methods because they are associated with a specific object or class. The NetView program supplies sample methods that you can use for automation tasks.

A set of NetView services named the *RODM automation platform* makes automation easier. The NetView automation table, command lists, and applications can issue requests to RODM to change values of fields and trigger methods. A method that is supplied with the NetView program sends commands to be issued by a NetView task. And the RODM automation platform provides an enhanced API which enables applications in the NetView address space to issue RODM functions with less programming effort.

Automation Example

A typical automation implementation can use methods, a user application, and the RODM automation platform. For example, you can use a notification method to notify your automation application when a resource fails. Your automation application can query RODM to find the resources in the network that are related to the resource that failed. By querying the status of the related resources, your automation application can determine the most likely location of the problem and can issue commands to correct the problem.

You can create methods associated with specific objects in RODM that issue NetView commands using the RODM automation platform. An object-specific method can contain the commands to activate the resource that the method is associated with. When triggered by your automation application, the object-specific method sends the commands to the EKGSPPI method that is supplied with the NetView program, the commands are passed to the NetView program and issued by an autotask. This enables the same application to activate different types of resources without knowing the commands specific to each resource.

For More Information

This document contains two chapters specifically about automation. Read Chapter 7, “Writing Automation Code,” on page 181 for more information about automation with the GMFHS data model. Read Chapter 8, “Using the RODM Automation Platform,” on page 189 for more information about the RODM automation platform services.

RODM Programming Tasks

While this overview has focused on using RODM to support NetView management console and network automation, RODM can support other types of network and system management programs. This section describes RODM programming tasks in general.

RODM can be used for any task that requires a high-speed data cache manager. RODM provides an application programming interface for user application programs, and another application programming interface for methods. It also provides a load function to simplify loading data into the data cache and maintaining the data.

User applications and methods have very similar interfaces to RODM. Many of the functions that RODM provides can be used by both types of programs. Both user applications and methods send function requests to RODM. RODM replies with a return code and reason code to indicate if the request was successful. Some function requests cause RODM to return data as well. A single function request made to RODM and the response from RODM make up a *transaction*.

RODM Transactions

Many transactions request RODM to take some action on a particular class, object, field, or subfield in the data cache. For example, a user application sends a request to RODM to change the value of a field that represents the status of a network resource. The particular class, object, field, or subfield that the transaction specifies is the *target* of the transaction. In general, a transaction has a single target.

Each transaction is made using a call to RODM that passes the required parameters for that transaction. The parameters are grouped into six control blocks:

- Access block
- Transaction information block
- Function block
- Response block
- Entity access information block
- Field access information block

Specific transactions use different blocks as needed.

The *access block* identifies the user application to RODM. Methods run within RODM, so they never use an access block. The RODM automation platform services CNMQAPI and DSINOR take care of the access block for applications running in the NetView address space.

The *transaction information block* is used to track each transaction with RODM. RODM places the return code and reason code for the transaction in this control block. All transactions use this block.

The *function block* specifies the RODM function to be run. It contains the particular parameters that RODM needs to run the function. All transactions use this block.

The *response block* contains any data requested from RODM. Functions that request data, such as query functions, use a response block.

The *entity access information block* identifies the specific class and object that is the target of a transaction. This block is used when a class, object, field, or subfield is the target of a transaction.

The *field access information block* identifies the specific field that is the target of a transaction. This block is used when a field or subfield is the target of a transaction.

RODM Functions

RODM provides functions for user applications and methods. Some functions are available only to user applications, and some are available only to methods. Many functions are available to both. Each function requires a particular authorization level, so you can limit the functions available to particular applications.

RODM provides functions to connect to and disconnect from RODM. It provides functions to checkpoint RODM and stop RODM.

RODM provides a set of functions to change the structure of the elements in the data cache. There are functions to create and delete classes, objects, fields, and subfields. Link and unlink functions enable you to define relationships among objects.

RODM provides a set of functions to change the values of the fields and subfields of classes and objects. Changing the value of a field triggers its change method if one has been defined. Changing the value of a subfield does not trigger the change method.

RODM provides query functions to get information about the classes and objects in the data cache. Programs can query the value of any field or subfield. Querying the value of a field triggers its query method if one has been defined. Querying the value of a subfield does not trigger the query method. Programs can also query the structure of the elements in the data cache. RODM also provides the ability to locate objects in RODM based on the value of a character field.

RODM provides functions to support the notification process. Programs can add and delete notification subscriptions. User applications can get information from the notification queue. Notification methods support the RODM notification process.

Other functions enable you to write diagnostic information to the RODM log and trigger methods. You can issue a list of functions in a single call to RODM. You can also issue asynchronous requests to RODM.

Each function is described in detail in Chapter 14, “Application Programming Reference,” on page 367. There are sample function blocks and programming examples for each function RODM provides.

Programming Languages

User applications access RODM using the RODM user application programming interface. User applications can be written in any programming language supported by your z/OS environment. However, RODM samples and examples are provided only in PL/I and C.

Methods access RODM using the RODM method application programming interface. RODM methods can be written only in PL/I or C. Many methods that are supplied with the NetView program are supplied in source format. You can use these methods as models to write your own RODM methods.

RODM Notification Process

The RODM *notification process* enables user applications to receive asynchronous notification of events. User applications *subscribe* to fields in the data cache. When the value of the field changes, the notification method associated with the field is triggered. The notification method writes information about the change to a *notification queue* and RODM posts the *event control block* (ECB) for the user application.

The user application waits until its ECB is posted by RODM. The user application calls the EKGWAIT module to wait until the ECB is posted. The user application gets the information from the notification queue and takes the appropriate actions. When it finishes processing an event, the user application waits to be notified of the next event.

RODM Load Function

The RODM load function provides an easy way to load the class structure and objects into the RODM data cache. See the *IBM Tivoli NetView for z/OS Data Model Reference* for more information about data models, class structures, fields, and objects.

You create input statements for each class and object which are processed by the load function. You can use the load function to load the initial structure and objects into the data cache, and you can also use it to update and maintain the data cache at any time.

The RODM load function accepts two types of input statements:

- *High-level RODM load function statements* enable you to create and delete classes and objects. Each create statement defines one class or object and all of its fields. A single high-level RODM load function statement can do the work of many RODM transactions.
- *RODM load function primitive statements* enable you to make changes to the RODM data cache that are not possible with the high-level RODM load function statements. For example, you can trigger an object-independent method or change the value of a subfield in the data cache using RODM load function primitive statements.

Additional RODM Documentation

This document contains information about defining a network to the GMFHS data model, loading the data model into the RODM data cache, and writing application programs and methods that use RODM. Other documents in the NetView library contain information about RODM that can be of use to you when you are performing the tasks that are outlined in this document:

IBM Tivoli NetView for z/OS Installation: Configuring Graphical Components
Describes procedures for installing the NetView program and for customizing your system and tailoring your network for your needs, including the following information:
- Defining RODM as an MVS Subsystem
- Setting up Security
- Defining the RODM Log
- Updating the RODM Start Procedure
- Defining Global Variables for RODM
- Defining RODM Using the EKGCUST Member
- Defining Initialization Values for RODM DSIQTSK Task

IBM Tivoli NetView for z/OS Administration Reference
Contains the following information:
- The statements that are used to define RODM and the RODM automation task
- Customizing RODM using the EKGCUST member

IBM Tivoli NetView for z/OS Security Reference
This document contains information for defining RODM security.

IBM Tivoli NetView for z/OS Automation Guide
Describes how to use RODM as part of NetView automation.

IBM Tivoli NetView for z/OS Troubleshooting Guide
Contains information about diagnostics and troubleshooting, including the following information:
- Debugging methods
- The RODM log
- The RODM dump utility
- The RODM load utility error listing
- Using RODM API statistics to improve RODM performance

IBM Tivoli NetView for z/OS Messages and Codes Volume 2 (DUI-IHS)
Describes the messages that are returned by RODM. RODM messages are prefixed with EKG.

IBM Tivoli NetView for z/OS SNA Topology Manager Implementation Guide
Describes how to use the SNA topology manager.

IBM Tivoli NetView for z/OS Data Model Reference
Describes the GMFHS, SNA topology manager, and MultiSystem manager data models.

IBM Tivoli NetView for z/OS Tuning Guide
Provides information for tuning RODM and GMFHS.

IBM Tivoli NetView for z/OS User's Guide: NetView
Provides information for operators and system programmers on how to use the NetView program, including RODM and GMFHS.

Tools for RODM

The NetView program provides the following tools for use with RODM:
- RODMView
- RODM unload function
- FLCARODM (RODM Access Facility)
- RODM Collection Manager
- Visual BLDVIEWS (VBV)
- BLDVIEWS
- DELVIEWS

For more information about these tools, see Appendix A, "RODM Tools," on page 503.

RODM Samples and Macros

The NetView program provides sample code that you can use to set up your own network in RODM and to learn how to write application programs and methods. It also supplies macros for you to include in the application programs and methods that you write. The sample code and macros, which are shipped with the NetView product, can be found in the following libraries:

NETVIEW.V6R1M0.CNMSAMP
This library contains sample code that you can use to define and load your network into RODM. Additionally, this library contains sample code that you can use to learn how to connect to RODM and how to write application programs and methods that use GMFHS automation. The names of the function samples have prefixes EKG5 and EKG6.

NETVIEW.V6R1M0.SCNMMAC1
This library contains the macros that you include in your application programs and methods. The names of these macros have prefixes EKG1, EKG2, EKG3, and EKG4. For more information about these macros, see Chapter 14, "Application Programming Reference," on page 367.

Some of these macros and parts of the sample code are described in this document. The names of the specific macros or functions are listed in the sections in which they are described.

Part 2. Defining Resources to NetView

Chapter 2. Defining Your Network to GMFHS 17
Manual Network Definition Overview 17
Sample Network 18
  SNA Components of the Sample Network 18
  Non-SNA Components of the Sample Network 19
    Service Points 19
    Sysplex Network 19
    Ethernet Network 20
    Token-Ring Local Area Network 20
    ITNM Network 21
Identifying Network Elements to Define 22
  Identifying Management Objects 22
    SNA Domains 22
    Network Management Gateways 22
    Non-SNA Domains 23
  Identifying Managed Objects 23
    GMFHS_Shadow_Objects_Class Objects 23
    GMFHS_Managed_Real_Objects_Class Objects 24
    GMFHS_Aggregate_Objects_Class Objects 24
  Identifying Connectivity Relationships 25
    ComposedOfLogical and IsPartOf 26
    ComposedOfPhysical and IsPartOf 26
    AggregationParent and AggregationChild 26
    ParentAccess and ChildAccess 27
    PhysicalConnPP 27
    LogicalConnPP 27
    PhysicalConnUpstream and PhysicalConnDownstream 28
    LogicalConnUpstream and LogicalConnDownstream 28
    BackboneConnPP 28
  Identifying Views 28
    Exception Views 28
    Network Views 29
    Configuration Views 31
    More Detail Views 33
Defining Your Configuration to RODM 33
  Defining Management Objects 33
    Defining SNA Domains 33
    Defining Network Management Gateways 34
    Defining Non-SNA Domains 35
  Defining Managed Objects 36
    Defining SNA Resources 36
    Defining Non-SNA Real Resources 37
    Defining GMFHS Aggregate Objects 38
  Defining Connectivity Relationships Between Objects 40
    Defining Logical Connectivity 40
    Defining Physical Connectivity 41
    Defining Parent-Child Relationships 41
  Defining Views 41
    Defining Exception Views 41
    Defining Network Views 42
    Defining Configuration Views 43
    Defining More Detail Views 46
  Defining Layout Parameters 46
    Defining Layout Parameters for Exception Views 46
    Defining Layout Parameters for Network, Configuration, and More Detail Views 46
    Defining Layout Parameters for Dynamically Built More Detail Views 49
  Putting It All Together 54

Chapter 3. Loading the GMFHS Data Model 57
Loading the Data Models and Network Definitions 57
Changing Network Definitions When GMFHS Is Running 57
  Selecting the Required GMFHS CONFIG Command 58
    Non_SNA_Domain_Class Changes 59
    SNA_Domain_Class Changes 59
    NMG_Class Changes 60
    GMFHS_Managed_Real_Objects_Class Changes 60
  Adding NMGs and Domains When GMFHS Is Active 60

Chapter 4. Communicating with Network Management Gateways 61
Defining Non-SNA Presentation Protocol 62
  DOMP010 Presentation Protocol 62
  DOMP020 Presentation Protocol 63
  PASSTHRU Presentation Protocol 64
  NONE Presentation Protocol 64
  Output Formatting For All Presentation Protocols 64
    DOMP020 and PASSTHRU Output Formatting 64
    DOMP010 Output Formatting 65
  DOMP010 Formatting Rules 65
    General Packet Format 65
    Keyword and Value Definitions 65
    Command Execution—CE 66
    Command—CM 66
    Component ID—CP 67
    Domain—DM 68
    Protocol—PT 68
    Reason—RN 69
    Response—RP 70
    Command Sender ID—SN 70
    Message Sequence Number—SQ 70
    Status—ST 70
    Time Stamp—TM 71
    Text—TX 71
  Command Formatting and Protocol Examples 72
    Single-Response Protocol 72
    Multiple-Response Protocol 73
  Timing Considerations 74
    Alerts 74
    Command Responses 75
Defining Non-SNA Session Protocols 75
  DOMS010 75
  PASSTHRU 76

  - NONE . . . 76
  - Session Establishment for DOMS010 . . . 76
  - Session Establishment for DOMS010 Using CNMS4406 . . . 76
  - GMFHS-Initiated Session Establishment . . . 77
  - INIT Generic Alert for Session Establishment . . . 78
  - Session Termination . . . 80
- Defining Non-SNA Transport Protocols . . . 81
  - COS Gateway Support. . . . 81
  - Program-to-Program Interface Gateway . . . 82
  - OST/PPT Gateway . . . 82
  - Monitoring Non-Network Devices . . . 83
  - Types of NMGs . . . 83
    - Common Operations Services NMGs . . . 83
    - Operator Station Task NMGs . . . 83
    - Program-to-Program Interface NMGs . . . 83
    - PPI Command Transport Envelope . . . 84

Chapter 5. How GMFHS Uses RODM . . . 87

- GMFHS Initialization . . . 87
  - Aggregation Warm Start . . . 87
  - Resource Status Warm Start . . . 87
  - GMFHS Initialization Process Overview . . . 88
    - Setup Subprocess . . . 88
    - Session Establishment Subprocess . . . 88
- Monitoring Topology Managers . . . 89
- Building Views . . . 89
  - Object Discovery Process . . . 89
    - Predefined Views . . . 89
    - Dynamically Built Views . . . 89
    - Object Discovery Process Description for Specific Views . . . 94
  - Object Connectivity Process . . . 100
  - Defining Exception View Objects and Criteria 100
    - Defining Exception Criteria. . . . 101
    - Defining Candidates for Exception Views . . . 103
    - Defining the ExceptionViewFilter Field . . . 103
    - Customizing the DisplayStatus Mapping Table for Exception Views . . . 104
    - Default Values for Classes . . . 109
    - Specifying Resource Names for DisplayStatus Mapping . . . 109
    - Examples of Customizing DisplayStatus Mapping . . . 110
    - Creating a DisplayStatus Method for Exception Views . . . 111
    - Implementing Exception View Processing for MultiSystem Manager . . . 112
  - Locate Resource Function . . . 113
  - Restricting Recursive Views . . . 113
  - Refreshing Open Views . . . 114
- Applying Span-of-Control to Views . . . 114
  - Views . . . 115
    - Defining Predefined Views to Spans . . . 115
    - Defining Dynamically Built Views to Spans 115
    - Examples of Defining Views to Spans . . . 116
  - Resources. . . . 117
    - Examples of Restricting Resources Within Views Using Spans . . . 119
  - Helpful Hints . . . 120
    - No Views in the View List Are in the Operator's Span-of-Control . . . 120
    - No Resource in the View Is in the Operator's Span-of-Control . . . 120
    - Selected Object Is Not in the Operator's Span-of-Control . . . 120
    - Changing the NGMFVSPN Attribute . . . 121
    - RACF Is Used for RODM Security . . . 121
- Applying Span-of-Control to Set and Clear Operator Status. . . . 121
- Applying Policy to Views . . . 122
  - Representing Policy Definitions in RODM . . . 122
  - Resources Belonging to Multiple Policies . . . 124
  - Resources Suspended from Aggregation Due to Policy . . . 127
  - Suspending Aggregation Using an Aggregate 128
  - System Status Updates No Longer Sent to Resources Due to Policy . . . 129
  - Additional Information . . . 129
- Aggregation Concepts . . . 130
  - Aggregation Overview . . . 130
  - Creating an Aggregation Hierarchy . . . 131
  - Building the Aggregation Hierarchy in RODM 132
  - Updating Status . . . 133
    - How Status Affects Aggregation . . . 134
    - Using the DisplayStatus of Real Objects . . . 134
    - Calculating the Aggregate Parent Status . . . 136
    - Aggregation Problems . . . 139
    - UserStatus Field . . . 139
    - Events That Start the Aggregation Process 139
    - Aggregation Methods . . . 142
  - Status Groups . . . 142
  - Using Status Groups . . . 142
  - Examples of Customizing Aggregate DisplayStatus . . . 143
- Using the Collection Definition Objects . . . 143
  - Collection Definition Objects . . . 144
    - Collection Definition Object Fields . . . 144
  - Using Collection Specifications . . . 145
    - Conditional Statements . . . 145
    - Postfix Notation in Conditional Statements 146
    - Complex Conditional Statements . . . 147
    - Stack Model Postfix Processing . . . 148
    - Collection Specification Syntax . . . 149
    - Collection Specification Values. . . . 150
    - Values and Data Types . . . 153
  - Examples of Collection Definition Objects . . . 155
- Using NetView Resource Manager . . . 159
  - NetView Resource Manager Views . . . 159
    - NetView Resource Manager Object Information . . . 161
    - NetView Management Console Command Support for NetView Resource Manager . . . 162
  - Modifying DUIFSMT for NetView Resource Manager . . . 164
  - Using DUIFVINS with NetView Resource Manager . . . 164
  - NetView Resource Manager Sample Loader Files . . . 164
    - Customizing Sample Loader Files . . . 165

Chapter 6. Customizing GMFHS to Process and Receive Alerts and Resolutions 167
Receiving and Monitoring Alerts or Resolutions 167
  What GMFHS Receives from the Hardware Monitor 167
  Objects in RODM Representing SNA Resources 168
  Objects in RODM Representing NMGs 169
  Objects in RODM Representing Non-SNA Domains 169
    First Method 169
    Second Method. 170
  Objects in RODM Representing Non-SNA Resources 171
    Single Non-SNA Resource 171
    Multiple Non-SNA Resources 171
DUIFEDEF Alert Processing 172
  Parameters 172
    Pointer to a reentrant work area 172
    Pointer to a second reentrant work area . . 173
    Value of the EMDomain field 173
    Value of the DomainCharacteristics field . . 173
    Pointer to an array of structures 173
    Pointer to hardware monitor resource hierarchy 173
    Pointer to the length of the hardware monitor resource hierarchy 174
    Register 15 Conventions 174
    Default DUIFEDEF Actions. 174
  Alert Translation Tables 176

Chapter 2. Defining Your Network to GMFHS

You can manually define your network configuration to the NetView program based on the GMFHS data model. This chapter first describes a sample network, and then shows the steps for manually defining a network.

Notes:

1. You can use the SNA topology manager to define your SNA network to RODM. Refer to the *IBM Tivoli NetView for z/OS SNA Topology Manager Implementation Guide* for more information.
2. You can use the MultiSystem Manager Access facility to define your non-SNA network to RODM.

To help you manually define your network to RODM, a sample object load file, DUIFSNET, is provided with the NetView program. The sample file contains the RODM load function statements that define the sample network to RODM.

You manually define your network using RODM load function statements. You can generate these statements in any of the following ways:

- If you have configuration information stored in a repository, write a conversion program to convert the information to the RODM load file format presented in Chapter 10, “Using the RODM Load Function,” on page 239.
- Create the configuration definitions with a text editor.

You can also define your network without using the RODM load function. If you have your network configuration information stored in a database, you can write a RODM user application that places the configuration information directly into RODM. Your user application puts the data into RODM by issuing calls to the RODM user API. See Chapter 11, “Writing Applications that Use RODM,” on page 301 for information about writing RODM user applications.

Manual Network Definition Overview

To manually define your network configuration to RODM, perform the following tasks in the order listed:

1. Analyze your configuration and identify the network elements that you need to define to RODM.
2. Define the management objects in your network. Management objects are:
   - SNA domains
   - Network management gateways
   - Non-SNA domains
3. Define the managed objects in your networks. Managed objects are:
   - Real non-SNA objects for which you are to receive status, alerts, or both through a service point
   - SNA objects that appear in views with non-SNA objects
   - Aggregate objects
4. Define connectivity relationships for the resources in your network. Examples of connectivity relationships include logical and physical connectivity, parent-child, composed-of-logical, composed-of-physical, and is-part-of.

5. Define the types of views of your configuration that you want the operator to see.

Sample Network

This chapter uses a sample network (as shown in Figure 2) to describe how to define your network to RODM. This network contains both SNA and non-SNA components.

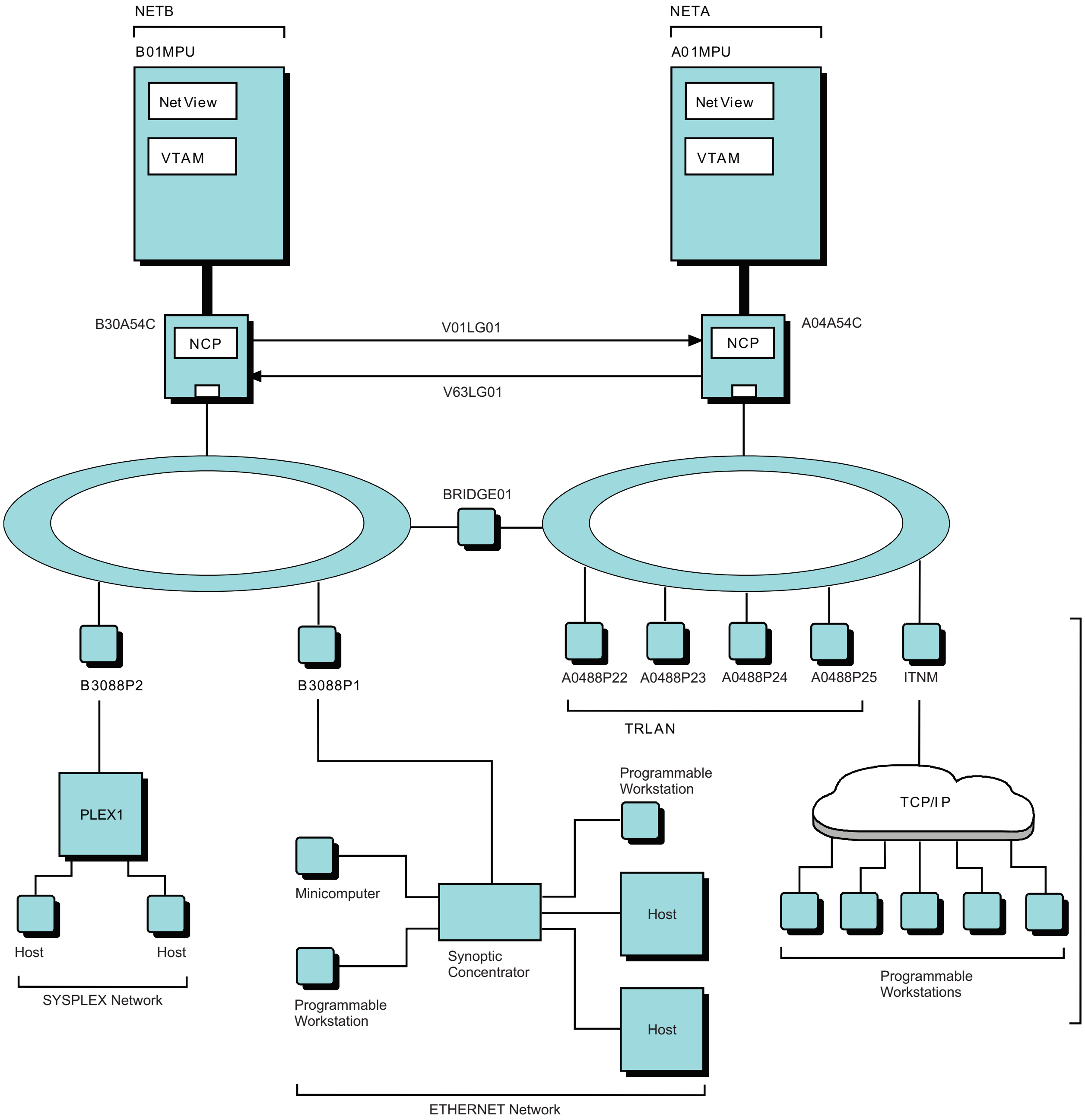


Figure 2. Sample Network

SNA Components of the Sample Network

The sample network consists of two network domains: network NETA and network NETB.

Network NETA consists of the following components:
- Host processor A01MPU, running a NetView program and VTAM
- NCP A04A54C, which connects the host processor to a token-ring LAN
- NMG A0488P21, which manages the token-ring LAN network (TRLAN network)
- NMG A0488P31, which manages the ITNM network
- Token-ring LAN network
- IP network managed by IBM Tivoli Network Manager (ITNM network)

Network NETB consists of the following components:
- Host B01MPU, running a NetView program and VTAM
- NCP B30A54C, which connects the host to a token-ring
- NMG B3088P1, which manages the Ethernet network
- NMG B3088P2, which manages the sysplex network
- Ethernet network
- Sysplex network

The two host systems are connected by two logical gateway connectors, V01LG01 and V63LG01, through NCP/Token-Ring interconnection (NTRI). These logical gateway connectors between the two NCPs are associated with the two token-ring LANs with a bridge between them. The SNA links connecting the service points to their NCPs also use token rings for their underlying physical connectivity.

The hosts, NCPs, service points, gateway connectors, and link connectors in the sample network are SNA resources managed by the NetView and VTAM programs. The focal point NetView program, GMFHS, and RODM run in host A01MPU. The NetView management console monitors these SNA resources and generates views for them.

Non-SNA Components of the Sample Network

NetView management console does not recognize the non-SNA components of the sample network. For a NetView management console to manage these non-SNA components, they must be defined to RODM using the GMFHS data model.

Service Points

The following four service points, defined as network management gateways, are in the sample network:
- NMG B3088P1 runs the SYNOPTAP transaction program, which manages the Ethernet network.
- NMG B3088P2 runs the NetView program, which manages the remote sysplex.
- NMG A0488P21 runs in the token-ring LAN and manages the TRLAN network.
- NMG A0488P31 runs transaction program A94306F8, which manages the ITNM network.

Sysplex Network

Figure 3 on page 20 shows more detail about the sysplex network shown in the sample network. The sysplex network consists of:
- PLEX1 reports to enterprise master NetView program in NMG B3088P2.
- The RALXT1 and RALXT2 z/OS systems connect to PLEX1.

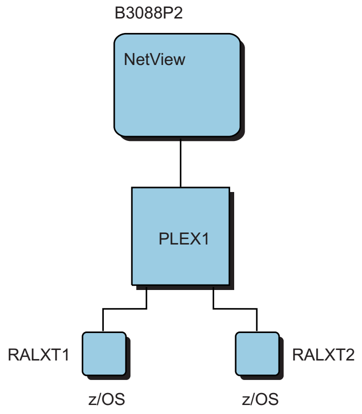


Figure 3. Sysplex Network

Ethernet Network

Figure 4 shows more detail of the Ethernet network in the sample network. An adapter on service point B3088P1 connects the service point to synoptic concentrator CNTR3000. The concentrator is connected to the host systems through three connectors:

- Connector NSL_ENET, which is associated with the Windows system and the AIX system
- Connector NSL_B202, which is associated with host AS400.
- Connector OEMLAB, which is associated with two non-SNA (Solaris and Linux) hosts

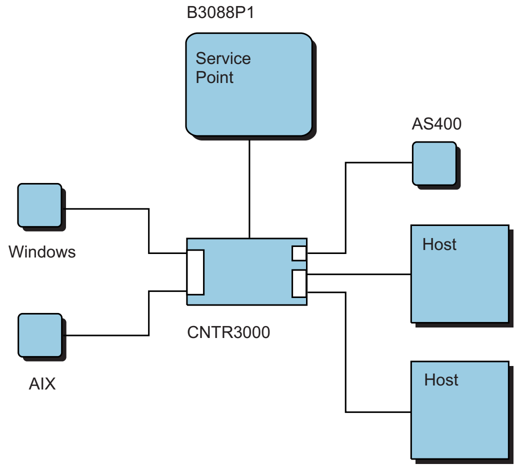


Figure 4. Ethernet Network

Token-Ring Local Area Network

Figure 5 on page 21 shows token-ring network TRLAN. It consists of the following resources:

- Adapter TRADPTR, which connects NCP A04A54C to the token ring
- Resource A04N1088, which is the SNA line representing the token-ring interface coupler (TIC)
- Resource A04P1088, which is defined for the SNA physical unit (PU) for the TIC

- Resources A0488P21 through A0488P25, which are token-ring adapters for programmable workstations and are associated with the appropriate adapter addresses in the LAN Manager
- BRIDGE01, which is a bridge on the LAN that connects to another token ring in NETB

The sample network defines SNA PU 2 resources representing the programmable workstations to SNA, and has named the SNA PUs A0488P21 through A0488P25, associating the SNA PUs to the adapter resident in each workstation that supports a PU. The sample network uses the DisplayResourceName field to specify the name that is displayed for each resource in the token-ring network. For example, the object LANMGR.10005AC35CA0 has its DisplayResourceName field set to A0488P21. This enables you to display names for resources that are meaningful to your operators.

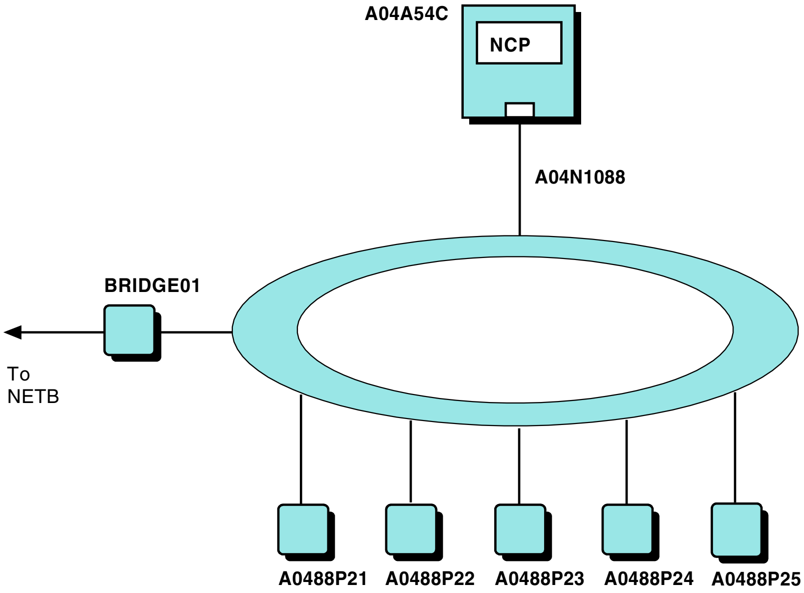


Figure 5. Token-Ring LAN

ITNM Network

Figure 6 on page 22 shows more detail for the ITNM network in the sample network. The ITNM network is the IP network that is managed by IBM Tivoli Network Manager. The ITNM network consists of the following resources:
- IBM Tivoli Network Manager
- Workstations T46A, T47A, T47B, T48A, and T48B

IBM Tivoli Network Manager is configured on the A0488P31 system. Workstations T46A, T47A, T47B, T48A, and T48B are connected to the TCP/IP network in which IBM Tivoli Network Manager resides. IBM Tivoli Network Manager converts selected traps related to these resources into alerts, which are then sent to the focal point host A01MPU.

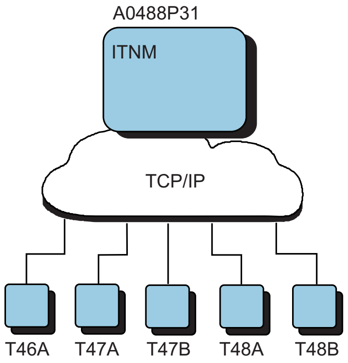


Figure 6. ITNM Network

Identifying Network Elements to Define

To properly define your network to RODM, assess your network components and their configuration, and then identify the network elements. The elements to identify are:

- Management objects
- Managed objects
- Connectivity relationships
- Desired views

Identifying Management Objects

Management objects represent the programs that control the components of a network and connect the components to the NetView program. These programs send alerts to the NetView program to update the status of resources in the network and receive commands from the NetView program for the resources that they control. Three types of management objects need to be identified to RODM:

- SNA domains
- Network management gateways
- Non-SNA domains

SNA Domains

An SNA domain represents one NetView program. You need to define to RODM one SNA domain for each NetView program that can originate alerts for SNA resources, if these SNA resources are defined as shadow objects to RODM.

You also need to define an SNA domain for each NetView program that has a non-SNA domain reporting to it, even if it has no SNA shadow objects defined on it. This ensures command support for the non-SNA objects and enables GMFHS to determine if the status of resources in the non-SNA domain is known. For information about shadow objects, see “Identifying Managed Objects” on page 23.

In the sample network, one SNA domain is defined for each of the NetView programs that reside in hosts B01MPU and A01MPU.

Network Management Gateways

A network management gateway (NMG) is a gateway between the NetView program and the network management function of one or more non-SNA networks. The AIX and IBM Tivoli Network Manager service points are examples of NMGs. An NMG can also be a user-written service point that uses service point command service (SPCS) support or sends alerts by some other means.

Two other NetView facilities that support network management gateways are the program-to-program interface (PPI) and operator station tasks (OSTs). The program-to-program interface provides a path for the exchange of network management information and commands for applications that manage non-SNA resources and run in the focal point host in address spaces other than the NetView address space. OSTs run command procedures and command processors that accept network management commands for, and provide status of, non-SNA resources.

In the sample network, four service points are defined as network management gateways:
- B3088P2
- B3088P1
- A0488P31
- A0488P21

Non-SNA Domains

You must define a non-SNA domain for each non-SNA network being monitored. A non-SNA domain is uniquely identified by any combination of service point, transaction program, and element management system.

Depending on the transaction program used, the transaction program and element management system might or might not identify themselves in alerts coming to the NetView program for non-SNA resources. A Non_SNA_Domain_Class object needs to be defined for each combination of service point, transaction program, and element management system that is identified in alerts flowing to the NetView program.

In the sample network, a non-SNA domain is defined for each of the following networks:
- The Ethernet network, which has a service point named B3088P1
- The sysplex network, which is connected by NMG B3088P2
- The TRLAN network, which has a service point named A0488P21
- The ITNM network, which has a service point named A0488P31

Identifying Managed Objects

Managed objects represent the network resources that you manage. These objects contain status and configuration information about the network resources that they represent. Managed objects require management objects to send status to the NetView program and to receive commands for the resource. You identify one managed object for each network resource that you want to manage using RODM. Four types of managed objects can be defined to RODM:
- SNA topology manager class objects. The SNA topology manager objects are not included in the sample network DUIFSNET. For more information, refer to the*IBM Tivoli NetView for z/OS SNA Topology Manager Implementation Guide*.
- GMFHS_Shadow_Objects_Class objects
- GMFHS_Managed_Real_Objects_Class objects
- GMFHS_Aggregate_Objects_Class objects

GMFHS_Shadow_Objects_Class Objects

The SNA topology manager creates SNA objects for resources that it manages. If you have other SNA resources that are not managed by SNA topology manager, you can create GMFHS_Shadow_Objects_Class objects to represent them. GMFHS_Shadow_Objects_Class objects represent SNA resources that you want to relate to non-SNA resources. The status of shadow objects is not kept in RODM, but is maintained by the NetView management console SNA support. When a view containing shadow objects is displayed at the NetView management console workstation, NetView management console fills in and maintains each object's status.

Note: The NetView management console does not maintain shadow object status. Shadow objects are displayed on the NetView management console, but the status is always unknown.

If you want to relate SNA resources to non-SNA resources such as those in the four non-SNA networks in the sample network, you need to define the SNA resources as objects on the GMFHS_Shadow_Objects_Class. These GMFHS_Shadow_Objects_Class objects are SNA resources, such as PUs, logical units (LUs), and link connections, that are defined in RODM so that they can be related to associated non-SNA resources.

In the sample network, logical link connectors V01LG01 and V63LG01 have been defined and are related to the physical path that connects the two NCPs and the two token-ring LANs. If either of the logical link connectors is displayed with a status of unsatisfactory, the operator can select the connector and request more detailed information about the resource. GMFHS then locates the GMFHS_Shadow_Objects_Class object for the connector in RODM, follows the configuration relationships to determine what resources made up the connector, and dynamically constructs and displays a view consisting of more detailed information.

GMFHS_Managed_Real_Objects_Class Objects

GMFHS_Managed_Real_Objects_Class objects represent non-SNA resources that are managed by a NetView management console. The status of each of these resources is determined by alerts and command responses sent through the network and is stored in RODM. Examples of these resources include multiplexers, modems, software applications, and T1 element managers. You must define a GMFHS_Managed_Real_Objects_Class object to GMFHS for each resource that you manage. If you have added child classes to the GMFHS_Managed_Real_Objects_Class, create objects of the child classes instead. For more information, refer to the *IBM Tivoli NetView for z/OS Data Model Reference*.

In the sample network, a GMFHS_Managed_Real_Objects_Class object is defined for each resource of interest in the four non-SNA networks. For example, in the sysplex network shown in Figure 3, a GMFHS_Managed_Real_Objects_Class object is defined for the following resources:
- A sysplex named PLEX1
- z/OS systems RALXT1 and RALXT2

GMFHS_Aggregate_Objects_Class Objects

GMFHS_Aggregate_Objects_Class objects represent a group of objects. This group of objects can consist of any number and combination of real objects and aggregate objects. Examples of aggregate objects are data centers, complex circuits composed of multiple components, and arbitrary groups of resources.

You can define an aggregate object to GMFHS and relate it to underlying GMFHS_Managed_Real_Objects_Class objects. The status of the aggregate object is determined by the status of the real objects that the aggregate object represents. If you have added child classes to the GMFHS_Aggregate_Objects_Class, you need to create objects of the child classes instead.

You can also define an aggregate object that is composed of other aggregate objects. The status of this higher-level aggregate object is determined by the status of the real objects that contribute to the status of the lower-level aggregate objects. The status of the lower-level aggregate objects does not contribute to the status of the higher-level aggregate object; only real objects contribute to the status of aggregate objects.

Because GMFHS_Shadow_Objects_Class objects do not have status fields, the real resources that they represent do not contribute to the status of an aggregate object.

GMFHS supports up to nine levels of aggregation. A level of aggregation is one aggregate object composed of one or more real or aggregate objects. If a real object is defined as a child of an aggregate parent object and that aggregate parent object is defined as a child of another parent aggregate object, two levels of aggregation have been defined.

Aggregate objects must be defined in a strict hierarchy. An aggregate object cannot be defined as a child aggregate object of an aggregate object that is below it in the aggregation hierarchy.

For more information about using aggregation, see “Aggregation Concepts” on page 130.

In the sample network, an aggregate object has been defined for each of the non-SNA networks: Ethernet, Sysplex, ITNM, and TRLAN. Each of these aggregate objects represents all of the real resources in the respective network. The status of each of these aggregate objects reflects the collective status of the underlying real resources.

Two other aggregate objects are also defined:
- Aggregate object WESTCTR is composed of the ETHERNET and SYSPLEX aggregate objects. The status of WESTCTR is determined by the status of the real resources in the Ethernet and sysplex networks.
- Aggregate object EASTCTR is composed of aggregate objects ITNM and TRLAN. The status of EASTCTR is determined by the status of the real resources in the ITNM and TRLAN networks.

These aggregate objects appear in the high-level view described in “Identifying Views” on page 28.

Identifying Connectivity Relationships

Connectivity relationships are ways in which resources defined in RODM can be connected to each other. These relationships can be physical, logical, or peer. The GMFHS data model supports the following relationships:
- ComposedOfLogical and IsPartOf
- ComposedOfPhysical and IsPartOf
- AggregationParent and AggregationChild
- ParentAccess and ChildAccess
- PhysicalConnPP
- LogicalConnPP
- PhysicalConnUpstream and PhysicalConnDownstream
- LogicalConnUpstream and LogicalConnDownstream
- BackboneConnPP

ComposedOfLogical and IsPartOf

ComposedOfLogical and IsPartOf create a logical relationship in which one object is logically composed of other objects. The other objects, in turn, are part of the first object. This logical relationship can be between any number of real objects, aggregate objects, or shadow objects.

In the sample network, shadow object NETV.WECONN represents the gateway connectors between NCP A04A54C and NCP B30A54C. It has a ComposedOfLogical relationship with the shadow objects V01LG01 and V63LG01. These GMFHS_Shadow_Objects_Class objects in turn have an IsPartOf relationship with the GMFHS_Shadow_Objects_Class object NETV.WECONN.

If the SNA topology manager is installed, the ComposedOfLogical relationship can be done using the SNA topology manager object instead of the shadow object.

When an operator selects the NETV.WECONN object in a view and requests more detail, GMFHS follows the ComposedOfLogical relationship for the NETV.WECONN object to retrieve all objects satisfying this relationship. GMFHS builds a view consisting of these objects, and sends it to the workstation for display. If a ComposedOfPhysical relationship is also defined on the NETV.WECONN object, GMFHS also builds a view of that relationship and sends it to the workstation for display.

ComposedOfPhysical and IsPartOf

ComposedOfPhysical and IsPartOf create a physical relationship in which one object is physically composed of other objects. The other objects are, in turn, part of the first object.

In the sample network, the GMFHS_Aggregate_Objects_Class object named SYSPLEX, representing an entire non-SNA network, has a ComposedOfPhysical relationship with objects in RODM representing the host and two minicomputers, as shown in Figure 3 on page 20. The GMFHS_Managed_Real_Objects_Class objects in RODM representing these resources, in turn, have an IsPartOf relationship with aggregate object SYSPLEX.

If an operator selects the SYSPLEX object in a view and asks for more detail, GMFHS follows the ComposedOfPhysical relationship for the SYSPLEX object to retrieve all objects satisfying this relationship from RODM, builds a view consisting of these objects, and sends it to the workstation for display to the requesting operator. If a ComposedOfLogical relationship is also defined on the SYSPLEX object, GMFHS builds a view of that relationship also and sends it to the workstation for display, along with the ComposedOfPhysical relationship view.

Although ComposedOfPhysical and IsPartOf are generally used to define a relationship between an aggregate object and underlying real objects, this is not the only use for this relationship. For example, you can define an object of the GMFHS_Managed_Real_Objects_Class as being composed of other GMFHS_Managed_Real_Objects_Class objects. In this case no aggregation occurs, but if the operator selects the first object and asks for more detail, a view of the objects that the first object is composed of is displayed.

AggregationParent and AggregationChild

AggregationParent and AggregationChild create a relationship in which one object is the aggregate parent for one or more aggregation children. The status of the aggregate parent is determined by the status of the aggregation children.

The AggregationParent field of a real object links to all of the aggregate objects to which that real object contributes status; a real object can contribute status to any number of aggregate objects. The AggregationChild field of an aggregate object links to all of the real objects that contribute status to that aggregate object.

You do not directly create links between the AggregationParent fields and AggregationChild fields in the GMFHS data model. Instead, GMFHS supplies a method, DUIFCUAP, that links these fields. For example, the following RODM load function primitive statement links the AggregationParent field of the real object Sysplex.PLEX1.RALXT2 to the AggregationChild field of the aggregate object Sysplex:

```
OP DUIFCUAP INVOKED_WITH (SELFDEFINING)
   ((CHARVAR)'LINK'
   (CHARVAR)'GMFHS_Managed_Real_Objects_Class.Sysplex.PLEX1.RALXT2'
   (CHARVAR)'GMFHS_Aggregate_Objects_Class.Sysplex');
```

The DUIFCUAP method is also used to remove these links.

ParentAccess and ChildAccess

The ParentAccess and ChildAccess fields are used by GMFHS to build Configuration Parents views and Configuration Children views. ParentAccess and ChildAccess create a relationship in which one object is the parent for one or more children objects.

When an operator selects a resource and asks for a Configuration Parents view, GMFHS retrieves the resource from RODM and determines the resource's entire ancestry. It then builds a view of the objects that satisfy this relationship and displays the view at the workstation.

This relationship is often useful in hierarchically-arranged networks for determining a path to an owner of a resource. Define both the ParentAccess and ChildAccess relationships if you want to use either the Configuration Parents view or the Configuration Children view.

PhysicalConnPP

PhysicalConnPP creates a relationship in which one resource is physically connected to another resource in a peer-to-peer relationship. This connection can be either a node to link connection or a node to node connection. If the connection is node to node, GMFHS inserts a null connector between the two nodes when it displays a view containing the two objects.

In the sample network, the host in the sysplex network is connected by PhysicalConnPP relationships to two links, which are in turn connected by PhysicalConnPP relationships to minicomputers. When the operator selects a resource and asks to see a view consisting of those resources that are physically connected, GMFHS uses this relationship to build and display the view.

LogicalConnPP

The LogicalConnPP relationship works the same way as the PhysicalConnPP relationship, except that this relationship is logical rather than physical.

In the sample network, NCP B30A54C is connected to gateway connector V01LG01 through the LogicalConnPP relationship. Gateway connector V01LG01 is in turn connected to NCP A04A54C by this same relationship.

PhysicalConnUpstream and PhysicalConnDownstream

PhysicalConnUpstream and PhysicalConnDownstream are used to physically connect objects in which direction is important. These relationships are used when it is important to group resources at one or the other end of a connection.

For example, if you are defining a multipoint link and the resources connected to it, you can use PhysicalConnUpstream to link a controller to the link, and PhysicalConnDownstream to link several terminals to the link. In this case, when the operator asked for a view showing physical connectivity, the controller is linked at one end of the link, and the terminals are all linked at the other end.

LogicalConnUpstream and LogicalConnDownstream

LogicalConnUpstream and LogicalConnDownstream are used to logically connect objects in which direction is important. These relationships are the logical counterpart of the PhysicalConnUpstream and PhysicalConnDownstream relationships.

BackboneConnPP

BackboneConnPP is used to show objects that are part of a subarea backbone.

Identifying Views

GMFHS builds most views based on the relationships defined among the objects that are displayed at the workstation. However, you can define four types of views in which you specify the objects that are to be displayed: exception, network, configuration, or more detail views. The views you define depend upon your network.

Exception Views

An exception view is a collection of real, shadow, and aggregate objects that have been defined as exceptions. No connectivity relationship is shown among these objects. An exception view is simply a graphical list of objects. This list can be filtered by DisplayStatus or UserStatus values of the resource object.

The following list offers just a few examples of how you can define exception views to meet your varying business needs.

- To display all NCPs that are inactive.
- To display all NCPs that are inactive except for those that are being reactivated by an automation routine.
- To define views that contain failing resources that are specific to an operators area of responsibility.
- To show all lines that have failed.
- To define the time of day that a resource can be included in an exception view. For example, suppose you have a workstation on a token-ring LAN that is represented as a PU. During the day, you want to monitor the workstation to ensure that its status is satisfactory. When you turn off the workstation at the end of the day, the status of the PU changes to unsatisfactory. Depending on your exception view definition, the PU is included in an exception view. To prevent this, you can create two definitions: one for regular hours and one for off hours. At the end of the business day a timer starts an automation routine to change from the regular hours definition to the off hours definition, and the PUs is then excluded from the exception view. For more information, see “Defining Exception View Objects and Criteria” on page 100.

Figure 7 shows an example of an exception view.

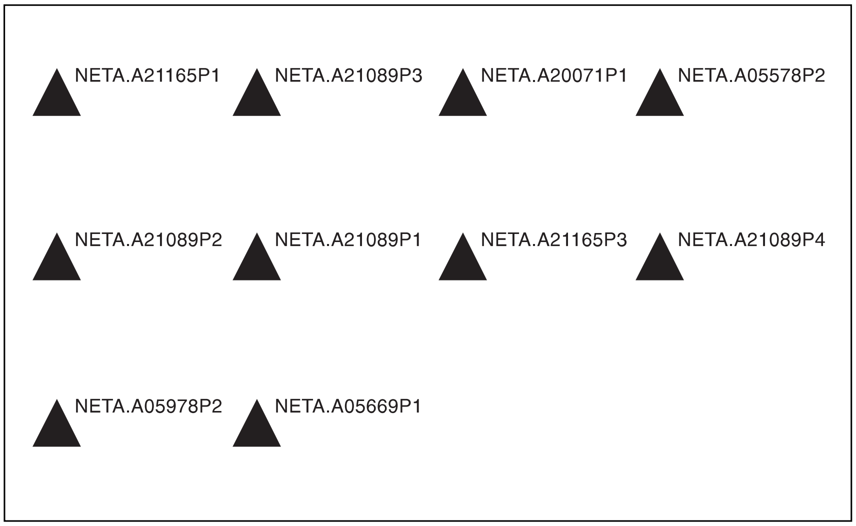


Figure 7. Exception View Example

Network Views

A network view is a collection of real, aggregate, and shadow objects that the operator is to view together. When the operator selects a network view, GMFHS retrieves the appropriate view object from RODM and determines what objects are specified as being part of this view. GMFHS then retrieves these objects, builds a view containing them, and displays the view at the workstation. If the objects have any logical or physical connectivity relationships defined among them, these relationships are shown in the view.

Two of the network views defined for the sample network are:

- A high-level view named BIGPIC, which shows the status of the non-SNA components of the network at a high level.
- A management view named SAMPNET, which shows the major SNA and non-SNA components of the network that are involved in managing the non-SNA networks.

Figure 8 on page 30 shows the high-level view named BIGPIC. In this view, WESTCTR is an aggregate object composed of the ETHERNET and SYSPLEX aggregate objects. EASTCTR is an aggregate object composed of aggregate objects TRLAN and ITNM. Aggregate objects ETHERNET, SYSPLEX TRLAN, and ITNM represent the real objects in each of the non-SNA networks being managed.

When real objects change status, their status is reflected up to aggregate objects ETHERNET, SYSPLEX TRLAN, and ITNM, and also to aggregate objects WESTCTR and EASTCTR. High-level view BIGPIC, therefore, presents operators with a view that represents all of the non-SNA real objects being managed.

If the status of WESTCTR changes from satisfactory to degraded, the operator can select the WESTCTR object and ask for more detail. A view consisting of the ETHERNET and SYSPLEX aggregate objects is displayed. Or the operator can select the object and request a fast path to failing resource view. This view consists of the real objects in the ETHERNET and SYSPLEX aggregate objects that are in an exception state. This type of view can be valuable in a network that contains many real and aggregate objects.

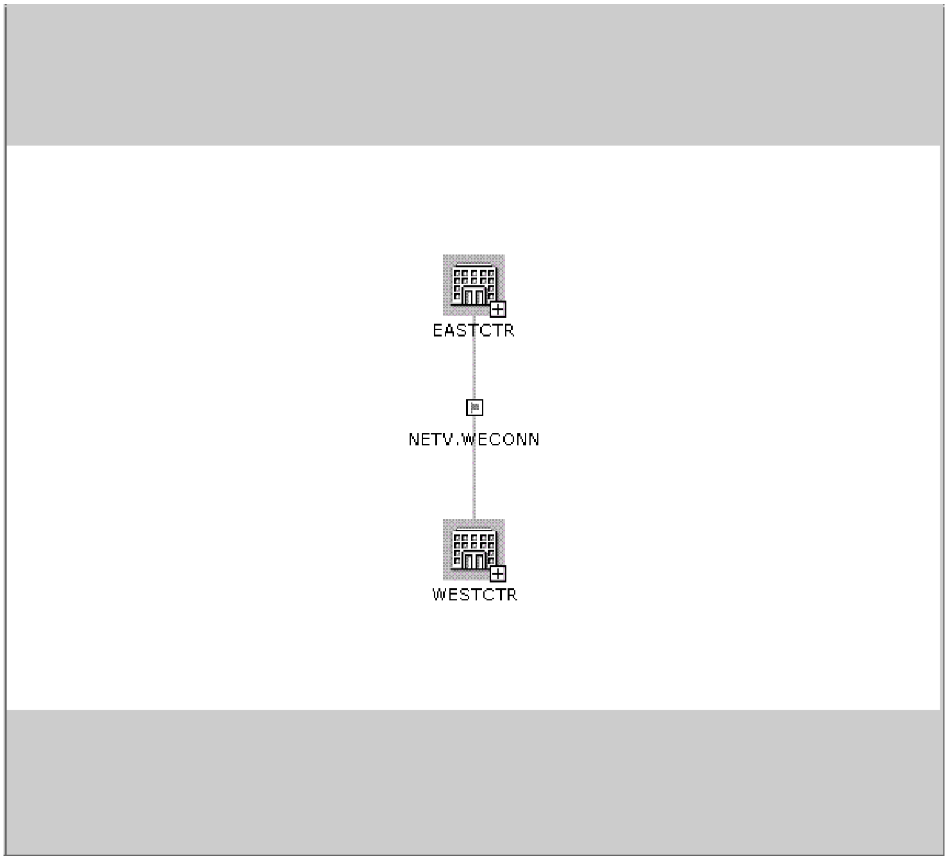


Figure 8. High-Level View BIGPIC

Figure 9 on page 31 shows the management view named SAMPNET. This view displays the major SNA and non-SNA components of the network. It contains the SNA hosts, NCPs, and service points as well as the logical gateway connectors linking the two NCPs. Connected to the service points that are network management gateways are the aggregate objects ETHERNET, SYSPLEX, TRLAN, and ITNM. The SNA resources shown are defined to GMFHS as GMFHS_Shadow_Objects_Class objects.

This view shows the major SNA and non-SNA components involved in managing the non-SNA networks in the sample, and the relationships among them. The operator can see the status of both the SNA and the non-SNA objects. If a non-SNA aggregate changes status, the operator can select it and ask for a more detailed view to find the source of the status change.

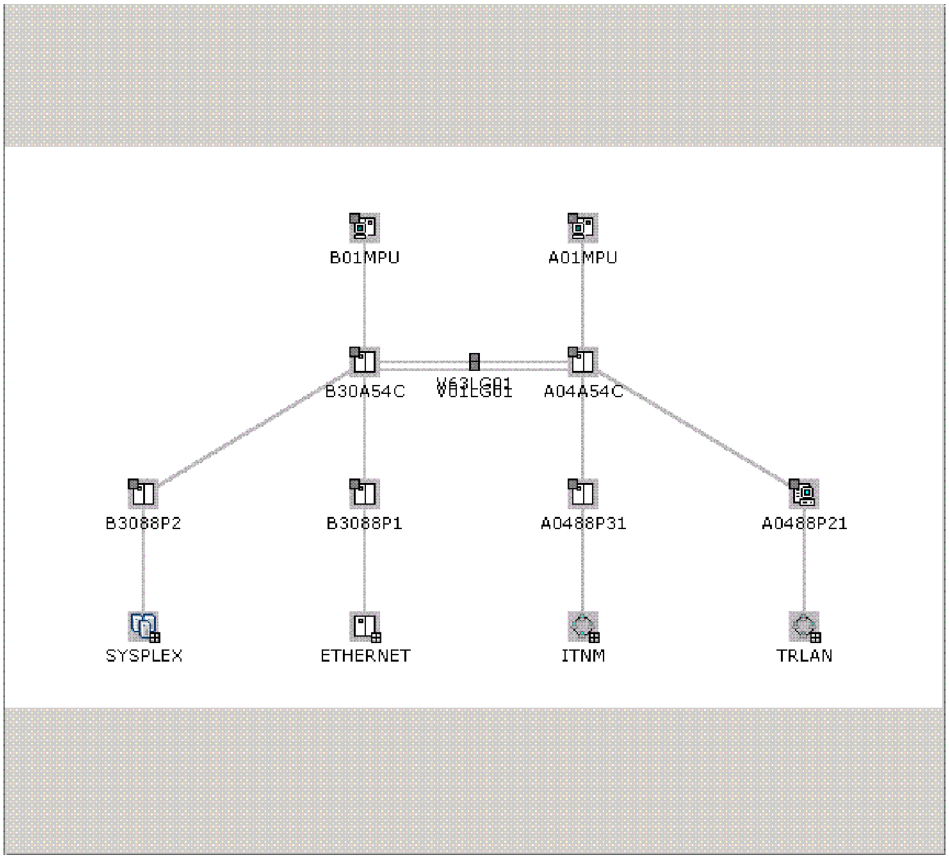


Figure 9. Management View SAMPNET

Configuration Views

The following configuration views are predefined views. They are used to show objects in relationship to other objects.

View Type	Description
Peer	Displays objects that have a peer relationship.
Physical	Displays objects in a network based on a physical relationship between objects.
Logical	Displays objects in a network based on a logical relationship between objects.
Backbone	Displays objects that constitute a subarea backbone.

The following configuration views can also be dynamically built views:
- Backbone
- Logical
- Physical

For more information about configuration views, see “Object Discovery Process Description for Specific Views” on page 94. The sample network contains a configuration peer view, which is described next.

A configuration peer view is a collection of objects that share a peer relationship in the network displayed in a view. You specify the objects that are to appear in a configuration peer view when you define the view. Although you can specify any type of displayable object in a peer view, select only those objects that share a peer relationship. It is up to you to decide which objects have such a relationship.

When the operator selects a resource in a view and asks to see any peer views in which that object is defined, GMFHS uses the peer view objects you define to construct the appropriate views and sends them to the requesting operator's workstation for display. As with network views, if the objects have any logical or physical connectivity relationships defined among them, these relationships are shown in the view.

Figure 10 is a peer view containing three objects from the ETHERNET network in the sample network. This view contains:
- Connector OEMLAB
- Connector NSL_ENET
- Connector NSL_B202

The names used in this peer view are determined by the DisplayResourceName field of the objects. For example, the MyName value of the object displayed as OEMLAB is LATTVIEW.656_MAIN.CNTR3000.SL02P0.

Each of the three objects in this peer view are linked to the DisplayResourceType object DUIXC_RTN_LAN_ADAPTER. The DUIU5N01 icon and the trapezoid-shaped terminal symbol are specified by the link to DUIXC_RTN_LAN_ADAPTER. No relationships are defined between these objects in the sample network definition, so none are displayed in the view.

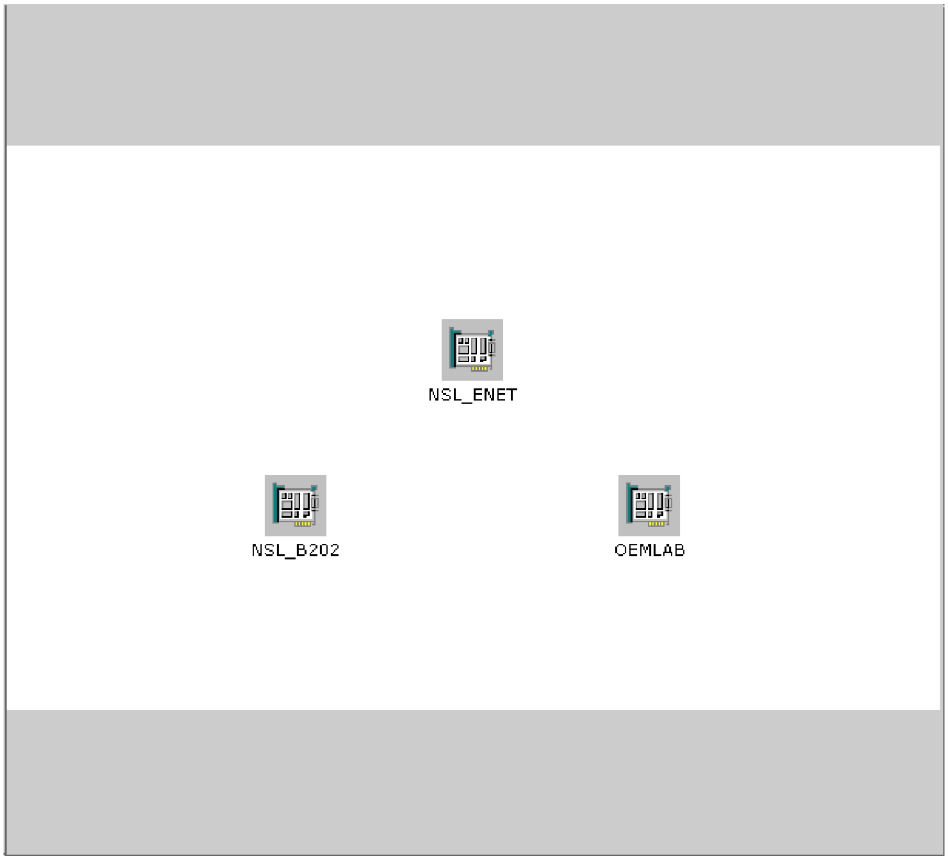


Figure 10. Peer View of ETHERNET Network

More Detail Views

The following more detail views are predefined views. They are used to show objects in relationship to other objects.

View Type	Description
Logical	Displays the next lower layer of objects in a network based on a logical relationship between objects.
Physical	Displays the next lower layer of objects in a network based on a physical relationship between objects.

More detail views can also be dynamically built. For more information, see “More Detail Views” on page 97.

Defining Your Configuration to RODM

You can use the SNA topology manager to define APPN and subarea networks, and you can use MultiSystem Manager to define non-SNA resources in RODM. You can also manually define non-SNA resources in RODM as described next.

After you identify the resources in your network that you want to monitor with GMFHS, you then define those resources to RODM. All resources are defined in terms of RODM load function statements and the GMFHS data model. The source for your definition is one or more RODM load files containing the definition statements.

This section describes how to define each of the objects you identified in the previous section to RODM. For each type of object described previously, a description about how that type of object is defined, the fields that must be defined for that type of object are identified, and a sample object using the RODM load function statements is defined. For more information about the RODM load function statements, see Chapter 10, “Using the RODM Load Function,” on page 239.

You can create the RODM load function statements required to define your network to GMFHS using an editor, or you can write a program to convert from your own configuration database format to the format required by the RODM load function.

Defining Management Objects

Management objects include network management gateways, SNA domains, and non-SNA domains. Create one NMG_Class object for each network management gateway. Create one SNA_Domain_Class object for each SNA domain. Create one or more Non_SNA_Domain_Class objects for each non-SNA domain, depending on the specific information contained in alerts sent from the domain.

Defining SNA Domains

Define one SNA_Domain_Class object for each SNA domain identified in your configuration that provides access to service points that are contained in SNA resources. This object can be displayed in a view; however, the status of SNA_Domain_Class objects is not maintained by GMFHS.

In the sample network, SNA domain B01NV is defined by the following RODM load function statement:

```
-- Create SNA Domain Object for B01NV --
CREATE INVOKER  ::= 0000003;
       OBJCLASS ::= SNA_Domain_Class;
       OBJINST  ::= MyName = (CHARVAR) 'B01NV';

       ATTRLIST
       SNANet   ::= (CHARVAR) 'NETB';
END;
```

The name of an SNA_Domain_Class object in RODM is the 5–character NetView domain identifier.

In this example, the SNA_Domain_Class object named B01NV is in an SNA network named NETB. The object name is specified on the OBJINST parameter and the network name is specified in the field SNANet of the ATTRLIST parameter associated with the CREATE statement for this object. If you are defining more than one SNA domain, the basic information in the definition remains the same for each domain; you need only provide the name of the object and the SNA network to which the domain is related.

Defining Network Management Gateways

Create a network management gateway object for each network management gateway in your network.

In the sample network, the B3088P2 network management gateway is defined by the following RODM load function statement:

```
-- Create NMG Object for B3088P2 --
CREATE INVOKER  ::= 0000004;
       OBJCLASS ::= NMG_Class;
       OBJINST  ::= MyName = (CHARVAR) 'B3088P2';

       ATTRLIST
       Domain   ::= (OBJECTLINK)
       ('SNA_Domain_Class'.'B01NV'.'ContainsResource'),
       CommandRouteLUName ::= (CHARVAR) 'B01NV',
       NMGCharacteristics ::= (ANONYMOUSVAR) x'80',
       AgentStatusEffect      ::= (ANONYMOUSVAR) x'80',
       TransportProtocolName  ::= (CHARVAR) 'COS',
       WindowSize             ::= (INTEGER) 1;
END;
```

The name of the network management gateway object in RODM is determined as follows:

- If the gateway uses the common operator services (COS) facilities of the NetView program to receive commands, the name of the network management gateway object is the PU or LU name associated with the SNA resource that contains the service point.
- If the gateway uses PPI interface to deliver commands and receive command responses and alerts, the network management gateway object name is the program-to-program interface receiver name associated with the network management application to which the commands are sent.
- If the gateway uses command processors and procedures running on an OST, the network management gateway object name can be any name that is unique for objects of this type.

In this example, the value of the TransportProtocolName field is COS, which specifies that either an SSCP-PU or an LU-LU session using the common operations services (COS) architecture is used to transport commands and alerts between service point B3088P2 and the NetView program. The window size is 1, specifying that only 1 command can be outstanding against the NMG.

The CommandRouteLUName field is set to B01NV, specifying that commands in host A01MPU be routed to the service point B3088P2 by a RMTCMD command, which specifies that the commands are first sent over a NetView-NetView session to the NetView program residing in host B01MPU. This NetView program sends a RUNCMD command to service point B3088P2 and routes responses back to the NetView program in host A01MPU.

The TransportProtocolName field specifies how GMFHS communicates with the network management gateway when delivering commands and accepting responses to commands. Valid values for this field are:
- COS
- PPI
- OST
- NONE

Defining Non-SNA Domains

Define one Non_SNA_Domain_Class object for each unique combination of service point (SP), transaction program (TP), and element management subsystem (EMS) in your network. The following combinations uniquely specify an object of the Non_SNA_Domain_Class:
- SP
- SP.TP
- SP.TP.EMS
- TP
- TP.EMS

Note that only the first three entries in the preceding list are valid for the DOMS010 session protocol.

The value of the DisplayStatus field of an object in the Non_SNA_Domain_Class represents the status of the command and response communication session between GMFHS and the transaction program associated with the domain. It does not indicate whether the transaction program is able to forward alert information about the domain to GMFHS. For more information about alert handling, see Chapter 6, “Customizing GMFHS to Process and Receive Alerts and Resolutions,” on page 167.

In the sample network, Non_SNA_Domain_Class object RTP1 is defined by the following RODM load function statement:

```
-- Create Non-SNA Domain Object for RTP1 --
CREATE INVOKER  ::= 0000003;
       OBJCLASS ::= Non_SNA_Domain_Class;
       OBJINST  ::= MyName = (CHARVAR) 'A0488P31.A94306F8.RTP1';

       ATTRLIST
       EMDomain ::= (CHARVAR) 'RTP1',
       DomainCharacteristics ::= (ANONYMOUSVAR) x'0072',
       InitialResourceStatus ::= (INTEGER) 132,
       PresentationProtocolName ::= (CHARVAR) 'PASSTHRU',
       SessionProtocolName ::= (CHARVAR) 'PASSTHRU',
       TransactionProgram ::= (CHARVAR) 'A94306F8',
       ReportsToAgent ::= (OBJECTLINK)
         ('NMG_Class'.'A0488P31'.'ReportsOnDomain');
END;
```

In this example the following field values are specified for the object of the Non_SNA_Domain_Class:

- The MyName field consists of three names, separated by periods:
  - The name of the service point (A0488P31)
  - The name of the transaction program (A94306F8)
  - The name of the element management subsystem (RTP1)

  The element management subsystem contains only the element management domain name; RTP1 in this example.
- The DomainCharacteristics field specifies:
  - The transaction program NAP supports native commands, display status, activate, and deactivate commands.
  - Resource name elements are concatenated with periods building the full name of the reported-on resource.
  - The transaction program returns responses for commands.
  - The soliciting of resource status of real objects in the domain is suppressed.
- The InitialResourceStatus field specifies that a satisfactory status is reported for resources managed by transaction program NAP until the actual resource status is reported by alerts or by response to a command.
- The PresentationProtocolName field specifies DOMP020. The DOMP020 protocol specifies that GMFHS substitutes a command string for each generic command. GMFHS uses the command string from the object of GMFHS_Managed_Real_Objects_Class that is the target of the generic command, or from the object of the Non_SNA_Domain_Class that is the domain of the target of the generic command. For example, GMFHS substitutes the value of the ActivateCommandText field when an activate generic command is selected.
- The SessionProtocolName field specifies PASSTHRU, which means that GMFHS assumes a session exists with the transaction program associated with this domain.
- The ReportsToAgent field specifies that the domain is associated with the service point and the NMG_Class object defined for that service point (A0488P31).

Because in this sample the domain is not displayed in any views, no connectivity is defined for it.

Defining Managed Objects

Managed objects include SNA resources, non-SNA real resources, and aggregate resources. You can use the SNA topology manager to load SNA objects into RODM, or you can manually define GMFHS_Shadow_Objects_Class objects using the process described next. This section describes how to define these resources to RODM.

Note: Because the alerts sent to the NetView program identify resources that have changed status, assign names to managed objects that match the names that are supplied by the alerts. For information about how GMFHS uses resource names from alerts, see Chapter 6, “Customizing GMFHS to Process and Receive Alerts and Resolutions,” on page 167.

Defining SNA Resources

Define one object of the GMFHS_Shadow_Objects_Class for each SNA resource that you want to define to RODM. Although the status of SNA resources is not stored in RODM, you might want to define SNA resources to RODM for one or more of the following reasons:

- To show the relationship between SNA and non-SNA resources

- To obtain alert history for SNA resources
- To obtain SNA alert pending user status

In the sample network, the shadow object for SNA host B01MPU is defined by the following RODM load function statement:

```
-- Create GMFHS Shadow Object for SNA Host B01MPU --
CREATE INVOKER  ::= 0000003;
       OBJCLASS ::= GMFHS_Shadow_Objects_Class;
       OBJINST  ::= MyName = (CHARVAR) 'NETB.B01MPU';
       ATTRLIST
       LocateName ::= (INDEXLIST)((CHARVAR) 'NETB.B01MPU'),
       DisplayResourceName ::= (CHARVAR) 'B01MPU';
END;
OP DUIFCLRT INVOKED_WITH (SELFDEFINING)
   ((CHARVAR)'LINK'
   (CHARVAR)'GMFHS_Shadow_Objects_Class.NETB.B01MPU'
   (CHARVAR)'Display_Resource_Type_Class.DUIXC_RTS_HOST');
```

The name of a shadow object is the SNA network name of the network that contains the SNA object, a period (.), and the SNA name of the resource. In this example, the name is NETB.B01MPU.

In this example, the host B01MPU has a DisplayResourceName of B01MPU; this name is displayed next to the resource in all views that contain the resource. The shadow object is assigned the DisplayResourceType of DUIXC_RTS_HOST, indicating that it is an SNA Host.

You do not define the relationships for GMFHS_Shadow_Objects_Class objects when defining the objects themselves, but do so only after all objects are defined. Therefore, linkages to other objects are defined later in this section.

Defining Non-SNA Real Resources

Define an object of the GMFHS_Managed_Real_Objects_Class for each non-SNA real resource you want to define to RODM. The name of this object is used to correlate alerts received for the resource to the object that represents the resource.

If you added child classes to the GMFHS_Managed_Real_Objects_Class, you need to create fields and objects on the child classes instead. Refer to the *IBM Tivoli NetView for z/OS Data Model Reference* for more information.

If the object you are defining is to be displayed in predefined network, configuration, or more detail views using certain layout algorithms, you might need to define an object of the Layout_Parameters_For_Object_Class for this object. The definition of the Layout_Parameters_For_Object_Class object is described in “Defining Layout Parameters for Network, Configuration, and More Detail Views” on page 46.

In the sample network, the z/OS system RALXT1 is a real resource residing in the Sysplex network. RALXT1 is defined to RODM as a GMFHS_Managed_Real_Objects_Class object by the following RODM load function statement:

```
-- Create a GMFHS Managed Real Object for RALXT1 --
CREATE INVOKER  ::= 0000003;
       OBJCLASS ::= GMFHS_Managed_Real_Objects_Class;
       OBJINST  ::= MyName = (CHARVAR) 'SYSPLEX.PLEX1.RALXT1';

       ATTRLIST
       LocateName ::= (INDEXLIST)((CHARVAR) 'SYSPLEX.PLEX1.RALXT1'),
       DisplayResourceName ::= (CHARVAR) 'RALXT1',
```

```
       Domain   ::= (OBJECTLINK)
       ('Non_SNA_Domain_Class'.'B3088P2.NAP.SYSPLEX'.'ContainsResource')
END;
OP DUIFCLRT INVOKED_WITH (SELFDEFINING)
   ((CHARVAR)'LINK'
   (CHARVAR)'GMFHS_Managed_Real_Objects_Class.SYSPLEX.PLEX1.RALXT1'
   (CHARVAR)'Display_Resource_Type_Class.DUIXC_RTN_STM_SYSTEM')
```

The name of a GMFHS_Managed_Real_Objects_Class object is used to resolve alerts coming in for the real resource. It consists of the character string specified in the EMDomain field of the Non_SNA_Domain_Class object representing the non-SNA domain in which the real resource resides, and the name of the resource as known to its transaction program and element management system, separated by a period.

In this example, the z/OS system is associated with the Non_SNA_Domain_Class object B3088P2.NAP.SYSPLEX, and is given a DisplayResourceType of DUIXC_RTN_STM_SYSTEM. Because the DisplayResourceName field is specified, the name that is displayed to the operator in conjunction with this resource when it is displayed in views is RALXT1.

The link between an object of the GMFHS_Managed_Real_Objects_Class and an object of the Display_Resource_Type_Class is created by a RODM load function primitive statement that triggers the DUIFCLRT method. RODM load function primitive statements are described in “Load Function Primitive Statements” on page 242. The DUIFCLRT method is described in “DUIFCLRT: Link Resource Type Method” on page 489.

Defining GMFHS Aggregate Objects

Aggregate objects can be used to group resources into a higher-level resources for monitoring purposes. You can also use exception views to monitor the resources directly. For more information, see “Defining Exception View Objects and Criteria” on page 100.

Define one GMFHS_Aggregate_Objects_Class object for each aggregate object that you want to display in a view. If you have added child classes to the GMFHS_Aggregate_Objects_Class, you need to create objects of the child classes instead. To define a GMFHS aggregate object:

- Specify the composite relationships of the elements of the aggregate object.
- Specify which resources belong to the aggregate object.
- Set up the hierarchies between the aggregation parent and the aggregation children.

The sample network contains a SYSPLEX object that is composed of real resource representing the sysplex as a whole, two z/OS systems, and two links between the z/OS systems and the sysplex, as illustrated in Figure 3 on page 20. An aggregate object, named SYSPLEX, is defined to represent the sysplex network. The SYSPLEX aggregate object is included in a high-level view, and its status represents the collective status of the resources it represents. The GMFHS_Aggregate_Objects_Class object for the network SYSPLEX is defined by the following RODM load function statements:

```
-- Create a GMFHS Aggregate Object for SYSPLEX --
CREATE INVOKER  ::= 0000004;
       OBJCLASS ::= GMFHS_Aggregate_Objects_Class;
       OBJINST  ::= MyName = (CHARVAR) 'SYSPLEX';
       ATTRLIST
       ThresholdDegraded ::= (INTEGER) 1,
```

```
       ThresholdSeverelyDegraded ::= (INTEGER) 2,
       ThresholdUnsatisfactory ::= (INTEGER) 3,
       ComposedOfPhysical ::= (OBJECTLINKLIST)
('GMFHS_Managed_Real_Objects_Class'.'SYSPLEX.PLEX1'.'IsPartOf')
('GMFHS_Managed_Real_Objects_Class'.'SYSPLEX.PLEX1.RALXT1'.'IsPartOf'
('GMFHS_Managed_Real_Objects_Class'.'SYSPLEX.PLEX1.RALXT2'.'IsPartOf'
('GMFHS_Managed_Real_Objects_Class'.'SYSPLEX.PLEX1.LINK1'.'IsPartOf')
('GMFHS_Managed_Real_Objects_Class'.'SYSPLEX.PLEX1.LINK2'.'IsPartOf');
END;
OP DUIFCLRT INVOKED_WITH (SELFDEFINING)
   ((CHARVAR)'LINK'
   (CHARVAR)'GMFHS_Aggregate_Objects_Class.SYSPLEX'
 (CHARVAR)'Display_Resource_Type_Class.DUIXC_RTN_STM_SYSPLEX_AGG');

OP DUIFCUAP INVOKED_WITH (SELFDEFINING)
   ((CHARVAR)'LINK'
   (CHARVAR)'GMFHS_Managed_Real_Objects_Class.SYSPLEX.PLEX1'
   (CHARVAR)'GMFHS_Aggregate_Objects_Class.SYSPLEX');

OP DUIFCUAP INVOKED_WITH (SELFDEFINING)
   ((CHARVAR)'LINK'
   (CHARVAR)'GMFHS_Managed_Real_Objects_Class.SYSPLEX.PLEX1.RALXT1'
   (CHARVAR)'GMFHS_Aggregate_Objects_Class.SYSPLEX');

OP DUIFCUAP INVOKED_WITH (SELFDEFINING)
   ((CHARVAR)'LINK'
   (CHARVAR)'GMFHS_Managed_Real_Objects_Class.SYSPLEX.PLEX1.RALXT2'
   (CHARVAR)'GMFHS_Aggregate_Objects_Class.SYSPLEX');

OP DUIFCUAP INVOKED_WITH (SELFDEFINING)
   ((CHARVAR)'LINK'
   (CHARVAR)'GMFHS_Managed_Real_Objects_Class.SYSPLEX.PLEX1.LINK1'
   (CHARVAR)'GMFHS_Aggregate_Objects_Class.SYSPLEX');

OP DUIFCUAP INVOKED_WITH (SELFDEFINING)
   ((CHARVAR)'LINK'
   (CHARVAR)'GMFHS_Managed_Real_Objects_Class.SYSPLEX.PLEX1.LINK2'
   (CHARVAR)'GMFHS_Aggregate_Objects_Class.SYSPLEX');
```

The definition of an aggregate object involves two sets of relationships: the ComposedOfPhysical and IsPartOf relationship, and the AggregationParent and AggregationChild relationship. The ComposedOfPhysical and IsPartOf relationship determines which objects are displayed in a view when the operator selects an object in another view and asks for more detail. The AggregationParent and AggregationChild relationship determines which real resources are used to calculate the status of an aggregate resource.

In this example, the ComposedOfPhysical field of the SYSPLEX aggregate object is linked to the IsPartOf fields of the following GMFHS_Managed_Real_Objects_Class objects:
- SYSPLEX.PLEX1
- SYSPLEX.PLEX1.RALXT1
- SYSPLEX.PLEX1.RALXT2
- SYSPLEX.PLEX1.LINK1
- SYSPLEX.PLEX1.LINK2

This ComposedOfPhysical and IsPartOf relationship specifies that GMFHS is to construct a view consisting of the specified GMFHS_Managed_Real_Objects_Class objects and display that view at the workstation when the operator selects the v object in a view and asks for more detail.

The SYSPLEX aggregate object is assigned a DisplayResourceType of DUIXC_RTN_STM_SYSPLEX_AGG, which indicates that the object represents a non-SNA aggregate host. The link between an object of the GMFHS_Aggregate_Objects_Class and an object of the Display_Resource_Type_Class is created by a RODM load function primitive statement that triggers the DUIFCLRT method. The DUIFCLRT method is described in "DUIFCLRT: Link Resource Type Method" on page 489.

The SYSPLEX object is an aggregate host that represents the underlying real resources in the sysplex network. An AggregationParent and AggregationChild link is created between this aggregate parent and its aggregate children by RODM load function primitive statements using the DUIFCUAP method. The DUIFCUAP method is described in "DUIFCUAP: Update Aggregation Path Method" on page 491.

In general, the ComposedOfPhysical and IsPartOf relationship and the AggregationParent and AggregationChild relationship are used in conjunction; however, they can be used separately. For example, if you wanted a real resource to appear in a more detailed view for an aggregate resource but did not want it to contribute to the status of the aggregate resource, you can define the ComposedOfPhysical and IsPartOf relationship for the aggregate object and real object pair, but not define the AggregationParent and AggregationChild relationship.

As another example, you might want to define a GMFHS_Managed_Real_Objects_Class object as being composed of other GMFHS_Managed_Real_Objects_Class objects. Then, when the user selects the first object and asks for more detail, the objects that are defined as part of the first object are displayed. Because the first object is not an aggregate object, the AggregationParent and AggregationChild relationship is not defined in this case.

Defining Connectivity Relationships Between Objects

Connectivity relationships between objects can determine which objects appear in views and which resources contribute to the status of aggregate objects. With the exception of relationships involving shadow objects, these connectivity relationships, described in "Identifying Connectivity Relationships" on page 25, can be defined when the objects are defined or any time after the objects are defined. Connectivity relationships that include shadow objects can be defined only after the shadow objects have been defined. This section illustrates how to define some of these relationships using examples from the sample network.

Defining Logical Connectivity

Objects can be connected with logical links using the LogicalConnPP field or the LogicalConnUpstream and LogicalConnDownstream fields of the objects that are to be connected. In the sample network, the shadow object that represents SNA host B01MPU is logically connected to the shadow object that represents SNA NCP B30A54C to create the relationship illustrated in Figure 9 on page 31 by using the following RODM load function statement:

```
-- Link Host B01MPU to NCP B30A54C --
OP 'GMFHS_Shadow_Objects_Class'.'NETB.B01MPU'.'LogicalConnPP'
IS_LINKED_TO
'GMFHS_Shadow_Objects_Class'.'NETB.B30A54C'.'LogicalConnPP';
```

For each object that is to be linked, the class information for the object, the object name, and the field that determines the type of link that is being defined needs to be specified.

Defining Physical Connectivity

Objects can be connected with physical links by using the PhysicalConnPP field or the PhysicalConnUpstream and PhysicalConnDownstream fields of the objects that are to be connected. In the sample network, non-SNA host RALV4 is physically linked to link LINK1 by using the following RODM load function statements:

```
-- Link RALXT1 to Link LINK1 --
OP 'GMFHS_Managed_Real_Objects_Class'.
                                  'SYSPLEX.PLEX1.RALXT1'.'PhysicalConnPP'
IS_LINKED_TO
'GMFHS_Managed_Real_Objects_Class'.'SYSPLEX.PLEX1.LINK1'.'PhysicalConnPP'
```

For each object that is to be linked, the class information for the object, the object name, and the field that determines the type of link that is being defined needs to be specified.

Defining Parent-Child Relationships

Parent and Child links are defined using the ChildAccess and ParentAccess fields of the objects that are to be linked. In the sample network, the z/OS system RALXT1 is linked to the sysplex in the configuration illustrated in Figure 3 on page 20 by using the following RODM load function statement:

```
-- Link PLEX1 to RALXT1 --
OP 'GMFHS_Managed_Real_Objects_Class'.'SYSPLEX.PLEX1'.'ChildAccess'
IS_LINKED_TO
'GMFHS_Managed_Real_Objects_Class'.'SYSPLEX.PLEX1.RALXT1'.'ParentAccess'
```

For each object that is to be linked, the class information for the object, the object name, and the field that determines whether the object is the parent or the child needs to be specified.

Defining Views

The following kinds of views can be defined in RODM:
- Exception
- Network
- Configuration
- More detail

When defining view objects, always use the RODM high-level load function statements. RODM high-level load function statements allow all fields on the object to be defined before the object is used. If RODM primitive statements are used, GMFHS might attempt to access information about the view object before all of the information is defined, and this can result in unexpected errors. For more information about high-level load function and primitive statements, see to Chapter 10, “Using the RODM Load Function,” on page 239.

The views that are constructed in RODM are displayed by the NetView management console. The following sections describe parameters and layout algorithms that are used by the graphic facility. See Appendix B, “View Layout Facility,” on page 657 for more information about views.

Defining Exception Views

Exception views are represented by objects in the Exception_View_Class. Create one object in this class for each exception view you want to display. Use the NetView management console to display a list of all defined views.

The sample network does not include an exception view. However, sample DUIFDEXV provides an example of defining exception view objects, and the

RODM load function statements in this section can be used to define an exception view. Figure 11 shows an exception view of all objects in the GMFHS_Displayable_Objects_Parent_Class that have DisplayStatus of either severely degraded or unsatisfactory.

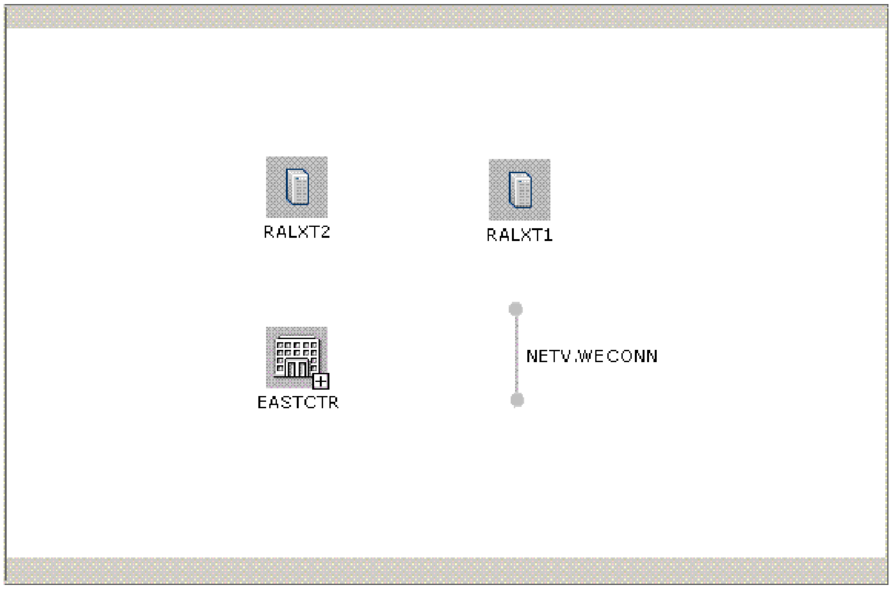


Figure 11. Exception View of a Network

The exception view EXCEPTIONVIEW1 is defined by the following RODM load function statement:

```
CREATE INVOKER  ::= 0000001;
       OBJCLASS ::= Exception_View_Class;
       OBJINST  ::= MyName = (CHARVAR) 'EXCEPTIONVIEW1';
       ATTRLIST
       Annotation ::= (CHARVAR) 'Monitored by Operator A',
       ExceptionViewName ::= (CHARVAR) 'EXVIEW1',
END;
```

Use the following statement to define all objects of the class GMFHS_Displayable_Objects_Parent_Class to be in EXCEPTIONVIEW1. Note that you do not have to define ExceptionViewList fields at the class level. You can also define the ExceptionViewList field at the object level.

```
OP 'GMFHS_Displayable_Objects_Parent_Class'..
   'ExceptionViewList'
HAS_VALUE (INDEXLIST)((CHARVAR) 'EXVIEW1');
```

For more information defining objects to exception views, see “Defining Exception View Objects and Criteria” on page 100.

Defining Network Views

Network views are represented by objects in the Network_View_Class. Create one object in this class for each network view you want to display. The NetView management console can display a list of all defined views.

Figure 12 on page 43 shows a network view of the SYSPLEX network component of the sample network. The icon and symbol displayed for each object are determined by the DisplayResourceType object to which it is linked. For example, the resource SYSPLEX.PLEX1.RALXT1 is linked to DUIXC_RTN_STM_SYSTEM. The icon DUIUEB2B and the square-shaped host symbol are specified by

DUIXC_RTN_STM_SYSTEM. The name RALXT1 shown in the view is specified by the DisplayResourceName field of object SYSPLEX.PLEX1.RALXT1.

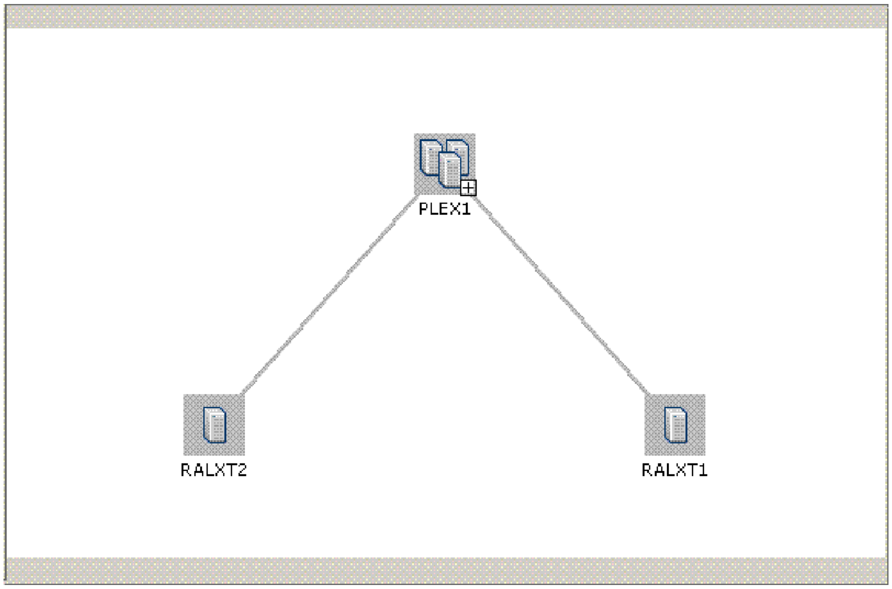


Figure 12. Network View of the Sysplex Network

The network view of the sysplex network is defined by the following RODM load function statement:

```
-- Create Network View for SYSPLEX --
CREATE INVOKER  ::= 0000004;
       OBJCLASS ::= Network_View_Class;
       OBJINST  ::= MyName = (CHARVAR) 'SYSPLEX';
       ATTRLIST
       Annotation ::= (CHARVAR) 'SYSPLEX NETWORK',
       ContainsObjects ::= (OBJECTLINKLIST)
('GMFHS_Managed_Real_Objects_Class'.
                             'SYSPLEX.PLEX1.RALXT1'.'ContainedInView')
('GMFHS_Managed_Real_Objects_Class'.
                             'SYSPLEX.PLEX1.RALXT2'.'ContainedInView')
('GMFHS_Managed_Real_Objects_Class'.
                             'SYSPLEX.PLEX1.LINK1'.'ContainedInView')
('GMFHS_Managed_Real_Objects_Class'.
                             'SYSPLEX.PLEX1.LINK2'.'ContainedInView')
('GMFHS_Managed_Real_Objects_Class'.
                             'SYSPLEX.PLEX1'.'ContainedInView');
END;
```

In this example, a Network_View_Class object named SYSPLEX is defined to represent the network view of the sysplex network. The Annotation field of the object is assigned the value `SYSPLEX NETWORK`, which is displayed at the workstation with the view. The ContainsObjects field of the SYSPLEX object is linked to the ContainedInView fields of the managed real objects that represent the real resources that make up the SYSPLEX network.

Defining Configuration Views

Configuration views are created by defining an object to represent the view on one of the following classes:

View Type	Class Defined
Peer	Configuration_Peer_View_Class

Physical	Configuration_Physical_Connectivity_View_Class
Logical	Configuration_Logical_Connectivity_View_Class
Backbone	Configuration_Backbone_View_Class

Create one object on its respective class for each configuration view you want to display. Because the sample network contains a configuration peer view, an example of defining a Configuration_Peer_View_Class object follows. Use a similar procedure to define objects on any of the other configuration view type classes. The following configuration views can also be dynamically built views:
- Backbone
- Logical
- Physical

For more information about configuration views, see "Object Discovery Process Description for Specific Views" on page 94.

Defining Peer Views: Figure 13 on page 45 is a peer view of the token-ring LAN component. Peer views are represented by objects in the Configuration_Peer_View_Class. Create one object in this class for each peer view you want to display.

Figure 13 on page 45 is a peer view of the token-ring LAN component of the sample network. The icon and symbol displayed for each object are determined by the DisplayResourceType object to which it is linked. For example, the aggregate resource BRIDGE01 is linked to DUIXC_RTN_BRIDGE_AGG. The icon DUIU4N02 and the hexagon-shaped node symbol are specified by DUIXC_RTN_BRIDGE_AGG. Because BRIDGE01 is an aggregate resource, the node symbol contains the smaller aggregate symbol as well. The name BRIDGE01 shown in the view is specified by the DisplayResourceName field of object BRIDGE01.

Note that the sample network defines a real object named LANMGR.BRIDGE01 that also has a DisplayResourceName value of BRIDGE01. The BRIDGE01 in this view is an object of the GMFHS_Aggregate_Objects_Class.

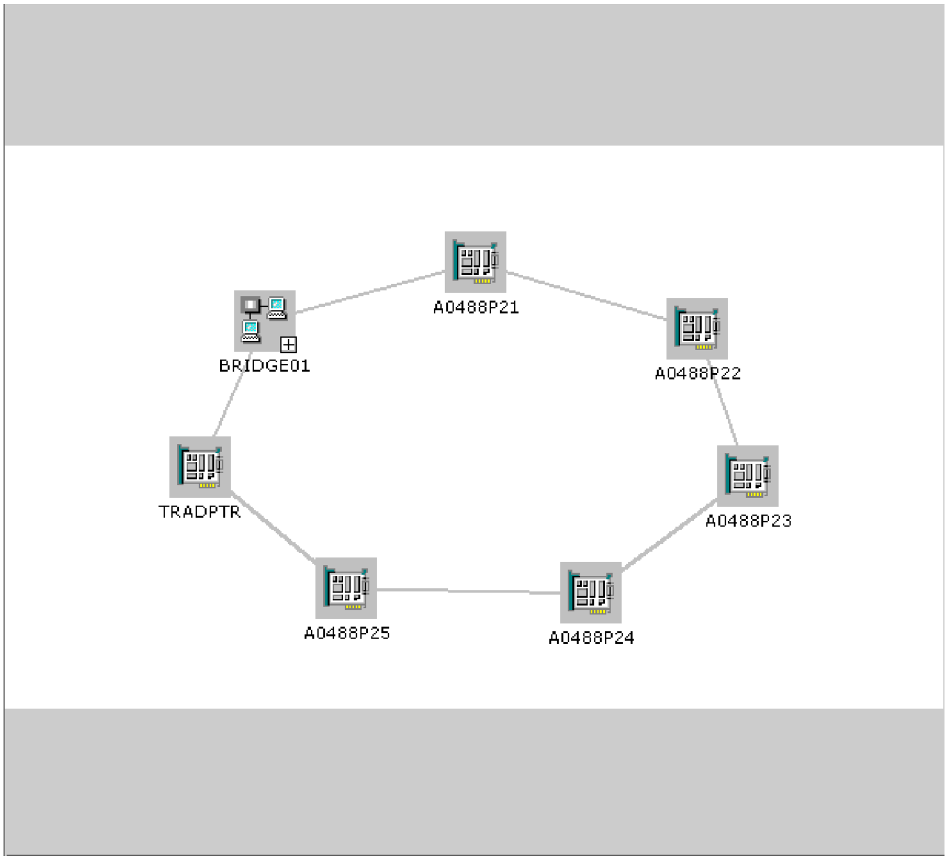


Figure 13. Peer View of Token-Ring Network TRLANNET

The configuration peer view of the token-ring LAN network is defined by the following RODM load function statement:

```
-- Create Configuration Peer View TRLANNET --
CREATE INVOKER  ::= 0000004;
       OBJCLASS ::= Configuration_Peer_View_Class;
       OBJINST  ::= MyName = (CHARVAR) 'TRLANNET_Peer';
       ATTRLIST
       Annotation ::= (CHARVAR) 'Token Ring Network',
       LayoutType ::= (INTEGER) 4,
       ConnType ::= (ANONYMOUSVAR) x'80',
       FirstNode  ::= (OBJECTLINK)
('GMFHS_Managed_Real_Objects_Class'.'LANMGR.10005AC35CA0'.'IsFirstNode'),
       SecondNode ::= (OBJECTLINK)
('GMFHS_Managed_Real_Objects_Class'.'LANMGR.10005A95E7CC'.'IsSecondNode'),
       ContainsObjects ::= (OBJECTLINKLIST)
('GMFHS_Managed_Real_Objects_Class'.'LANMGR.10005AC35CA0'.'ContainedInView')
('GMFHS_Managed_Real_Objects_Class'.'LANMGR.10005A95E7CC'.'ContainedInView')
('GMFHS_Managed_Real_Objects_Class'.'LANMGR.10005A89A267'.'ContainedInView')
('GMFHS_Managed_Real_Objects_Class'.'LANMGR.10005A966BAB'.'ContainedInView')
('GMFHS_Managed_Real_Objects_Class'.'LANMGR.10005A95A08C'.'ContainedInView')
('GMFHS_Managed_Real_Objects_Class'.'LANMGR.400076041088'.'ContainedInView')
('GMFHS_Aggregate_Objects_Class'.'BRIDGE01'.'ContainedInView');
END;
```

In this example, a Configuration_Peer_View_Class object named TRLANNET_Peer is defined to represent the configuration peer view of the token-ring LAN network. The Annotation field of the object is assigned the value `Token Ring Network`; the

LayoutType field is assigned the value 4, which specifies radial layout for token-ring networks. The ConnType field is assigned value 80, as in the previous network view example.

When you create a view, you specify the object names of the objects that appear in the view. The object names in the RODM load function statements in this example are different from the names shown in Figure 13 on page 45, because the sample network uses the DisplayResourceName field to specify the name that is displayed for each resource in the token-ring network. For example, the object LANMGR.10005AC35CA0 has its DisplayResourceName field set to A0488P21.

The FirstNode field of the TRLANNET_Peer object is linked to the IsFirstNode field of the object that is to be displayed at the top of the ring in the configuration peer view. The SecondNode field links to the object that is to be displayed to the right of the first node in the view. The ContainsObjects field links to the remaining objects that are to be displayed in the view. These objects are displayed in the view in the order in which they are defined.

Defining More Detail Views

More detail views are created by defining an object to represent the view on one of the following classes:

View Type	Class Defined
Physical	More Detail_Physical_View_Class
Logical	More Detail_Logical_View_Class

Create one object on its respective class for each more detail view you want to display. Note that these views can also be dynamically built views.

The sample network does not include a predefined more detail view. For more information about more detail views, see “More Detail Views” on page 97.

Defining Layout Parameters

Layout parameters can be specified for the following types of views:

- Network
- Configuration
- More detail
- Exception

Defining Layout Parameters for Exception Views

The grid layout is the only layout algorithm that can be used with exception views, and the only view parameter that can be defined for the grid layout algorithm is layout width. For information about the grid layout algorithm, see Appendix B, “View Layout Facility,” on page 657.

Defining Layout Parameters for Network, Configuration, and More Detail Views

When you define a network, configuration, or more detail view, you can specify the layout algorithm. You do this by specifying a value in the LayoutType field of the view object you define to represent the view. You can define view objects for the following classes:

- Network_View_Class
- Configuration_Peer_View_Class
- Configuration_Backbone_View_Class
- Configuration_Logical_Connectivity_View_Class

- Configuration_Physical_Connectivity_View_Class
- More_Detail_Logical_View_Class
- More_Detail_Physical_View_Class

If you do not specify a layout algorithm, the default radial by link type layout algorithm is used.

For information about choosing the kind of layout algorithm to use and the advantages and disadvantages of each layout algorithm, see Appendix B, "View Layout Facility," on page 657.

Certain layout algorithms require that you provide additional information to help it lay the view out correctly. Sometimes this information is specified in the fields of the view object itself; for example, the LinkCrossOptionValue field specifies the amount of effort the radial layout algorithm is to expend trying to untangle crossed links. As another example, the FirstNode and SecondNode fields specify which node is to be placed at the top of the ring, and which node is to be placed to the right of the top node, in the radial layout algorithm for token rings.

Additional information can also be specified in the fields of Layout_Parameters_For_Object_Class objects. These objects link a view and an object that is to be displayed in the view. They specify parameters that apply when that object is laid out in a particular view by a particular layout algorithm. One Layout_Parameters_For_Object_Class object can be linked to all objects that have the same layout parameters.

Examples are the RootNode field, which specifies that the resource linked to this Layout_Parameters_For_Object_Class object is to be the root node in a connectivity tree when the connectivity tree layout is used, and the LayoutSequence field, which specifies for certain layout algorithms where an object linked to this Layout_Parameters_For_Object_Class object appears in a sequence of objects.

Table 1 lists the fields that can be specified on objects of the following classes:

- Network_View_Class
- Configuration_Peer_View_Class
- Configuration_Backbone_View_Class
- Configuration_Logical_Connectivity_View_Class
- Configuration_Physical_Connectivity_View_Class
- More_Detail_Logical_View_Class
- More_Detail_Physical_View_Class

These fields can be optional, required, or not applicable, depending on the layout algorithm that is being used. Table 1 indicates the optional (**O**) and required (**R**) fields. **N/A** indicates that the parameter is not applicable for that type of layout algorithm.

Table 1. Layout Algorithms and View Parameters

Layout Algorithm	Link Cross Option Value	Bin Packing Flag	Bus Node	First Node	Second Node	Layout Orien- tation	Default Row Spacing	Ellipse Aspect Ratio Width/ Height	Layout Width
Radial by cluster ID	O	O	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Radial by link type	O	O	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Local area net	O	O	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Token-ring net	N/A	N/A	N/A	R	R	N/A	N/A	N/A	N/A

Table 1. Layout Algorithms and View Parameters (continued)

Layout Algorithm	Link Cross Option Value	Bin Packing Flag	Bus Node	First Node	Second Node	Layout Orien- tation	Default Row Spacing	Ellipse Aspect Ratio Width/ Height	Layout Width
LAN with central bus	N/A	N/A	R	N/A	N/A	N/A	N/A	N/A	N/A
Hierarchical with proximity	N/A	N/A	N/A	N/A	N/A	O	O	N/A	N/A
Single ellipse	N/A	N/A	N/A	N/A	N/A	N/A	N/A	O	N/A
Connectivity tree	N/A	N/A	N/A	N/A	N/A	O	O	N/A	N/A
Grid	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	O

For information about the layout parameters and about the layout algorithms, see Appendix B, “View Layout Facility,” on page 657.

Layout Parameters: Table 2 lists the layout parameters that can be specified on Layout_Parameters_For_Object_Class objects and indicates for which type of layout algorithms the layout parameters are optional (**O**) or required (**R**). **N/A** indicates that the parameter is not applicable for that type of layout algorithm. For more information about these layout parameters and the layout algorithms, see Appendix B, “View Layout Facility,” on page 657.

Table 2. Layout Algorithms and Layout Parameters

Layout Algorithm	Resource Layout Char.	Layout Sequence	Hierarch. Priority	Root Node	Cluster IDValue
Radial by cluster ID	N/A	N/A	N/A	N/A	R
Radial by link type	O	N/A	N/A	N/A	N/A
Local area net	N/A	O	N/A	N/A	N/A
Token-ring net	N/A	O	N/A	N/A	N/A
LAN with central bus	N/A	O	N/A	N/A	N/A
Hierarchical with proximity	N/A	N/A	R	N/A	N/A
Single ellipse	N/A	O	N/A	N/A	N/A
Connectivity tree	N/A	O	N/A	R	N/A
Grid	N/A	O	O	N/A	N/A

In the sample network, Layout_Parameters_For_Object_Class object LPTRLAN contains the parameters that specify how aggregate object TRLAN is to be displayed in network view SAMPNET, as illustrated in Figure 9 on page 31. The following is the RODM load function statement that defines the LPTRLAN object:

```
-- Create Layout Parameters for Object TRLAN --
CREATE INVOKER  ::= 0000004;
       OBJCLASS ::= Layout_Parameters_For_Object_Class;
       OBJINST  ::= MyName = (CHARVAR) 'LPTRLAN';
       ATTRLIST
       Object ::= (OBJECTLINK)
('GMFHS_Aggregate_Objects_Class'.'TRLAN'.'LayoutParmList'),
```

```
      View ::= (OBJECTLINKLIST)
('Network_View_Class'.'SAMPNET'.'LayoutParmList'),
      HierarchicalPriority ::= (INTEGER) 4;
END;
```

The Object field specifies the object to which the layout parameters apply; the View field specifies the view to which the layout parameters apply. The HierarchicalPriority field specifies that the TRLAN object is to appear in the fourth row of the hierarchical layout in the network view.

Layout_Parameters_For_Object_Class object LPB3088P2P contains the parameters that specify how shadow object NETB.B3088P2 is to be displayed in network view of the sysplex network, as illustrated in Figure 12 on page 43. The following RODM load function statement defines the LPB3088P2P layout parameters object:

```
-- Create Layout Parameters for Object B3088P2 --
CREATE INVOKER  ::= 0000004;
      OBJCLASS ::= Layout_Parameters_For_Object_Class;
      OBJINST  ::= MyName = (CHARVAR) 'LPB3088P2P';
      ATTRLIST
      Object ::= (OBJECTLINK)
('GMFHS_Shadow_Objects_Class'.'NETB.B3088P2'.'LayoutParmList'),
      View ::= (OBJECTLINKLIST)
('Network_View_Class'.'Sysplex'.'LayoutParmList'),
      LayoutSequence ::= (INTEGER) 0,
      RootNode ::= (ANONYMOUSVAR) X'80';
END;
```

As in the previous example, the Object and View fields specify the object and the view to which these parameters are associated. The LayoutSequence field is assigned the value 0, which specifies that the nodes are to be laid out in no particular order in the view. The RootNode field specifies that shadow object NET.B3088P2 is to be displayed as a root node in the connectivity tree.

Defining Layout Parameters for Dynamically Built More Detail Views

All types of more detail views can be dynamically built. You can specify the layout of more detail views even though you do not explicitly define the more detail views. More detail views are created when an NetView management console operator chooses **More Detail** from a context menu. GMFHS attempts to build the following more detail views for objects defined in RODM:

- The more detail logical view contains all of the objects specified by the ComposedOfLogical field of the selected object.
- The more detail physical view contains all of the objects specified by the ComposedOfPhysical field of the selected object.
- The configuration children II view contains all of the objects specified by the RelFieldNamesA field of the View_Information_Object_Class object for the configuration children II view.
- The configuration children III view contains all of the objects specified by the RelFieldNamesA field of the View_Information_Object_Class object for the configuration children III view.

If the value of the ComposedOfLogical field or the ComposedOfPhysical field is null, the corresponding view is not built. Refer to “Understanding Views” in the *IBM Tivoli NetView for z/OS User's Guide: NetView Management Console* for information about displaying more detail views.

You can specify layout parameters for each of the more detail views created from a selected object. Complete the following steps to specify layout parameters for more detail views. Figure 14 on page 51 shows the objects (A , B , and C) and links (1 and 2) you create.

1. Select the object for which you want to define more-detail-view layout parameters. You are defining layout parameters for the more detail views created when this object is selected in another view.

   For this example, select the aggregate object TRLAN (A) in the sample network.

2. Choose the more detail view for which you are defining layout parameters: more detail logical or more detail physical.

   The TRLAN object has valid values for both ComposedOfLogical and ComposedOfPhysical, so two more detail views are created. For this example, choose to define layout parameters for the more detail physical view.

3. Create an object of the Layout_Parameters_For_View_Class to represent the view.

   Hint: Layout_Parameters_For_View_Class objects are similar to Network_View_Class objects.

   The following is part of the RODM load function statement that creates the object (B) for this example. The sample member DUIFSNET contains the complete statements.

   ```
   CREATE INVOKER  ::= 0000004;
          OBJCLASS ::= Layout_Parameters_For_View_Class;
          OBJINST  ::= MyName = (CHARVAR)
                    'View_Layout_Parms_For_TRLAN_More_Detail_Physical';
   ```

4. Link the SelectedResource field of the object you created in Step 3 to the DetailViewLayoutForSelectedResource field of the object you selected in Step 1.

   The following is part of the RODM load function statement that creates this link, shown as 1 in Figure 14 on page 51:

   ```
   SelectedResource ::= (OBJECTLINKLIST) ('GMFHS_Aggregate_Objects_Class'.
   'TRLAN'.'DetailViewLayoutForSelectedResource'),
   ```

5. Specify which more detail view type this Layout_Parameters_For_View_Class object (B) represents. You specify the view type by linking the ViewClass field of this object to the DetailViewLayout field of an object (C) in the View_Information_Reference_Class that represents the view type:
   - More_Detail_Logical_View_Reference
   - More_Detail_Physical_View_Reference
   - Configuration_Children_II_View_Reference
   - Configuration_Children_III_View_Reference

   The following is part of the RODM load function statement that creates the link specifying a more detail physical view, shown as 2 in Figure 14 on page 51:

   ```
   ViewClass ::= (OBJECTLINKLIST) ('View_Information_Reference_Class'.
   'More_Detail_Physical_View_Reference'.'DetailViewLayout'),
   ```

6. Specify the layout parameters for the view you are defining. The remaining fields of the Layout_Parameters_For_View_Class object (B) specify the layout algorithm and other view parameters. Table 1 on page 47 lists the required parameters for each layout algorithm.

   For this example, choose radial layout for token ring networks as the layout algorithm. Table 1 on page 47 shows that the FirstNode field and SecondNode field are required for this layout. The following is part of the RODM load function statement that specifies the layout algorithm and the FirstNode and SecondNode fields:

```
LayoutType ::= (INTEGER) 4,
FirstNode  ::= (OBJECTLINK) ('GMFHS_Managed_Real_Objects_Class'.
     'LANMGR.10005AC35CA0'.'IsFirstNode'),
SecondNode ::= (OBJECTLINK) ('GMFHS_Managed_Real_Objects_Class'.
     'LANMGR.10005A95E7CC'.'IsSecondNode');
```

7. If you want to use this same Layout_Parameters_For_View_Class object for additional objects or views, create additional links. All of the link fields accept multiple values.
8. If you need to control the layout of individual objects in the more detail view, define layout parameters for the objects. Some layout algorithms require layout parameters for the objects: Table 2 on page 48 lists required parameters.

   See "Adding Layout Parameters for Objects in More Detail Views" for instructions on defining layout parameters.

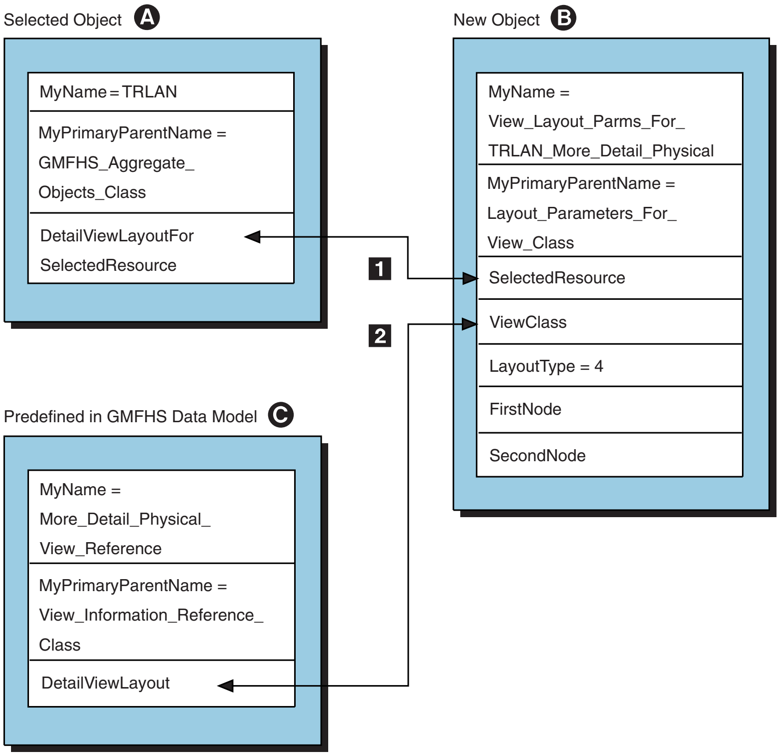


Figure 14. Defining Layout Parameters for More Detail Views

Adding Layout Parameters for Objects in More Detail Views:

Note: You can also define layout parameters for individual objects that appear in more detail views. You define these layout parameters with Layout_Parameters_For_Object_Class objects. Links specify which objects and views the layout parameters apply to. Complete the following steps to specify layout parameters for more detail views. Figure 15 on page 54 shows the objects and links you create.

1. Identify the objects in a more detail view that you want to define layout parameters for. The objects must be specified by the ComposedOfLogical, the ComposedOfPhysical, or the RelFieldNamesA field of the original object you specified in Step 1 on page 50 to appear in the more detail view.

For this example, define layout parameters for the object (**E**) LANMGR.10005A89A267 of the GMFHS_Managed_Real_Objects_Class.

2. Create an object of the Layout_Parameters_For_Object_Class to represent the layout parameters for the object when it is in a particular view.

   The following is part of the RODM load function statement (not in the DUIFSNET sample) that creates this object (**D**), shown in Figure 15 on page 54:

   ```
   CREATE INVOKER   ::= 0000004;
          OBJCLASS  ::= Layout_Parameters_For_Object_Class;
          OBJINST   ::= Detail_Layout_LANMGR.10005A89A267;
   ```

3. Link the Object field of the Layout_Parameters_For_Object_Class object you created in Step 2 to the DetailLayoutParmList field of the object represented.

   In this example, link the Object field of the Detail_Layout_LANMGR.10005A89A267 object (**D**) to the DetailLayoutParmList field of the object (**E**) LANMGR.10005A89A267. The following is part of the RODM load function statement that creates this link, shown as **3** in Figure 15 on page 54:

   ```
   Object ::= (OBJECTLINKLIST) ('GMFHS_Managed_Real_Objects_Class'.
   'Detail_Layout_LANMGR.10005A89A267'.'DetailLayoutParmList'),
   ```

4. Specify the view that these layout parameters apply to:

   a. Link the SelectedResource field of the Layout_Parameters_For_Object_Class object to the DetailLayoutParmListForSelectedResource field on the object which is selected to generate this more detail view (the object selected in 1 on page 50).

      In this example, link the SelectedResource field of object (**D**) Detail_Layout_LANMGR.10005A89A267 to the DetailLayoutParmListForSelectedResource field of object (**A**) TRLAN. The following is part of the RODM load function statement that creates this link, shown as **4** in Figure 15 on page 54:

      ```
      SelectedResource ::= (OBJECTLINKLIST)
      ('GMFHS_Aggregate_Objects_Class'.
      'TRLAN'.'DetailLayoutParmListForSelectedResource'),
      ```

   b. Specify which more detail view type these layout parameters apply to. You specify the view type by linking the ViewClass field of this object (**D**) to the DetailLayoutParmList field of an object (**C**) in the View_Information_Reference_Class that represents the view type:

      - More_Detail_Logical_View_Reference
      - More_Detail_Physical_View_Reference
      - Configuration_Children_II_View_Reference
      - Configuration_Children_III_View_Reference

      The following is part of the RODM load function statement that creates the link specifying the more detail physical view, shown as **5** in Figure 15 on page 54:

      ```
      ViewClass ::= (OBJECTLINKLIST)
      ('View_Information_Reference_Class'.
      'More_Detail_Physical_View_Reference'.
      'DetailLayoutParmList'),
      ```

5. Specify the layout parameters for the object. Table 2 on page 48 lists the optional and required layout parameters for each layout algorithm.

   For this example, the radial layout for token ring algorithm is used. Table 2 on page 48 shows that the LayoutSequence field is the only optional parameter you can specify. Specify a value of 3 for the LayoutSequence field of this object (**D**). The following is part of the RODM load function statement that sets the value of the LayoutSequence field:

```
LayoutSequence ::= (INTEGER) 3;
```

6. If you want to use this same Layout_Parameters_For_Object_Class object for additional objects or views, create additional links. All of the link fields accept multiple values.

   For example, use this same object to define the layout parameters for object LANMGR.10005A89A267 when it is in the more detail physical view generated when an object of the GMFHS_Aggregate_Objects_Class named OTHER_AGG is selected (OTHER_AGG is not part of the sample network). Create a link from the SelectedResource field of object Detail_Layout_LANMGR.10005A89A267 to the DetailLayoutParmListForSelectedResource field of object OTHER_AGG. The following is a RODM load function primitive statement that creates this link:

   ```
   OP 'Layout_Parameters_For_Object_Class'.
      'Detail_Layout_LANMGR.10005A89A267'.'SelectedResource'
   IS_LINKED_TO 'GMFHS_Aggregate_Objects_Class'.'TRLAN'.
      'DetailLayoutParmListForSelectedResource';
   ```

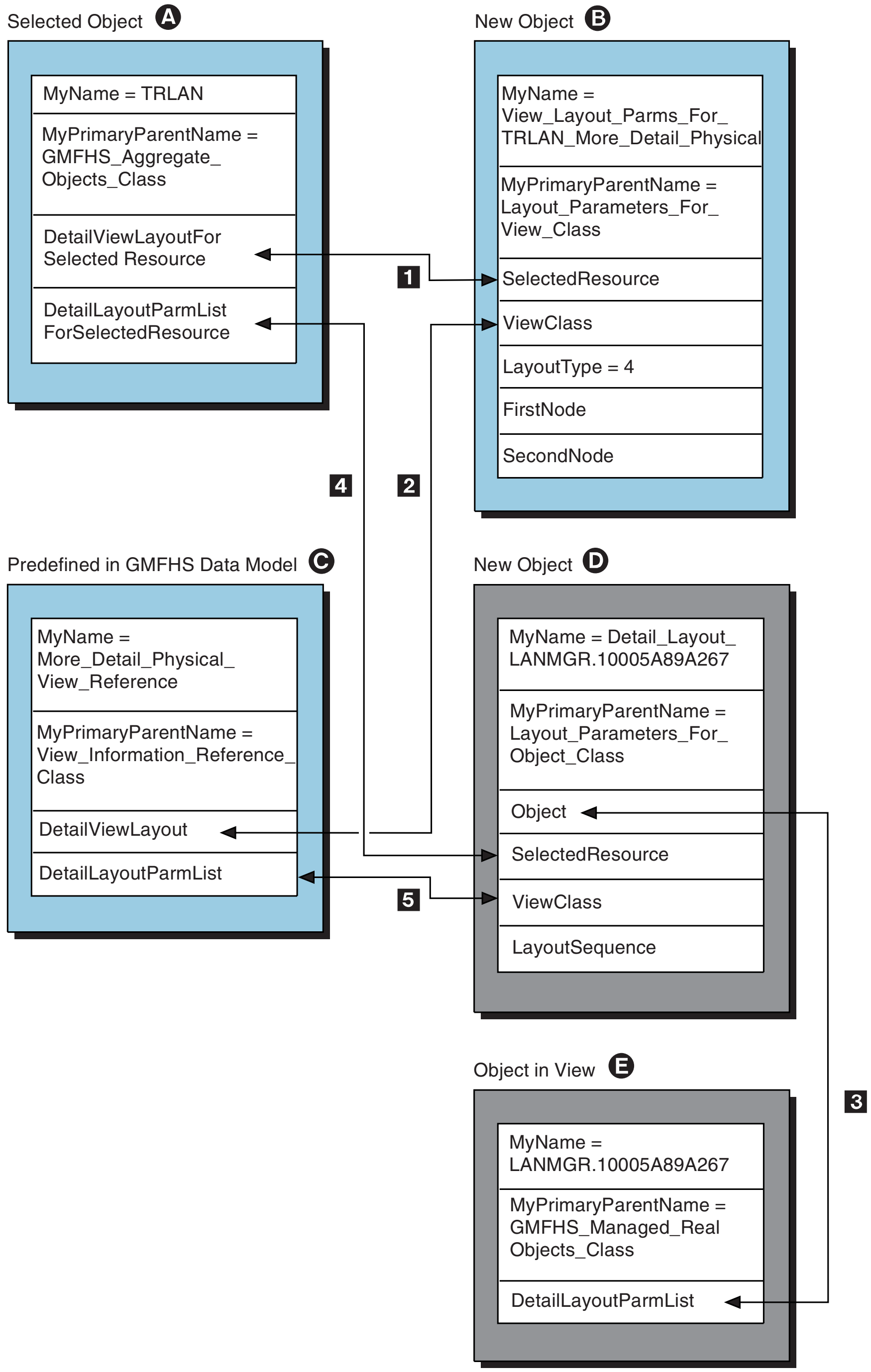


Figure 15. Defining Layout Parameters for Objects in More Detail Views

Putting It All Together

After you have defined the objects that represent your configurations and networks, load them into RODM using the RODM load function. Chapter 3, "Loading the GMFHS Data Model," on page 57 contains directions for doing this.

You need to load the class definition before you load the definitions of the objects of that class. By the same token, you need to define objects that are to be linked before you can actually link them. Use the load function statements in sample member DUIFSNET as an example of the order to follow. The objects and links in the sample network are arranged for loading in the following order:

1. SNA_Domain_Class objects
2. GMFHS_Shadow_Objects_Class objects
3. NMG_Class objects
4. Non_SNA_Domain_Class objects
5. GMFHS_Managed_Real_Objects_Class objects
6. GMFHS_Aggregate_Objects_Class objects
7. Linkages among objects
   - Logical links
   - Physical links
   - Parent/Child links
8. Exception_View_Class objects
9. Network_View_Class objects
10. Configuration_Peer_View_Class objects
11. Layout_Parameters_For_View_Class objects
12. Layout_Parameters_For_Object_Class objects

Note: Although the sample network defined in sample load file DUIFSNET does not include an exception view, it is included in the preceding list in the position that it must be loaded.

Study the network in the sample load file DUIFSNET carefully before defining your own network. For information about RODM load function syntax, see Chapter 10, "Using the RODM Load Function," on page 239.

Chapter 3. Loading the GMFHS Data Model

This chapter describes how to load the GMFHS and SNA topology manager data models, your network definition, and methods into RODM. This chapter also describes how to make additions, changes, or deletions to objects when GMFHS is active.

The GMFHS class structure is provided in RODM load function input file, DUIFSTRC, which is shipped with the NetView program.

The class structure for the SNA topology manager is provided in RODM load function input files, FLBTRDM*x*, which is also shipped with the NetView program. For more information about the FLBTRDM*x* load function input files, refer to *IBM Tivoli NetView for z/OS Installation: Configuring Graphical Components*.

DUIFSTRC and all the FLBTRDM*x* input files are loaded using sample CNMSJH12. Both the DUIFSTRC and all of the FLBTRDM*x* input files must be loaded for GMFHS operation. Note that input file DUIFSTRC must be loaded before any FLBTRDM*x* input files are loaded. This is the order specified in sample CNMSJH12 and it must not be changed.

Loading the Data Models and Network Definitions

With RODM running, use sample CNMSJH12 to load the GMFHS data model and your network definition.

1. Create RODM statements to define your non-SNA network. See Chapter 2, “Defining Your Network to GMFHS,” on page 17 for information about how to define your network to RODM.
2. Update the sample job CNMSJH12 as follows:
   a. Change the JOB statement to specify your installation's accounting information.
   b. Enter the names of the RODM load files that were created in Step 1 into the EKGIN1 DD statement on the last line of the sample. For example, if your object definitions are in the data set NETVIEW.V6R1M0.MYDEFS(OBJECTS), the last line of CNMSJH12 is:
   ```
   //         DD DSN=NETVIEW.V6R1M0.MYDEFS(OBJECTS),DISP=SHR
   ```
   c. Replace `RODMNAME` with the name of your RODM in the EXEC statement.
3. Ensure that RODM is running.
4. Start CNMSJH12.
5. Start GMFHS.

Changing Network Definitions When GMFHS Is Running

If GMFHS is running when non-SNA objects are to be added, changed, or deleted in the RODM data cache, the GMFHS CONFIG command might be required. The GMFHS CONFIG command identifies, to GMFHS, the scope of the changes and the type of processing needed to respond to them.

Subarea resources that are managed by SNA topology manager can be changed anytime without using the GMFHS CONFIG commands.

Notes:

1. NMGs and domains can be added dynamically without using the GMFHS CONFIG command. See “Adding NMGs and Domains When GMFHS Is Active” on page 60 for more information.
2. When you change the GMFHS data stored in RODM while GMFHS is active, you might get unpredictable results until the appropriate GMFHS CONFIG command is issued and completes.

The three GMFHS CONFIG command types are: DOMAIN, NETWORK and VIEW. The following sections list which GMFHS CONFIG command to issue based on the field and class you are changing:

DOMAIN
Used when the changes include changing the association of GMFHS_Managed_Real_Objects_Class objects with Non_SNA_Domain_Class objects, but do not include changes that require that the GMFHS CONFIG NETWORK command be used. See the NetView online help for details on the behavior of the CONFIG DOMAIN command.

NETWORK
Used only when the changes being made include changes to information that describes the characteristics and structure of the NMGs and domains.

VIEW Not needed, has been left in only for migration purposes.

The GMFHS CONFIG command also has a LOAD parameter. If the default LOAD=NO is specified with CONFIG VIEW, no operation is performed. For CONFIG DOMAIN and NETWORK, if the default LOAD=NO is specified, all command processing is completed except for the invocation of the RODM load function. For example, if the contents of the cache are changed by running the RODM load function by job posting or by some RODM application other than GMFHS, use the GMFHS CONFIG command with LOAD=NO specified. This causes the processing within GMFHS, required for the changes, to be completed.

If LOAD=YES is specified, the RODM load function is run as part of the command processing. If the INDD=*ddname* the data set or sets identified by *ddname* will be passed to the RODM load function as the input. If the INDD parameter is not specified the default is EKGIN3.

Note: Use the GMFHS CONFIG command with caution. This command can reinitialize some RODM objects that are under one or more non-SNA domains. This can result in significant CPU utilization depending on the number of real objects that are defined. The amount of CPU utilization can be similar to the amount used when GMFHS was initially started.

See the NetView online help for more information about the GMFHS CONFIG command.

Selecting the Required GMFHS CONFIG Command

The following tables show which GMFHS CONFIG command is required when objects in the RODM cache have their field values changed. To determine what CONFIG command must be used, use the first of the following rules that applies:

- If any object field change being made requires a CONFIG NETWORK command, use that command.

- If any object field change requires a CONFIG DOMAIN command, use that command.
- Finally, if the field is not listed, no CONFIG command is required for any of the object additions or deletions or object field value changes being made. However, issue the RODM CHKPT command after the completion of the RODM load function job. This causes a new checkpoint image of the RODM cache to be written so that it is available for cache recovery if needed.

There is no separate table provided for the addition or deletion of the objects themselves. This is because, with the exception of SNA Domain objects, a new object has no effect until it is linked to another object, and an object cannot be deleted until all of its links to other objects have been deleted. The establishment and deletion of object links is done by changing field values for fields with data type OBJECTLINK or OBJECTLINKLIST. Changes to fields of these types are covered by the tables.

Non_SNA_Domain_Class Changes

Table 3 shows which GMFHS CONFIG command to use when changing a field of an object in the Non_SNA_Domain_Class.

Table 3. GMFHS CONFIG Command for Non_SNA_Domain_Class Objects

Field	GMFHS CONFIG Command
AlertProc	NETWORK
CommandTimeoutInterval	NETWORK
ContainsResource	NETWORK, DOMAIN (see note)
DomainCharacteristics	NETWORK
DomainCharacteristics2	NETWORK
EMDomain	NETWORK
InitialResourceStatus	NETWORK
PresentationProtocolName	NETWORK
ReportsToAgent	NETWORK
SessionProtocolName	NETWORK
TransactionProgram	NETWORK
WindowSize	NETWORK

Note: The ContainsResource field of Non_SNA_Domain_Class objects can specify either GMFHS-managed real resources or GMFHS-NMG objects that belong to the domain. If the Resources field of the non-SNA Domain object is linked to the Domain field of a GMFHS-NMG object, use the CONFIG NETWORK command. If only GMFHS-managed real resources are being linked to or unlinked from non-SNA domain objects, the CONFIG DOMAIN command can be used. See the NetView online help for a complete description of the CONFIG DOMAIN command before you use it.

SNA_Domain_Class Changes

Table 4 shows which GMFHS CONFIG command to use when changing a field of an object in the SNA_Domain_Class. Issue the GMFHS CONFIG NETWORK command when you create or delete an object of the SNA_Domain_Class.

Table 4. GMFHS CONFIG Command for SNA_Domain_Class Objects

Field	GMFHS CONFIG Command
ContainsResource	NETWORK

Table 4. GMFHS CONFIG Command for SNA_Domain_Class Objects (continued)

Field	**GMFHS CONFIG Command**
SNANet	NETWORK

NMG_Class Changes

Table 5 shows which GMFHS CONFIG command to use when changing a field of an object in the NMG_Class.

Table 5. GMFHS CONFIG Command for NMG_Class Objects

Field	**GMFHS CONFIG Command**
AgentStatusEffect	NETWORK
CommandRouteLUName	NETWORK
Domain	NETWORK
NMGCharacteristics	NETWORK
ReportsOnDomain	NETWORK
TransportProtocolName	NETWORK
WindowSize	NETWORK

GMFHS_Managed_Real_Objects_Class Changes

Table 6 shows which GMFHS CONFIG command to use when changing a field of an object in the GMFHS_Managed_Real_Objects_Class.

Table 6. GMFHS CONFIG Command for GMFHS_Managed_Real_Objects_Class Objects

Field	**GMFHS CONFIG Command**
Domain	DOMAIN (see note)

Note: If only GMFHS-managed real resources are being linked to or unlinked from non-SNA domain objects, the CONFIG DOMAIN command can be used. See the NetView online help for a complete description of the CONFIG DOMAIN command before you use it.

Adding NMGs and Domains When GMFHS Is Active

NMGs and non-SNA domains can be added to RODM while GMFHS is running without using the GMFHS CONFIG command. Use the following guidelines when defining the objects in RODM.

- Set the appropriate bit to indicate that you want to dynamically add an NMG or non-SNA domain.
- Set the appropriate bit in the DomainCharacteristics field to indicate that you do not want GMFHS to apply initial or unknown status to resources under a non-SNA domain.

  Note: This only applies when GMFHS initially processes the NMG or non-SNA domain. GMFHS applies initial and unknown status normally for all subsequent processing.
- If you do not want GMFHS to solicit resource status for a non-SNA domain, set the appropriate bit in the DomainCharacteristics field.
- Link an NMG to a non-SNA domain after the NMG and domain have been defined in RODM. GMFHS uses this link as a signal to start processing a new NMG or domain.

Chapter 4. Communicating with Network Management Gateways

This chapter describes how GMFHS communicates with network management gateways (NMGs). The NMGs send status information about non-SNA networks to GMFHS. GMFHS sends commands for the non-SNA networks to the NMGs.

Non-SNA resources are associated with a non-SNA domain in GMFHS. When you define non-SNA domains to GMFHS, you specify the NMG that owns each non-SNA domain and its associated resources. You also specify how GMFHS communicates with the NMG.

The clock on the workstation on which the NMG is running needs to be synchronized with the clock on which the host GMFHS is running. The DOMP010 presentation protocol synchronizes these clocks. For other presentation protocols, create your own routine to synchronize the clocks.

Refer to the *Service Point Application Router™ and Remote Operations Service Guide* for information about using the ROP services. If the clocks are not synchronized, GMFHS might not process alerts correctly.

Use this chapter to help you select the correct values for the following GMFHS fields:
- PresentationProtocolName
- SessionProtocolName
- TransportProtocolName

This chapter also helps you select the correct values for some of the bits of the DomainCharacteristics field.

You can also use this chapter to understand what GMFHS expects from an NMG. You need this information to create your own service points or NMGs.

Table 7 shows the values for the three GMFHS protocol fields for typical NMGs.

Table 7. GMFHS Protocol Values for Typical NMGs

NMG name	Presentation ProtocolName	Session ProtocolName	Transport ProtocolName
NAP	DOMP010	DOMS010	COS
NetView OST[1]	DOMP020	PASSTHRU	OST
NetView OST	PASSTHRU	PASSTHRU	OST
NetView PPI	NONE	NONE	PPI
Open Topology Interface Agent[2]	DOMP010	NONE	COS
PPI	DOMP020	PASSTHRU	PPI

[1] Use the DOMP020 presentation protocol if you want to use parameter substitution.

[2] IBM Tivoli NetView for z/OS Open Topology Interface Agent.

Remember that this table lists typical values for the protocol parameters. Other combinations of parameter values are possible and the values you use depend on what your NMGs support.

Defining Non-SNA Presentation Protocol

The presentation protocol translates commands to and from the syntax used by the element management system. The translation is done according to the rules for the domain associated with the resource that is the target of the command.

The PresentationProtocolName field of the Non_SNA_Domain_Class object specifies which protocol is used for the non-SNA domain. The valid protocol names are:
- DOMP010
- DOMP020
- PASSTHRU
- NONE

DOMP010 Presentation Protocol

The DOMP010 protocol enables generic commands to be translated for delivery to the gateway associated with the domain and also enables the responses to commands formatted using the DOMP010 protocol to be translated to DisplayStatus. The DisplayStatus is reflected in the appearance of objects in the views. Native and resource-specific commands can also be delivered using the DOMP010 protocol supported by the native-element manager or transaction program associated with the domain.

The DOMP010 presentation protocol specifies that the command messages and command response messages from the NMG are formatted according to the rules described in “DOMP010 Formatting Rules” on page 65.

The DOMP010 protocol provides translation of the following types of commands:
- Generic commands:
  - Activate
  - Display Abnormal Status
  - Display Status
  - Inactivate
  - Reconfigure
  - Recycle
- Session protocol commands
- Native and resource-specific command text

The DOMP010 protocol also provides for the translation of command responses from native element managers for any command.

For native commands, DOMP010 performs parameter substitution on the command entered by the operator. GMHFS replaces the tokens in the command as follows:

Token **Action taken by GMFHS**

%APPL%
Replace with the value of the TransactionProgram field of the Non_SNA_Domain_Class object.

%DOMAIN%
Replace with the value of the EMDomain field of the Non_SNA_Domain_Class object.

%RESOURCE%
Replace with the value of the MyName field of the resource.

%SPNAME%
Replace with the value of the MyName field of the NMG_Class object.

%TYPE%
Replace with the value of the TypeName field of the Display_Resource_Type_Class object associated with the resource.

GMFHS accepts the following parameters in native OST text:
- %RESPONSE%
- %NORESPONSE%

The %RESPONSE% parameter forces all valid command responses to be returned to the workstation. The %RESPONSE% parameter overrides the Response Expected bit of the Non_SNA_Domain_Class DomainCharacteristics field. The %NORESPONSE% parameter forces the native command to be issued at the OST console, and no response is returned to the workstation.

The DOMP010 presentation protocol is applicable only on COS and program-to-program interface NMGs.

DOMP020 Presentation Protocol

The DOMP020 protocol enables generic commands to be translated for delivery to the NMG associated with the domain. The DOMP020 protocol supports native and resource-specific command text. Responses to these commands are returned unchanged to the command response window of the originating workstation. GMFHS does not extract status information from these responses.

The text of generic commands is retrieved from RODM. GMFHS requests the command text from the GMFHS_Managed_Real_Objects_Class object that represents the target of the command. If this object does not define the command text, GMFHS then requests the command text from the Non_SNA_Domain_Class object that represents the domain of the command's target. The Display Abnormal Status and Reconfigure generic commands are valid only if the target of the command is an object of the Non_SNA_Domain_Class. The fields used for generic commands follow:

Generic Command	GMFHS Field
Activate	ActivateCommandText
Deactivate	DeactivateCommandText
Display Abnormal Status	DisplayAbnormalStatusCommandText
Display Status	DisplayStatusCommandText
Reconfigure	ReconfigureCommandText
Recycle	RecycleCommandText

When GMFHS locates the command, it performs parameter substitution. GMFHS looks for any of the following tokens in the command, and replaces them as follows:

Token **Action taken by GMFHS**

%APPL% Replace with the value of the TransactionProgram field of the Non_SNA_Domain_Class object.

%DOMAIN%
Replace with the value of the EMDomain field of the Non_SNA_Domain_Class object.

%RESOURCE%
Replace with the value of the MyName field of the resource.

%SPNAME%
Replace with the value of the MyName field of the NMG_Class object.

%TYPE%
Replace with the value of the TypeName field of the Display_Resource_Type_Class object associated with the resource.

Note: Display Abnormal Status and Reconfigure commands pertain only to domains; therefore only the domain object is searched for the command text.

The DOMP020 protocol is used with all NMG types. The gateways allow commands to be delivered to the OST associated with a workstation operator or to the central site NetView primary program operator interface task (PPT) if the command is from GMFHS. The command procedure or processor that is run for the command might directly or indirectly generate an alert. The alert reports the resulting resource status.

PASSTHRU Presentation Protocol

The PASSTHRU protocol specifies that native network command text entered by a workstation operator passes directly to the native element management system unchanged, and that native network command response text returns to the workstation operator without interpretation by GMFHS.

The PASSTHRU presentation protocol specifies that the actual text of the commands is retrieved from RODM. The differences between PASSTHRU and DOMP020 are that PASSTHRU does not support generic commands and does not perform parameter substitution.

NONE Presentation Protocol

Specify NONE for the PresentationProtocolName value for a domain if commands are not sent to the NMG associated with the domain. For example, specify NONE when domains are defined to only receive alerts for the resources they contain.

Output Formatting For All Presentation Protocols

This section describes output formatting for the DOMP020 and PASSTHRU protocols and for the DOMP010 protocol.

DOMP020 and PASSTHRU Output Formatting

If the NMG is using the COS transport protocol, the subvector 31 contains the response to a RUNCMD. The response in subvector 31 is formatted as follows: when the native element manager sends multiple lines of response text to GMFHS, each line of response text must be put in a separate subvector 31. This ensures that each separate line of response text is displayed in the workstation Command Responses window as a separate line of text.

DOMP010 Output Formatting

Each separate line of text in a multiple line response is preceded by a separate text keyword (TX). See "Text—TX" on page 71 for more information about the use of the TX keyword for the DOMP010 protocol.

DOMP010 Formatting Rules

This section describes the format of the textual data contained in either the commands for COS NMGs or the data delivered to program-to-program interface NMGs. In this section, the term *packet* refers to the information in these subvectors.

General Packet Format

A packet is made up of one or more comma-delimited keyword parameters. These parameters perform such functions as identifying the command or response. All values in the text packet are displayable characters.

- In the NetView/PC API/CS environment, the displayable characters are coded in ASCII.
- In the SNA network, the characters are coded in displayable EBCDIC. NetView/PC API/CS performs the necessary code set translations.

Each parameter has the following general format:

```
keyword=value
```

Each keyword is 2 - characters long, and the equal sign is always present. The value is of variable length. For example, if CP is a keyword that has the value MINIA, the keyword parameter is:

```
CP=MINIA
```

Keyword values can be made up of more than one data item, delimited with commas and surrounded by one set of parentheses, for example:

```
CP=(MINIA,MINIB)
```

In a typical packet, several keyword parameters are specified. The keyword parameters are also delimited by commas, for example:

```
CM=AE,SQ=10,DM=DOMAIN,CP=(MINIA,MINIB)
RP=AE,SQ=10,DM=DOMAIN,CP=MINIA,ST=U,TM=930601120000,CP=MINIB,ST=U,TM=930601120000,
```

In most cases, the order of the individual parameters is unimportant. Exceptions to this rule are noted in the descriptions of the keywords.

Keyword and Value Definitions

The packet keywords and their descriptions follow:

Keyword	**Description**
CE	Command execution
CM	Command identifier, required for commands
CP	Component identifier
DM	Domain identifier
PT	Protocol text
RN	Reason
RP	Response identifier, required for responses
SN	Command sender identifier
SQ	Message sequence number, required for commands and responses
ST	Status identifier
TM	Time stamp
TX	Native command or response text

The following sections describe each keyword and its values.

Command Execution—CE

The command execution status keyword (CE) indicates a failure to successfully run a command. It differs from a negative response (RP=X) in that the negative response applies to the entire command. A command execution failure applies to a subset of the command.

The keyword values for CE are value lists contained in a text string. The values are the same as those for the reason (RN) keyword. See “Reason—RN” on page 69 for these values.

When the command is Display Status (CM=D) or Display Abnormal Status (CM=A), and the statuses of more than one component are carried in the response, a command execution failed for any one of the components. This is indicated by the following:

```
CP=component_name,ST=X,CE=(reason text)
```

The same command response carries the status of those components for which the command was successful. If command execution fails for each component individually, the CE keyword and ST=X are returned for each component.

Note: The use of ST=X, is required, and indicates that any status already reported for this component is still in effect.

The CE keyword is position dependent. CE must follow the CP keyword for its subject component, and precede any other components. That is, the CP and CE pair for a given component must not be split by another CP keyword.

The CE keyword is supported for Display Status and Display Abnormal Status commands (CP=A and CP=D).

Command—CM

The command keyword, CM, is the command issued to the element manager. This keyword is required on any packet sent from the host to an element manager.

CM values have a two-part definition:

- The first byte of the value is the command type. The command type classifies the type of command you issue to the non-SNA device. The following list describes the command types.

Value	Description
A	Display abnormal status
C	Reconfigure domain
D	Display status for a named resource or resources
I	Inactivate resource
N	Native command
P	Protocol message
R	Recycle resource
V	Activate resource
X	Negative response

- The second byte is the continuation.

  The continuation byte is used in conjunction with command types that can require multiple responses.

  Value **Description**

E This is either an initial request or the last response to an initial request.

M This is either a continuation request or not the last response when multiple responses are required to service an initial request.

For more information about the importance of the continuation byte, see "Multiple-Response Protocol" on page 73.

Component ID—CP

The component ID provided by the CP keyword must match the resource portion of the MyName value of a GMFHS_Managed_Real_Resource object in the RODM data cache. For example, if the MyName of the resource is OTTAWA.MINIA, specify `CP=MINIA`.

You can specify multiple resources with one CP keyword by using a value list. For example, if three resources are included in one command, the CP keyword is:

```
CP=(MINIA,MINIB,MINIC)
```

Note: Command responses use multiple CP values, rather than a component ID list, if the response is for multiple resources.

The size of the CP keyword value depends on the following:
- The type of NMG containing the element manager
- The size of required keywords in the command
- The size of optional keywords in the command

The maximum command size depends on the NMG type. The maximum size can be one of the following:
- 240 characters for the COS gateway
- 256 characters for OST gateways
- 253 characters for program-to-program interface gateways

To determine the valid maximum size of the resource names in the CP keyword, do the following:

1. Add the number of characters in the base command and the number of characters in the CP keyword syntax.
2. Subtract that total from the maximum length that the NMG supports.

For example, the following command contains 24 characters:

```
CM=DE,SQ=5,DM=DOMAIN,CP=aaa
```

Therefore, the maximum size of the resource name *aaa* is 216 characters for the COS gateway, 232 characters for OST gateways, and 229 characters for program-to-program interface gateways.

The following command contains 28 characters:

```
CM=DE,SQ=5,DM=DOMAIN,CP=(aaa,bbb,ccc)
```

Therefore, the maximum size of the resource names *aaa*, *bbb*, and *ccc* is 212 characters for the COS gateway, 228 characters for OST gateways, and 225 characters for program-to-program interface gateways.

If you specify multiple components in the command and the size of the command exceeds the maximum, GMFHS automatically reduces the number of resources in the command to reduce the command size.

Domain—DM

The domain keyword, DM, specifies the non-SNA domain of a resource when multiple non-SNA domains are supported. The domain keyword is optional.

DM signifies the domain in which the GMFHS associates a resource specified with the CP keyword. DM needs to match the EMDomain field of the Non_SNA_Domain_Class object. For example, if the MyName of the resource is OTTAWA.MINIA, the keyword parameter format is:

```
DM=OTTAWA
```

The DM value can be up to 8 characters in length.

Protocol—PT

The protocol keyword, PT, is used when a command identifier (CM) or response identifier (RP) command type equals protocol command (P); for example, CM=PE (E is the continuation byte).

The PT values are protocol commands that control the communication session between two cooperating processes: on the host, and on the target of the command (the native element manager). Because all commands require responses, any protocol command request must have a protocol-type response.

Table 8 lists the defined PT values and displays the session protocol commands used for the DOMS010 protocol.

Table 8. Protocol Command Values

Protocol Command	Meaning
SESSION_REQUEST	Sent by GMFHS to the element manager to request that a session be established.
SESSION_REQUEST_ACCEPT	A response acknowledging a SESSION_REQUEST protocol command. This command does not indicate that a session is established.
INIT_ACCEPT	Returned by GMFHS to acknowledge receipt of the INIT alert.
INIT_ACCEPT_ACCEPT	A response acknowledging the INIT_ACCEPT protocol command.
SET_CLOCK	Sent by GMFHS after it receives the INIT_ACCEPT_ACCEPT protocol command and if the SET_CLOCK protocol command is supported by the domain's native element manager. This message is sent only if the support set clock bit is set to "on" in the DomainCharacteristics field. SET_CLOCK provides the current local time in its TM parameter value. This message is issued every 24 hours for as long as the session remains active.
SET_CLOCK_ACCEPT	Returned by the native element manager to acknowledge the SET_CLOCK protocol command.

Note: The values for the PT keyword in commands coming from GMFHS are lowercase. GMFHS is not case-sensitive on the response values.

For example, if the GMFHS is responding to an INIT alert from the NMG, the format of the packet is:

```
CM=PE,DM=DURHAM,SQ=7,PT=(INIT_ACCEPT)
```

The response to the INIT_ACCEPT is:

```
RP=PE,DM=DURHAM,SQ=7,PT=(INIT_ACCEPT_ACCEPT)
```

If the SET_CLOCK protocol command is supported, GMFHS sends it to the NMG every 24 hours, allowing the NMG to set its clock to the correct time. The current time is carried by the TM keyword and accounts for the NMG's offset specified in the INIT alert. For example:

```
CM=PE,SQ=8,DM=DURHAM,PT=(SET_CLOCK),TM=930101120000
RP=PE,SQ=8,DM=DURHAM,PT=(SET_CLOCK_ACCEPT)
```

See “Session Establishment for DOMS010” on page 76 for more information about these protocols.

Reason—RN

The reason keyword (RN) indicates why a request was not honored. RP=XE is always used with the RN keyword.

The reason value is a text string in value list format. For example:

```
RN=(execution node inaccessible)
```

Table 9 lists the supported text values.

Table 9. Reason Values

Value	Description
Aborted	An error occurred prohibiting the completion of a request (failure in memory, CPU, disk, and so on).
Canceled	The request was canceled before it can be completed.
Component unknown	The target component is unknown.
Currently not allowed	The command type is supported but cannot be run by the target component at this time.
Execution node inaccessible	The target node that runs the requested command is not accessible.
Failed	The command processing completed, but failed to achieve the expected results (ACTIVATE did not result in the component becoming active).
Invalid command ID	The command type is not valid.
Invalid parameter	A keyword parameter was incorrect and prohibited the execution of the command.
No resources	There were insufficient resources available to run the request (memory, CPU, disk, and so on).
Not allowed	The command type is supported but is not allowed for the target component.
Not supported	The command type is not supported by the entity processing the command.
Preempted	The request was preempted by another process before it can be completed.
Timed out	The request timed out before a valid response can be processed.

Note: GMFHS is not case-sensitive on the response values.

Response—RP

The response keyword, RP, identifies a command response packet. The response keyword values are the same as described for the command keyword, CM, under "Command—CM" on page 66. RP values also use the continuation byte as described in the CM values.

For example, if you issue a Display Status command for a single component, the response is positive and no continuation message is required. The format of the keyword parameter is:

```
RP=DE,SQ=5,DM=DOMAIN,CP=MINIA
```

If the response to a request is negative (request cannot be successfully completed), an X is placed in the first byte for the command type. For example:

```
RP=XE,SQ=5,DM=DOMAIN,RN=(no resources)
```

Command Sender ID—SN

The command sender ID keyword, SN, identifies the sender of the command. The SN keyword is included in all commands. The keyword value is always GMFHS:

```
SN=GMFHS
```

Message Sequence Number—SQ

The message sequence number keyword, SQ, contains a unique message sequence number that identifies either the request or response. The message sequence number of a response is identical to the sequence number used in the original request. For example, if you issue a Display Status command for one component with a sequence number of 6, the response to that request also has a sequence number of 6.

SQ provides a correlation for the continuation responses. If a single request requires multiple responses, the message sequence number is used to correlate all of the responses to the original request. For example, if you issue a Display Abnormal Status COMPONENTS command with a message sequence number of 35, the first response in a series of responses has a message sequence number of 35 and the continuation byte set to more (M). For example:

```
CM=AM,SQ=35
```

The originator can send another request with the continuation byte set to M and a message sequence number of 35. When the responder receives this request, it knows to continue sending the data that does not fit in the previous response packet. This multiple exchange continues until the original request is satisfied with the continuation byte in the response being set to end (E).

Message sequence numbers roll over after reaching 999.

Status—ST

The status keyword, ST, can be used to describe either of the following:

- The status of a component in response to a display status (CM=A or CM=D) command
- The resulting component status in response to an activate (CM=V), deactivate (CM=I), or recycle (CM=R) command

The value for a status keyword can be the GMFHS external status of the resource, which is a 1-byte value.

Only one status value type is enabled for any given resource in a response message.

When status is reported on multiple resources, the ST keyword parameter and value must immediately follow each associated component ID keyword (CP). If the ST and TM keywords are sent together, their specific order does not matter, as long as they both follow the associated CP keyword.

If the GMFHS external status of a resource is unsatisfactory, the format of the ST keyword parameter is:

```
ST=U
```

If the GMFHS external statuses of components NODE1 and NODE2 are being reported, and their respective statuses are satisfactory and unsatisfactory, the format of the ST keyword parameter is as follows:

```
CP=NODE1,ST=S,TM=890315120801,CP=NODE2,ST=U,TM=890315120814
```

Time Stamp—TM

The time-stamp keyword, TM, describes the local date and time. The TM value and keyword are required whenever a command response provides a component, and for each component status provided in the response. This includes D, A, I, V, and R commands. The time-stamp keyword can be in other responses but is ignored. The TM keyword is also included on a SET_CLOCK session protocol command to specify the element manager's clock setting.

When time is reported on multiple resources, the TM keyword parameter and value must immediately follow each associated component ID keyword (CP). If the TM and ST keywords are sent together, their specific order does not matter, as long as they both follow the associated CP keyword. The format of the time stamp is:

```
TM=yymmddhhmmss
```

The time stamp variables are defined as:

yy	year
mm	month (01 - 12)
dd	day (01 - 31)
hh	hour (00 - 23)
mm	minute (00 - 59)
ss	second (00 - 59)

For example, if a status is being reported as of 3:58:21 p.m. local time on 28 May, 1993, the TM keyword parameter is:

```
TM=930528155821
```

Text—TX

The text keyword, TX, provides support for native commands and their responses. The value for TX is a string of text.

For commands, the TX value is the text of a native network command, such as a command entered at the native element manager's console. The following is the data item format for the SHOW CIRCUIT A native command:

```
TX=(SHOW CIRCUIT A)
```

For responses, TX is the response text received at the native element manager's console. Command responses are shown in the Command Response window, if the command was issued by the operator. Each occurrence of the TX keyword results in one line of text displayed at the NetView workstation. The following is the format of the response keyword parameter, if the response to the command is CIRCUIT A CONFIGURED AND OPERATIONAL:

```
TX=(CIRCUIT A CONFIGURED AND OPERATIONAL)
```

If the response to the command is a multiple line response, the format of the response keyword parameters is:

```
TX=(    COMMAND FAILURE STATISTICS),
TX=(ROUTES    ERRORS    HITS    MISSES),
TX=(  40       250      2000       4)
```

Commas separate the individual parameter lines. In the case of text responses, the order of the parameter lines is important, and each separate TX keyword results in a separate line of text in the Command Response window.

A) character (right parenthesis) ends the TX text string. If the text includes an imbedded) character, precede the) with a second) character. The following is the format of the response keyword parameter, if the response to the command is CIRCUIT (A) CONFIGURED AND OPERATIONAL:

```
TX=(CIRCUIT (A)) CONFIGURED AND OPERATIONAL)
```

Command Formatting and Protocol Examples

This section provides examples of the required presentation processing protocol. Functionally, there are two protocols:
- Single-response protocol
- Multiple-response protocol

See “Keyword and Value Definitions” on page 65 for a description of the various keywords and values that make up the command and response packets of the command. See “Command—CM” on page 66 for a list of the command types and continuation bytes.

Single-Response Protocol

The single-response protocol consists of a command designated as an initial command and a response designated as a last response. Figure 16 shows the packets exchanged for a Display Status command and response.

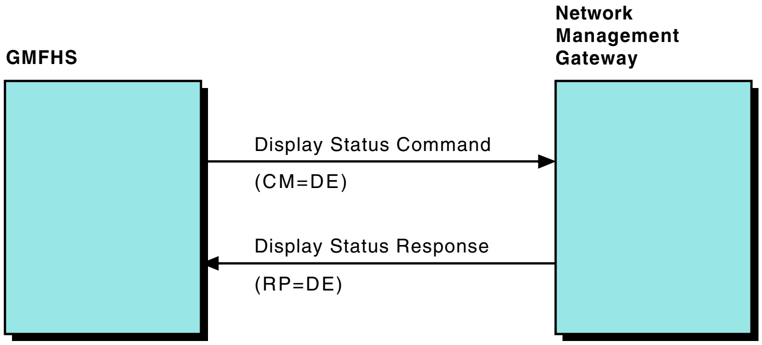


Figure 16. Single-Response Protocol

The command, sent from GMFHS, contains the CM keyword. Maintaining the protocol, the first character of the CM value, D, is interchangeable. It signifies the display status command type. This value can also be any command type valid for the command.

However, the E value in the continuation character specifies an initial command. This character must always be in the first occurrence of a command packet, regardless of whether or not additional command packet continuations (continuation value = M) are required.

In the response from the native element manager, the RP keyword has the value DE. The command type character is interchangeable. The E value in the continuation character specifies that the response is the last response generated.

The protocol has an additional check in the SQ keyword. The SQ value for a response must equal the SQ value for the command.

As the following example shows, the single-response protocol allows for a response containing data for more than a single resource.

The command requests the status of three resources, RALV4.RALXT1, RALV4.RALXT2, and RALV4.TX02, in a single CP keyword parameter.

```
CM=DE,DM=EASTSIDE,CP=(RALV4.RALXT1,RALV4.RALXT2,RALV4.TX02),SQ=1
```

The response contains separate CP keywords for each requested resource.

```
RP=DE,DM=EASTSIDE,CP=RALV4.RALXT1,ST=N,TM=901201135901,
CP=RALV4.RALXT2,ST=N,TM=901201135912,
CP=RALV4.TX02,ST=D,TM=901201135914,SQ=1
```

Note: The CM and SQ keyword parameters are in the command. RP and SQ parameters are in the response.

Multiple-Response Protocol

When the response data is too large to fit in a single response, GMFHS and the NMG use the multiple-response protocol.

The multiple-response protocol consists of:

- A command designated as an initial command
- An unlimited number of continuation responses and commands
- A last response

Figure 17 shows the packets exchanged for a Display Status command and the response in the simplest multiple-response case.

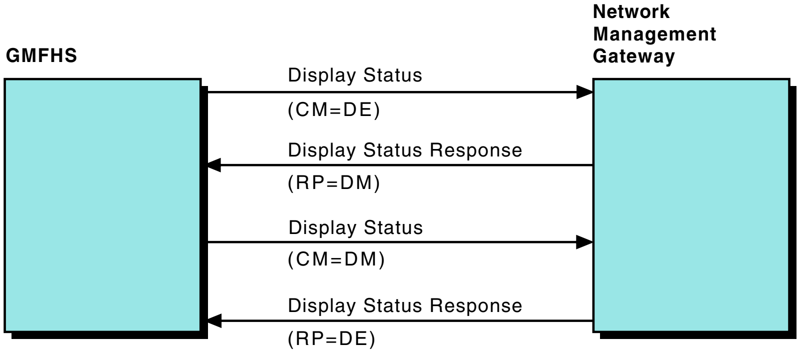


Figure 17. Multiple-Response Protocol

The initial command, sent from the NetView program, contains the CM keyword with the continuation character set to E (CM=AE). The NMG response indicates that the response does not contain all of the data by including the value M as the RP keyword continuation parameter (RP=AM).

To get more of the response data, GMFHS reissues the request. All request parameters are the same as the initial request except for the continuation parameter, which is set to M (CM=DM). The NMG sends the remaining data and indicates that no more data will be sent by setting the continuation parameter to E (RP=AE).

The following initial command calls for a display of all resources in the non-SNA domain B3088P2 that have a status of abnormal:

```
CM=AE,DM=B3088P2,SQ=44
```

The following response results:

```
RP=AM,DM=B3088P2,SQ=44,CP=TIM,ST=A,TM=911231235959,
CP=A0488P23,ST=C,TM=920101000000,
CP=A0488P24,ST=U,TM=920101000001
```

This response indicates that there is a continuation of the response (RP = AM) and provides the statuses of three resources, A0488P22, A0488P23, and A0488P24.

The command is sent again:

```
CM=AM,DM=B3088P2,SQ=44
```

The continuation character is set to M (CM = AM), indicating that the command is a continuation of the previous command with sequence number 44 (SQ=44).

Finally, another response ends the exchange:

```
RP=AE,DM=B3088P2,SQ=44,CP=RALV4.TX02,ST=A,TM=920101000002
```

The continuation character is set to E (RP=AE), indicating that this is the last response.

Timing Considerations

Because status information is contained in both generic alerts and command responses, GMFHS provides a time stamp at the time it processes the alert or response. The date and time of an alert, are provided by the native element manager or its agent in the NMG.

Alerts

The NetView program assumes that the effective time of an alert when the alert is received by the NetView program.

This standard presents problems for non-SNA alerts reported through an NMG. The alert can be delayed significantly in the non-SNA network and in the NMG before it is delivered to the VTAM program and then to GMFHS. Delays can result in inaccurate alert time-stamping that complicates or defeats efforts at network problem resolution. GMFHS uses the following rules to overcome these shortcomings:

- The alert originator can include a date/time subvector in the alert. It overrides the time that the NetView program receives the alert. The Greenwich mean time (GMT) offset in the subvector is used, if in the optional GMT offset subfield.
- If the alert date/time subvector does not include the GMT offset and the native element manager reported its GMT offset at session establishment, the native element manager's offset is used.

- If the alert date/time subvector does not include the GMT offset, and session establishment does not provide an offset, the time in the date/time subvector is used and normalized with the NetView program's local GMT offset.

Command Responses

GMFHS requires that the time-stamp keyword parameter (TM) be included in any command response containing a component status. However, a status response can arrive at GMFHS after a more recent alert for the same component. This happens if the native element manager is assembling a response with statuses from multiple components, and the status of one component changes after it is in the response, but before the response is sent. If the native element manager sends an alert for this component before it sends the command response, GMFHS receives the status indications in the wrong order.

GMFHS recovers from this situation by comparing time stamps. If a status update (either an alert or a command response) is time stamped earlier than the most recent status reported, GMFHS does not apply the new status. GMFHS logs an audit message and a console message.

The time-stamp keyword does not include the GMT offset. GMFHS normalizes time stamps to compare them. If the INIT alert used to establish the session between GMFHS and the native element manager contains the native element manager's GMT offset, this offset is used. Otherwise, the GMFHS local GMT offset is used.

Defining Non-SNA Session Protocols

The session protocol you specify for a non-SNA domain indicates how GMFHS establishes, maintains, and ends command and response communication sessions for that domain. The presentation protocol used for a domain is specified in the SessionProtocolName field of the non-SNA domain object in RODM. The valid session protocol names are:
- DOMS010
- PASSTHRU
- NONE

GMFHS is also responsible for establishing, maintaining, and ending communication sessions with the element managers. GMFHS uses the value of the SessionProtocolName field of the Non_SNA_Domain_Class object to determine how to establish a session with the element manager.

DOMS010

The DOMS010 protocol specifies a set of rules and a command syntax that coordinate the establishment of a command session between GMFHS and the non-SNA domain.

The DOMS010 session protocol specifies that GMFHS and the element manager must verify each other's identities before GMFHS determines that a session exists. The commands GMFHS sends the element manager, and the responses it expects, are described in “Protocol—PT” on page 68. In addition, “Session Establishment for DOMS010” on page 76 contains examples of the identification sequence.

If the domain specifies DOMS010, the commands are formatted according to the DOMP010 formatting rules, regardless of the values in the PresentationProtocolName field.

PASSTHRU

The PASSTHRU protocol specifies that a command session is to exist between GMFHS and the non-SNA domain without any exchange of session establishment information. GMFHS assumes the command session is active immediately upon GMFHS initialization.

NONE

The NONE protocol indicates that there is no command support for the domain.

Session Establishment for DOMS010

The DOMS010 session protocol stipulates that GMFHS must acquire a session with the domain before any other commands are available. Sessions are initiated by GMFHS, or from the element manager. Figure 18 shows a session establishment initiated from the element manager.

To view what GMFHS is reporting as the status of a domain, use the GMFHS SHOW DOMAIN command. Refer to NetView online help for information about the SHOW command.

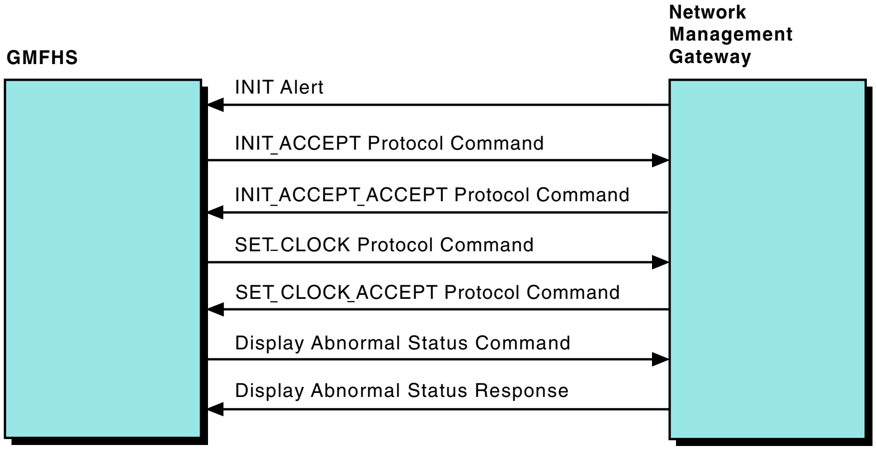


Figure 18. Session Establishment at the Request of the NMG. The commands shown in this figure are described in “Protocol—PT” on page 68.

The element manager can initiate a session with GMFHS by sending an INIT generic alert. When GMFHS receives the alert, it does the following:

- Responds to the NMG with an INIT_ACCEPT protocol command. The INIT alert is described in “INIT Generic Alert for Session Establishment” on page 78.
- Sends a SET_CLOCK protocol command, if supported.
- Sends one or more Display Abnormal Status or Display Status generic commands to retrieve the current status of all the resources. If Display Abnormal Status is not supported, GMFHS issues a Display Status generic command, if supported, for every resource. Whether these commands are supported is specified by the DomainCharacteristics field of the Non_SNA_Domain_Class object that defines the domain to GMFHS.

Session Establishment for DOMS010 Using CNMS4406

Some service points that implement the DOMS010 session protocol might not provide alerts that are adequate for determining whether the service point is running. The CNMS4406 sample is provided by the NetView program to facilitate session establishment between GMFHS and these service points.

This sample provides the INIT and DOWN alert portion of DOMS010 session establishment. Use this sample to specify the following items:

- The three named elements of a non-SNA domain: the service point (SP), the transaction program (TP), and the element management subsystem (EMS). For more information about these, see “Defining Non-SNA Domains” on page 35.
- Whether to send an INIT or DOWN alert. This alert then matches a similarly named domain object in RODM with the service point.

The CNMS4406 sample has the following syntax:

```
►►──CNMS4406──┬─DOWN─┬── sp_name── tp_name── domain_name─────────────────────►◄
              └─INIT─┘
```

Where:

sp_name
Indicates the service point (SP) name as defined to VTAM.

tp_name
Specifies the transaction program (TP) name.

domain_name
Specifies the domain name (also known as the EMS).

The CNMS4406 sample is a NetView command processor coded in the C language. To use this sample, it must first be compiled using C with the LONGNAME compile option and placed in an executable NetView library.

Note: For information about how to compile samples, see *IBM Tivoli NetView for z/OS Programming: PL/I and C*. For information about the LONGNAME compile option, see the *z/OS XL C/C++ Programming Guide* (SC09-4765).

You must also place the following CMDDEF statements in the CNMCMD member in the DSIPARM data set (use included file CNMCMDU for migration purposes):

```
CMDDEF.CNMS4406.MOD=CNMS4406
CMDDEF.CNMS4406.RES=N
```

For example, to run sample CNMS4406 for a domain object named `A0488P31.A94306F8.NETVIEW`, an INIT alert can be sent using the following command from either the NetView command facility or the NetView automation table:

```
CNMS4406 INIT A0488P31 A94306F8 NETVIEW
```

To establish a session between GMFHS and the service point when both are active, place this sample in your automation table to always send the appropriate INIT and DOWN alerts.

GMFHS-Initiated Session Establishment

Although GMFHS is a passive session partner, it can prompt the element manager to initiate a session. The DomainCharacteristics field of a Non_SNA_Domain_Class object confirms that a GMFHS session has been established and solicits status from the NMG for the domain. This prompting can occur:

- At GMFHS startup, and at user-defined time intervals until the session is acquired
- When GMFHS detects an NMG status change to satisfactory, and GMFHS does not have a session with an element manager under the NMG

The DOMS010 protocol uses the same protocol commands shown in Table 8 on page 68 for the DOMP010 protocol. The exchange occurs as illustrated in Figure 19.

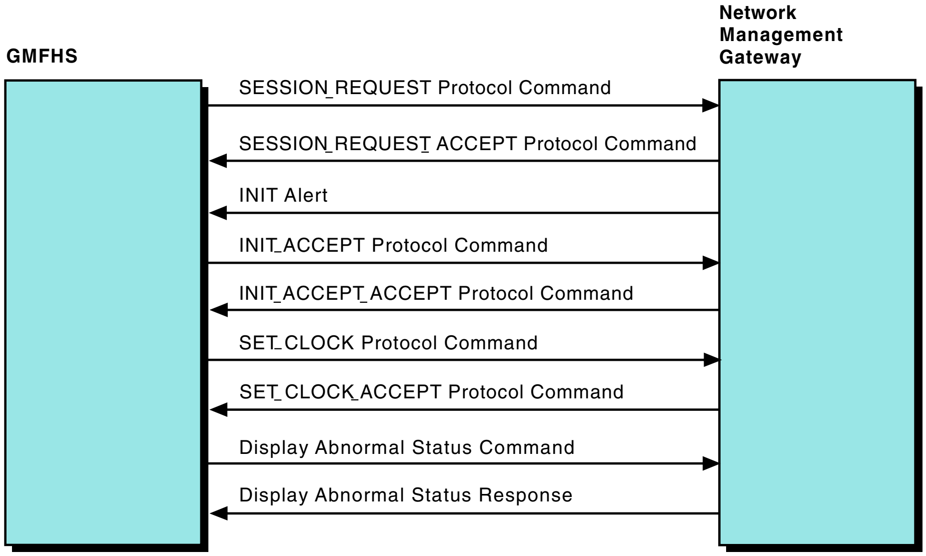


Figure 19. Session Establishment at the Request of GMFHS

GMFHS initiates a session with an element manager by sending a SESSION_REQUEST protocol command. When the element manager receives this command, it responds with SESSION_REQUEST_ACCEPT protocol command and generates the generic INIT alert. The rest of this process is described in “Session Establishment for DOMS010” on page 76.

INIT Generic Alert for Session Establishment

In addition to protocol commands, the DOMS010 protocol includes the INIT alert. An element manager generates an INIT alert to establish a session with GMFHS.

Table 10 lists the subvectors and data that need to appear in the INIT generic alert.

Note: Unless noted as optional, all subvectors and data are required.

Table 10. Generic Alert Subvectors

Subvector	Description
Generic alert data subvector	Alert Type: X'12' (unknown) Alert description code: X'FE00' (undetermined error)
Probable cause subvector	Probable cause code point: X'1001' (application program)
Cause undetermined subvector	Recommended action code point: X'0700' (no action necessary)

Table 10. Generic Alert Subvectors (continued)

Subvector	Description
First product set ID subvector	Product classification: X'xC' (non-IBM software) Software product common name: Identifier of the NMG application (in the non-SNA network) that communicates across the NMG API. Software product common level: 000000 Software product program number: USER0 **Note:** The first product set ID subvector is included to comply with SNA but does not carry significant information.
Second product set ID subvector	Product classification: X'xC' (non-IBM software) Software product common name: name of the native element manager that receives commands Software product common level: 000000 Software product program number: USER0 **Note:** The second product set ID subvector is included to comply with SNA but does not carry significant information.
Date/Time subvector (optional)	An X'01' subvector containing date and time information.
Hierarchy resource list subvector	First resource name (mandatory): Name of the service point First resource type identifier (mandatory): X'81' (service point) Transaction program resource (optional): Transaction program identifier (optional): X'18' (transaction program) Additional resource name (optional): As required, to uniquely identify the domain Additional resource type identifier (optional): Any **Note:** The concatenation of resource names, beginning with the service point, with a period (.) as a delimiter between names, needs to be identical to the MyName field of an object in the RODM Non_SNA_Domain_Class object.

Table 10. Generic Alert Subvectors (continued)

Subvector	Description
Self-defining text message subvector	Text message: INIT[,GMT=*chhmm*] The optional GMT keyword parameter describes the offset to Greenwich mean time (GMT) for all alerts and command responses that contain status information. The keyword value is formatted as follows: *c* is the GMT time modifier code: +, -, or Z. • Specify + to add the GMT modifier to the local time. • Specify - to subtract the GMT modifier from the local time. • Specify Z if the local time is already GMT. In this case *hhmm* is 0000. *hhmm* is the GMT modifier in hours and minutes: • For *hh*, the valid range in 24-hour format is 00—23 . • For *mm* The valid minute range is, 00—59.

Session Termination

Figure 20 shows the alert exchange during session termination.

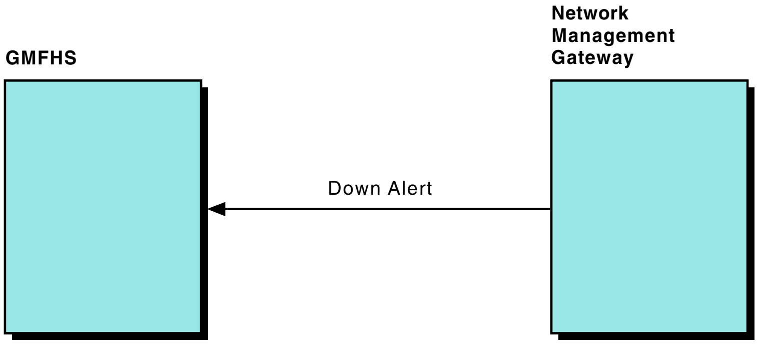


Figure 20. Session Termination

Note: The session termination alert is identical to the alert described in "INIT Generic Alert for Session Establishment" on page 78, except that the self-defining text message subvector contains the text DOWN.

After GMFHS receives this alert, it considers the session down, and sends no commands to the NMG until the session is re-established.

GMFHS also ends the session if it detects a down state for one of the following reasons:
- The status of an NMG changes to Unsatisfactory.
- An alert reports a status change of the element manager to Unsatisfactory.
- GMFHS receives an INIT alert from the element manager.

If an INIT alert is received, the session is ended and immediately re-established.

Defining Non-SNA Transport Protocols

The transport protocol definitions control how network control commands are transported to their non-SNA resource destinations. Depending on the transport protocol you define, you can issue commands at the workstation to control non-SNA resources.

The transport protocol field specifies how GMFHS communicates with the network management gateway (NMG) when delivering commands and accepting responses to commands. The valid protocol names are:

- COS indicates that the NMG is a service point and that GMFHS use RUNCMD commands to communicate with the service point.
- PPI indicates that the NMG uses a program-to-program interface (PPI) and that GMFHS use the PPI to communicate with a system or network management transaction program running in another address space on the focal point host communicating with the NetView management console.
- OST specifies that the NMG is the NetView program and that commands are delivered to a NetView OST.
- NONE specifies that this NMG does not accept commands.

Note: If the NMG represents a service point, its name must be the SNA name of the service point. If the NMG uses the PPI, its name must be the PPI receiver ID used by the NMG. If the NMG is an OST, its name can be any 1-to 8-character name.

COS Gateway Support

The NetView common operations services (COS) gateway support uses the RUNCMD command to deliver network control commands to, and receive command responses from, service points owned by the central site SSCP or remote SSCPs on distributed hosts. Because these service points are accessed by the service point command service (SPCS) of the NetView program, GMFHS does not directly use the communications network management interface (CNMI) of VTAM for this communication.

When you issue a network control command, the transport layer checks the network management gateway (NMG) object TransportProtocolName field. If the field value is COS, the GMFHS host delivers the command to the GMFHS scope checker OPT running in the NetView address space. The scope checker passes the command to the GMFHS COS command processor running on a separate autotask. The COS command processor saves some context information for the command, and creates and issues a RUNCMD command containing the command. The responses to the RUNCMD command are received by GMFHS COS command processor, are correlated to the outstanding command, and are returned to GMFHS. The command list issues the RUNCMD command and obtains responses for it. When all responses are available, they are returned to the COS command processor. The command processor correlates the responses to the command context it retained and returns the responses to GMFHS.

If the service point resides in a distributed NetView system, the COS command processor routes the command over an LU 6.2 session using the MS transport. The NetView program routes the command to the distributed NetView system, runs the command on a distributed router autotask, and returns the responses to the central site NetView program where they are delivered to the COS command processor. The command responses are returned to GMFHS the same way they are returned for responses from a local service point.

To use the COS transport protocol, set the value of the TransportProtocolName field to COS in the NMG_Class object for that gateway.

If the NetView program is communicating with a service point using LU 6.2 and the service point LU has a different NETID than the NetView program that issues the RUNCMD, a bit in the NMGCharacteristics field must specify that the SNA network name be included in the `NETID=` keyword parameter of the RUNCMD.

If the NetView program is communicating with a service point using an SSCP-PU session and the NetView program that issues the RUNCMD does not own the CNMI that communicates with the service point PU, specify the domain name of the NetView program that owns the CNMI on the CommandRouteLUName field of the NMG_Class object for the service point.

Program-to-Program Interface Gateway

The program-to-program interface (PPI) for gateway transport allows a process in an address space other than GMFHS or NetView to receive generic and native network commands from GMFHS, and to return command responses. To use the PPI transport type, define an NMG object with a TransportProtocolName field value of PPI. The MyName field of this NMG object must be the PPI receiver name to which GMFHS will send commands for this gateway.

The messages exchanged through the program-to-program interface use the execute run major vector and the reply-to-execute major vector, except as follows:

- If you specified on the DomainCharacteristics field that command responses are expected from the native element manager, the execute major vector must include a supporting data correlation MS common subvector. The PCID in the supporting data correlation subfield contains the command correlator.
- If GMFHS can not deliver the execute command, the sense data subvector contains the PPI return code that describes why the PPI send request failed. Refer to the *IBM Tivoli NetView for z/OS Application Programmer's Guide* for information about PPI return codes.

OST/PPT Gateway

The NetView OST/PPT provides a gateway transport facility that allows network control commands to be issued using the NetView operator station task (OST) associated with the workstation originating the command, or using the primary program operator interface (POI) task (PPT), if there is no associated workstation operator. NetView command lists and command processors are initiated in response to commands entered by workstation operators. The following characteristics are in effect for this gateway:

- Some OST/PPT commands do not produce a command response, even if the expect responses bit of the DomainCharacteristics field is on.
- Command lists or command processors initiated by this gateway can use the NetView GENALERT facility to report current or resulting resource status so that is reflected in the views. If a command initiated by this facility causes a change that otherwise results in an alert being generated for the target resource, the use of the GENALERT is not necessary.

Monitoring Non-Network Devices

The NetView program enables you to monitor non-network devices, such as a line printer. You can write a command list that issues a GENALERT command that generates a generic alert. Define the names of your RODM real resources representing non-network devices and your RODM non-SNA domain objects that report on these devices, so that they follow the naming conventions used by the GENALERT alert resource hierarchy.

Types of NMGs

GMFHS can communicate with three types of NMGs:
- Common operations services NMGs
- Operator station task NMGs
- Program-to-program interface NMGs

The type of NMG is determined by the TransportProtocol field of the NMG_Class object. All domains managed by an NMG must be of the same type.

Common Operations Services NMGs

GMFHS communicates with common operations services (COS) NMGs with the NetView RUNCMD command. The network command manager task creates the command text according to the presentation and session protocols, then uses the COS gateway command processor autotask to issue the RUNCMD command and wait for the response. For more information about RUNCMD, see NetView online help.

COS NMGs provide the following benefits:
- GMFHS can receive command responses.
- Depending on the presentation protocol, the command responses can contain status information that the network command manager task can interpret.
- Several current service point applications conform to this architecture.
- The responses to operator-initiated commands are displayed in the Non-SNA Command Response window.

The maximum size of a command to a COS NMG is 240 bytes. If the command text length for a presentation or session protocol command exceeds 240 bytes after substitution of any command variables, GMFHS rejects the command.

Operator Station Task NMGs

GMFHS communicates with operator station task (OST) NMGs by sending the command to the requesting operator's OST, or to the PPT for GMFHS-initiated commands. The network command manager task creates the command text according to the presentation and session protocols, then uses the host task manager OPT message queuing service to send the command to the operator's OST or PPT. GMFHS cannot interpret OST command responses, so all status changes must be reported to GMFHS as alerts.

The maximum size of a command to an OST NMG is 256 bytes. If the command text length for a presentation or session protocol command exceeds 256 bytes after substitution of any command variables, GMFHS rejects the command.

Program-to-Program Interface NMGs

GMFHS communicates with program-to-program interface NMGs by exchanging information with another application registered to the program-to-program interface. Commands are formatted within an execute command major vector

(X'8061'). Command responses are returned in two response major vectors (X'0061' and X'1300'). The network command manager task creates the command text according to the presentation and session protocols, and sends it across the program- to-program interface to the element manager. The element manager responds to GMFHS over the program-to-program interface.

Program-to-program interface NMGs provide the following benefits:

- GMFHS can receive command responses.
- Depending on the presentation protocol, the command responses can contain status information that the network command manager task can interpret.
- The responses to operator-initiated commands are displayed in the Non-SNA Command Response window.

The maximum size of a command to a program-to-program interface NMG is 253 bytes. If the command text length for a presentation or session protocol command exceeds 253 bytes after substitution of any command variables, GMFHS rejects the command.

PPI Command Transport Envelope

The text of GMFHS commands is transported to the program-to-program interface NMG in the execute command major vector (X'8061'). This major vector is described in the *System Network Architecture Formats*. However, because GMFHS must have a correlator in command responses, and the architecture of the execute command major vector does not include a correlator subvector, GMFHS departs from the architecture by including a subvector that contains a correlator. This additional correlator is the supporting data correlation subvector (X'48').

Table 11 shows the subvectors and subfields that are included in the execute command major vector.

Table 11. Subvectors and Subfields in the Execute Command Major Vector

Subvector	Subfield	Description
Name list	Destination application name	Value of TransactionProgram field in Non_SNA_Domain_Class object.
Self-defining text message	Coded character set ID	X'00000037'
Self-defining text message	Text message	Command text created by the presentation layer
Supporting data correlation	Fully qualified session PCID	PCID: GMFHS internal correlator Network-qualified CP name: GMFHS.NETCMD

The command response consists of two major vectors:

- Reply to execute command
- Text data parameter

GMFHS ignores all subvectors in the reply-to-execute-command major vector; no subvectors are required. Table 12 on page 85 shows the subvectors and subfields of the text data parameter major vector.

Table 12. Subvectors and Subfields in the Text Data Parameter Major Vector

Subvector	Subfield	Description
Supporting data correlation	Fully qualified session PCID	Must be identical to the subvector in the command PCID: GMFHS internal correlator Network-qualified CP name: GMFHS.NETCMD
Self-defining text message	Text message	Command response text
Self-defining text message	Other subfields	GMFHS ignores all other subfields in this subvector.

Chapter 5. How GMFHS Uses RODM

The Graphic Monitor Facility host subsystem (GMFHS) works with RODM and a NetView management console to display graphic views of networks and issue commands to resources that you select from the view. The views contain both status and configuration information about network resources. This chapter describes how GMFHS uses RODM. Using this information, you can then modify the contents of RODM to change how GMFHS and NetView management console perform.

GMFHS Initialization

GMFHS can be started with either of two options:
- Aggregation warm start
- Resource status warm start

The default is that the options are not run and GMFHS is started normally.

Aggregation Warm Start

An aggregation warm start is caused by coding the AGGRST=YES parameter in the GMFHS startup procedure, CNMGMFHS. An object-independent method, DUIFFAWS, is run to initialize the fields related to status aggregation in the real and aggregate objects in the RODM data cache. See "DUIFFAWS: Aggregation Warm Start Method" on page 495 for more information.

Resource Status Warm Start

A resource status warm start is caused by coding the RESWS=YES parameter in the GMFHS startup procedure, CNMGMFHS.

Resource status warm start provides a mechanism for quickly restoring GMFHS. Use the resource status warm start option if GMFHS has been abnormally ended, and the status of the resources in RODM that were managed by GMFHS are still accurate. GMFHS bypasses the normal resource status initialization process for all domain resources and uses the existing status information in RODM instead.

GMFHS sets the status of resources on a domain basis. For a resource status warm start to occur, a domain must meet one of the following conditions:
- Status solicitation of resources was completed successfully the last time GMFHS was initialized.
- Status solicitation is not supported.
- Skip Status solicitation is indicated.

Resource status warm start requires current status data in RODM. To ensure the current status is maintained in RODM, periodic checkpoints of RODM are required to save the current domain and resource values. RODM can then be loaded using the data sets containing the previous checkpoint data.

Notes:
1. All status updates are lost for the period between the last checkpoint of RODM and when GMFHS was reinitialized.

2. If GMFHS and RODM are warm started on a backup host, the DASD that contains the checkpoint file must be accessible by the backup host.

GMFHS Initialization Process Overview

Normal GMFHS initialization has two subprocesses:

- Setup
- Session Establishment

These subprocesses determine the initial status of the resources in each non-SNA domain. However, under certain circumstances GMFHS does not perform these steps; this is determined by the values of the following GMFHS start option and RODM fields:

- GMFHS warm start option (resws=yes|no)
- The AgentStatus field defined on a NMG_Class object
- The AgentStatusEffect field defined on a NMG_Class object
- The DomainCharacteristics field defined on a Non_SNA_Class object
- The DomainCharacteristics2 field defined on a Non_SNA_Class object

Setup Subprocess

Resources under each domain are set to initial, or unknown, status except under the following conditions:

- GMFHS is started with the resource status warm start option (resws=yes) and the status complete bit is turned **on** in the DomainCharacteristics2 field.
- The skip status setup bit of the DomainCharacteristics field is turned **on**.

Session Establishment Subprocess

The status of the resources within each domain is solicited if status solicitation is supported. For more information about status solicitation, see Chapter 4, “Communicating with Network Management Gateways,” on page 61.

GMFHS does not perform the session-establishment subprocess for a domain if GMFHS is started with the resource status warm-start option (resws=yes), and the status complete bit of the DomainCharacteristics2 field is turned **on**. However, if GMFHS is started with the resource status warm start option (resws=yes), and the status complete bit of the DomainCharacteristics2 field is turned **off**, GMFHS performs the session-establishment subprocess for the domain.

If status solicitation is not supported for a domain, resource status is set according to the following conditions:

- If the value of the AgentStatusEffect field is X'80' and the status complete bit is turned **on** in the DomainCharacteristics2 field, the status of the resources is not changed.
- If the value of the AgentStatusEffect field is X'80' and the status complete bit is turned **off** in the DomainCharacteristics2 field:
  - If the value of the AgentStatus field is either 1 or 3, the status of the resources is set to the status that is indicated by the value of the InitialResourceStatus field.
  - If the value of the AgentStatus field is either 0 or 2, the status of the resources is set to Unknown.
- If the value of the AgentStatusEffect field is X'00', the status of the resources is set to the status that is indicated by the value of the InitialResourceStatus field.

Monitoring Topology Managers

GMFHS can monitor the status of topology managers and indicate this status to operators. Create one object under the Topology_Manager class to represent each topology manager. Note that the SNA topology manager automatically creates this object for you.

Using fields on the Topology_Manager class object, each manager can specify:

- Its status
- The interval within which it must indicate its status before GMFHS assumes it is unavailable
- Its command indicator range

Each manager must periodically update the StatusIndicator field on its object to notify GMFHS that it is active. If this field is not updated within the interval specified by StatusInterval field, GMFHS reports that the manager is unavailable. Topology manager status is displayed in the status area in a NetView management console business view, and is summarized on the status bar for open views.

Building Views

GMFHS builds all views using a 2-step process:

- Object discovery
- Object connectivity

Object *discovery* is the process used to determine the list of objects to display in a view. This process varies depending on the type of view that is requested.

Object *connectivity* is the process used to determine how the objects in the list are interconnected in a view. This process is the same for each type of view. See “Object Connectivity Process” on page 100 for a description of this process.

Object Discovery Process

All of the views that GMFHS builds can be classified in two categories:

- Predefined
- Dynamically built

Predefined Views

Predefined views are represented by a view object in RODM. The view object contains links to each resource that are in the view. The only object discovery processing needed is to query the list of objects currently linked to the view object. Note that objects in exception views are not linked.

Dynamically Built Views

Dynamically built views are not represented by a view object in RODM. Dynamically built views are selected by either choosing an object on an open view and issuing an action against it or by issuing a Locate Resource request for a specific object. In either case, GMFHS receives the request and determines which field on the specified object is queried to find the set of objects necessary to build the view. The fields that are queried depend on the type of view.

For some dynamically built views, GMFHS uses a recursive process to determine the complete list of objects that are to be displayed in a view. For example, when a configuration parent view is requested for an object, GMFHS determines the parent of the object. It then determines whether this parent has a parent. This process is repeated until a parent object is found that has no parent. See “Restricting Recursive Views” on page 113 for more information. The views that use this process are identified in “Object Discovery Process Description for Specific Views” on page 94.

The following objects have important roles in the view building process:
- Display_Resource_Type_Class objects
- View_Information_Object_Class objects

The following overview describes these objects, and “Object Discovery Process Description for Specific Views” on page 94 contains a description of how these objects are used for each type of view.

Display_Resource_Type_Class Object: A Display_Resource_Type_Class object is used to associate an icon with the resource when it is displayed. Displayable objects that can be placed in a view must be linked to an object of the Display_Resource_Type_Class. Linking the displayable object to the Display_Resource_Type_Class object can be done two ways, which are described and illustrated in the following figures:

Note: A displayable object can be linked to a Display_Resource_Type_Class object both ways. When GMFHS encounters this situation, the technique shown in Figure 21 is used.

Prior to NetView Version 3, method DUIFCLRT was usually run to perform the link. DUIFCLRT links the DisplayResourceType field of the displayable object to the Resources field of the Display_Resource_Type_Class object as shown in Figure 21. The disadvantage of this is that you have to run this method for each object.

Figure 21. Technique for Linking Display_Resource_Type_Class Objects Prior to NetView Version 3

You can now associate a Display_Resource_Type_Class object with an object class in RODM as shown in Figure 22 on page 91. This is done by creating a View_Information_Reference_Class object, and placing its object ID in the ViewInfoRefObjDRT field on the object class. The DisplayResourceType field of the View_Information_Reference_Class object is then linked to the ResourceClasses field of the Display_Resource_Type_Class object using method DUIFCLRT. The View_Information_Reference_Class object is used, because links cannot be defined at the class level. The ViewInfoRefObjDRT field is inherited by all objects of the class. The advantage to this technique is that the link is defined only once at the class level instead of individually for each object.

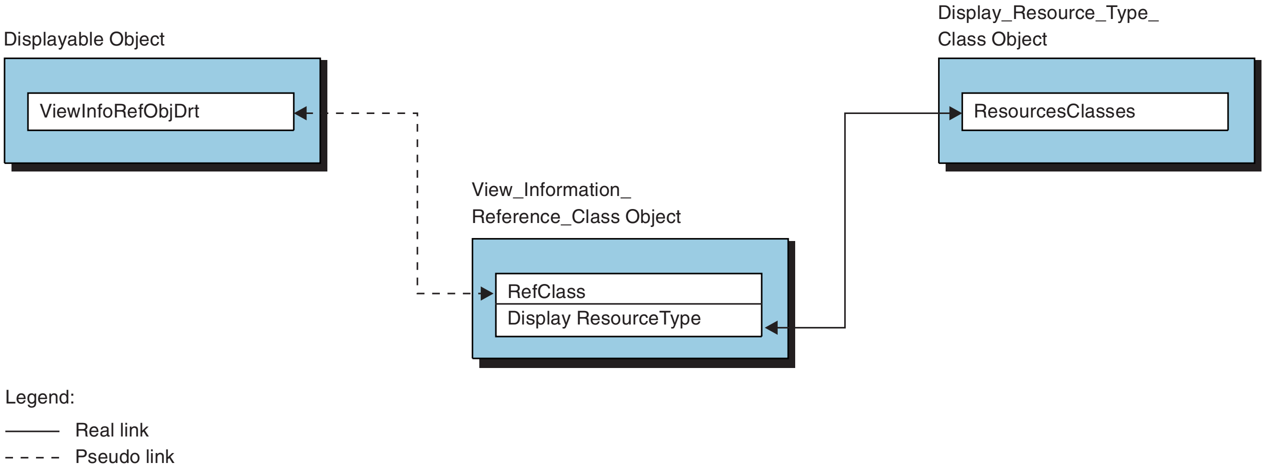


Figure 22. Technique for Linking Display_Resource_Type_Class Objects Now

View_Information_Object_Class object: GMFHS uses View_Information_Object_Class objects for the following purposes:

- To determine which fields on an object to query to find all other related objects when building some dynamically built views.
- To determine how objects in a view are connected. See “Object Connectivity Process” on page 100 for more information.

For both purposes, however, GMFHS uses a common technique to determine which View_Information_Object_Class object to use. There is one View_Information_Object_Class object for every resource-type and view-type pair that GMFHS defines. All resource types ultimately point to the View_Information_Object_Class objects that represent in which types of views they can be displayed in.

All view types ultimately point to the View_Information_Object_Class objects that represent the resource types that can be displayed in a particular type of view. For each object-type and view-type pair, there is only one valid View_Information_Object_Class object to represent the combination. Two techniques can be used to determine the View_Information_Object_Class object, **A** , for a resource:

1. The first technique was the only technique available prior to NetView Version 3. The objects and fields used by this technique are illustrated in Figure 23 on page 92.
2. Starting with NetView Version 3, the second technique is available. The objects and fields used by this technique are illustrated in Figure 24 on page 93.

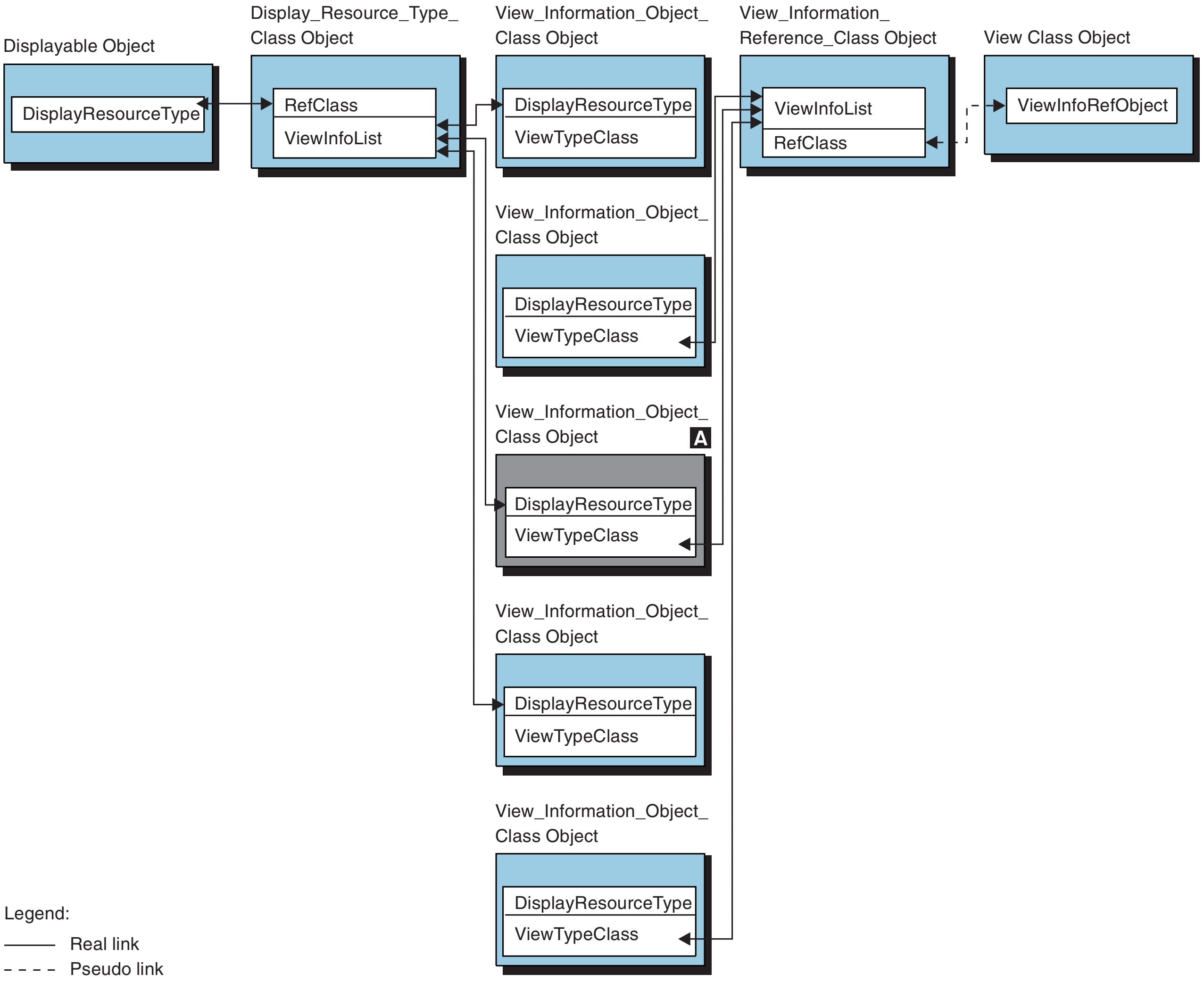


Figure 23. View_Information_Object_Class Object Determination Technique One

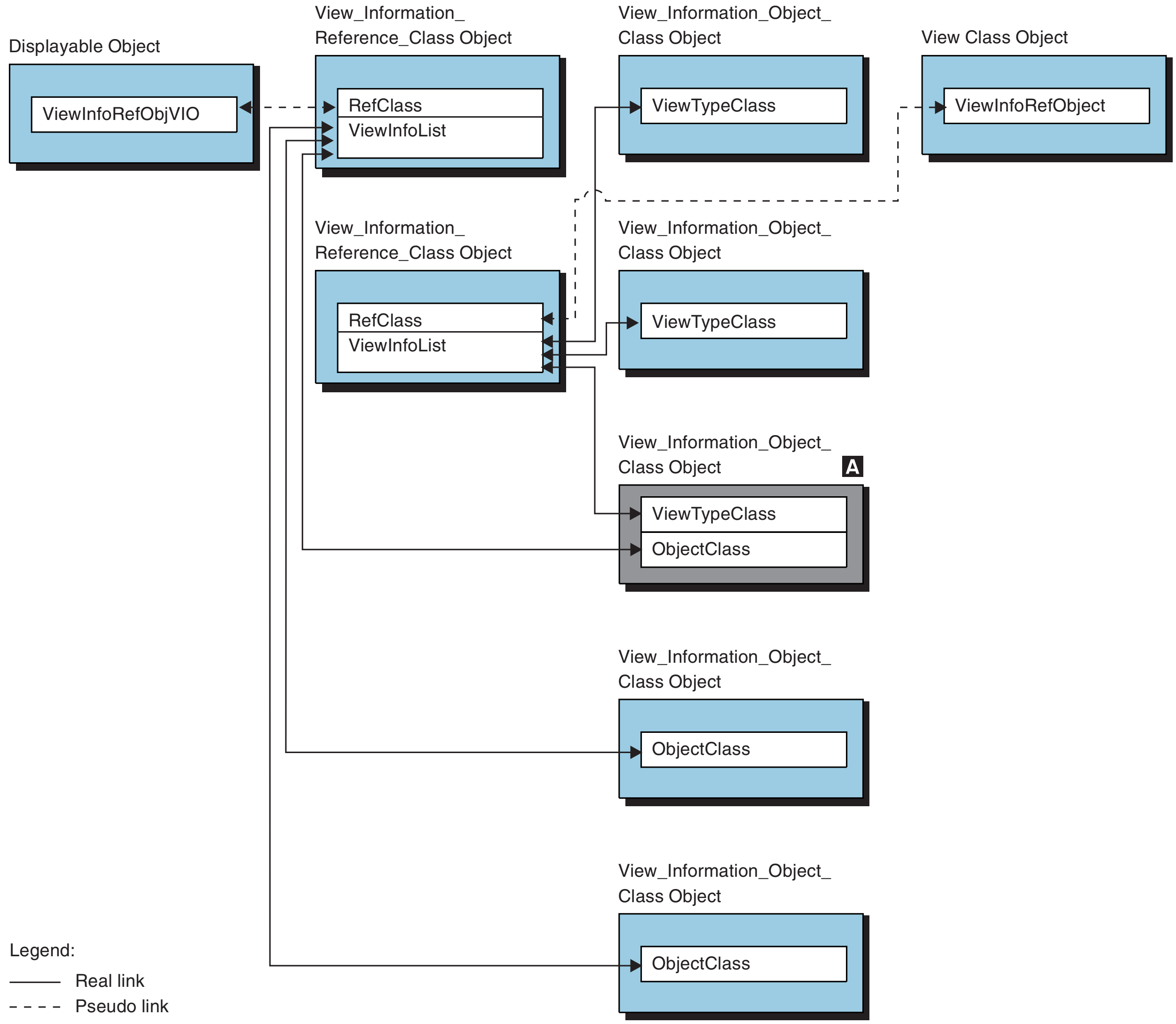


Figure 24. View_Information_Object_Class Object Determination Technique Two

A displayable object can specify a View_Information_Object_Class object using both the DisplayResourceType field (as shown in Figure 23 on page 92) and the ViewInfoRefObjVIO field (as shown in Figure 24). When GMFHS encounters this situation, it uses the View_Information_Object_Class object pointed to by the ViewInfoRefObjVIO field.

Either of two scenarios can occur where GMFHS cannot find a valid View_Information_Object_Class object for a displayable object:

- A View_Information_Object_Class object is not found when an operator selects a view type that is not defined for a resource object, called the root object. In this case, GMFHS displays a message stating that the view type is not enabled for this type of object.
- If an object other than a root object is to be in a view but GMFHS cannot find its View_Information_Object_Class object, GMFHS omits the object and builds the view. Prior to NetView Version 3, if GMFHS cannot find a View_Information_Object_Class object for a resource object, it cannot build the view.

Object Discovery Process Description for Specific Views

This section describes how GMFHS determines which objects to include in a view. Network and exception views are opened by selecting them from the NMC tree view. All other types of views are opened by selecting an object rather than a view name.

The following information is provided for each view:

- Whether the view is predefined or dynamically built. Note that some views can be either predefined or dynamically built.
- A high level description of the logic that GMFHS uses to discover all of the objects.
- The fields that are used by the object discovery process.

Network Views: Network views are predefined views. Each view is represented by a Network_View_Class object in RODM. Every object under this class is queried when the NetView management console server establishes a session with GMFHS, and is displayed in the NMC tree view. Whenever you add or delete network views, this list of views is automatically refreshed. The name of the view that is displayed in the list is the value of the MyName field of the Network_View_Class object.

When a network view is opened, the request is passed to GMFHS. GMFHS queries the ContainsObjects field of the Network_View_Class object. The list of objects that is returned is used by the GMFHS connectivity process. See "Object Connectivity Process" on page 100 for a description of this process.

Configuration Peer Views: Configuration peer views are predefined views. Each view is represented by a Configuration_Peer_View_Class object in RODM. Configuration peer views are similar to network views, but there are two significant differences:

- Configuration views are not available in the NMC tree view.
- A configuration view is called by object, not by name.

When a configuration peer view is opened, the request is passed to GMFHS. GMFHS queries the ContainedInView field on the selected resource object. This field points to every predefined view to which this resource is currently defined. For each of these view objects, GMFHS determines its view type by finding the class on which the object was created. For each Configuration_Peer_View_Class object, GMFHS queries the ContainsObjects field on the specified view object to get the list of objects that are to be placed in the view. The list of objects that is returned is used by the GMFHS connectivity process. See "Object Connectivity Process" on page 100 for a description of this process.

NMC Locate Failing Resources Views: NMC locate failing resources views are dynamically built views which are requested by selecting an aggregate object in an open view and requesting an NMC locate failing resources view.

When an NMC locate failing resources view is opened, NetView management console passes the request to GMFHS. GMFHS queries the AggregationChild field of the selected aggregate object to get a list of all aggregate children objects and real children objects of the aggregate object. For each aggregate child object, GMFHS queries the AggregationChild field of that object to get its children objects. This process is repeated until GMFHS has the complete list of all real objects under the original aggregate.

GMFHS removes all aggregate objects from the list and real objects that meet *any* of the following criteria:

- Does not map to an exception state (ResourceTraits contains NOXCPT).
- Has a UserStatus that indicates the object is suspended from aggregation (UserStatus bit 0x40 is on).
- Has an AggregationPriorityValue that indicates aggregation is not in use (AggregationPriorityValue = -1).

A list of objects that do not meet any of these criteria is passed to the GMFHS connectivity process. See "Object Connectivity Process" on page 100 for a description of this process.

Customizing Fast Path to Failing Resource Views: You can determine which objects appear in an NMC locate failing resources view by customizing how the DisplayStatus of an object maps to the exception state of an object. See "Defining Exception View Objects and Criteria" on page 100 for more information about mapping display status to exception state.

Configuration Children Views: The configuration children view is a dynamically built view which is requested by selecting an object in an open view and selecting a configuration children view. This view shows the operator all children defined to the selected object. To find the children objects of the selected object, GMFHS uses the following process:

- Find the View_Information_Object_Class object.
- Query the RelFieldNamesA field of the View_Information_Object_Class object. For the base GMFHS data model, this field specifies the ChildAccess field. Note that the RelFieldNamesA field is user modifiable and can contain other values.
- The ChildAccess field contains a pointer to all objects that are children of the object.

This process is repeated for each child object of the selected object until the complete list of children is identified. The list of objects is passed to the GMFHS connectivity process. See "Object Connectivity Process" on page 100 for a description of this process.

Configuration Parent Views: The configuration parent view is a dynamically built view which is requested by selecting an object from an open view and selecting a configuration parent view. This view shows the selected object, connection to intermediate parents, and connection to the ultimate parent of the selected object. To find the parent objects of the selected object, GMFHS uses the following process:

- Find the View_Information_Object_Class object.
- Query the RelFieldNamesA field of the View_Information_Object_Class object. For the base GMFHS data model, this field specifies the ParentAccess field. Note that the RelFieldNamesA field is user modifiable, and can contain other values.
- The ParentAccess field contains a pointer to all objects that are parent objects of the selected object.

This process is repeated for each parent object of the selected object until the complete list of parent objects is identified. The list of objects is passed to the GMFHS connectivity process. See "Object Connectivity Process" on page 100 for a description of this process.

Configuration Logical Views: The configuration logical view is requested by selecting an object in an open view and then selecting a configuration logical view. This view shows the selected object and all resource objects that are logically connected to it. Configuration logical views can be dynamically built or predefined.

For dynamically built configuration logical views, GMFHS uses the following process to find the objects that are logically connected to the selected object:

- Find the View_Information_Object_Class object.
- Query the following fields for the base GMFHS data model:
  - RelFieldNamesA, which specifies the LogicalConnUpstream field
  - RelFieldNamesB, which specifies the LogicalConnDownstream field
  - RelFieldNamesAB, which specifies the LogicalConnPP field.

  Note that the RelFieldNamesA, RelFieldNamesB, and RelFieldNamesAB fields are user modifiable and can contain other values.
- These fields contain pointers to the objects that are logically connected to the selected object.

This process is repeated for each resource object that is logically connected to the selected object until the complete list of objects is identified.

For predefined configuration logical views, the request is passed to GMFHS. GMFHS queries the ContainedInView field on the selected resource object. This field points to every predefined view to which this resource is currently defined. For each of these view objects, GMFHS determines its view type by finding the class on which the object was created. For each Configuration_Logical_View_Class object, GMFHS queries the ContainsObjects field on the specified view object to get the list of objects that are to be placed in the view.

For both dynamically built and predefined configuration logical views, the list of objects is passed to the GMFHS connectivity process. See “Object Connectivity Process” on page 100 for a description of this process.

Configuration Physical Views: The configuration physical view is requested by selecting an object from an open view and then selecting a configuration physical view. This view shows the selected object, and all resource objects that are physically connected to it. Configuration physical views can be dynamically built or predefined.

For dynamically built configuration physical views, GMFHS uses the following process to find the objects that are physically connected to the selected object:

- Find the View_Information_Object_Class object.
- Query the following fields for the base GMFHS data model:
  - RelFieldNamesA, which specifies the PhysicalConnUpstream field
  - RelFieldNamesB, which specifies the PhysicalConnDownstream field
  - RelFieldNamesAB, which specifies the PhysicalConnPP field

  Note that the RelFieldNamesA, RelFieldNamesB, and RelFieldNamesAB fields are user modifiable and can contain other values.
- These fields contain pointers to the objects that are physically connected to the selected object.

This process is repeated for each resource object that is physically connected to the selected object until the complete list of objects is identified.

For predefined configuration physical views, the request is passed to GMFHS. GMFHS queries the ContainedInView field on the selected resource object. This field points to every predefined view to which this resource is currently defined. For each of these view objects, GMFHS determines its view type by finding the class on which the object was created. For each Configuration_Physical_View_Class object, GMFHS queries the ContainsObjects field on the specified view object to get the list of objects that are to be placed in the view.

For both dynamically built and predefined configuration physical views, the list of objects is passed to the GMFHS connectivity process. See "Object Connectivity Process" on page 100 for a description of this process.

Configuration Backbone Views: The configuration backbone view is requested by selecting an object from an open view and selecting a configuration backbone view. This view shows the subarea backbone. Configuration backbone views can be dynamically built or predefined.

For dynamically built configuration backbone views, GMFHS uses the following process to find the backbone objects that are related to the selected object:
- Find the View_Information_Object_Class object.
- Query the RelFieldNamesA field of the View_Information_Object_Class object. For the base GMFHS data model, this field specifies the BackboneConnPP field. Note that the RelFieldNamesA field is user modifiable and can contain other values.
- The BackboneConnPP field contains a pointer to all objects that are part of the SNA backbone.

This process is repeated for each backbone object that is related to the selected object until the complete list of backbone objects is identified.

For predefined configuration backbone views, the request is passed to GMFHS. GMFHS queries the ContainedInView field on the selected resource object. This field points to every predefined view to which this resource is currently defined. For each of these view objects, GMFHS determines its view type by finding the class on which the object was created. For each Configuration_Backbone_View_Class object, GMFHS queries the ContainsObjects field on the specified view object to get the list of objects that are to be placed in the view.

For both dynamically built and predefined configuration backbone views, the list of objects is passed to the GMFHS connectivity process. See "Object Connectivity Process" on page 100 for a description of this process.

More Detail Views: More detail views display the next lower layer of child resources for the selected object. There are four types of more detail views:
- More detail logical
- More detail physical
- Configuration child II
- Configuration child III

One or more of these views can be displayed for the selected resource depending on its resource type.

If any of these views yield a view with no objects, the view is not returned to the workstation. If no views can be built, a message is displayed at the workstation saying the view cannot be found.

The following topics describe how GMFHS builds the four types of more detail views.

More Detail Logical: A more detail logical view can be dynamically built or predefined. When a more detail logical view is opened, the request is passed to GMFHS. To determine which objects are in the view, GMFHS performs the following actions:
- Query the ContainsLogical field of the selected object to find the name of the field that is queried to get the list of objects. For the base GMFHS data model, this field specifies the ComposedOfLogical field. The ComposedOfLogical field contains the list of objects that make up the next lower layer of the selected object.
- Pass the list of objects to the GMFHS connectivity process. See "Object Connectivity Process" on page 100 for a description of this process.

More Detail Physical: A more detail physical view can be dynamically built or predefined. When a more detail physical view is opened, the request is passed to GMFHS. To determine which objects are in the view, GMFHS performs the following actions:
- Query the ContainsPhysical field of the selected object to find the name of the field that is queried to get the list of objects. For the base GMFHS data model, this field specifies the ComposedOfPhysical field. The ComposedOfPhysical field contains the list of objects that make up the next lower layer of the selected object.

  Pass the list of objects to the GMFHS connectivity process. See "Object Connectivity Process" on page 100 for a description of this process.

Configuration Child II View: A configuration child II view is a dynamically built view, which shows a subset of the children defined to the selected logical unit object. To find the subset of children of the selected object, GMFHS uses the following process:
- Find the View_Information_Object_Class object.
- Query the RelFieldNamesA field of the View_Information_Object_Class object. This field specifies the list of fields to query to determine the list of the first-level children.

This process is repeated for each child object of the selected object until the complete list of children objects is identified. The list of objects is passed to the GMFHS connectivity process. See "Object Connectivity Process" on page 100 for a description of this process.

If one or more of the fields specified by the RelFieldNamesA field is present on the selected object, the view is displayed even if there are no children. In this case, only the selected object is displayed. This view is displayed with a radial layout with the selected object as the root node.

The following SNA topology manager resource classes use this view type to display the LU-type objects attached to the selected object:
- appnEN
- appnNN
- crossDomainResource

- interchangeNode
- logicalLink
- logicalUnit
- luGroup
- migrationDataHost
- snaNode
- t5Node

Configuration Child III View: A configuration child III view is a dynamically built view, which shows a subset of the children defined to the selected definition group object. To find the subset of children of the selected object, GMFHS uses the following process:

- Find the View_Information_Object_Class object.
- Query the RelFieldNamesA field of the View_Information_Object_Class object. This field specifies the list of fields to query to determine the list of the first-level children.

This process is repeated for each child object of the selected object until the complete list of children objects is identified. The list of objects is passed to the GMFHS connectivity process. See "Object Connectivity Process" on page 100 for a description of this process.

If one or more of the fields specified by the RelFieldNamesA field is present on the selected object, the view is displayed even if there are no children. In this case, only the selected object is displayed. This view is displayed with a hierarchical layout with the selected object as the root node.

The following SNA topology manager resource classes use this view type to display the definition group objects attached to the selected object:

- t5Node
- interchangeNode
- migrationDataHost
- appnEN
- appnNN
- definitionGroup

Exception Views: Exception views are predefined views. Each view is represented by an object created on the Exception_View_Class in RODM. Every object in this class is queried when the NetView management console graphic data server or NetView management console server establishes a session with GMFHS, and is displayed in the NMC tree view. When you add or delete an exception view, this list of views is automatically refreshed. The view name displayed is the value of the MyName field of the Exception_View_Class object.

The object discovery process for exception views is different from other predefined views because the view object does not contain links to each resource in the view. For exception views, object discovery is accomplished by defining a list of candidate objects that can be in an exception view and a series of filters that is constantly applied to that list. These filters reduce the list to include only those objects that you want to be displayed in the exception view. For example, you can define all of your NCPs to an exception view, and set it up so that the only ones displayed in the view are the ones having problems that need attention.

When an exception view is opened, the request is passed to GMFHS, which determines the list of candidate objects. The list of candidate objects is found by first querying the ExceptionViewName field of the Exception_View_Class object.

Then GMFHS issues a locate request for the value of that field against the ExceptionViewList field in RODM. All objects that are defined as candidates are returned with this locate request.

The ExceptionViewFilter field of the Exception_View_Class object contains the filters used to reduce this list. For example, using these filters you can filter out objects that are currently suspended or marked, or objects whose status is not considered a problem. This yields a list of resources that are in a problem state. The list of objects, even if empty, is then passed to the NetView management console to be displayed.

GMFHS keeps all open exception views current. This is done by determining whether views specified in the ExceptionViewList of the resource are open. After comparing the filter for each view to the resource, GMFHS determines if the resource is either added to, or deleted from, an open exception view.

Object Connectivity Process

After the object determination process has determined the list of objects that are in a view, the list is passed to the object connectivity process. GMFHS must now determine how the objects that are listed are interconnected in the view. GMFHS does this by performing the following process, sequentially, for each object listed. For each object, GMFHS performs the following actions:

- Find the View_Information_Object_Class object.
- Query the RelFieldNames*x* field. This field specifies which fields are queried on the object.
- Query those fields on the object.
- Compares the object list returned by the query request to the initial object list that was passed to the connectivity process. All objects that are contained in both lists are connected.
- Pass the view to the NetView management console.

Notes:

1. For exception views, GMFHS does not use this process. All objects are displayed in a grid, and there is no connectivity relationship among these objects.
2. If GMFHS determines that a node is connected to another node, it inserts a null connector link between the two nodes.
3. If a link does not have real nodes as end points, GMFHS inserts null connector nodes.

Defining Exception View Objects and Criteria

To define an exception view complete the following tasks:

1. Create an exception view object and define the criteria for what is considered an exception. This step provides the filters that are applied to the exception view candidate list, which ultimately defines the object to be displayed in a view.
2. Define the objects in RODM that are candidates for exception views.

All exception views are defined on the NetView host; you cannot customize these views from the NetView management console.

Sample DUIFDEXV, Define Exception Views, provides examples for creating four exception view objects and setting two ExceptionViewList values for both the

GMFHS_Managed_Real_Objects_Class and the GMFHS_Aggregate_Objects_Class. The prologue of sample DUIFDEXV contains information about how to define an exception view for GMFHS objects.

Defining Exception Criteria

You can define what constitutes an exception for any given exception view and resource, thus determining when an object is placed in an exception view. The following fields are used to determine when a resource is displayed in an exception view:

- The value of the UserStatus field of the object
- The value of the DisplayStatus field of the object
- The value of the ResourceTraits field of the object
- The ExceptionViewFilter field of the Exception_View_Class object

The UserStatus field of an object allows you to specify whether an object is displayed in an exception view based on an operator entry or an automation program. For example, operators can mark the objects on which they are working, and you can choose to exclude the marked objects from exception views. Or, if your automation routine is trying to recover a failed resource, the automation routine can set the automation-in-progress bit of the object, and you can choose to exclude these objects from exception views. Use the ExceptionViewFilter to customize the processing of these UserStatus values for each exception view.

The DisplayStatus field of an object contains the basic status information used to decide whether an object is placed in an exception view. For example, if the DisplayStatus value is 129 (satisfactory), you probably do not want to display the object in an exception view. If the DisplayStatus value changes to 130 (unsatisfactory), you probably do want to display the object. However, you might want to display some objects with a DisplayStatus value of 132 (unknown) but not display others.

NetView supplies a sample table, DUIFSMT, that maps the DisplayStatus of objects and classes to exceptions or non-exceptions. This mapping is referred to as the exception state of an object.

```
DUIFSMT  CSECT
         DUIFSMTE CLASS=APPNNN,                                        C
               XCPT=(UNSAT,UNKWN,DS152,DS153,DS154,DS155,DS156,DS157,DSC
               158,DS159,MEDUN,LOWUN)
         DUIFSMTE CLASS=INTERCHANGENODE,                               C
               XCPT=(UNSAT,UNKWN,DS152,DS153,DS154,DS155,DS156,DS157,DSC
               158,DS159,MEDUN,LOWUN)
         DUIFSMTE CLASS=MIGRATIONDATAHOST,                             C
               XCPT=(UNSAT,UNKWN,DS152,DS153,DS154,DS155,DS156,DS157,DSC
               158,DS159,MEDUN,LOWUN)
         DUIFSMTE CLASS=T5NODE,                                        C
               XCPT=(UNSAT,UNKWN,DS152,DS153,DS154,DS155,DS156,DS157,DSC
               158,DS159,MEDUN,LOWUN)
         DUIFSMTE CLASS=APPNTRANSMISSIONGROUP,                         C
               XCPT=(UNSAT,UNKWN,DS152,DS153,DS154,DS155,DS156,DS157,DSC
               158,DS159,MEDUN,LOWUN)
         DUIFSMTE CLASS=APPNTRANSMISSIONGROUPCIRCUIT,                  C
               XCPT=(UNSAT,UNKWN,DS152,DS153,DS154,DS155,DS156,DS157,DSC
               158,DS159,MEDUN,LOWUN)
         DUIFSMTE CLASS=T4NODE,                                        C
               XCPT=(UNSAT,UNKWN,DS152,DS153,DS154,DS155,DS156,DS157,DSC
               158,DS159,MEDUN,LOWUN)
         DUIFSMTE CLASS=GMFHS_Managed_Real_Objects_Class,              C
               XCPT=(UNSAT,DS152,DS153,DS154,DS155,DS156,DS157,DS158,DSC
               159,MEDUN,LOWUN)
         DUIFSMTE CLASS=ALL,                                           C
               XCPT=(UNSAT,DEGRD,SDGRD,DS152,DS153,DS154,DS155,DS156,DSC
               157,DS158,DS159)
LAST     DUIFSMTE END
```

Figure 25. Sample Table DUIFSMT

You can customize how the DisplayStatus is interpreted by modifying the DUIFSMT table. See “Customizing the DisplayStatus Mapping Table for Exception Views” on page 104 for more information.

You can also create a RODM user method, which allows you to access RODM data and override the table. See “Creating a DisplayStatus Method for Exception Views” on page 111 for more information.

Note: The exception state of an object is one of the criteria used to determine which real objects are included in an NMC Locate Failing Resources view. Only real objects that map to an exception state are included in an NMC Locate Failing Resources view. See “NMC Locate Failing Resources Views” on page 94 for more information.

The ResourceTraits field of an object contains the value of how DisplayStatus has been interpreted and the state of all UserStatus bits. The ResourceTraits field of an object is used when an exception view is built to determine when an object meets the criteria for inclusion in an exception view.

The ExceptionViewFilter field of an object is defined on all objects of the Exception_View_Class. This field defines the state an object must be in to be displayed in an exception view. The value of the ExceptionViewFilter field is compared to the values for the DisplayStatus and UserStatus fields of the resource object as reflected in the ResourceTraits field. If the values of the ExceptionViewFilter field and ResourceTraits field match, the object is considered an exception and is placed in the defined exception view. See “Defining the ExceptionViewFilter Field” on page 103 for a complete description of ExceptionViewFilter customization.

Defining Candidates for Exception Views

The following fields are used to define in which exception views an object can be displayed:

- The ExceptionViewName field of the Exception_View_Class object
- The ExceptionViewList field of the object

The ExceptionViewName field contains the unique name of the Exception_View_Class object that you created. You must create one Exception_View_Class object for each exception view that you define, and the name of each object must be unique.

The ExceptionViewList field of a resource object contains a list of ExceptionViewNames. You must specify the ExceptionViewName of each exception view in which you want this resource to be displayed when the resource has an exception state. Because a resource can be displayed in more than one exception view, the ExceptionViewList field can contain a list of names.

If you create a resource object to be displayed in an open exception view, one of the following tasks is required:

- Change the ExceptionViewList field from a null value to the list of candidate views.
- Close and then reopen the exception view.

If you want to delete a resource object from RODM that is in an open exception view, remove the ExceptionViewName from the ExceptionViewList before you delete the resource object. If you delete the resource object from RODM before you remove it from the ExceptionViewList, the resource object remains in the view until it is closed because GMFHS cannot send updates for deleted objects.

For SNA resources managed by SNA topology manager, the ExceptionViewList field is set by NetView when the object is created. The NetView program determines the value of this field based on the class of the object. You can change the default mapping of classes to exception views by customizing the FLBEXV table. For more information about customizing the FLBEXV table, see the *IBM Tivoli NetView for z/OS SNA Topology Manager Implementation Guide*.

Defining the ExceptionViewFilter Field

The ExceptionViewFilter field is used to define the state that an object must be in to be placed in an exception view. There are 5 values in the field; each represents a different status filter. Filter 1 is for DisplayStatus, and the remaining 4 filters are for UserStatus.

The default for the ExceptionViewFilter is X'4000' (bit value '0100 0000 0000 0000'), which indicates that:

- Only objects in an exception state are candidates for the view. Objects in an exception state are those objects that have the value XCPT in the ResourceTraits field.
- No filtering is done on UserStatus.

This means that if an object maps to an exception state, it is displayed in an exception view regardless of its UserStatus. The default value of the ExceptionViewFilter can be changed at either the class or object level.

DisplayStatus Filter: Set the ExceptionViewFilter for DisplayStatus to 0 (zero) if you want all objects to be considered candidates for an exception view regardless of the DisplayStatus. If you want only objects that are in an exception state to be considered candidates for an exception view, leave the ExceptionViewFilter for DisplayStatus set to 1, which is the default value.

Shadow objects do not have a DisplayStatus field, so they are not considered to be monitorable objects. However, if you set the filter for DisplayStatus in the ExceptionViewFilter field to 0 (zero), shadow objects are candidates for the view. Shadow objects must adhere to all of the criteria specified in the ExceptionViewFilter field of the view object and the ExceptionViewList field of the shadow object must contain the ExceptionViewName of the view.

UserStatus Filters: Set the UserStatus filters in the ExceptionViewFilter to indicate which UserStatuses are filtered out of the exception view. For example, if you want to filter out objects that have a UserStatus of “mark” set the mark UserStatus filter in the ExceptionViewFilter field to bit value X'01'. If you want to filter all objects that are *not* marked, set the mark UserStatus filter in the ExceptionViewFilter field to bit value X'10'.

An object is not displayed in an exception view if the following bits for UserStatus are on:
- X'02' (not monitored)
- X'40' (aggregation is suspended)

This means that you cannot filter on these bits, because they are automatically filtered from an exception view.

Use the “List Suspended Resources” at the NetView management console to determine which objects have been suspended from aggregation.

Table 13 contains examples of alternate values for the ExceptionViewFilter field and the resultant exception view:

Table 13. Examples of ExceptionViewFilter Field Values and Resultant Views

Value	Objects in View
'0000 0000 0000 0000' (X'0000')	All objects defined to the view regardless of the DisplayStatus or UserStatus.
'0101 0000 0000 0000' (X'5000')	All objects in an exception state defined to the view that are ***not*** marked. All marked objects are filtered out of the view.
'0110 0000 0000 0000' (X'6000')	All objects in an exception state defined to the view that are marked. All objects that are ***not*** marked are filtered out of the view.

Customizing the DisplayStatus Mapping Table for Exception Views

You can customize the mapping of DisplayStatus values using the table DUIFSMT. This table consists of statements created by the DUIFSMTE macro.

To customize the table, change the DUIFSMTE statements in sample DUIFSMT to reflect the desired DisplayStatus mapping and then use sample CNMSJH13 to:
- Assemble and link-edit the table to create a load module.
- Refresh the DisplayStatus change method.
- Trigger a recalculation of the DisplayStatus mapping for all real and aggregate objects in RODM.

Recalculate the DisplayStatus mapping so that the new status is immediately available for exception views. If you do not want to recalculate until the DisplayStatus of the object is changed, comment out the following statement in sample CNMSJH13:

```
OP DUIFRFDS INVOKED_WITH;
```

Figure 26 shows the syntax of the DUIFSMTE macro. You specify the default values for classes not included in the DUIFSMT table using the value `ALL` for *class_name*.

The macro format is shown in Figure 26.

DUIFSMTE

```
►►──DUIFSMTE ──┬─┤ Class ├─┬───────────────────────────────────────────────►◄
               └─END───────┘
```

Class:

```
├──CLASS=class_name──┬───────────────────────┬──┬──────────────────────────┬──────►
                     │      ┌─,──────┐       │  ├─,MYNAME=resource_name────┤
                     │      ▼        │       │  └─,RESOURCE=resource_name──┘
                     └─,XCPT=(──value──)─────┘

►──┬─────────────────────────┬──┬───────────────────────────┬─────────────────┤
   │         ┌─,──────┐      │  └─,USRXMETH=method_name─────┘
   │         ▼        │      │
   └─,STGRPn=(──value──)─────┘
```

Figure 26. Macro DUIFSMTE Syntax

More than one keyword can be specified, but no keyword can be specified more than once.

Where:

CLASS=*class_name*

The name of the class in RODM for which you are customizing DisplayStatus mapping. If you want to specify the default values for classes not included in the DUIFSMT table, use the value `ALL` for *class_name*.

To customize the DisplayStatus mapping for all of the objects of a class, one statement for that class is necessary. To customize the DisplayStatus mapping for specific objects, or groups of objects, of a class, multiple statements are required. Each statement with the same value for *class_name* requires a different value for the RESOURCE or MYNAME keyword.

Note: RODM names are case-sensitive.

For classes managed by SNA topology manager, you can use alias values for class. Table 14 on page 106 lists the aliases you can enter and their corresponding actual class names as known to RODM; both are accepted by the DUIFSMTE macro.

Table 14. Aliases for RODM Class Names

Alias for Class	MyName Value for Class
APPNEN	1.3.18.0.0.1821
APPNNN	1.3.18.0.0.1822
APPNTRANSMISSIONGROUP	1.3.18.0.0.1823
APPNTRANSMISSIONGROUPCIRCUIT	1.3.18.0.0.2058
CROSSDOMAINRESOURCE	1.3.18.0.0.2281
CROSSDOMAINRESOURCEMANAGER	1.3.18.0.0.2278
DEFINITIONGROUP	1.3.18.0.0.2267
INTERCHANGENODE	1.3.18.0.0.1826
LENNODE	1.3.18.0.0.1827
LOGICALLINK	1.3.18.0.0.2085
LOGICALUNIT	1.3.18.0.0.1829
MIGRATIONDATAHOST	1.3.18.0.0.2155
PORT	1.3.18.0.0.2089
T2-1NODE	1.3.18.0.0.1843
T4NODE	1.3.18.0.0.1844
T5NODE	1.3.18.0.0.1845
VIRTUALROUTINGNODE	1.3.18.0.0.1845

See “Implementing Exception View Processing for MultiSystem Manager” on page 112 for information on exception view processing.

XCPT=*value*

Specifies DisplayStatus values of objects considered to be in an exception state. More than one value can be specified, but no value can be specified more than once. Objects with these DisplayStatus values are added to an exception view if the UserStatus and ExceptionViewList criteria are also met.

Note: If XCPT is not specified, or if the value for XCPT is null, the object is not included in an exception view that is defined to only include exception objects. CrossDomainResourceManager in Figure 30 on page 111 is not displayed in an exception view that has an ExceptionViewFilter of X'4000'.

The following **XCPT** values are possible:

DEGRD
Specifies objects with a DisplayStatus value of 133 (degraded).

INTER
Specifies objects with a DisplayStatus value of 131 (intermediate).

LOWSA
Specifies objects with a DisplayStatus value of 145 (low satisfactory).

LOWUN
Specifies objects with a DisplayStatus value of 161 (low unsatisfactory).

MEDSA
Specifies objects with a DisplayStatus value of 144 (medium satisfactory).

MEDUN
Specifies objects with a DisplayStatus value of 160 (medium unsatisfactory).

SATIS Specifies objects with a DisplayStatus value of 129 (satisfactory).

SDGRD
Specifies objects with a DisplayStatus value of 134 (severely degraded).

UNKWN
Specifies objects with a DisplayStatus value of 132 (unknown).

UNSAT
Specifies objects with a DisplayStatus value of 130 (unsatisfactory).

There are 16 possible user-defined DisplayStatus values that are reserved for customer use only. Possible user-defined values for **XCPT** are:

DS136 Specifies objects with a user-defined DisplayStatus value of 136.

DS137 Specifies objects with a user-defined DisplayStatus value of 137.

DS138 Specifies objects with a user-defined DisplayStatus value of 138.

DS139 Specifies objects with a user-defined DisplayStatus value of 139.

DS140 Specifies objects with a user-defined DisplayStatus value of 140.

DS141 Specifies objects with a user-defined DisplayStatus value of 141.

DS142 Specifies objects with a user-defined DisplayStatus value of 142.

DS143 Specifies objects with a user-defined DisplayStatus value of 143.

DS152 Specifies objects with a user-defined DisplayStatus value of 152.

DS153 Specifies objects with a user-defined DisplayStatus value of 153.

DS154 Specifies objects with a user-defined DisplayStatus value of 154.

DS155 Specifies objects with a user-defined DisplayStatus value of 155.

DS156 Specifies objects with a user-defined DisplayStatus value of 156.

DS157 Specifies objects with a user-defined DisplayStatus value of 157.

DS158 Specifies objects with a user-defined DisplayStatus value of 158.

DS159 Specifies objects with a user-defined DisplayStatus value of 159.

STGRPn=*value*, where n is a number from 1 to 8

Specifies a group of DisplayStatus values for status group aggregation (see "Status Groups" on page 142). More than one value can be specified, but no value can be specified more than once per status group. If the DisplayStatus value of a real object matches a DisplayStatus value in a status group, any parent aggregate objects is assigned the DisplayStatus value from the same status group if the status group is defined for the parent aggregate object. If more than one DisplayStatus value is defined in the status group for the aggregate object, the first DisplayStatus value is used.

The groups are prioritized from 1 (high) to 8 (low). For any STGRPn, if the keyword is not specified or is null on either a real or aggregate object then there can be no status override for that status group.

The possible STGRPn values are the same as those listed for the XCPT keyword.

RESOURCE=*resource_name*

The DisplayResourceName of the specific resource or group of resources to which these values apply. You can use the wildcard character * (asterisk) at the end of the resource name to specify groups of resources. You cannot use a wildcard character * embedded in a resource name. See "Specifying Resource Names for DisplayStatus Mapping" on page 109 for more information.

Note: The RESOURCE and MYNAME keywords cannot both be specified in the same DUIFSMTE statement.

MYNAME=*resource_name*

The MyName of the resource or group of resources to which these values apply. You can use the wildcard character * (asterisk) at the end of the resource name to specify groups of resources. You cannot use a wildcard character * embedded in a resource name.

Note: The MYNAME and RESOURCE keywords cannot both be specified in the same DUIFSMTE statement.

USRXMETH=*method_name*

The name of a RODM user method to be triggered for objects in this class; if specified, the method might override the DisplayStatus mapping. See "Creating a DisplayStatus Method for Exception Views" on page 111 for more information.

END

This keyword ends table processing. DUIFSMTE END must be the last statement in your source for the table.

Usage Notes®:

1. In sample DUIFSMT, DUIFSMTE must start in column 10. You can code the keywords in the columns following DUIFSMTE, separated by a space.
2. If a statement exceeds 71 characters, put a continuation character in column 72 and continue the statement in column 16 of the next line.
3. If you enter more than one statement with the same *class_name* and *resource_name* values, the first statement is used and the other statements are ignored; a warning message is issued.

Default Values for Classes

To specify the default values for all classes not defined in the DUIFSMT table, use the value `ALL` for *class_name*. For example:

```
DUIFSMTE CLASS=ALL,XCPT=(DEGRD,INTER,SDGRD,UNSAT)
```

These values apply to all classes unless they are overridden by other statements. You only need to code the specific classes that differ from the values you specify for `CLASS=ALL`.

Specifying Resource Names for DisplayStatus Mapping

You can specify the DisplayStatus mapping for specific resources or groups of resources within a class. To specify the resource name, use the RESOURCE or MYNAME keyword of the DUIFSMTE macro. You can use an asterisk (*), the wildcard character, at the end of the resource name to specify groups of resources. You cannot embed wildcard characters in the resource name.

If you want to customize a specific resource, code the statement for that resource before other generic statements that match in its class. (See Usage Note 3 on page 108.) For example, assume that you have a resource in the GMFHS_Managed_Real_Objects_Class whose DisplayResourceName is RALXT1 and MyName is SYSPLEX.PLEX1.RALXT1. If you want resource SYSPLEX.PLEX1.RALXT1 to map to XCPT if it has an unsatisfactory status, but you do not want other resources in that class to do the same, code the statement for the resource first as shown in Figure 27:.

```
DUIFSMTE CLASS=GMFHS_Managed_Real_Objects_Class,              C
      RESOURCE=RALXT1,XCPT=(UNSAT)
DUIFSMTE CLASS=GMFHS_Managed_Real_Objects_Class,              C
      XCPT=(INTER)
DUIFSMTE CLASS=ALL,                                           C
      XCPT=(UNSAT,UNKWN)
```

Figure 27. Customizing a Resource

If, in Figure 27, the second DUIFSMTE statement had been coded before the first DUIFSMTE statement, resource SYSPLEX.PLEX1.RALXT1 and all other objects in the GMFHS_Managed_Real_Objects_Class map to an exception only when they have an intermediate status.

The rules for the RESOURCE keyword are the same as the rules for the RESOURCE keyword in the customization tables of the SNA topology manager. See the *IBM Tivoli NetView for z/OS SNA Topology Manager Implementation Guide* for more information.

Figure 28 on page 110 illustrates an example of coding both a MYNAME keyword and a RESOURCE keyword for the same class. Assume that you have a resource object in the GMFHS_Managed_Real_Objects_Class whose MyName is SYSPLEX.PLEX1.RALXT1 and DisplayResourceName is RALXT1. If you coded DUIFSMTE entries as shown in Figure 28 on page 110, the resource matches against all three DUIFSMTE entries. However, because the order in which the statements are coded is important, the first DUIFSMTE entry is the one that matches the exception state. This object is an exception only if its DisplayStatus is intermediate.

```
DUIFSMTE CLASS=GMFHS_Managed_Real_Objects_Class,               C
      MYNAME=SYSPLEX.*,                                          C
      XCPT=(INTER)
DUIFSMTE CLASS=GMFHS_Managed_Real_Objects_Class,               C
      RESOURCE=RALV*,                                           C
      XCPT=(SATIS)
DUIFSMTE CLASS=ALL,                                             C
      XCPT=(UNSAT)
DUIFSMTE END
```

Figure 28. Example of a MYNAME and RESOURCE Keyword in the Same DUIFSMTE Entry

Examples of Customizing DisplayStatus Mapping

The examples in this topic are provided to give you a better understanding of mapping DisplayStatus to an exception state. In the first example (shown in Figure 29), assume the following conditions:

- You want to display all objects of the t4Node (1.3.18.0.0.1844) class with a DisplayStatus of unsatisfactory or unknown in an exception view. (Use the alias from Table 14 on page 106 for the class name.)
- You want to display all objects of the appnEN (1.3.18.0.0.1821) class with a DisplayStatus of unsatisfactory, intermediate, or unknown in an exception view. (Use the actual MyName value from Table 14 on page 106 for the class name.)
- You want to display all objects of the GMFHS_Aggregate_Objects_Class in an exception view if their DisplayStatus value is severely degraded.
- For objects in all other classes, you want to place them in exception views only if their DisplayStatus is unsatisfactory or severely degraded.

Using the previously listed conditions, Figure 29 shows the coding of the DisplayStatus mapping table. Note that the fourth statement sets the defaults.

```
DUIFSMTE CLASS=T4NODE,XCPT=(UNSAT,UNKWN)
DUIFSMTE CLASS=1.3.18.0.0.1821,XCPT=(UNSAT,INTER,UNKWN)
DUIFSMTE CLASS=GMFHS_Aggregate_Objects_Class,XCPT=(SDGRD)
DUIFSMTE CLASS=ALL,XCPT=(UNSAT,SDGRD)
```

Figure 29. DisplayStatus Mapping Table Coding Example 1

For the second example (shown in Figure 30 on page 111), assume the following conditions:

- You have created a RODM method named CUSTMTH1 to decide whether objects of the t2-1Node are to be displayed in exception views based on the values of other fields in RODM.
- You do not want objects of the crossDomainResourceManager class to be displayed in any exception view that has an ExceptionViewFilter value of X'4000'.
- You want the object in the appnEN class with a DisplayResourceName of USIBMNT.NCPPU1 to be displayed in an exception view regardless of its status. No user-defined DisplayStatus values are defined.
- You want objects in the appnEN class with the SNA network ID portion of the DisplayResourceName of USIBMNT to be displayed in exception views if their status is not satisfactory. No user-defined DisplayStatus values are defined.

Using the previously listed conditions, Figure 30 on page 111 shows the coding for the DisplayStatus mapping table.

```
DUIFSMTE CLASS=T2-1NODE,USRXMETH=CUSTMTH1
DUIFSMTE CLASS=CROSSDOMAINRESOURCEMANAGER
DUIFSMTE CLASS=APPNEN,                                         C
      RESOURCE=USIBMNT.NCPPU1,                                 C
      XCPT=(DEGRD,INTER,SATIS,SDGRD,UNKWN,UNSAT,MEDSA,MEDUN,LOC
      WSA,LOWUN)
DUIFSMTE CLASS=APPNEN,                                         C
      RESOURCE=USIBMNT.*,                                      C
      XCPT=(DEGRD,INTER,SDGRD,UNKWN,UNSAT,MEDSA,MEDUN,LOWSA,LOC
      WUN)
```

Figure 30. DisplayStatus Mapping Table Coding Example 2

Creating a DisplayStatus Method for Exception Views

You can code an object independent method to provide an extra level of DisplayStatus exception processing in addition to what is provided by the DUIFSMT table. A sample user method, DUIFCUXM, is provided for this purpose. See this sample when writing your user method.

If you specify a method name with the USRXMETH keyword in the DUIFSMT table, that method is triggered asynchronously each time the DisplayStatus of the specified object changes. This method must follow the guidelines for RODM methods. For more information about writing RODM methods, see Chapter 13, “Writing RODM Methods,” on page 339.

The method is triggered asynchronously from the DUIFCRDC method and is passed the object ID for which a DisplayStatus change has occurred. This method has the following input parameters:

```
Smallint   Total_length;
Smallint   Data_Type;
Smallint   Data_Length;
ObjectID   Resource_Object_ID;
Integer    Requested_exception_status;
```

Because the user method is asynchronous, the original conditions that cause it to be driven might not be true when the user method gains control. Therefore, no prequeried field values are passed to the user method from method DUIFCRDC.

Be aware that timing and error handling problems can occur. For example, the mapping of exception state from DUIFSMT can cause an object to be added to an exception view, but the user method can change the exception state of the same object so that it is removed a second later. Errors in the user method must be resolved by the user method. For more information about asynchronous error handling in RODM, see to Chapter 11, “Writing Applications that Use RODM,” on page 301.

If you are receiving unexpected results from your user method and suspect that it is not being triggered, the user method might be installed incorrectly. In this case, RODM issues a return code and reason code in the transaction information block. This error is written to the RODM log as a UAPI trace entry, depending on the values of LOG_LEVEL and MLOG_LEVEL that are set in the customization file. The log entry contains the following information:

- Return code: 8
- Reason code: 81
- Function ID: 1416 (Trigger an Object Independent Method)
- Data: *your user method name*

Note: To test the installation of your user method, you can trigger it using RODMVIEW.

The user method accepts any criteria, including information in RODM, to determine the exception state of an object. When the exception state is determined, method DUIFVCFT, which is provided by IBM, is triggered from the user method to implement the status in the ResourceTraits field of the specified object.

Case 1: Change exception state of an object to XCPT.

1. From the user method, pass Requested_exception_status=1 to method DUIFVCFT.
2. DUIFVCFT changes the ResourceTraits field to XCPT.

Case 2: Change exception state of an object to NOXCPT.

1. From the user method, pass Requested_exception_status=0 to method DUIFVCFT.
2. DUIFVCFT changes the ResourceTraits field to NOXCPT.

In either case, the setting of the ResourceTraits field can result in an object being added to, or deleted from, an open exception view. This determination is made by method DUIFVCFT.

The input parameters to method DUIFVCFT are the same as the input to the user method, except Requested_exception_status is filled in only when you trigger DUIFVCFT. Trigger DUIFVCFT only if the user method determines that the exception state of the input object needs to change.

You can also write a user method to filter resources from a view that are marked as failing because of a higher-level resource failure. Method DUIFCUX2 is provided as a sample method that performs this function.

Implementing Exception View Processing for MultiSystem Manager

An exception view is a graphic list of objects that can be filtered by the value of the DisplayStatus or UserStatus fields of the object. Enabling exception view processing for MultiSystem Manager objects enables you to recognize failing resources in a timely manner.

To implement exception view processing:

1. Modify NetView part DUIFSMT to include the statements from sample FLCSSMT. DUIFSMT is an assembler part and does not support the %INCLUDE statement. As a result, you must include these statements into DUIFSMT by manually editing the file.

   Sample FLCSSMT is the sample table that maps the DisplayStatus of MultiSystem Manager objects and classes to exceptions or non-exceptions. FLCSSMT is shipped in the CNMSAMP data set.
2. Run the NetView JCL sample CNMSJH13 to assemble and link-edit DUIFSMT. This results in:
   - Assembling and link-editing the table to create a load module.
   - Refreshing the DisplayStatus change method.
   - Recalculating the DisplayStatus mapping for all real and aggregate objects in RODM.
3. Modify the MultiSystem Manager exception view file.

The MultiSystem Manager exception view table lists the names of the exception views that a RODM object is associated with when the RODM object is created by MultiSystem Manager.

If you have already implemented exception view processing for MultiSystem Manager, modify the existing MultiSystem Manager exception view table.

If you have not already implemented exception view processing for MultiSystem Manager, copy the FLCSEXV sample to a data set accessible from the DSIPARM DD concatenation defined in your NetView start procedure. Rename the sample file to a name appropriate for your environment. The FLCSEXV sample resides in the CNMSAMP data set.

The FLCSEXV sample contains sample exception view statements for all of the MultiSystem Manager real object classes. You can add exception views for aggregate objects. You can also create an object in the Exception_View_Class class (see sample FLCSDM6 for an example) and then use the MyName field of the Exception_View_Class object as the value for the EXVWNAME keyword.

All of the statements are commented in the sample. If you want to perform exception view processing for a particular object class, uncomment the statements associated with that object class.

FLCSEXV does support the %INCLUDE statement. See the prologue of sample FLCSEXV for information regarding the syntax of the table.

4. Specify the name of the MultiSystem Manager exception view table on the (MSM)COMMON.FLC_EXCEPTION_VIEW_FILE statement in the CNMSTUSR or C*xx*STGEN member.
5. The MultiSystem Manager data model is loaded using the NetView CNMSJH12 sample. The prologue of each of these samples contains a short description of the data model members that are shipped with MultiSystem Manager.

   If you are using the Open feature, uncomment the statement in the CNMSJH12 JCL sample for the FLCSDM6O data model sample.

If you want information about...	**Refer to...**
Exception view processing	*IBM Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer's Guide*
DUIFSMT	*IBM Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer's Guide*

Locate Resource Function

The locate resource function enables the operator to display a resource when the name of the view that contains it is unknown. Multiple types of views can be searched and built when the object is found in RODM.

When the locate resource function is selected, the request is passed to GMFHS. GMFHS issues a locate request for the LocateName field and the DisplayResourceName field for the uppercase version of the entry. The requested views are built for objects in either list. Note that the LocateName field is of type IndexList and can have multiple values. Therefore, you can have multiple aliases for the object, and locate the object using any of them. Remember that the locate is on an uppercase string, so the values in LocateName must also be uppercase. The value of DisplayResourceName field does not have to be uppercase.

Restricting Recursive Views

While building some types of views, GMFHS queries a large number of objects to find all of the objects that belong in a view. This can result in views that are unusable because they have too many objects in them. You can use the HopCount field to restrict the number of objects that GMFHS queries. For example, if you set the value of the HopCount field to 3, GMFHS only queries up to 3 levels of objects from the selected object. If you want GMFHS to query all objects, set the value of the HopCount field to 0 (zero).

Refreshing Open Views

GMFHS sends a view change notification to the workstation when an object, or connectivity field, used in building the view has changed in RODM. This is done by a notification method, DUIFVNOT, that is installed on all connectivity fields as well as fields on objects or classes that control how views are built. The method is installed by sample FLBTRDME when the data model is loaded. FLBTRDME calls an object independent method, DUIFVINS, which installs DUIFVNOT on each field.

Note that the notification method is inherited by the objects of a class. For a list of all the fields on which GMFHS installs DUIFVNOT, see sample FLBTRDME.

Method DUIFVINS must be run for each new class or connectivity field that is added to the data model. See "DUIFVINS: Install View Granularity Method (DUIFVNOT)" on page 499 for a description of method DUIFVINS.

Applying Span-of-Control to Views

This section shows how GMFHS determines which resource and view names are used to check span authorization when building span-restricted views.

This section often refers to the NGMFVSPN and CTL attributes. These are not RODM attributes. They are attributes defined in either the NetView operator profiles in the DSIPRF data set or the NETVIEW segment of the USER profiles in a system authorization facility (SAF) product, such as RACF®. See the *IBM Tivoli NetView for z/OS Security Reference* for more information about these attributes.

Spans can be used to restrict operators from seeing views and resources within views. To apply span-of-control to views:

- Use the NGMFVSPN attribute to specify whether each operator is subject to span checking for views and resources within views.
- Use the NetView span table to define views and resources within views to spans.
- Use the CTL attribute to specify that span checking must be done for this operator.

For more information about defining resources and views to spans in the NetView span table, see the *IBM Tivoli NetView for z/OS Security Reference*.

Before you can use spans to restrict views and resources within views, you need to understand the naming convention used by RODM to identify views and resources. Resource and view names are represented in the NetView span table as resource and view identifiers. These identifiers, which can contain wildcard characters, must match exactly the names used by GMFHS during the view building process. The GMFHS rules for determining resource and view names are described in this section.

Views

As described in “Object Discovery Process” on page 89, all of the views built by GMFHS can be classified as either predefined or dynamically built. GMFHS uses a different procedure to determine the view name, depending on whether the view is predefined or dynamic.

Defining Predefined Views to Spans

Predefined views are defined by the customer. Each predefined view is represented by a view object in RODM. The following types of views can be predefined to RODM:
- Network
- Exception
- Configuration peer
- Configuration backbone
- Configuration logical
- Configuration physical
- More Detail logical
- More Detail physical

Network, exception, and configuration peer views can only be predefined; they are never dynamically built by RODM. The other views in the above list can be either predefined or dynamically built.

When you define a predefined view to a span in the NetView span table, the view identifier must be equal to the MyName attribute of the view object. To see how predefined views can be defined to spans, consider this example. Suppose a network view is predefined to RODM and the MyName field is equal to `MY_NETWORK_VIEW`. If the *span_level* position of the NGMFVSPN attribute specifies that view names are checked for span authorization, GMFHS verifies that the operator requesting the view has span authorization for view name `MY_NETWORK_VIEW`.

If the following statement is defined in the NetView span table, an operator, with span SPAN1 started, can access the view:

```
SPANDEF SPAN=SPAN1,VIEW=MY_NETWORK_VIEW;
```

Alternatively, a SPANDEF statement can be defined using wildcard characters that matches the `MY_NETWORK_VIEW` view name, such as in the following examples:
- `SPANDEF SPAN=SPAN1,VIEW=*VIEW;`
- `SPANDEF SPAN=SPAN1,VIEW=M*;`
- `SPANDEF SPAN=SPAN1,VIEW=*NETWORK*;`

Defining Dynamically Built Views to Spans

Dynamically built views are not represented by a view object in RODM. When you define a dynamically built view to a span in the NetView span table, the view identifier must be equal to the DisplayResourceName field of the selected resource, appended with a three or four character suffix designating the type of view.

The following types of views can be dynamically built by GMFHS:

View Type	**Suffix**
Configuration Backbone	-BAK
Configuration Child	-CHD
Configuration Child II (More Detail LU)	-MLU

Configuration Child III (More Detail Definition Group)	-MDF
Configuration Logical	-LOG
Configuration Logical/Physical	-LP
Configuration Parent	-PAR
Configuration Physical	-PHY
Fast Path	-FP
More Detail Logical	-MDL
More Detail Physical	-MDP

Note: The hyphen is part of the suffix.

This example shows how a dynamically built view can be defined to a span. Suppose an NMC locate failing resource view is selected for an aggregate resource whose DisplayResourceName field is equal to `MyAggResource`. If the span_level position of the NGMFVSPN attribute specifies span checking for view names, GMFHS verifies that the operator requesting the view has span authorization for view name `MyAggResource-FP`.

As another example, suppose a configuration parent view is selected for a real resource whose DisplayResourceName field is equal to `NETA.NCP1`. If the span_level position of the NGMFVSPN attribute specifies span checking for view names, GMFHS verifies that the operator requesting the view has span authorization for view name `NETA.NCP1-PAR`.

When you are defining views to spans, especially dynamically built views, it can be advantageous to use wildcard characters. For more information about wildcard characters, see the *IBM Tivoli NetView for z/OS Security Reference*.

Examples of Defining Views to Spans

The following examples are provided to help you understand how to define views to spans. The examples assume:

- CTL=SPECIFIC has been defined for the operator requesting the view.
- The *span_level* position of NGMFVSPN specifies span checking for view names.
- The operator requesting the view has span SPAN1 started.
- There are no other SPANDEF statements defined in the span table that matches the view names other than those that are defined in the examples.

Example 1: SPANDEF statements that define view identifiers to spans do not exist in the NetView span table. The operator cannot open any views until one or more view identifiers have been defined to span SPAN1 with SPANDEF statements in the NetView span table.

Example 2: Because dynamically built views derive their view names from the resource by which they were selected, resource identifiers can be defined to spans based on the name of the resource. For example, assume all resource names in network A begin with the characters `NETA` and the following statement is defined in the NetView span table:

- `SPANDEF SPAN=SPAN1,VIEW=NETA*;`

An operator with span SPAN1 started can display any view whose view name begins with `NETA`, such as `NETA.NCP-FP`, `NETA_NETWORK_VIEW`, `NETA.HOST-MDL` or `NETA`.

Example 3: If restricting operators by resource name is not feasible, perhaps access to views are restricted by view type. For example, to authorize an operator to see only NMC locate failing resource or more detail views, define the following statement in the NetView span table:

- `SPANDEF SPAN=SPAN1,VIEW=(*-FP,*-MD*);`

An operator with span SPAN1 started can display any NMC locate failing resource or more detail view.

Example 4: To give an operator span authorization for all NMC locate failing resource views except those that are generated by resources in network A, define the following statement in the NetView span table:

- `SPANDEF SPAN=SPAN1,VIEW=(*-FP<NETA*-FP>);`

An operator with span SPAN1 started can display any NMC locate failing resource view except those that are generated by a resource whose DisplayResourceName begins with the characters `NETA`.

Example 5: To give an operator span authorization for all views except more detail views, define the following statement in the NetView span table:

- `SPANDEF SPAN=SPAN1,VIEW=*<*-M*>;`

An operator with span SPAN1 started can display any view except for any type of more detail view.

Example 6: View names are truncated at a maximum of 32 characters. If you have a resource whose DisplayResourceName field is greater than 32 characters, for example, a DisplayResourceName value of `NETWORKA.OPCENTER22.OPERATOR.SHIFT1`. If this resource is selected and a configuration parent view is requested, the resulting dynamic view name sh be `NETWORKA.OPCENTER22.OPERATOR.SHIFT1-PAR`. However, the view name is truncated to 32 characters which results in `NETWORKA.OPCENTER22.OPERATOR-PAR`. Even though the DisplayResourceName is 32 characters, it is truncated because the suffix must be contained within the 32 character view name. The suffix is never truncated from the view name.

Depending on your SPANDEF definitions, this truncation might cause you problems in your span table. Assume that you have set the DisplayResourceName of a group of resources to indicate which shift of operators are responsible for monitoring them. To give an operator span authorization for all resources designated as SHIFT1 resources, you defined the following statement in the NetView span table:

- `SPANDEF SPAN=SPAN1,VIEW=*SHIFT1*;`

View name `NETWORKA.OPCENTER22.OPERATOR-PAR` does not match this SPANDEF statement and the operator cannot display the view. You must either set the value of DisplayResourceName so the length of the value is less than 28 characters or define SPANDEF statements that do not reference truncated characters of the DisplayResourceName.

Resources

If the *span_level* position of the NGMFVSPN attribute specifies span checking for resource names, only those resources that are authorized to a span started for the operator requesting the view are displayed in the view. Before you define resource identifiers to spans in the NetView span table, understand which resource names are used by GMFHS to determine span authorization.

A resource is monitorable if it can be displayed in a view and is not a shadow object. For example, all resources defined in the GMFHS data model under class GMFHS_Monitorable_Objects_Parent_Class are monitorable objects. All monitorable objects in RODM have the following fields:
- MyName
- DisplayResourceName
- UserSpanName

You can assign a value to the MyName field when you create an object in RODM, but you cannot modify the MyName value after the object is created.

You can assign and modify the DisplayResourceName field. This field is used to create the resource names displayed in NetView management console views.

The DisplayResourceName can be set by GMFHS method DUIFCLRT. This method is used to link the DisplayResourceType field of a resource object to the Resources field of an object of the Display_Resource_Type_Class. If the DisplayResourceName is null when the method is triggered, the method sets the value of the DisplayResourceName field equal to the value of the MyName field. If the DisplayResourceName is not null when the method is triggered, no change is made to the DisplayResourceName.

Note: Remember that MultiSystem Manager, SNA topology manager, and other user applications can modify the DisplayResourceName.

You can also create and modify the UserSpanName field. MultiSystem Manager, as well as other user applications, can modify the UserSpanName field. For more information about how MultiSystem Manager uses this field, see *IBM Tivoli NetView for z/OS Installation: Configuring Graphical Components*.

SNA objects defined in RODM as shadow objects, that is, objects defined in the GMFHS_Shadow_Objects_Class, do not have a UserSpanName field. To ensure consistency across RODM-based and workstation-based views, only the MyName field is used to determine span authorization for shadow objects. Even though the DisplayResourceName field can be defined for a shadow object and this name is displayed in a view, the name is not used to determine span authorization.

Depending on how you use RODM, you can assign a different value to each of these fields for a given resource object. For example, when defining a given workstation in your network, you can define the MyName field as *netid.resource_type.real_resource_name* and use this field to keep track of the resources in your network.

You can then define DisplayResourceName for that workstation as the *userid* of the user who owns the workstation. Because the DisplayResourceName value is displayed as the resource identifier in views, this can make it easier for operators to determine the office in which a failing resource is located.

Similarly, you can define the UserSpanName as the *netid* for the network that contains the workstation. You can then use the UserSpanName to define a group of workstations that are all in the same *netid*.

GMFHS uses the following logic to determine span authorization for a resource in a view:
- If the resource is a shadow object, the MyName field is always used to determine span authorization.

- If the resource is not a shadow object:
  - If a value exists for UserSpanName, the UserSpanName field is used to determine span authorization.
  - If a value does not exist for UserSpanName, but a value does exist for DisplayResourceName, the DisplayResourceName field is used to determine span authorization.
  - If a value does not exist for UserSpanName or DisplayResourceName, the MyName field is used to determine span authorization.

Examples of Restricting Resources Within Views Using Spans

The following examples show how you can restrict resources within views. The examples are based on the following assumptions:

- CTL=SPECIFIC was defined for the operator requesting the view.
- The span_level position of NGMFVSPN specifies span checking for resource names.
- The operator requesting the view started span SPAN1.
- There are no other SPANDEF statements defined in the span table that match the resource name.

Note: If a CHARVAR field has a zero (0) length, it is considered to be null. MyName, DisplayResourceName, and UserSpanName are all CHARVAR fields.

Example 1: If DisplayResourceName and UserSpanName are both null, the MyName field determines span authorization for the resource. For example, a monitorable resource in RODM has a MyName value of `SYSPLEX.PLEX1.RALXT1`. The DisplayResourceName and UserSpanName are null. The following statement is defined in the NetView span table:

- `SPANDEF SPAN=SPAN1,RESOURCE=SYSPLEX.PLEX1.RALXT1;`

Thus, an operator with span SPAN1 started can display resource `SYSPLEX.PLEX1.RALXT1` in a view.

Example 2: If UserSpanName is null and DisplayResourceName has a value (in other words, DisplayResourceName is not null), the DisplayResourceName field determines span authorization for the resource. For example, a monitorable resource in RODM has a MyName value of `SYSPLEX.PLEX1.RALXT1` and a DisplayResourceName value of `RALXT1`. The UserSpanName is null. The following statement is defined in the NetView span table:

- `SPANDEF SPAN=SPAN1,RESOURCE=RALXT1;`

An operator with span SPAN1 started can display this resource in a view. Because DisplayResourceName is not null and the resource is not a shadow object, the DisplayResourceName field determines span authorization.

In this situation, it is useful to use a wildcard in the resource definition. If the statement is defined in the NetView span table instead of the previous statements, an operator with span SPAN1 started can display this resource whether or not the DisplayResourceName value is `RALXT1`. If the DisplayResourceName is null, the MyName value of `SYSPLEX.PLEX1.RALXT1` is used to determine span authorization. For example:

- `SPANDEF SPAN=SPAN1,RESOURCE=*RALXT1;`

Example 3: The DisplayResourceName is used to create the resource names displayed in views. While the DisplayResourceName value can be useful to describe resources displayed within views, it might not be useful when determining span authorization. This value can be overridden by setting the UserSpanName field. The DisplayResourceName is still displayed in views, but the UserSpanName value is used for span authorization.

For example, a monitorable resource in RODM has:
- A MyName value of `SYSPLEX.PLEX1.RALXT1`
- A DisplayResourceName value of `RALXT1`
- A UserSpanName value of `BUILDING500.RALXT1`

In this example, the following statement is defined in the NetView span table:
- `SPANDEF SPAN=SPAN1,RESOURCE=BUILDING500.*;`

An operator with span SPAN1 started can display resource `SYSPLEX.PLEX1.RALXT1` in a view.

Now suppose one of the following statements was defined in the NetView span table instead of the previous statement:
- `SPANDEF SPAN=SPAN1,RESOURCE=SYSPLEX.PLEX1.RALXT1;`
- `SPANDEF SPAN=SPAN1,RESOURCE=RALXT1;`

In this case, the operator is denied span authorization to the resource. Because UserSpanName has a value, it is used to determine span authorization for the resource. DisplayResourceName and MyName are not used to determine span authorization when UserSpanName has a value.

Helpful Hints

Occasionally, your resource, view, and span definitions do not yield the results you expect. The following sections describe some helpful hints that you can use in debugging unexpected conditions.

No Views in the View List Are in the Operator's Span-of-Control

If span-of-control is applied to views at the view level, all views are span checked before they are opened and in most cases, before they are put in a view list. If none of the views in the view list are in the operator's span-of-control, depending on the NGMFVSPN value, an informational message is issued that indicates why a view list is not returned.

No Resource in the View Is in the Operator's Span-of-Control

If span-of-control is applied to views at the resource level, all resources in a view are span checked before the view is opened. If none of the resources in the view are in the operator's span-of-control, an informational message is issued that indicates why the view is not opened.

Selected Object Is Not in the Operator's Span-of-Control

If a locate resource is requested for a resource that is not in the operator's span-of-control, an informational message is issued that indicates why a view is not opened.

Similarly, if views (such as, more detail views) are requested for a selected resource in an open view but that resource is no longer in the operator's span-of-control, an informational message is issued that indicates why the view is not opened. This situation can occur only when one of the following conditions is true:

- The operator stopped the span to which the resource had been defined in the NetView span table.
- The NetView span table was changed (and subsequently refreshed) such that the resource is no longer defined to a span that the operator has started.

Resources are not removed from open views when the NetView span table is changed or because spans are started or stopped. These changes are made when the open view is refreshed.

Changing the NGMFVSPN Attribute

The NGMFVSPN attribute that is assigned in the profile of the NetView management console operator remains in effect for the duration of the session of that operator. A changed NGMFVSPN attribute is retrieved only if the NetView operator signs off and signs back on with the new NGMFVSPN attribute and the NetView management console operator signs off and signs back on after the NetView operator is signed back on.

Because of this restriction, a change to the NGMFVSPN attribute does not affect open NetView management console views. All NetView management console views are refreshed after the operator signs back on.

RACF Is Used for RODM Security

If you are using RACF for RODM security, ensure that the NetView domain name is defined to RACF and has a minimum of RODM security level 2. If these security requirements are not satisfied, RODM queries can fail, resulting in span authorization errors.

Applying Span-of-Control to Set and Clear Operator Status

Span of control is applied to the following subset of Set operator status and Clear operator status actions:
- Marker
- Suspended, manually clear
- Suspended, automatically clear

If the operator has an access level of UPDATE(U) to a span-of-control, a marker or suspend action for a selected resource in the span is completed and the operator status is set or cleared as requested by the operator. An access level of UPDATE(U) is required for marker and suspend actions for resources in a span-of-control.

If the operator has only an access level of READ(R) to a span-of-control containing the resource or if the resource is not in a span accessed by the operator, the marker or suspend action for the selected resource is ignored.

Marker or suspend actions against VTAM resources, including shadow objects, is span checked similar to the way they are for commands. If you are using the NetView span table, span checking for marker and suspend actions for RODM objects utilizes the hierarchy of the UserSpanName, DisplayResourceName and MyName fields.

Marker and suspend actions are not optional for span-of-Control. If span-of-control is implemented, an active span for an operator must contain UPDATE(U) access for the resource receiving the marker or suspend action.

- For more information about the hierarchy of the UserSpanName, DisplayResourceName, and MyName fields, see “Resources” on page 117.

- For more information about using spans to protect resources, see the *IBM Tivoli NetView for z/OS Security Reference*.

Applying Policy to Views

Using NMCSTATUS policy definitions, you can define time schedules for resources in NetView management console views. With these schedules, policy is applied to views to specify when the displayable status of one or more resources in a view is disabled at the NetView management console or when one or more resources in a view is suspended from aggregation.

When your NMCSTATUS policy definitions are processed, CHRON timers are set to indicate when the policy is activated and deactivated. Each policy definition specifies a group of resources and actions to be applied to that group of resources during the specified time period.

When the beginning timer pops, the policy is activated. The NMCSTATUS policy code creates a RODM object in the Aggregate_Collection_Class to represent the policy definition. This triggers the RODM Collection Manager to create an aggregate object in the GMFHS_Aggregate_Objects_Class to represent the collection of resource objects based on the RODM field values of the object in the Aggregate_Collection_Class. Resources belonging to the collection are linked to the aggregate by way of the AggregateParent/AggregateChild and ComposedOfLogical/IsPartOf fields. The actions specified on the policy definition are applied to all resources in the collection.

When the ending timer pops, the policy is deactivated. The NMCSTATUS policy code deletes the RODM object from the Aggregate_Collection_Class. This triggers the RODM Collection Manager to delete the corresponding aggregate object in the GMFHS_Aggregate_Objects_Class representing the collection of resource objects belonging to the policy. Any resource object matching the collection is removed from the collection. Status updates are resumed and suspended resources are unsuspended based on the policy definition. If the resource object belongs to another active policy it is not removed from the collection. See "Resources Belonging to Multiple Policies" on page 124 for more information.

Representing Policy Definitions in RODM

Each active policy is represented in RODM by an object in the Aggregate_Collection_Class. Values from the NMCSTATUS keywords are used to set RODM fields on the object. The following list shows the key fields on the object and how the value is derived from the policy definition.

MyName
The name of the object is created by concatenating the timer handle of the CHRON timer that popped, to indicate the beginning of the policy, with the name of the policy definition. For example, if timer handle NMC1 is the beginning timer for policy definition POLICY1, the MyName field of the RODM object is set to NMC1POLICY1.

CollectionSpec1
The RODM Collection Manager language that specifies the collection of resources is generated from the CLASS, MYNAME and RESOURCE keywords or the BLDVIEWSSPEC keyword or the COLLECTIONSPEC keyword. CollectionSpec1 contains 32K of data. If the value is greater than 32K, the additional data is stored in RODM fields CollectionSpec2, CollectionSpec3, or

CollectionSpec4, as needed. Each of these fields also contain 32K of data and are defined in the GMFHS data model (DUIFSTRC).

RequestFlags Indicates which actions apply to the policy. If keyword SUSPENDAGG=YES is specified, the action suspends all the resources in the collection. If keyword STOPUPDATE=YES is specified, the action disables system status updates at the NetView management console for resources in the collection. Both actions can be applied to the same collection of resources.

CollectionLocateName
Value of 'NMCSTATUS' is added to this indexed list field to indicate the object represents a policy definition.

Example 1: At 6:00 a.m., a RODM object is created in the Aggregate_Collection_Class with field values as shown in this example. The timer handle is NMC1.

```
Policy definition:
  NMCSTATUS POLICY1
   CLASS=(GMFHS_Managed_Real_Objects_Class)
   TIME=(06.00.00,18.00.00)
   STOPUPDATE=YES

RODM field values:
  MyName='NMC1POLICY1'
  CollectionSpec1='|GMFHS_Managed_Real_Objects_Class|MyName|*|.CONTAINS.'
  RequestFlags='80000000'x
  CollectionLocateName='NMCSTATUS'
```

Example 2: At 6:00 a.m., a RODM object is created in the Aggregate_Collection_Class with field values as shown in this example. The timer handle is NMC1.

```
Policy definition:
  NMCSTATUS POLICY2
   CLASS=(GMFHS_Managed_Real_Objects_Class)
   RESOURCE=(RALXT1)
   TIME=(06.00.00,18.00.00)
   STOPUPDATE=YES
   SUSPENDAGG=YES

RODM field values:
  MyName='NMC1POLICY2'
  CollectionSpec1='|GMFHS_Managed_Real_Objects_Class|
      DisplayResourceName|RALXT1|.EQ.'
  RequestFlags='C0000000'x
  CollectionLocateName='NMCSTATUS'
```

Example 3: At 6:00 a.m., a RODM object is created in the Aggregate_Collection_Class with field values as shown in this example. The timer handle is NMC1.

```
Policy definition:
  NMCSTATUS POLICY3
   CLASS=(GMFHS_Managed_Real_Objects_Class)
   MYNAME=(SYSPLEX*)
   TIME=(06.00.00,18.00.00)
   SUSPENDAGG=YES

RODM field values:
  MyName='NMC1POLICY3'
  CollectionSpec1='|GMFHS_Managed_Real_Objects_Class|MyName|SYSPLEX*|.CONTAINS.'
  RequestFlags='40000000'x
  CollectionLocateName='NMCSTATUS'
```

Example 4: At 6:00 a.m., a RODM object is created in the Aggregate_Collection_Class with field values as shown in this example. The timer handle is NMC1.

```
FILE1 contains the following BLDVIEWS statements:
  Majnode=NETA.A01M,
  Type=XCA

Policy definition:
  NMCSTATUS POLICY4
   BLDVIEWSSPEC=(QSAMDSN,USER.INIT(FILE1))
   TIME=(06.00.00,18.00.00)
   STOPUPDATE=YES

RODM field values:
  MyName='NMC1POLICY4'
  CollectionSpec1='|1.3.18.0.0.3315.8.3.7|MyName|1.3.18.0.2.4.6=*;
                   1.3.18.0.0.2032=*;1.3.18.0.0.2032=XCA.NETA.A01M|.CONTAINS.'
  RequestFlags='80000000'x
  CollectionLocateName='NMCSTATUS'
```

Example 5: At 6:00 a.m., a RODM object is created in the Aggregate_Collection_Class with field values as shown in this example. The timer handle is NMC1.

```
DDFFILE2 is a data definition file allocated with command
  ALLOCATE FILE(DDFFILE2) DATASET(USER.INIT(FILE2)) SHR

DDFFILE2 contains the following BLDVIEWS statements:
  NONSNA=*

Policy definition:
  NMCSTATUS POLICY5
   BLDVIEWSSPEC=(QSAMDD,DDFFILE2)
   TIME=(06.00.00,18.00.00)
   STOPUPDATE=YES

RODM field values:
  MyName='NMC1POLICY5'
  CollectionSpec1='|GMFHS_Managed_Real_Objects_Class|MyName|*|.CONTAINS.'
  RequestFlags='80000000'x
  CollectionLocateName='NMCSTATUS'
```

Resources Belonging to Multiple Policies

A resource can be defined to multiple policy definitions. A count of the number of active policies the resource belongs to is saved in a counter field. Each displayable resource object has two counter fields defined:

PolicyCtrSU	Represents the number of active policies this resource belongs to where the action applied to the resource is *stop updates*.
PolicyCtrSA	Represents the number of active policies this resource belongs to where the action applied to the resource is *suspend aggregation*.

These fields ensure that actions are not removed from a resource belonging to other active policies. When a resource is removed from a policy, the applicable counter is decreased by one. When the counter is zero, the action is removed from the resource. If the counter is not zero, the resource belongs to another active policy and the action remains in place.

Example 1: POLICY1 specifies status updates sh not be sent to resource ABC on Saturdays. POLICY2 specifies status updates sh not be sent to real resources beginning with the letter A, i.e. RESOURCE=A* from 8 a.m. to 10 a.m. every day, including Saturdays.

```
Policy definitions:
  NMCSTATUS POLICY1
   CLASS=(GMFHS_Managed_Real_Objects_Class)
   RESOURCE=(ABC)
   DAYOFWEEK=(SAT)
   TIME=(00.00.00,23.59.59)
   STOPUPDATE=YES
  NMCSTATUS POLICY2
   CLASS=(GMFHS_Managed_Real_Objects_Class)
   RESOURCE=(A*)
   TIME=(08.00.00,10.00.00)
   STOPUPDATE=YES
```

1. Saturday at 12:00 a.m., a timer pops and POLICY1 is activated. The PolicyCtrSU field of resource ABC is increased by one. PolicyCtrSU=1 for resource ABC and status updates are not sent to the resource.
2. Saturday at 8 a.m., a timer pops and POLICY2 is activated. The PolicyCtrSU field of all real resources A* in the collection is increased by one. PolicyCtrSU=2 for resource ABC because the resource belongs to both collections. PolicyCtrSU=1 for the resources belonging only to the POLICY2 collection. Status updates are not sent for any resource whose PolicyCtrSU field is not zero.
3. Saturday at 10 a.m., a timer pops and POLICY2 is deactivated. The PolicyCtrSU field of all real resources A* in the collection is decreased by one. PolicyCtrSU=1 for resource ABC since the resource still belongs to the POLICY1 collection. PolicyCtrSU=0 for the resources belonging only to the POLICY2 collection. Status updates are sent for these resources but not for resource ABC.
4. Saturday at 11:59 p.m., a timer pops and POLICY1 is deactivated. The PolicyCtrSU field of resource ABC is decreased by one. PolicyCtrSU=0 for resource ABC. Status updates are now sent.

Example 2: POLICY1 specifies aggregation is suspended for resource ABC on Saturdays. POLICY2 specifies aggregation is suspended for real resources beginning with the letter A, i.e. RESOURCE=A* from 8 a.m. to 10 a.m. every day, including Saturdays.

```
Policy definitions:
  NMCSTATUS POLICY1
   CLASS=(GMFHS_Managed_Real_Objects_Class)
   RESOURCE=(ABC)
   DAYOFWEEK=(SAT)
   TIME=(00.00.00,23.59.59)
   SUSPENDAGG=YES
  NMCSTATUS POLICY2
   CLASS=(GMFHS_Managed_Real_Objects_Class)
   RESOURCE=(A*)
   TIME=(08.00.00,10.00.00)
   SUSPENDAGG=YES
```

1. Saturday at 12:00 a.m., a timer pops and POLICY1 is activated. The PolicyCtrSA field of resource ABC is increased by one. PolicyCtrSA=1 for resource ABC and aggregation is suspended for resource ABC.
2. Saturday at 8 a.m., a timer pops and POLICY2 is activated. The PolicyCtrSA field of all real resources A* in the collection is increased by one. PolicyCtrSA=2 for resource ABC because the resource belongs to both collections. PolicyCtrSA=1 for the resources belonging only to the POLICY2 collection. Aggregation is suspended for any resource whose PolicyCtrSA field is not zero.
3. Saturday at 10 a.m., a timer pops and POLICY2 is deactivated. The PolicyCtrSA field of all real resources A* in the collection is decreased by one. PolicyCtrSA=1 for resource ABC since the resource still belongs to the POLICY1 collection. PolicyCtrSA=0 for the resources belonging only to the POLICY2

collection. Aggregation is no longer suspended for these resources but continues to be suspended for resource ABC.

4. Saturday at 11:59 p.m., a timer pops and POLICY1 is deactivated. The PolicyCtrSA field of resource ABC is decreased by one. PolicyCtrSA=0 for resource ABC. The resource is no longer suspended from aggregation.

Example 3: A NetView management console operator can resume aggregation for a resource that is currently suspended from aggregation by a policy. Setting or clearing the suspend flag from NetView management console overrides any policy that is active. However, the PolicyCtrSA field is increased and decreased only when the resource is added or removed from a collection. In this example, POLICY1 specifies that resource PC1 is suspended from aggregation on Saturdays. POLICY2 specifies that resource PC1 is suspended from aggregation from 8 a.m. to 10 a.m. every day, including Saturdays. An operator can change the value of the suspend flag of a resource; however, policy continues to update the suspend flag when policies are activated and deactivated.

```
Policy definitions:
  NMCSTATUS POLICY1
   CLASS=(GMFHS_Managed_Real_Objects_Class)
   RESOURCE=(PC1)
   DAYOFWEEK=(SAT)
   TIME=(00.00.00,23.59.59)
   SUSPENDAGG=YES
  NMCSTATUS POLICY2
   CLASS=(GMFHS_Managed_Real_Objects_Class)
   RESOURCE=(PC1)
   TIME=(08.00.00,10.00.00)
   SUSPENDAGG=YES
```

1. Saturday at 12:00 a.m., a timer pops and POLICY1 is activated. The PolicyCtrSA field of resource PC1 is increased by one. PolicyCtrSA=1 for resource PC1 and aggregation is suspended for resource PC1.
2. Saturday at 8 a.m., a timer pops and POLICY2 is activated. The PolicyCtrSA field of resource PC1 is increased by one. PolicyCtrSA=2 for resource PC1 because the resource belongs to both collections. The resource remains suspended from aggregation.
3. Saturday at 10 a.m., a timer pops and POLICY2 is deactivated. The PolicyCtrSA field of resource PC1 is decreased by one. PolicyCtrSA=1 for resource PC1 because the resource still belongs to the POLICY1 collection. The resource remains suspended from aggregation.
4. Saturday at 3 p.m., a NetView management console operator clears the suspend flag for resource PC1. The PolicyCtrSA field remains unchanged (it is still equal to 1) but the resource is no longer suspended from aggregation.
5. Saturday at 11:59:59 p.m., a timer pops and POLICY1 is deactivated. The PolicyCtrSA field of resource ABC is decreased by one. PolicyCtrSA=0 for resource ABC. In this example, the suspend flag has already been cleared but if it hadn't, the suspend flag is cleared and resource PC1 is no longer suspended from aggregation.

Although a NetView management console operator can change the value of the suspend flag of a resource, policy continues to update the suspend flag when policies are activated and deactivated.

Example 4: A policy can specify that a resource is suspended from aggregation and does not receive status. In this situation, both counters are used to keep track of the number of active policies the resource belongs to for each action. In this example, POLICY1 specifies that status updates are not sent to resource PC1 on

Saturdays. POLICY2 specifies that resource PC1 is suspended from aggregation on Saturdays from 8 a.m. to 5 p.m. POLICY3 specifies that status updates are not sent to resource PC1 and resource PC1 is suspended from aggregation from 2 p.m. to 4 p.m. on Saturdays.

```
Policy definitions:
  NMCSTATUS POLICY1
   CLASS=(GMFHS_Managed_Real_Objects_Class)
   RESOURCE=(PC1)
   DAYOFWEEK=(SAT)
   TIME=(00.00.00,23.59.59)
   STOPUPDATE=YES
  NMCSTATUS POLICY2
   CLASS=(GMFHS_Managed_Real_Objects_Class)
   RESOURCE=(PC1)
   DAYOFWEEK=(SAT)
   TIME=(08.00.00,17.00.00)
   SUSPENDAGG=YES
  NMCSTATUS POLICY3
   CLASS=(GMFHS_Managed_Real_Objects_Class)
   RESOURCE=(PC1)
   DAYOFWEEK=(SAT)
   TIME=(14.00.00,16.00.00)
   STOPUPDATE=YES
   SUSPENDAGG=YES
```

1. Saturday at 12:00 a.m., a timer pops and POLICY1 is activated. The PolicyCtrSU field of resource PC1 is increased by one. Counter field values are PolicyCtrSA=0 and PolicyCtrSU=1. Status updates are no longer sent to resource PC1.
2. Saturday at 8 a.m., a timer pops and POLICY2 is activated. The PolicyCtrSA field of resource PC1 is increased by one. Counter field values are PolicyCtrSA=1 and PolicyCtrSU=1. Status updates are still not sent to resource PC1 and the resource is also suspended from aggregation.
3. Saturday at 2 p.m., a timer pops and POLICY3 is activated. Both counter fields are increased by one. Counter field values are PolicyCtrSA=2 and PolicyCtrSU=2. Status updates are still not sent to resource PC1 and the resource remains suspended from aggregation.
4. Saturday at 4 p.m., a timer pops and POLICY3 is deactivated. Both counter fields are decreased by one. Counter field values are PolicyCtrSA=1 and PolicyCtrSU=1. Status updates are still not sent to resource PC1 and the resource remains suspended from aggregation.
5. Saturday at 5 p.m., a timer pops and POLICY2 is deactivated. The PolicyCtrSA field of resource PC1 is decreased by one. Counter field values are PolicyCtrSA=0 and PolicyCtrSU=1. Status updates are still not sent to resource PC1. The resource is no longer suspended from aggregation.
6. Saturday at 11:59:59 p.m., a timer pops and POLICY1 is deactivated. The PolicyCtrSU field of resource ABC is decreased by one. Counter field values are PolicyCtrSA=0 and PolicyCtrSU=0. Status updates are now sent to resource PC1.

Resources Suspended from Aggregation Due to Policy

When a real resource is suspended from aggregation because of a scheduled policy definition, the resource is added to a collection representing the policy and the following actions occur in GMFHS:

- The suspend flag of the resource is set.
- The suspend flag note of the resource is set to *Scheduled*.
- One is added to the PolicyCtrSA of the resource.

When aggregation is resumed for a real resource because of a policy definition, the resource is removed from the collection representing the policy and the following actions occur in GMFHS:

- The suspend flag of the resource is cleared.
- The suspend flag note of the resource is cleared.
- One is subtracted from the PolicyCtrSA of the resource.

The suspend flag is cleared only if the value of the note is "Scheduled" and was set by operator ID *GMFHS*.

If a policy definition specifies SUSPENDAGG=YES and STOPUPDATE=NO, the affected resources do not change to the *Scheduled* system status. The resources are suspended from aggregation but continue to receive system status updates.

A NetView management console operator can override the setting of the suspend flag. See "Resources Belonging to Multiple Policies" on page 124 for more information.

Suspending Aggregation Using an Aggregate

When an aggregate is suspended from aggregation, the aggregate itself is not suspended from aggregation. Instead, all of the real objects currently reporting status to the aggregate are suspended from aggregation. The following actions occur in GMFHS:

- The suspend flag of the real resource is set.
- The suspend flag of the real resource note is set to *Scheduled*.
- One is added to the PolicyCtrSA of the real resource.
- The suspended flag of the aggregate child is set.
- The suspended flag note of the aggregate child is set to *Scheduled*.

The child suspended flag is also set for any aggregates in the AggregateChild/ AggregateParent path between the aggregate affected by policy and the real resources reporting status to that aggregate. However the child suspended flag note field is not set to *Scheduled* for these intermediate aggregate resources.

When aggregation is resumed for an aggregate, the aggregate itself is not resumed. Instead aggregation is resumed for all of the real objects currently reporting status to the aggregate. The following actions occur in GMFHS:

- The suspend flag of the real resource is cleared
- The suspend flag of the real resource note is cleared.
- One is subtracted from the PolicyCtrSA of the real resource.
- The suspended flag of the aggregate child is cleared.
- The suspended flag note of the aggregate child is cleared.

Example: AGGPOLICY specifies aggregation is suspended for aggregate resource AGG1 on Saturdays.

```
Policy definitions:
  NMCSTATUS AGGPOLICY
   CLASS=(GMFHS_Aggregate_Objects_Class)
   RESOURCE=(AGG1)
   DAYOFWEEK=(SAT)
   TIME=(00.00.00,23.59.59)
   SUSPENDAGG=YES
```

1. Saturday at 12:00 a.m., a timer pops and AGGPOLICY is activated. Aggregate resource AGG1 is added to the collection and the action (suspending aggregation) is applied to the resource. Suspending an aggregate from aggregation is a shortcut request to suspend all real resources currently

reporting status to the aggregate from aggregation. The PolicyCtrSA field of each real resource is increased by one. The PolicyCtrSA field of the aggregate is not updated because the aggregate itself is not suspended.

2. Saturday at 11:59:59 p.m., a timer pops and AGGPOLICY is deactivated. Aggregate resource AGG1 is removed from the collection and the action (suspending aggregation) is removed from each resource. Unsuspending an aggregate from aggregation is a shortcut request to resume aggregation for all real resources currently reporting status to the aggregate. The PolicyCtrSA field of each real resource is decreased by one. The PolicyCtrSA field of the aggregate is not updated because the aggregate itself was never suspended and can not be unsuspended.

If additional real resources begin to report status to aggregate AGG1 after the policy is activated, they are not suspended by the policy definition AGGPOLICY. Actions can only be applied to a member of the collection. The real resources are suspended and resumed only because of an action to aggregate AGG1, a member of the collection.

System Status Updates No Longer Sent to Resources Due to Policy

When system status updates occur, the DisplayStatus field of the resource is updated with the new status. A change to the DisplayStatus field triggers an update to the resource if it is displayed in an open NetView management console view.

When system status updates are no longer sent to a resource because of a scheduled policy definition, the resource is added to a collection representing the policy. For the case where this is the only active policy the resource belongs to, the following actions occur in GMFHS:

- The PolicyDisplayStatus field is set to the current value of the DisplayStatus field.
- The DisplayStatus field is set to *Scheduled*.
- The system status update sends *Scheduled* to the resource if it is displayed in an open NetView management console view.
- One is added to the PolicyCtrSU field of the resource.

Any system status updates received for this resource while it belongs to an active policy are saved in the PolicyDisplayStatus field rather than the DisplayStatus field. Thus system status updates are not sent to NetView management console.

When system status updates are resumed, the resource is removed from the collection representing the policy. The following actions occur in GMFHS.

- One is subtracted from the PolicyCtrSU field of the resource.
- If the PolicyCtrSU field=0, then the DisplayStatus field is set to the current value of the PolicyDisplayStatus field. This drives a NetView management console update to change the resource from *Scheduled* status to its current system status.
- If the PolicyCtrSU field is greater than zero, the DisplayStatus field remains *Scheduled* and any system status updates are saved in the PolicyDisplayStatus field. No update is sent while the resource belongs to a collection representing a policy where STOPUPDATE=YES was specified.

Additional Information

See the *IBM Tivoli NetView for z/OS Administration Reference* for information about creating and loading a policy file containing NMCSTATUS policy definitions.

See *IBM Tivoli NetView for z/OS Installation: Configuring Graphical Components* for information about the tasks necessary to process NMCSTATUS policy definitions.

See the *IBM Tivoli NetView for z/OS Data Model Reference* for information about specific RODM fields.

Aggregation Concepts

This section describes aggregation for network resources. The topology of network resources is managed by RODM. Network resources, including aggregate resources, are displayed in NetView management console views, based on information gathered by GMFHS.

Aggregation Overview

Aggregation is the process of creating, connecting, and updating the status of aggregate objects. *Aggregate objects* represent a collection of real objects. A *real object* represents an actual resource. Aggregate objects do not correspond to real, physical devices. Aggregate objects provide two types of information about the real objects associated with them:
- Connectivity information for fast path to failing resource views. For more information about these views, see "NMC Locate Failing Resources Views" on page 94.
- A single DisplayStatus (also referred to as status) representation for the group of real objects based on a set of rules.

Both aggregate and real objects can exist under any class within RODM. GMFHS uses the ResourceTraits field to determine whether an object is an aggregate or real object. The ResourceTraits field is of data type INDEXLIST and can have multiple values; all values are padded to eight characters with blanks. The GMFHS, SNA topology manager, and MultiSystem Manager data models set the ResourceTraits field at the class level for both real and aggregate classes. When an aggregate object is created, the value AGG is set in the ResourceTraits field to indicate that the object is an aggregate object. Similarly, when a real object is created, the value REAL is set in the ResourceTraits field to indicate that the object is a real object. An object cannot have both values in the ResourceTraits field; that is, it cannot be both a real and an aggregate object.

In Figure 31 on page 131, objects labeled ***A*** represent aggregate objects and objects labeled ***R*** represent real objects.

The *aggregation level* of an object is the number of aggregate objects traversed in an aggregation path, including the current aggregate object. The aggregation level of real objects is always 0. For example, in Figure 31 on page 131, the aggregation level of R4 is always 0. The aggregation level of A34 is 2 on the R10→A41→A34→A22→A12 path, and it is 1 on the R9→A34→A22→A12 path. The aggregation level of A35 is always 1.

For an object in the aggregation hierarchy that has no aggregate children, an *aggregation path* defines a unique traversal of the aggregation hierarchy using the AggregationParent field. The path includes only one object at each level of the hierarchy, and continues until the current object in the path has no aggregate parents. For example, in Figure 31 on page 131, R8→A32→A21→A12 form an aggregation path. R8→A33→A22→A12 form another aggregation path that begins and ends with the same objects.

An *aggregate child* is a real or aggregate object that is linked by the AggregationChild field. This link can be either direct (also referred to as immediate) or indirect. A direct child is a real or aggregate object that is directly linked to the AggregationChild field of an object. An indirect child is a real or aggregate object that can be reached by following the chain of AggregationChild links through the aggregation hierarchy starting from the direct child of an object. For example, in Figure 31, the direct children of A21 are R3, R4, A31 and A32. An indirect child of A12 is R9. The indirect children of A22 are R8, R9, R10, R11, R12, R13, and A41.

An *aggregate parent* is an aggregate object that is linked to an object by the AggregationParent field. This link can be either direct (also referred to as immediate) or indirect. A direct parent is any aggregate object that is directly linked to the AggregationParent field of an object. An indirect parent is an aggregate object that can be reached by following the chain of AggregationParent links through the aggregation hierarchy starting from the direct parent of an object. For example, in Figure 31, direct parents of R1 are A11 and A12. The direct parent of A34 is A22. An indirect parent of R11 is A12. The indirect parents of A41 are A22 and A12.

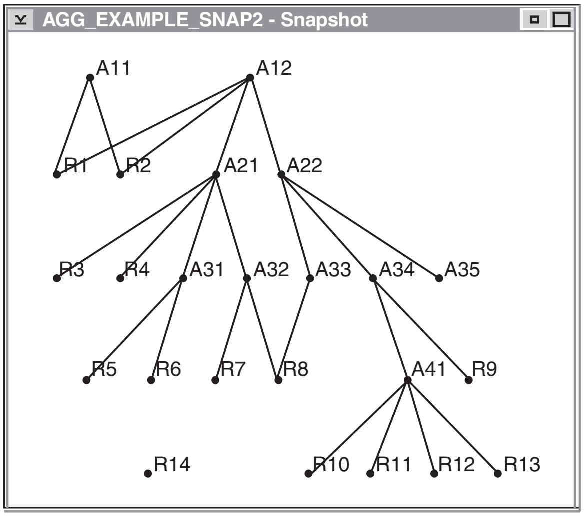


Figure 31. Aggregation Example Using Real (R) and Aggregate (A) Objects

Creating an Aggregation Hierarchy

An *aggregation hierarchy* is the topology of aggregate and underlying real objects. The aggregation hierarchy is built using the AggregationParent and AggregationChild fields of the objects.

Although real objects are part of an aggregation hierarchy, an aggregation hierarchy does not exist until at least one aggregate object is created in RODM. Figure 31 is one example of an aggregation hierarchy. An aggregation hierarchy is defined by the following rules:

- For each path in the hierarchy, the least significant child of the path can be either a real or an aggregate object. A *least significant child* is a real or aggregate object that has no aggregation children and therefore begins zero or more aggregation paths. For example, in Figure 31, R2, R7 and A35 are examples of least significant children.
- For each path in the hierarchy, the most significant parent of the path must be an aggregate object. A *most significant parent* is an aggregate object that has no

aggregation parents and therefore ends one or more aggregation paths. For example, in Figure 31 on page 131, A11 and A12 are examples of most significant parents. A real object can never be the most significant parent because a real object must have at least one aggregate parent to be considered part of the aggregation hierarchy. For example, in Figure 31 on page 131, R14 is not part of the aggregation hierarchy because it does not have an aggregate parent.

- A real object cannot be an aggregate parent.
- There is no restriction on the number of levels in an aggregation hierarchy. The number of levels in an aggregation hierarchy is equal to the number of levels in the longest aggregation path in the hierarchy.

  Note: Aggregation priority functions are restricted to 9 levels of aggregation. For more information, see "Aggregation Priority" on page 137.
- An object can be the direct child of more than one aggregate object, and an aggregate object can have more than one direct child. R1 is a direct child of both A11 and A12. R3, R4, A31 and A32 are direct children of A21.
- For GMFHS to perform aggregation correctly, there must be no aggregation hierarchy loops. An *aggregation hierarchy loop* exists when an aggregate object is a parent of itself. For example, A12 c not be a child of A33. This results in the path A12→A33→A22→A12→A33→A22..., which loops indefinitely.
- A parent-child relationship can exist between objects on more than one path. For each path, the child appears to be a unique object to the parent. For example, in Figure 31 on page 131, R8 and A12 belong to the same two aggregation paths: R8→A32→A21→A12 and R8→A33→A22→A12. From the perspective of A12, R8 is two separate real objects that have identical characteristics.
- All objects in the aggregation hierarchy need not be interconnected. For example, another subset of the aggregation hierarchy can be composed of objects that form a hierarchy similar to that shown in Figure 31 on page 131, but with no common objects between the two subsets of the hierarchy. The hierarchy subsets together form the entire aggregation hierarchy.

Building the Aggregation Hierarchy in RODM

Objects can be linked to or unlinked from the aggregation hierarchy at any time. The aggregation hierarchy is created using two RODM fields: AggregationParent and AggregationChild. For a description of these fields, see the *IBM Tivoli NetView for z/OS Data Model Reference*. The fields are of RODM type OBJECTLINKLIST. For any object, the AggregationParent field contains links to all of the direct parent objects. The AggregationChild field contains links to all of the direct child objects.

In Figure 32 on page 133, the AggregationParent field of R2 contains links to two objects, A11 and A12. The AggregationParent field of A22 contains links to one object, A12. The AggregationChild field of A22 contains links to three objects: A33, A34, and A35.

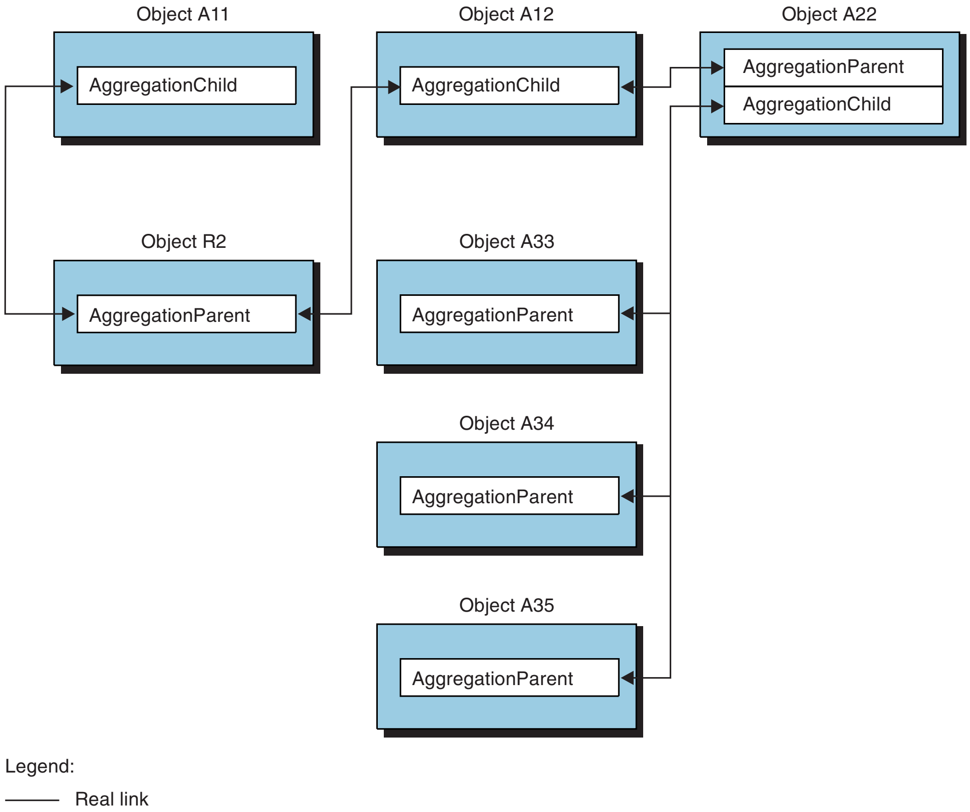


Figure 32. Links Between AggregationChild and AggregationParent Fields

For GMFHS to perform aggregation correctly, the link or unlink of the AggregationParent and AggregationChild fields of two objects must be performed by method DUIFCUAP. RODM does not prevent this operation or issue a warning if the operation is done without using the DUIFCUAP method; however, status values of all aggregate objects above the child object being linked or unlinked cannot be correctly calculated if this method is not used. Method DUIFCUAP also prevents aggregation hierarchy loops. GMFHS performs unpredictably if an aggregation hierarchy loop is introduced into the aggregation hierarchy. For more information about how to use method DUIFCUAP, see "DUIFCUAP: Update Aggregation Path Method" on page 491.

Using RODM methods and notifications, the aggregation hierarchy can be modified at any time. Whole sections of the hierarchy can be linked or unlinked. For example, in Figure 31 on page 131, A34 can be unlinked from A22 and linked to A31. This procedure has no effect on the status of A11 because the same objects still report to A11. However, the logical group of objects reporting to A21, A31, and A22 has changed as a result of the hierarchy change, and the statuses of these aggregate objects can be different. GMFHS dynamically handles these hierarchy changes when a link or unlink is done using the DUIFCUAP method.

Note: A12 can experience a temporary status change, depending on the length of time between the unlinking and relinking of A34.

Updating Status

Aggregation is performed on an aggregation hierarchy from the time that the first AggregationParent to AggregationChild link occurs to the time that the last AggregationParent from AggregationChild unlink occurs. The central purpose of aggregation is to keep the statuses of all aggregate objects in the aggregation hierarchy accurate at all times. The statuses of the aggregate objects are determined by collecting the status of all real object children under an aggregate object, and then performing a set of aggregation rules on the collected statuses using RODM fields defined on both the aggregate and real objects.

How Status Affects Aggregation

Only the statuses of real object children contribute to the status value of an aggregate parent. The statuses of child aggregate objects do not contribute to the statuses of parent aggregate objects, because these objects do not represent a real entity. For example, in Figure 31 on page 131, real object children R10, R11, R12, and R13 contribute statuses to aggregate objects A41 and A34; however, object A41 does not contribute status to aggregate object A34.

The aggregation process can be summarized as follows:

1. An event occurs that affects the status of aggregate objects in the aggregation hierarchy. See “Events That Start the Aggregation Process” on page 139. for more information.
2. Gather the statuses of all real objects that affect the aggregate objects.
3. Calculate the status of the aggregate object as described in “Using the DisplayStatus of Real Objects.”
4. Update the status of the aggregate object if it has changed.
5. Return to step 1 and wait for the next event.

Using the DisplayStatus of Real Objects

Although many RODM fields are used during the aggregation process, the DisplayStatus field is central to this process. Step 3 of the aggregation process listed under “How Status Affects Aggregation” uses the DisplayStatus field as follows:

- Counts the number of children contributing to the XCPT group.
- For each object contributing to the XCPT group, further categorizes the object into a number of status groups based on the status of the object.
- Counts the number of object children in each status group.
- Applies the aggregation rules listed in “Aggregation Rules” on page 138 to the XCPT group and status group counts to determine the status of the aggregate object.
- Updates the status of the aggregate object if it has changed.

XCPT Groups and Status Groups: Real objects can be members of the XCPT group and in zero to eight status groups, depending on their status values. These groups provide a way to prioritize and define the contribution of a real object to the status of an aggregate object. The eight different status groups are STGRP1 (Status Group 1) through STGRP8.

A real object is a member of an XCPT group, a status group, or both when the status of the real object matches one of the status values defined for the group. The status values defined for each group are customizable. For more information about defining XCPT and status group status values, see “Customizing the DisplayStatus Mapping Table for Exception Views” on page 104.

The XCPT group is used for exception view processing and aggregation processing. For aggregation processing, the status of each real object under an aggregate object is used to categorize the real object as having been in an exception (XCPT) or a non-exception (NOXCPT) state. All real objects in the XCPT state are counted in the XCPT group. For more information about the XCPT group and the status groups, see “Defining Exception View Objects and Criteria” on page 100.

Note: For a real object to be further categorized into the 8 status groups, the real object must also be counted in the XCPT group.

Example: In Figure 31 on page 131, aggregate A41 has real object children R10, R11, R12, and R13. Assume the following DUIFSMTE statements are coded in the DUIFSMT table:

```
DUIFSMTE CLASS=R10s_Class,MYNAME=R10,                              C
         XCPT=(UNSAT,INTER,DS136,DS137,DS142,DS143),               C
         STGRP1=(UNSAT,INTER),STGRP2=(DS136,DS142),                C
         STGRP6=(DS137,DS158,UNSAT)
DUIFSMTE CLASS=R11s_Class,MYNAME=R11,                              C
         XCPT=(UNSAT,LOWSA,LOWUN,DS140),                           C
         STGRP3=(LOWSA,LOWUN),STGRP5=(DS140)
DUIFSMTE CLASS=R12s_Class,MYNAME=R12,                              C
         XCPT=(INTER,LOWSA,DS154,DS158),                           C
         STGRP1=(DS158),STGRP4=(LOWSA),STGRP6=(DS154),             C
         STGRP8=(INTER,DS158)
DUIFSMTE CLASS=R13s_Class,MYNAME=R13,XCPT=(UNKWN),STGRP8=(UNKWN)
```

Figure 33. Example DUIFSMTE Statements in Table DUIFSMT

Also assume that the actual status values of the objects are:
- R10 is UNSAT
- R11 is DS140
- R12 is DS158
- R13 is UNKWN

In this example, all four resources are in an exception state and are counted in the XCPT group. R10 is a member of status groups 1 and 6; R11 is a member of status group 5; R12 is a member of status groups 1 and 8; R13 is a member of status group 8. For aggregate object A41, there are:
- Four real objects in the XCPT group.
- Two real objects in status groups 1 and 8.
- One real object in status groups 5 and 6.
- Zero real objects in status groups 2, 3, 4, and 7.

Notes:
1. For any DUIFSMTE macro definition, the status values defined for each status group sh be a subset of the status values defined for the XCPT group. An attempt to define a status group status value that is not also an XCPT group status value is not prevented; however, it has no affect on aggregation status calculations.
2. The first DUIFSMTE statement in Figure 33 has a status value of DS158 defined for STGRP6. This is enabled by the DUIFSMTE statement, but a status of DS158 is not counted toward STGRP6 because DS158 is not also in the XCPT group.
3. A status value in the XCPT group does not have to be defined as a status value in any of the status groups; a real object can contribute to the XCPT group without contributing to any of the status groups.

Suspended Resources: Real objects can be temporarily removed from the aggregation hierarchy without actually changing the AggregationParent and AggregationChild fields. This logical removal is referred to as *suspending* the object. The following techniques can be used to suspend objects:

- Using NetView management console, you can set the suspend flag of a resource from the Resource Properties window or clear suspended resources from the List of Suspended Resources window. For more information, see the NetView management console online help.
- Using RODMView, you can set the UserStatus field directly in RODM, . For more information, see the *IBM Tivoli NetView for z/OS Data Model Reference*.

Real objects can be suspended by an operator for any reason. In most cases, the object is suspended when problem resolution for the real resource represented by the object is being done. The object is said to be *resumed* when it is logically placed back into the aggregation hierarchy.

GMFHS uses the SuspendedCount field to track the number of resources that have been suspended. A real resource does not contribute status to its aggregation parents if one of the following actions occurred:

- The suspend flag of the UserStatus field is on.
- The AggregationPriorityValue field has a value of −1 (Ignore).
- The AggregationPriorityValue field has a value of −2 (Resource type default). The DefaultAggregationPriorityCopy field contains a copy of the value in the DefaultAggregationPriorityValue field of the Display_Resource_Type_Class object that is linked to the DisplayResourceType field of the real object. If the DefaultAggregationPriorityCopy field is −1 (Ignore) and the AggregationPriorityValue field is −2 (Resource type default), this resource does not participate in status calculations for aggregation.

Note: Setting the AggregationPriorityValue or DefaultAggregationPriorityValue fields to −1 (Ignore) does not affect the suspend flag of the UserStatus field. These actions are independent of each other and do not cause the other to occur.

Calculating the Aggregate Parent Status

After categorizing the status of each real object child into the XCPT group and status groups, and then counting the number of real object children in each group for a particular aggregate object, independent methods are used to calculate the status of an aggregate object. Aggregation rules are then used to resolve any conflicting status results produced by each of the methods.

Aggregation Thresholds: The status of an aggregate parent is determined based on whether the XCPT group count is above or below a threshold value. There are three threshold values defined as RODM fields on all aggregate objects. The values are listed below in order of severity: :

- ThresholdDegraded (lowest severity)
- ThresholdSeverelyDegraded
- ThresholdUnsatisfactory (highest severity)

A threshold is met if the XCPT group count for an aggregate object is greater than or equal to the threshold value. The ThresholdSeverelyDegraded value must be less than or equal to the ThresholdUnsatisfactory value, and the ThresholdDegraded value must be less than or equal to the ThresholdSeverelyDegraded value.

The valid values for these fields are described in the *IBM Tivoli NetView for z/OS Data Model Reference*. The values are as follows:

- A value of `-2` indicates that the value of the default field from the Display_Resource_Type_Class object (either DefaultThresholdDegraded, DefaultThresholdSeverelyDegraded, or DefaultThresholdUnsatisfactory) is used to define the threshold value. The default values can be -1, 0, or any positive integer. These default values substitute directly for the actual threshold values.

- A value of `-1` in the threshold field indicates that this threshold calculation is disabled for the aggregate object.
- A value of `0` in the threshold field indicates that the object always changes to the threshold status, no matter what the XCPT group count for the aggregate parent is. If more than one threshold has a 0 value, then the highest priority threshold takes effect.
- A positive number indicates that the XCPT group count must be equal to or greater than the number to cause the aggregate object to change to the threshold status value. The highest priority threshold that meets this condition is the threshold that is used to apply the status.
- A value between -100 and -200 (inclusive) in the threshold field indicates that the XCPT group count must be equal to or greater than the following value:

  (value + 100) × (total number of real objects reporting to the aggregate) × 0.01

  In effect, the value is a percentage of the total number of real objects currently attached to the aggregate object.

Aggregation Priority: Aggregation priority allows real objects to be designated as critical resources. If a critical resource contributes to the XCPT group of an aggregate parent, this constitutes an automatic match with the degraded threshold. Additional critical resources that contribute to the XCPT group has no additional effect. When the last critical resource no longer contributes to the XCPT group, the degraded threshold is no longer matched.

The AggregationPriorityValue field is defined on all real objects and it is used to define a real object as a critical resource. The valid values for this field are described in the *IBM Tivoli NetView for z/OS Data Model Reference*. Generally, the values are:

- A value of `-2` indicates that the value of the default field from the Display_Resource_Type_Class objects DefaultAggregationPriorityValue field is to be used to define the priority value. The default values can be -1, 0, or any positive number in the range of 1–9. These default values substitute directly for the actual priority values.
- A value of `-1` indicates that the real object is suspended from aggregation.
- A value of `0` indicates that the real object is not a critical resource.
- A positive number from 1 through 9 indicates that the real object is a critical resource. The number also indicates the number of levels up the aggregation hierarchy to which this object contributes its critical nature if the object does contribute to the XCPT group. The critical nature of a resource cannot be propagated more than 9 levels up the aggregation hierarchy.

Note: An aggregation hierarchy can have any number of levels. A real object is counted in the XCPT group for any aggregate at any level of the hierarchy. However, if the object is also a critical resource, the critical nature only be propagates a maximum of 9 levels above the real object. Therefore, there is a degraded threshold match for aggregate objects that are at a level less than or equal to the level specified in the AggregationPriorityValue field.

Status Group Customization: Both thresholding and priority aggregation allow the status of a parent aggregate object to be set to one of five predetermined values: Unknown, Satisfactory, Degraded, SeverelyDegraded, or Unsatisfactory.

The eight status groups are used to customize the actual state of the aggregate object. Status group customization is very similar to aggregation priority, without the 9 level limit on the aggregation hierarchy.

With status group customization, the final status of the aggregate parent can be customized to be a value other than one of the five predetermined values. All real objects that are a member of a particular status group are counted. This is done for each status group. If the number of real objects in a status group is greater than zero, the status group definitions on the aggregate object are used to determine the status of the aggregate object.

The status groups are prioritized from STGRP1 (highest) to STGRP8 (lowest). If more than one status group has a count greater than zero, and there is more than one matching status group definition for the aggregate object, then the first status value in the highest priority status group definition for the aggregate object is used as the status for the aggregate object.

Unknown Resources: The status values of real object children can contribute directly to the status values of aggregate parents without necessarily contributing to the XCPT group. The total number of real objects with Unknown statuses under an aggregate parent is compared to the value in the UnknownThreshold field of the Global_Aggregation_Parameters_Class. If this threshold is equaled or exceeded, then further aggregation processing for this aggregate parent is not valid and the status of the aggregate parent becomes Unknown.

Unlike the three thresholds defined under "Aggregation Thresholds" on page 136, this threshold is a number from 1 through 100 that represents a percentage. The percentage is applied to the total number of real children objects under the aggregate parent that are actively participating in aggregation (not suspended).

Aggregation Rules: Suspended resources, unknown resources, aggregation thresholds, aggregation priority, and status group customization are used to calculate the status of an aggregate object. The following aggregation rules are used in the order listed to resolve conflicts among the aggregation methods:

1. Logically remove suspended real object children from the aggregation hierarchy. This was already done by not allowing suspended real objects to be counted in the XCPT and status groups, but the total count of all objects reporting to an aggregate parent is now changed to reflect the removal of the suspended resources.
2. If the total number of real object children is now zero, *or* if there is no DisplayResourceType object currently linked to the aggregate parent and a default threshold from this object is needed, the status of the aggregate object is set to Unknown and the status calculation ends.
3. If the percentage of real object children with an Unknown status is greater than the UnknownThreshold, the status of the aggregate object is set to Unknown and the status calculation ends.
4. If there is a status group customization match with the aggregate object, the aggregate object takes on the first status defined in the highest matching status group of the aggregate object. The status calculation ends.
5. If the number of real object children in the XCPT group is greater than or equal to the Unsatisfactory threshold, the status of the aggregate object becomes

Unsatisfactory and the status calculation ends. The Unsatisfactory threshold can be expressed as an absolute count or as a percentage.

6. If the number of real object children in an XCPT group is greater than or equal to the SeverelyDegraded threshold, the status of the aggregate object becomes SeverelyDegraded and the status calculation ends. The ServerelyDegraded threshold can be expressed as an absolute count or as a percentage.
7. If the number of real object children in an XCPT group is greater than or equal to the Degraded threshold, the status of the aggregate object becomes Degraded and the status calculation ends. The Degraded threshold can be expressed as an absolute count or as a percentage.
8. If the number of real object children counted in the XCPT group that are critical resources is greater than zero, the status of the aggregate object becomes Degraded and the status calculation ends. Remember that the AggregationPriorityValue field for any real object child might not allow it to be counted as a critical resource for the current level of aggregate object.
9. If none of the previous conditions apply, the status of the aggregate object becomes Satisfactory and the status calculation ends.

Aggregation Problems

Aggregation is accomplished using various RODM fields. Some of these fields can be modified by the customer, and some are for GMFHS method use only. Although a customer sh never modify a field that is for GMFHS method use only, RODM does not prevent this from happening.

Inconsistencies can arise when:

- Internal counts are not equal for each aggregate object.
- Threshold values are greater than the total number of real object children of an aggregate parent, or threshold values that do not follow the restrictions defined in “Aggregation Thresholds” on page 136

An indicator in the UserStatus field is used to indicate possible inconsistencies during aggregation processing.

UserStatus Field

The UserStatus field on an aggregate object contains information used to set the operator status of the object in a view. There are five bits in the UserStatus field that contribute to the operator status of an aggregate object:

- The resource marked bit
- The threshold inconsistency bit (set as a result of aggregation problems described above)
- The suspended bit
- The resume bit
- The suspend resources under aggregate bit

The resource marked, suspended, resume, and suspend resources under aggregate bits are set as a result of an operator action or by setting the UserStatus field directly in RODM (using RODMView for example). The threshold inconsistency bit is set during the aggregation process if an inconsistency is detected.

Events That Start the Aggregation Process

A number of events can start the aggregation process. In general, aggregation is triggered based on a change to one of the RODM fields used for the aggregation process. For example, a link is made using the AggregationParent and AggregationChild field of two objects, or a DisplayStatus change occurs for a real object in the aggregation hierarchy. The following topics describe each of the events that trigger the aggregation process.

Changing the DisplayStatus of a Real Object: This is the most common event that triggers the aggregation process. The DisplayStatus value of a real object can change for a variety of reasons, such as a status change request from a NetView management console or a NetView alert. Any time the status of a real object that is a member of the aggregation hierarchy changes, the status of all aggregate parents of that real object might also need to be changed.

If the real object was suspended with the automatic resume feature and the status of the object is now Satisfactory, the object is logically relinked to the aggregation hierarchy and aggregation for the object is resumed.

If there is no change in the contribution of an object to the XCPT group or a status group, and the object does not change to or from Unknown status, then there is no change to the aggregate parent status.

Linking and Unlinking Using Method DUIFCUAP: The AggregationParent and AggregationChild fields of the child object and parent object passed to the DUIFCUAP method are updated. Although a link or unlink operation involves only two objects (the child object and the parent object), the action can affect the status values of many aggregate objects in the aggregation hierarchy.

After a link or unlink operation, the status of the immediate parent aggregate object and all parent objects of the immediate parent aggregate object can need to be changed.

Changing the AggregationPriorityValue: If the AggregationPriorityValue of a real object is changed, then the status of all aggregate parents of the real object might need to be changed. If the real object is not counted in the XCPT group for the aggregate parent object, there is no change to the aggregate parent status. The following techniques can be used to change the value of the AggregationPriorityValue field:

- Use the NetView management console workstation. For more information, see in the *IBM Tivoli NetView for z/OS User's Guide: NetView Management Console*.
- Use NetView management console. For more information, see the NetView management console online help.
- Set the AggregationPriorityValue field directly in RODM (using RODMView, for example). For more information, see the *IBM Tivoli NetView for z/OS Data Model Reference*.

Changing an Aggregate Object Threshold: If any of these thresholds are changed, the status of that specific aggregate object might need to be changed. The following techniques can be used to change the value of the ThresholdDegraded, ThresholdSeverelyDegraded, and ThresholdUnsatisfactory fields:

- Use NetView management console. For more information, see the NetView management console online help.
- Set the fields directly in RODM (using RODMView, for example). For more information, see the *IBM Tivoli NetView for z/OS Data Model Reference*.

Changing the Unknown Threshold: If this threshold is changed, the status of all aggregate objects in the aggregation hierarchy might need to be changed. Two techniques can be used to change the value of the UnknownThreshold field of the Global_Aggregation_Parameters_Class:

- By setting the UnknownThreshold field directly in RODM (using RODMView for example). For more information, see the *IBM Tivoli NetView for z/OS Data Model Reference*.

Note: You cannot use NetView management console to change the value of the UnknownThreshold field.

Suspending a Real Object: If a resource is suspended, the status of all aggregate parents of that real object might need to be changed. A real object can be suspended from participating in aggregation at the workstation. The following techniques can be used to suspend a real object from participating in aggregation:

- Use NetView management console. For more information, see the NetView management console online help.
- Set the UserStatus field directly in RODM (using RODMView, for example). For more information, see *IBM Tivoli NetView for z/OS Data Model Reference*.

Changing Resource Type Defaults: The AggregationPriorityValue field for a real object can indicate that the value of the DefaultAggregationPriorityValue field from the Display_Resource_Type_Class object linked to the real object sh be used for priority aggregation. The ThresholdDegraded, ThresholdSeverelyDegraded, and ThresholdUnsatisfactory fields for aggregate objects can indicate that the value of the default fields from the Display_Resource_Type_Class object linked to the aggregate object sh be used for threshold aggregation.

For a real or aggregate object using these defaults, the effect is the same as if the priority value or threshold field directly on the object had changed. The primary difference is that multiple real or aggregate objects can be changed because a Display_Resource_Type_Class object can be linked to multiple objects.

The following techniques can be used to change the value of the ThresholdDegraded, ThresholdSeverelyDegraded, and ThresholdUnsatisfactory fields:

- Use NetView management console. For more information, see the NetView management console online help.
- Set the field directly in RODM (using RODMView, for example). For more information, see the *IBM Tivoli NetView for z/OS Data Model Reference*.

Linking and Unlinking Using Method DUIFCLRT: Method DUIFCLRT is used to associate a real or aggregate object with an object of the Display_Resource_Type_Class. For real objects, this can affect the priority aggregation value of the object if the default value from the Display_Resource_Type_Class object is being used. For aggregate objects, this can affect any of the Degraded, SeverelyDegraded, or Unsatisfactory thresholds of the object if the default value from the Display_Resource_Type_Class object is being used.

For a real or aggregate object using any of these defaults, the effect is the same as if the priority value or threshold field directly on the object had changed.

Changing the Status Mapping Table: The status mapping table can be dynamically updated using sample CNMSJH13. Because the definition of the XCPT group or any of the eight status groups can change, this sample optionally allows the DisplayStatus value of each real object in RODM to be updated (changed to the same value that it currently has) to trigger exception view and aggregation status recalculations.

Aggregation Methods

"GMFHS Methods" on page 488 provides a list of GMFHS methods. Each of the methods that are described, beginning with DUIFCLRT, contribute at least indirectly to aggregation. Three of these methods, DUIFCUAP, DUIFFAWS, and DUIFFRAS contribute directly to aggregation.

Methods DUIFFAWS and DUIFFIRS are used to synchronize the aggregation hierarchy if the UserStatus field of an object indicates that there is a threshold inconsistency, or any time that an operator decides that the status of aggregate objects might be incorrect. DUIFFRAS performs a subset of the function performed by DUIFFAWS. DUIFFRAS causes the status of each aggregate object to be recalculated based on the existing XCPT group and status group counts for each aggregate object. DUIFFAWS extends DUIFFRAS by accumulating all of the XCPT group and status group counts for each aggregate object before recalculating the status of the aggregate object.

See "GMFHS Methods" on page 488 for a description of these methods.

Status Groups

The status (the value of the DisplayStatus field) of an aggregate object can be customized based on the status of real object children under the aggregate.

The sample table DUIFSMT described in "Defining Exception Criteria" on page 101 is used for this purpose. The STGRP*n* keywords (where *n* = 1 through 8) of the DUIFSMTE macro are used to map the status of real children objects to the desired status of the aggregate parent. For more information about the DUIFSMTE macro and how to refresh the DUIFSMT table, see "Customizing the DisplayStatus Mapping Table for Exception Views" on page 104.

The STGRPn keywords are used to group DisplayStatus values in the same way that the XCPT keyword is used for exception views. The groups are organized in a priority manner, with STGRP1 being the highest priority group and STGRP8 being the lowest. The same status value can belong to more than one status group; in effect, all status values can be placed in every status group. The DisplayStatus value must also be an XCPT value for it to register as a STGRP*n* keyword.

Status groups are used to map the status of a real object to the status of any parent aggregate objects. If a real object changes to a status value that is in any of the status groups, then the corresponding status group for all parent aggregate objects are used to determine the status value of the aggregate objects. If the real object status value is listed in more than one group, then the highest priority group that contains the status value is used.

The exception state of the real object is used to determine the status of any aggregate parents under the following conditions:

- The real object has no status groups, or the status value of the real object is not contained in any status group.
- The matching status group for the parent aggregate object is not defined.

Using Status Groups

The following list contains additional operational characteristics of performing aggregation using status groups:

- A status group match for an aggregate parent overrides the previous status of that parent. The status group override remains in effect until *either*:

1. A higher priority status group match occurs for the aggregate parent.
2. The status value of the last real object that is contributing to the current highest priority status group for the aggregate parent no longer matches that status group, or the real object is unlinked from the hierarchy or is suspended from aggregation.

- A status group match overrides the status value of an aggregate parent at any level of the aggregation hierarchy; there is no level limit as there is with aggregation priority values.
- As with exception based aggregation, suspended objects do not participate in status group aggregation.
- The aggregate object threshold for the Unknown status of real objects is not overridden by status group aggregation.

Examples of Customizing Aggregate DisplayStatus

The following example is provided to give an understanding of using status groups to customize the DisplayStatus value of an aggregate object. For the example, assume the following conditions:

- All objects of the T4NODE class contribute to exception state aggregation with a DisplayStatus of unsatisfactory or unknown. If the DisplayStatus is unsatisfactory, it is tagged to status group 1.
- All objects of the 1.3.18.0.0.1821 class contribute to exception state aggregation with a DisplayStatus of unsatisfactory, intermediate, or unknown. If the DisplayStatus is intermediate or unknown, it is tagged to status group 2.
- All aggregate objects have a status group match for status groups 1 and 2. An object of the T4NODE class with an unsatisfactory status results in the status of any aggregate parent to be DS136. An object of the 1.3.18.0.0.1821 class that has either an unsatisfactory or an intermediate status results in the status of an aggregate parent to be DS137, as long as this status is not overridden by a status group 1 match.
- Any object not in one of the three previously defined classes contributes to exception state aggregation with a DisplayStatus of unsatisfactory or medium unsatisfactory. If the DisplayStatus is UNSAT, it is tagged to status group 3. Because there is no matching status group 3 definition on any aggregate object, a real object DisplayStatus of UNSAT never causes a status group 3 override on an aggregate parent.

Using the previously listed conditions, Figure 34 shows the coding of the DisplayStatus mapping table. The fourth statement sets the defaults.

```
DUIFSMTE CLASS=T4NODE,XCPT=(UNSAT,UNKWN),STGRP1=(UNSAT)
DUIFSMTE CLASS=1.3.18.0.0.1821,XCPT=(UNSAT,INTER,UNKWN),       C
         STGRP2=(INTER,UNKWN)
DUIFSMTE CLASS=GMFHS_Aggregate_Objects_Class,XCPT=(SDGRD),     C
         STGRP1=(DS136),STGRP2=(DS137)
DUIFSMTE CLASS=ALL,XCPT=(UNSAT,MEDUN),STGRP3=(UNSAT)
```

Figure 34. Example of Customizing Aggregate Display Status

Using the Collection Definition Objects

Collection definition objects are used by the GMFHS RODM Collection Manager function to define the contents of Network_View_Class and GMFHS_Aggregate_Objects_Class objects. Collection definition objects are created in either the Network_View_Collection_Class or the Aggregate_Collection_Class.

Each of these classes are subclasses of the Collection_Definition_Class. Objects must not be created on the Collection_Definition_Class.

The Network_View_Class and GMFHS_Aggregate_Objects_Class objects defined by the collection definition objects are called collection creation objects. Collection creation objects are created by the GMFHS RODM Collection Manager function from the information in a collection definition object. The RODM Collection Manager continuously watches for new collection definition objects to be created or deleted in RODM. It creates a corresponding collection creation object dynamically. In addition, changes to the resource collection on an existing collection definition object are monitored continuously. The changes are dynamically reflected to the corresponding collection creation object.

Collection Definition Objects

Fields on a collection definition object specify the following information:

- The RODM MyName of the collection creation object.
- If a Network_View_Collection_Class object, the Annotation of the Network_View_Class collection creation object.
- If an Aggregate_Collection_Class object, the DisplayResourceUserData of the GMFHS_Aggregate_Objects_Class collection creation object.
- If an Aggregate_Collection_Class object, the DisplayResourceName of the GMFHS_Aggregate_Objects_Class collection creation object.
- If an Aggregate_Collection_Class object, the DisplayResourceType of the GMFHS_Aggregate_Objects_Class collection creation object.
- If an Aggregate_Collection_Class object, the DisplayResourceOtherData of the GMFHS_Aggregate_Objects_Class collection creation object.
- If an Aggregate_Collection_Class object, the DegradedThreshold of the GMFHS_Aggregate_Objects_Class collection creation object.
- If an Aggregate_Collection_Class object, the SeverelyDegradedThreshold of the GMFHS_Aggregate_Objects_Class collection creation object.
- If an Aggregate_Collection_Class object, the UnsatisfactoryThreshold of the GMFHS_Aggregate_Objects_Class collection creation object.
- The LayoutType of the Network_View_Class of GMFHS_Aggregate_Objects_Class collection creation object.
- If an Aggregate_Collection_Class object, request-specific flags that are used to process the aggregate collection.
- A data field that holds information that is interpreted by the NetView management console.
- A logic tree of rules that an object must pass to be included in the Network_View_Class or GMFHS_Aggregate_Objects_Class collection creation object.

Collection Definition Object Fields

See the *IBM Tivoli NetView for z/OS Data Model Reference* for complete information about the collection definition object classes and fields.

Most of the fields on the collection definition object are copied directly to the field of the same name on the collection creation object. Some of the fields, such as the RequestFlags, CollectionLocateName, and WizardHints field, are used only by the RODM Collection Manager. They are not used to supply a value to a field on the collection creation object.

Some of the collection definition object fields are used to indirectly supply a value to a field on the collection creation object. The LayoutType field, when specified on an Aggregate_Collection_Class object, is converted to a character string and appended to the string "RCMLayoutParmViewType". This concatenated string is used as the name of a Layout_Parameters_For_View_Class object. This object is then linked to the DetailViewLayoutForSelectedResource field of the collection creation object.

In a similar way, the DisplayResourceType field is used as the name of a Display_Resource_Type_Class object. This object is then linked to the DisplayResourceType field of the collection creation object. The CollectionSpec*n* fields are used to populate the ContainsObjects field of a Network_View_Class collection creation object and the AggregationChild and IsPartOf fields of a GMFHS_Aggregate_Objects_Class collection creation object. See "Using Collection Specifications" for more information about the usage of these fields.

If the collection creation object already exists in RODM, it is deleted and recreated using the information in the collection definition object. Name your collection creation object objects carefully to ensure that they do not overwrite existing Network_View_Class or GMFHS_Aggregate_Object_Class objects. Adding a prefix or suffix to the collection creation object name that identifies it as an object that was created by the RODM Collection Manager is an easy way to prevent creating a duplicate collection creation object.

Using Collection Specifications

The collection specification is contained in the CollectionSpec*n* fields of the collection definition object. These fields are concatenated together in ascending numerical order of the *n* numeric portion of the field to create the full collection specification. The first CollectionSpec*n* field must be CollectionSpec1. A collection specification contains a set of rules that describe the objects to be in the Network_View_Class collection creation object ContainsObjects field or the GMFHS_Aggregate_Objects_Class AggregationChild and IsPartOf fields.

The rules in the collection specification are applied dynamically. The rules match objects that currently exist in RODM at the time the rules are initially processed by the RODM Collection Manager function as well as objects that are dynamically added to or deleted from RODM after the rules are initially processed. The RODM Collection Manager places a RODM notification on all fields in all classes that are specified in any collection specification for any collection definition object and is then notified when the value of these fields change for any object. As a result, the RODM Collection Manager can update the objects in a collection creation object whenever a change occurs in RODM that affects the collection creation object.

Conditional Statements

Conditional statements are logically joined together and are a part of a collection specification.. Each conditional statement is composed of a RODM field, a RODM class, a value (or optionally), a set of values, and an operation. For each object within the specified class, the specified field is compared to the value or list-of-values using the operation. If the operation compares successfully, then the object matches the condition. Otherwise, it fails the condition. The list of all objects that compare successfully with the condition are the result of the conditional statement. These objects are of RODM type ObjectList.

The simplest form of a collection specification is a single conditional statement, and can be expressed in the following general terms:

```
{Class/Field} operation {Value}   ==>  list_of_objects
```

For each object in the given Class, take the Field value of the object and compare it to Value using the comparison operation. If the values compare successfully, place the object in the output *list_of_objects*.

The {Value} term can also be a reference to a set of values, much like the {Class/Field} term indirectly references all objects on the Class. Each value is listed directly in the collection specification. When more than one value is listed in the {Value} term, the Field value of an object is compared against each value in the value list. One or more of the values in the value list must compare successfully for the object to be added to the *list_of_objects*.

The single conditional statement can also be expressed in the following terms:

```
{Value1} operation {Value2}  ==>  list_of_objects
```

Where both Value1 and Value2 can be either a single value or a value list. Value1 refers to the value of the Field on each object in the Class. Value2 refers to the list of values directly specified in the conditional statement. This generic syntax is useful when complex conditional statements are described.

In the case of the simple collection specification, the *list_of_objects* that results becomes the object list for either the ContainsObjects field or the AggregationChild/IsPartOf fields of the collection creation object. In effect, this *list_of_objects* is the final output from the collection specification

Postfix Notation in Conditional Statements

When a postfix notation is used to express the conditional statement, the statement is:

```
{Class/Field} {Value} operation ==> list_of_objects
```

or

```
{Value1} {Value2} operation   ==>  list_of_objects
```

Postfix notation is the notation used in the actual collection specification on the collection definition object in RODM.

For example, a simple collection specification is as follows:

```
 |GMFHS_Managed_Real_Objects_Class|DisplayStatus|132|.EQ.
```

This collection specification takes the value of the DisplayStatus field for each object in the GMFHS_Managed_Real_Objects_Class and compares it to 132. If the values are equal, the object is added to the *list_of_objects* that satisfy the conditional statement. After all objects have been compared, the *list_of_objects* is put into the object list field of the collection creation object.

The conditional statement is also referred to as a leaf specification. A leaf specification produces a *list_of_objects* from a comparison of two lists of values. It is a leaf in the processing tree that a collection specification represents conceptually. It is a leaf because its Value1 and Value2 operators are not produced by other conditional statement evaluations from the collection specification, but instead come directly from either the collection specification (Value2) or from a field on an object (Value1).

Complex Conditional Statements

Most collection specifications are not composed of only one conditional statement. For an object to be considered a candidate for a network view, for example, you can have its DisplayStatus be 132 AND its MyName be Chihuahua. In this case, the conjunction AND is used to link the two conditional statements together:

The syntax for linking conditional statements together in postfix notation is:

```
( {Class/Field}  operation {Value} )    ( {Class/Field}  operation
{Value} ) conjunction  ==>  list_of_objects
```

or

```
 (leaf_specification) (leaf_specification) conjunction  ==>  list_of_objects
```

Both leaf specifications produce an object list even if the list contains no objects; the final *list_of_objects* is determined by applying the conjunction operator (AND or OR) to the two object lists. If the conjunction is AND, then the object identifier must be in both lists for it to be in the resulting *list_of_objects*. If the conjunction is OR, then the object identifier must be in one or the other list for it to be in the resulting *list_of_objects*.

Since a leaf specification evaluates to a *list_of_objects*, the generic form of the above syntax is:

```
(list_of_objects)(list_of_objects) conjunction  ==>  list_of_objects
```

This syntax is also referred to as a node specification. A node specification uses the output from other conditional statements (object lists) as the operands of the conjunction. Since a node specification itself is a conditional statement that produces an object list, an unlimited complex conditional can be built by recursively substituting node specifications in the simple node specification as described here.

For example, consider the following complex conditional in postfix notation:

```
(a) (b) EQ (c) (d) EQ AND (e) (f) EQ (g) (h) EQ AND OR
```

To continue this example, we build it up to the generic form of a complex conditional. First, (a) (b) EQ is a leaf specification:

```
(leaf_specification) (c) (d) EQ AND (e) (f) EQ (g) (h) EQ AND OR
```

Next, (c) (d) EQ is also a leaf specification:

```
(leaf_specification) (leaf_specification) AND (e) (f) EQ (g) (h) EQ AND OR
```

or

```
(list_of_objects) (list_of_objects) AND (e) (f) EQ (g) (h) EQ AND OR
```

Next, (list_of_objects) (list_of_objects) AND is in the form of a node specification:

```
(node_specification) (e) (f) EQ (g) (h) EQ AND OR
```

or

```
(list_of_objects) (e) (f) EQ (g) (h) EQ AND OR
```

Next, (e) (f) EQ (g) (h) EQ is identical to (a) (b) EQ (c) (d) EQ:

```
(list_of_objects) (leaf_specification) (leaf_specification) AND OR
```

Evaluating the complex conditional that involves the leaf specifications, we have:

```
(list_of_objects) (node_specification) OR
```

or

```
(list_of_objects) (list_of_objects) OR
```

This final conditional matches the generic syntax described here, and produces the final object list for the complex conditional. See "Stack Model Postfix Processing" for more information about the method used to actually evaluate the postfix notation used in a collection specification.

Stack Model Postfix Processing

A collection specification is processed by using a virtual stack to hold the intermediate results from the conditional statements in the collection specification. Any output from a leaf specification, which is an object list, is added to the stack. When a conjunction is encountered in the collection specification, the last two object lists added to the stack are removed from the stack, the conjunction is applied to the object lists, and the resulting object list is added to the stack. This processing continues, left to right, to the end of the collection specification. At the end of the collection specification, there sh be one and only one object list left on the stack. If this is not the case, the collection specification is syntactically incorrect. The object list left on the stack is the final result of the collection specification. It is assigned directly to the ContainsObjects or AggregationChild/IsPartOf fields of the collection creation object.

Although leaf specifications are processed using the postfix notation, the input to the operator (Value1 and Value2) are not object lists. The stack only contains object lists. Therefore, leaf specifications are evaluated without using the stack. Their output, which is a list of objects, is added to the stack.

The following example shows the stack operations that occur while evaluating the example on page on page 147:

```
(a) (b) EQ (c) (d) EQ AND (e) (f) EQ (g) (h) EQ AND OR
```

Initially, the stack is empty. Reading the collection specification from left to right, the leaf specification (a) (b) EQ is evaluated to the object list *a_b_objects* and added to the stack. The result is:

Stack contains:	*a_b_objects*
Remaining specification:	(c) (d) *EQ AND* (e) (f) *EQ* (g) (h) *EQ AND OR*

Since (c) is not a conjunction, what follows must be another leaf specification; anything other than a conjunction or a valid leaf specification is syntactically incorrect. (c) (d) EQ is evaluated to the object list *c_d_objects* and added to the stack. The result is:

Stack contains:	*c_d_objects*
	a_b_objects
Remaining specification:	*AND* (e) (f) *EQ* (g) (h) *EQ AND OR*

AND is a conjunction, so the first two object lists on the stack (in this case, the only two), are removed, then evaluated using the conjunction, and the result is added to the stack. It is an error if the stack does not contain two or more object lists when a conjunction is evaluated. The result is:

Stack contains:	*a_b_AND_ c_d_objects*
Remaining specification:	(e) (f) *EQ* (g) (h) *EQ AND OR*

Because (e) is not a conjunction, what follows is another leaf specification. (e) (f) EQ is evaluated to the object list e_f_objects and is added to the stack. The result is:

Stack contains:	*e_f_objects*
	a_b_AND_ c_d_objects
Remaining specifications:	(g) (h) *EQ AND OR*

Because (g) is not a conjunction, what follows is another leaf specification. (g) (h) EQ is evaluated to the object list g_h_objects and is added to the stack. The result is:

Stack contains:	*g_h_objects*
	e_f_objects
	a_b_AND_c_d_objects
Remaining specifications	*AND OR*

AND is a conjunction, so the first two object lists on the stack are removed, evaluated using the conjunction, and the result is added to the stack. The result is:

Stack contains:	*e_f_AND_g_h_objects*
	a_b_objects AND c_d_objects
Remaining specifications:	*OR*

Finally, OR is a conjunction, so the last two object lists on the stack are removed, evaluated using the conjunction, and the result is added to the stack. The result is:

Stack contains:	*a_b_AND c_d_objects_OR_e_f_AND_g_h_objects)*
Remaining specifications:	

At this point, there sh be only one object list on the stack (there is) and nothing left in the collection specification. If either of these is not true, the collection specification was syntactically incorrect. The final object list is the result of the collection specification, and is copied to the collection creation object.

Collection Specification Syntax

The syntax for the collection specification field is:

```
<collection_specification> :: <separator><leaf_specification>  -or-
                                 <separator><node_specification>

<node_specification> ::
   <leaf_specification><separator><leaf_specification><separator>
     <conjunction>  -or-
<leaf_specification><separator><node_specification><separator>
     <conjunction>  -or-
<node_specification><separator><leaf_specification><separator><conjunction>
                     -or-
<node_specification><separator><node_specification><separator><conjunction>
```

```
<leaf_specification> ::
   <class_name><separator><field_name><separator><value_list>
   <separator><operator>

<value_list> ::
   <value>  -or-
   <value><separator><value_list>

<class_name> ::
      string of characters, maximum of 64, specifying a RODM Class, e.g.
NMG_Class

<field_name> ::
      string of characters, maximum of 64, specifying a RODM Field, e.g. MyName

<value> ::
      string of characters, specifying the value of a RODM Field, e.g. CNM01AGT

<separator> ::
      a single character; can be any character value, e.g. |

<operator> ::  .EQ. (equal)                                              -or-

               .NE. (not equal)                                          -or-

               .LT. (less than)                                          -or-

               .GT. (greater than)                                       -or-

               .LE. (less than or equal to)                              -or-

               .GE. (greater than or equal to)                           -or-

               .CONTAINS. (contains at least one of)                     -or-

               .CONTAINS=. (contains at least one of, sensitive to case)  -or-

               .NCONTAINS. (does not contain)                            -or-

               .NCONTAINS=. (does not contain, sensitive to case)

<conjunction> :: .AND.  -or-

                 .OR.
```

The character that separates the individual tokens in the collection specification is defined as a part of the collection specification. <separator> can be any character. This character is allowed to be user defined because any selected value can possibly be included in a <class_name>, <field_name>, or <value>. In the NetView management console interface, the vertical bar (|) is used as the default separator character.

Collection Specification Values

The {Value} portion of a leaf specification can be thought of as a pattern. A pattern is a sequence of characters, some of which have special meanings, that is matched against a specific value or set of values. The special characters allow a pattern to describe more than one value. A pattern with no special characters describes only one value, the value that is composed of exactly the characters in the pattern. A pattern with special characters is similar to a list of values, where the list of values is composed of all of the unique values that match the pattern. If {Value} is a list of values, each of the values within the list can be a pattern with special characters.

These patterns can be expressed using DOS wildcards or regular expressions. A regular expression is a set of characters and operators that define a string or group of strings in a search pattern. Regular expressions also contain metacharacters, which are characters with special meanings. The default notation for patterns is to use DOS wildcards. If the pattern uses regular expressions, the first character of the pattern must be the backslash (\). If the pattern does not use any of the special characters (in either DOS or regular expression notation), the pattern resolves to single unique value for the comparison operation.

If you want to use DOS wildcards and the first character of the DOS wildcard is a backslash (\), then you must escape it with a plus sign (+). That is, `+\value` is interpreted as a DOS wildcard value of `\value`. Also, if you want to use a DOS wildcard and the first character of the DOS wildcard is a plus sign, then you must escape that with another plus sign. Again, `++value` is interpreted as a DOS wildcarded value of `+value`. The plus sign as an escape character is effective only as the first character of the value, and only when followed by another plus sign or backslash.

The special characters for DOS patterns are an asterisk (*) and the question mark (?). An asterisk matches zero or more characters from where the asterisk is in the pattern. A question mark matches any one character in the pattern. Special characters for DOS patterns can be used anywhere in a pattern. The pattern `*re?*om*` matches any string that has an `re` that is preceded by zero or more other characters, at least one character after the `re`, then zero or more characters until `om`, followed by zero or more characters to the end of the string.

A pattern using DOS wildcard characters must always match the entire string that it is being compared with. In this example, if the pattern was `re?*om` without the preceding and ending asterisks, then the matched string must begin with `re` and end with `om`. This is slightly different from the way regular expressions work.

Regular expressions are used for more complex pattern matching. DOS patterns in a collection specification are converted to regular expressions by the RODM Collection Manager prior to matching the pattern against a value; all pattern matching is done by the RODM Collection Manager using regular expressions. The regular expression pattern is applied to the substrings of the input string; if it matches a substring, then the pattern is considered to have matched the entire input string. Because regular expressions match on a substring of the input string, the caret (^) metacharacter is added to the beginning of any converted DOS wildcard pattern, and the dollar sign ($) metacharacter is added to the end of the same converted DOS wildcard pattern in order to enforce the DOS wildcard constraint of matching the entire string.

The simplest form of regular expression is a string of characters with no special meaning. The following characters have special meaning; they are used to form extended regular expressions:

. (period)
The period symbol matches any one character except the terminal new-line character.

[string]
A string within square brackets specifies any of the characters in the string. Thus [abc], if compared to other strings, matches any that contains a, b, or c. If the string within the square brackets contains a character, followed by a hyphen, followed by another character, it indicates that all of the characters in the current collating sequence between the two intervening

characters are considered a part of the string. For example, [a-z] can be equivalent to [abc...xyz] or, with a different collating sequence, it can be equivalent to [aAbBcC...xXyYzZ]. If the string within the square brackets begins with the caret (^) symbol, it negates the characters within the square brackets. Thus [^abc], if compared to other strings, fails to match any that contains even one a, b, or c.

expression[m] or expression[m,] or expression[m,u]

Integer values enclosed in [ ] indicate the number of times to apply the preceding regular expression. The value for m is the minimum number, and u is the maximum number. The value for u must be less than 256. If you specify m, it indicates the exact number of times to apply the regular expression. [m,] is equivalent to [m,u], where u is an unbounded upper limit. They both match m or more occurrences of the expression. The plus sign (+) and asterisk (*) operations are equivalent to [1,] and [0,] respectively.

expression* (asterisk)

The asterisk symbol indicates zero or more of any characters. For example, a*e is equivalent to any of the following strings: 99ae9, aaaaae, a999e99.

$ (dollar symbol)

The dollar symbol matches the end of the string.

^ (caret)

The caret symbol matches the beginning of the string.

\ (backslash)

The backslash character turns off the special meaning of any character following the backslash, thereby forcing the character to be intepreted as itself in the pattern. For example, \. matches the . character, not a \ followed by any character.

expression+ (plus)

The plus sign specifies one or more occurrences of a character. Thus, smith+ern is equivalent to, for example, smithhhern.

(expression)

Groups a subexpression allowing an operator, such as *, +, or [ ], to work on the subexpression enclosed in parentheses. For example, (a*(cb+)*) matches any string that contained zero or more occurrences of `a`, followed by zero or more occurrences of the pattern `c` followed by one or more occurrences of `b`.

The asterisk (*) character in a DOS pattern becomes a period asterisk (.*) in a regular expression. The question mark (?) characters in a DOS pattern becomes a period (.) in a regular expression.

All DOS patterns are prepended with a caret (^) (which matches the beginning of a string), and appended with a dollar sign ($) (which matches the end of a string) when they are converted into a regular expression by the RODM Collection Manager. This forces the entire string to be matched, character for character.

For example, the pattern `*IS?R*` is a DOS pattern that matches the following strings:
- BISTRO
- MISERLY
- MISER

However, it does not match these strings:

- MISTER
- DISRUPT

The same pattern expressed as a regular expression is `\.*IS.R.*`

The pattern `\RE[AGLRU]+.E[^A-O]+.*ON` is a regular expression that matches the following strings:

- REGULAR EXPRESSION
- REGAL-EXPATRIATION

However, it does not match these strings:

- REGULATION
- REGENERATION

Values and Data Types

A {Value} in a leaf specification is always initially interpreted as a character string. The {Class/Field} that the {Value} is compared with can be one of a number of actual data types. If necessary, {Value} (each value, in the case of a list of values) is converted to the appropriate data type before the comparison is done. In general, only character data types can be expressed using DOS wildcards or regular expressions. Special characters for pattern matching are interpreted as the literal character if found in a {Value} that is to be matched against other data types.

Not all RODM data types are allowed for a {Class/Field} element of a leaf specification. The following table lists each of the RODM data types, indicates whether the data type is allowed in a leaf specification, indicates whether DOS wildcards or regular expressions are allowed for the data type, and shows how data is converted from a character string to match the data type.

RODM Data Type	**Allowed in Leaf Specification**	**Allows Wildcards /Regular Expressions**	**Conversion**
ANONYMOUS	No	N/A	N/A
ANONYMOUSVAR	Yes	No	{Value} contains only the characters '0' or '1', which are converted to an actual bitstring before the comparison.
APPLICATIONID	No	N/A	N/A
BERVAR	Yes	No	{Value} contains only the characters '0' or '1', which are converted to an actual bitstring before the comparison.
CHARVAR	Yes	Yes	None (treated as a character string)
CHARAVARADDR	No	N/A	N/A
CLASSID	No	No	None (treated as a character string)
CLASSIDLIST	No	N/A	N/A

RODM Data Type	Allowed in Leaf Specification	Allows Wildcards /Regular Expressions	Conversion
CLASSLINKLIST	No	N/A	N/A
ECBADDRESS	No	N/A	N/A
FIELDID	Yes	No	{Value} is converted to an integer. It is an error if {Value} contains characters that cannot be converted to a floating point variable.
FLOATING	Yes	No	{Value} is converted to a floating point variable. It is an error if {Value} contains characters that cannot be converted to a floating point variable.
GRAPHICVAR	No	N/A	N/A
INTEGER	Yes	No	{Value} is converted to an integer. It is an error if {Value} contains characters that cvannot be converted to an integer.
INDEXLIST	Yes	Yes	None (Each value in the IndexList is treated as a CharVar, regardless of the actual type. At least one value must compare successfully for the IndexList to compare successfully.
METHODNAME	No	N/A	N/A
METHODPARAMETERLIST	No	N/A	N/A
METHODSPEC	No	N/A	N/A
OBJECTID	No	N/A	N/A
OBJECTIDLIST	No	N/A	N/A
OBJECTLINK	No	N/A	N/A
OBJECTLINKLIST	No	N/A	N/A

RODM Data Type	Allowed in Leaf Specification	Allows Wildcards /Regular Expressions	Conversion
OBJECTNAME	Yes	Yes	None (treated as a character string)
RECIPIENTSPEC	No	N/A	N/A
SELFDEFINING	No	N/A	N/A
SHORTNAME	No	No	None (treated as a character string)
SMALLINT	Yes	No	{Value} is converted to a short integer. It is an error if {Value} contains characters that cannot be converted to a short integer,
SUBSCRIBEID	No	N/A	N/A
SUBSCRIPTSPEC	No	N/A	N/A
SUBSCRIPTSPECLIST	No	N/A	N/A
TIMESTAMP	No	N/A	N/A
TRANSID	No	N/A	N/A

Examples of Collection Definition Objects

This section contains examples of the Collection Definition Objects.

Example 1:

Collect all objects in the GMFHS_Managed_Real_Objects_Class whose DisplayStatus field is not equal to 129 and show them in a Network View. The vertical bar character (|) serves as the separator character on the CollectionSpec1 field.

The CDO object that describes this collection can be specified as follows in a RODM loader file:

```
CREATE INVOKER  ::= 0000003;
     OBJCLASS ::= Network_View_Collection_Class;
     OBJINST  ::= MyName = (CHARVAR) 'Example1';
     ATTRLIST
     Annotation ::= (CHARVAR) 'Example1 Annotation',
     LayoutType ::= (INTEGER) 1,
     CollectionSpec1 ::=
       (CHARVAR) '|GMFHS_Managed_Real_Objects_Class|
                               DisplayStatus|129|.NE.';
END;
```

This RODM Collection Manager creates a Network_View_Class object called "Example1" with a LayoutType of 1 and Annotation of "Example1 Annotation". The collection specification represents a single conditional (it is composed of a single leaf specification). The matching object list is copied to the ContainsObject field of the Example1 view.

Example 2:

Collect all objects in the appnTransmissionGroupCircuit class (actual class name is 1.3.18.0.0.2058) whose DisplayResourceOtherData field contains a CP as the first two characters, and Active as the last six characters AND all objects in the appnTransmissionGroupCircuit class whose AggregationPriorityValue is equal to 1, 2, or 3 , and put them into an Aggregate. The vertical bar character (|) serves as the separator character.

The CDO object that describes this collection can be specified as follows in a RODM loader file:

```
CREATE INVOKER  ::= 0000003;
         OBJCLASS ::= Aggregate_Collection_Class;
         OBJINST  ::= MyName = (CHARVAR) 'Example2';
         ATTRLIST
         DisplayResourceOtherData ::= (CHARVAR) 'Example2 Other Data',
         DisplayResourceUserData ::= (CHARVAR) 'Example2 User Data',
         CollectionSpec1 ::=
           (CHARVAR) '|1.3.18.0.0.2058|DisplayResourceOtherData|
                                    CP*Active|.CONTAINS=.',
         CollectionSpec2 ::=
           (CHARVAR) '|1.3.18.0.0.2058|AggregationPriorityValue|
                                    1|2|3|.EQ.|.AND.';
     END;
```

The RODM Collection Manager creates a GMFHS_Aggregate_Objects_Class object called Example2 with a DisplayResourceOtherData of "Example2 Other Data" and a DisplayResourceUserData of "Example2 User Data". The other fields that are not specified on the Aggregate_Collection_Class object are set to the defaults used for objects created on the GMFHS_Aggregate_Objects_Class.

The collection specification is represented in both of the CollectionSpec*n* fields. It can be placed entirely in either the CollectionSpec1 or CollectionSpec2 field; this example demonstrates the concatenation of the two fields. The actual collection specification, after concatenation, is:

```
|1.3.18.0.0.2058|DisplayResourceOtherData|CP*Active|.CONTAINS=.|1.3.18.0.0.2058|
AggregationPriorityValue|1|2|3|.EQ.|.AND.
```

This collection specification represents a complex conditional (it is composed of a two leaf specifications). DOS wildcards are used to find the objects that match the DisplayResourceOtherData value. If there are three objects in class 1.3.18.0.0.2058 with objects IDs 1, 2, and 3, and their corresponding DisplayResourceOtherData fields contain:

- CPCP-supportedActive
- CP-CP Session Support
- CPCP-supportedNotActive

and their corresponding AggregationPriorityValue fields contain:

- -1
- 2
- 3

After evaluating the first leaf specification, the virtual stack contains:

- {1, 3}

where {1, 3} is the object list produced from evaluating the leaf specification. After evaluating the second leaf specification, the virtual stack contains:

- {2, 3}
- {1, 3}

The .AND. conjunction causes the two object lists to be removed from the stack; their intersection results in the list {3} which is added to the stack. This object is the result of the entire complex conditional. It is linked into both the AggregationChild field (using the DUIFCUAP method) and the IsPartOf field on the Example2 object.

There is no benefit using two different classes in the individual leaf specifications. Both leaf specifications, by definition, produce object lists that contain no objects in common. The intersection of the lists requested by the .AND. conjunction therefore always produces an empty object list. If the conjunction is .OR., then using two different classes is acceptable.

Example 3:

Collect all objects in the GMFHS_Managed_Real_Objects class whose MyName matches TEST plus an alphabetic classification character plus some number of additional characters plus 1 plus a numeric range character; for example, "TESTACPU10", as long as the alphabetic classification character is not B, and whose DisplayStatus is either Satisfactory or Unsatisfactory. Add to this list the objects in the GMFHS_Aggregate_Objects_Class whose MyName matches TEST plus an alphabetic classification character plus some number of additional characters, for example, "TESTACPUALL", as long as the alphabetic classification character is not B. Enter them into a Network View.

The CDO object that describes this collection can be specified as follows in a RODM loader file:

```
CREATE INVOKER  ::= 0000003;
       OBJCLASS ::= Network_View_Collection_Class;
       OBJINST  ::= MyName = (CHARVAR) 'Example3';
       ATTRLIST
       Annotation ::= (CHARVAR) 'Example3 Annotation',
       CollectionSpec1 ::=
          (CHARVAR) '|GMFHS_Managed_Real_Objects_Class|MyName|\^TEST[A-C].*1.$|
                            .CONTAINS.'
                      '|GMFHS_Managed_Real_Objects_Class|MyName|TESTB*|
                            .NCONTAINS.|.AND.'
                      '|GMFHS_Managed_Real_Objects_Class|DisplayStatus|130|
                            .LE.|.AND.'
                    '|GMFHS_Aggregate_Objects_Class|MyName|\^TEST[A-C].*$|
                            .CONTAINS.'
                      '|GMFHS_Aggregate_Objects_Class|MyName|TESTB*|
                            .NCONTAINS.|.AND.|.OR.';   END;
```

Assume the following objects exist in the GMFHS_Managed_Real_Objects_Class:

Object ID	MyName	DisplayStatus
1	TESTACPU10	131
2	TESTACPU11	129
3	TESTBCPU10	130
4	TESTBCPU11	132
5	TESTCCPU10	129
6	TESTCCPU11	132
7	TESTCCPU12	129

Object ID	MyName	DisplayStatus
8	TESTCCPU12X	130
9	TESTDCPU10	129

Assume the following objects exist in the GMFHS_Aggregate_Objects_Class:

Object ID	MyName
10	TESTAAGGs
11	TESTBAGGS
12	TESTCAGGS
13	TESTDAGGS

The expression for the first leaf specification is given in regular expression notation. DOS wildcards do not have a way to specify that the 5th character must be between A and C, so the regular expression was used in this case. After evaluating the first leaf specification, the virtual stack contains:

```
{1, 2, 3, 4, 5, 6, 7}
```

After evaluating the second leaf specification, the virtual stack contains:

```
{1, 2, 5, 6, 7, 8, 9}
      {1, 2, 3, 4, 5, 6, 7}
```

The .AND. conjunction removes these two lists from the stack, and replaces them with:

```
{1, 2, 5, 6, 7}
```

After evaluating the third leaf specification, the virtual stack contains:

```
{2, 3, 5, 7, 8, 9}
{1, 2, 5, 6, 7}
```

The .AND. conjunction removes these two lists from the stack, and replaces them with:

```
{2, 5, 7}
```

After evaluating the fourth leaf specification, the virtual stack contains:

```
{10, 11, 12}
{2, 5, 7}
```

After evaluating the fifth (and final) leaf specification, the virtual stack contains:

```
{10, 12, 13}
{10, 11, 12}
{2, 5, 7}
```

The .AND. conjunction removes the top two lists from the stack, and replaces them with:

```
{10, 12}
{2, 5, 7}
```

Finally, the .OR. conjunction removes only two lists from the stack, and replaces them with:

```
{2, 5, 7, 10, 12}
```

This becomes the final object list returned by the complex conditional which is then linked in to the ContainsObjects field of the Example 3 object.

Using NetView Resource Manager

This section describes NetView Resource Manager views and how they can be customized. You can use NetView Resource Manager to graphically monitor and manage NetView tasks for resource utilization and status with the NetView management console. You can monitor all the NetView programs in your enterprise using one NetView management console. For more information about setting up and using NetView Resource Manager see the following books:

- *IBM Tivoli NetView for z/OS Installation: Configuring Graphical Components*
- *IBM Tivoli NetView for z/OS User's Guide: NetView*

NetView Resource Manager Views

When NetView Resource Manager is active, **NetViewTasks** is displayed in the NetView management console view tree. This opens a view of the NetView Resource Manager domain aggregate objects. You can navigate from this view to the NRM Task aggregate objects view. From a Task aggregate, you can navigate to a view with the following real objects, which represent statistical monitors:

- Status
- CPU (CPU utilization)
- STG (Storage)
- MSGCT (Message Queue Count)
- MQOUT (Output Message Rate)
- MQIN (Input Message Rate)
- I/O (I/O Rate)

To see the value of the monitors, click **Resource Properties** to open the Resource Properties notebook.

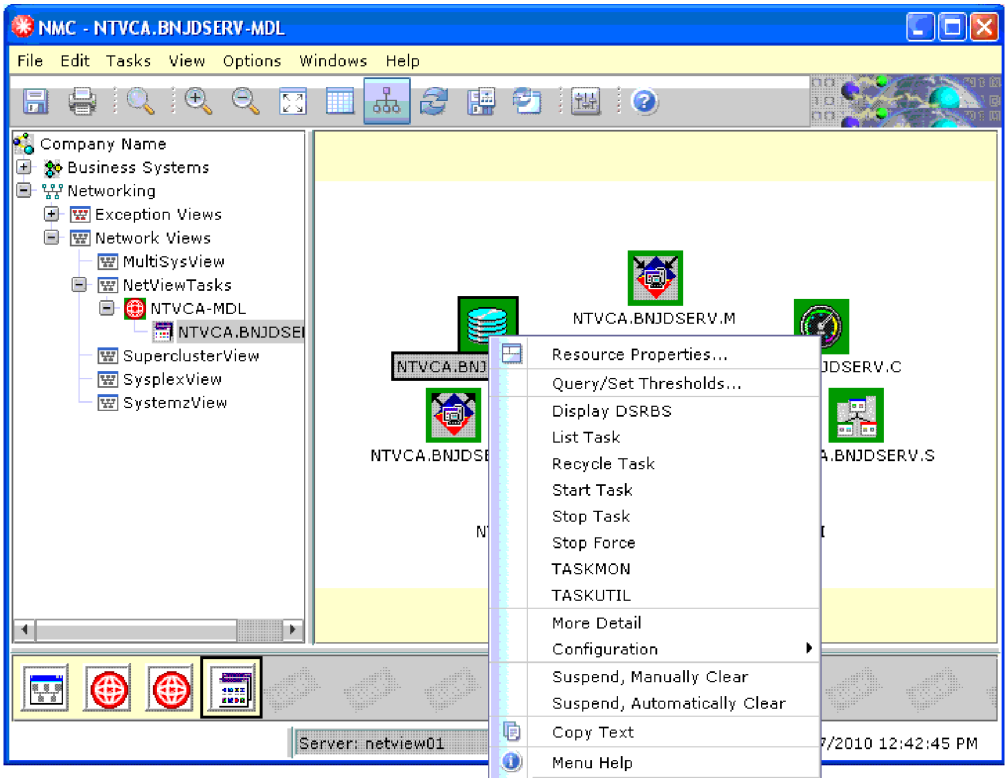
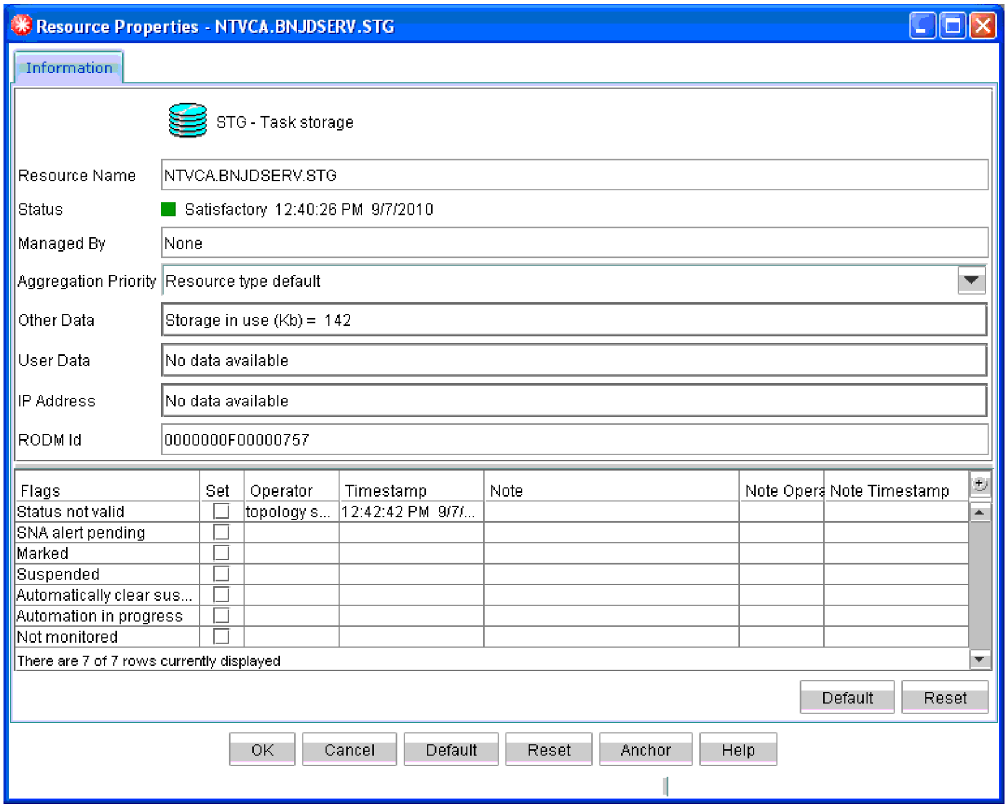


Figure 35. Resources Properties Notebook

The monitor value is in the **Other Data** field.



Resource Name: NTVCA.BNJDSERV.STG
Status: 🟩 Satisfactory 12:40:26 PM 9/7/2010
Managed By: None
Aggregation Priority: Resource type default
Other Data: Storage in use (Kb) = 142
User Data: No data available
IP Address: No data available
RODM Id: 0000000F00000757

Flags	Set	Operator	Timestamp	Note	Note Opera	Note Timestamp
Status not valid	☐	topology s...	12:42:42 PM 9/7/...			
SNA alert pending	☐					
Marked	☐					
Suspended	☐					
Automatically clear sus...	☐					
Automation in progress	☐					
Not monitored	☐					

There are 7 of 7 rows currently displayed

Figure 36. Other Data Field

This field is not automatically updated dynamically. If you want it to be updated dynamically, see "Using DUIFVINS with NetView Resource Manager" on page 164.

The default status values for NetView Resource Manager real objects are as follows:
- Task active - Satisfactory
- Task inactive - Unknown
- Task status unknown - Unknown
- Threshold 1 has been reached - Intermediate
- Threshold 2 has been reached - Medium Unsatisfactory
- Threshold 3 has been reached - Unsatisfactory

The status value is stored in the RODM DisplayStatus field for each NetView Resource Manager object that represents a statistical monitor.

Status values for the real objects can be customized. See the `Display Status` section in CNMSTYLE under `NetView Resource Manager Initialization Parameters` for information about how to do this customization. NetView Resource Manager real objects are in the GMFHS_Managed_Real_NRM_Objects_Class class, therefore a DisplayStatus of Unknown does not map to an exception state. If you want to map the Unknown DisplayStatus to an exception status for NetView Resource Manager objects, see "Modifying DUIFSMT for NetView Resource Manager" on page 164.

NetView Resource Manager Object Information

NetView Resource Manager aggregate objects are in the GMFHS_Aggregate_NRM_Objects_Class class. NetView Resource Manager real objects are in the GMFHS_Managed_Real_NRM_Objects_Class class. All NetView Resource Manager objects have an "NRM" prefix in the MyName field.

NetView Management Console Command Support for NetView Resource Manager

Commands are available for all NetView Resource Manager objects. The commands that are available depend on the type of task, as shown in Table 15. The available commands can be selected by right-clicking the selected object. Command results are displayed on the console log of the NetView management console.

Table 15. Supported NetView Management Console Commands

Command	**Tasks**
DSRBS	• DST
LIST SAFOP=opid	• OST • NNT • AOST (Autotask)
LIST taskname	• PPT • DST • OPT • OST • NNT • AOST (Autotask) • HCT
LIST STATUS=TASKS	• NetView Aggregate
LIST STATUS=VOST	• VOST (Virtual OST)
Message	• OST • NNT • AOST (Autotask) • VOST (Virtual OST)
Query/Set Thresholds[1]	• NetView Aggregate • MAINTASK • PPT • DST • OPT • OST • NNT • AOST (Autotask) • VOST (Virtual OST) • HCT
RECYCLET	• DST • OPT
RESOURCE	• NetView Aggregate
START HCL=hclname[1]	• HCT
START TASK=taskname[1]	• DST • OPT
STOP FORCE=taskname[1]	• DST • OPT • OST • NNT • AOST (Autotask) • VOST (Virtual OST) • HCT

Table 15. Supported NetView Management Console Commands (continued)

Command	Tasks
STOP TASK=taskname[1]	• DST • OPT • OST • NNT • AOST (Autotask) • VOST (Virtual OST) • HCT
TASKMON	• NetView Aggregate • MAINTASK • PPT • DST • OPT • OST • NNT • AOST (Autotask) • VOST (Virtual OST) • HCT
TASKUTIL	• NetView Aggregate • MAINTASK • PPT • DST • OPT • OST • NNT • AOST (Autotask) • VOST (Virtual OST) • HCT

The commands issued at the TASK aggregate are generally the same as the commands issued at the real objects, with the TASKMON command as an exception. TASKMON *taskname* is issued on aggregate TASK objects. TASKMON *taskname stat* is issued on the following objects:
- CPU
- STG
- IO
- MQIN
- MQOUT

TASKMON *taskname* is issued for the STATUS and MSGCT objects.

Note: For more information about autotasks, see the *IBM Tivoli NetView for z/OS Automation Guide*. With the exception of the DSIWEB and FLBTOPO tasks, all the tasks listed are valid for NetView Resource Manager.

Use the Query/Set Threshold command, which is presented as a dialog, to examine or change the effective NetView Resource Manager thresholds. This dialog is available for all objects except the STATUS object. The thresholds can also be set with the DEFAULTS and OVERRIDE commands. Use the Message command, which is also presented as a dialog, to send a message to the selected operator task.

---

[1] These commands are protected by the default security for the NetView program (CNMSCAT2/CNMSAF2).

Modifying DUIFSMT for NetView Resource Manager

Unknown resources (inactive tasks), by default, are not considered to be in an exception state. To map the DisplayStatus value of Unknown to an exception state for resources in the GMFHS_Managed_Real_NRM_Objects_Class class, use DUIFSMT.

Example:

```
DUIFSMTE CLASS=GMFHS_Managed_Real_NRM_Objects_Class,            C
         XCPT=(UNSAT,DS152,DS153,DS154,DS155,DS156,DS157,DS158,DSC
         159,MEDUN,LOWUN,UNKWN)
```

CNMSJH13 is provided to assemble and link-edit DUIFSMT. For more information about DUIFSMT, see “Customizing the DisplayStatus Mapping Table for Exception Views” on page 104.

Using DUIFVINS with NetView Resource Manager

To have the NetView Resource Manager monitor values updated dynamically, code the following RODM loader statement:

```
OP DUIFVINS INVOKED_WITH (SELFDEFINING)
   (
    (SMALLINT) 0
    (INTEGER)  7
    (OBJECTID) EKG_Method.DUIFVNOT
    (CLASSID)  GMFHS_Managed_Real_NRM_Objects_Class
    (FIELDID)  GMFHS_Managed_Real_NRM_Objects_Class.DisplayResourceOtherData
);
```

See “DUIFVINS: Install View Granularity Method (DUIFVNOT)” on page 499 for more information.

NetView Resource Manager Sample Loader Files

A sample of NetView Resource Manager objects views and aggregates that take advantage of the RODM Collection Manager is available. The RODM Collection Manager is a NetView function that actively monitors RODM contents and updates views and aggregates according to criteria you specify. One section of sample `JCL CNMSJH12` provides sample RODM loader files that build RODM Collection Manager collections of NetView Resource Manager objects.

Follow the instructions in `CNMSJH12` to uncomment the two DD statements containing DUIFNRM1 and DUIFNRM2 parts as shown in the following example:

```
// DD DSN=NETVIEW.V6R1M0.CNMSAMP(DUIFNRM1),DISP=SHR  <-NRM RCM SAMPLE
// DD DSN=NETVIEW.V6R1M0.CNMSAMP(DUIFNRM2),DISP=SHR  <-NRM RCM SAMPLE
```

The DUIFNRM1 sample contains the following views and aggregates:

- View - NRM_OSTs - All NetView users logged on
- View - NRM_CPU_USERS - Non-Satisfactory CPU users
- View - NRM_HEALTH - General health of the NetView program, containing the following aggregates:
  - Aggregate - NRM_HEALTH_CPU - All Non-Satisfactory CPU objects
  - Aggregate - NRM_HEALTH_IO - All Non-Satisfactory IO objects
  - Aggregate - NRM_HEALTH_MQS - All Non-Satisfactory MQIN and MQOUT objects
  - Aggregate - NRM_HEALTH_MESSAGES - All Non-Satisfactory MSG objects
  - Aggregate - NRM_HEALTH_STORAGE - All Non-Satisfactory STG objects

These views and aggregates collect data from all NetView programs that the NetView Resource Manager is currently managing, so they are best used on a single system. Or, they can be modified to select a single system by changing their criteria using the RODM Collection Manager, described in "Customizing Sample Loader Files."

The DUIFNRM2 sample is an example of selecting objects from a single NetView program. It contains the following view:

- View - NRM_DSI_TASKS - A01NV tasks starting with DSI

Customizing Sample Loader Files

After you load the sample RODM loader files, you can modify the collections using the NetView management console. As an administrator, click **Tasks** > **RODM Collection Manager**, to open the RODM Collection Manager GUI. From there, you can browse and modify the collections. To make your changes persistent across RODM cold starts, specify a data set or partitioned data set member to which to save your changes when sending your collections to the host. Then, when you restart RODM, load the data sets containing your modified collections, so that they are available to NetView management console users.

Chapter 6. Customizing GMFHS to Process and Receive Alerts and Resolutions

This chapter describes how GMFHS receives and processes alerts and resolutions. It describes how the customization changes you make affect this processing. Ensure the name of the objects you create in RODM match the resource names supplied by alerts.

Receiving and Monitoring Alerts or Resolutions

GMFHS monitors the status of non-SNA resources and the alert-received (event notification) user status of SNA resources by receiving copies of all alert and resolution major vectors that the hardware monitor automates. GMFHS identifies the resources on which these major vectors report. GMFHS relates each status report to the object in RODM that represents the resource.

Note: A *non-SNA domain* in GMFHS is any valid combination of a service point, transaction program, and element management system. A non-SNA domain in GMFHS functions as the interface between the NetView program and the non-SNA network.

There are seven elements involved in this process; customization can affect all of them:

- What GMFHS receives from the hardware monitor
- Objects in RODM representing SNA resources
- Objects in RODM representing network management gateways (NMGs)
- Objects in RODM representing non-SNA domains
- Objects in RODM representing non-SNA resources
- DUIFEDEF alert processing
- Alert translation tables

What GMFHS Receives from the Hardware Monitor

When NetView receives an alert, the alert is passed through the automation table where the DUIFECMV command processor is run. This command processor sends information to GMFHS and initiates GMFHS processing of the alert. The information received by GMFHS is:

- A copy of the major vector.
- The hardware monitor resource hierarchy created from the content of the hierarchy and resource list (H/RL) subvector or hierarchy name list (HNL) subvector.
- The name of the SNA domain from which the major vector originated.
- An optional set of parameters to DUIFECMV which bypass the DUIFEDEF alert processor. The parameters are CLASS, DOMAIN, INDICAT, OBJNAME, STATUS, and GMFHSDOM. If specified, the following parameters are required:
  - DOMAIN
  - CLASS
  - OBJNAME
  - INDICAT

  STATUS is required only if the value of parameter INDICAT is 2 or 4. GMFHSDOM is optional.

GMFHS checks the hardware monitor resource hierarchy rather than the H/RL or HNL subvectors for resource names. Some of its logic depends on the presence or absence of these two subvectors.

If parameters are specified for DUIFECMV, they cause GMFHS to bypass the processing described in "Objects in RODM Representing SNA Resources," "Objects in RODM Representing NMGs" on page 169, "Objects in RODM Representing Non-SNA Domains" on page 169, and "Objects in RODM Representing Non-SNA Resources" on page 171. CLASS, DOMAIN, and OBJNAME are used to identify the object to which the alert is logged, and STATUS specifies a value for the new resource status. INDICAT specifies the type of status processing to perform. When a value of 1 or 3 is specified for INDICAT, the procedure described in "Alert Translation Tables" on page 176 is used.

Note: Command processor DUIFECMV must run under the autotask DUIFEAUT. Refer to the NetView online help or the *IBM Tivoli NetView for z/OS Command Reference Volume 1 (A-N)* for more information about DUIFECMV and its operands.

Objects in RODM Representing SNA Resources

When GMFHS receives an alert or resolution major vector, it tries to determine whether the reported resource is an SNA resource or a non-SNA resource. If the major vector contains neither the H/RL subvector nor the HNL subvector, GMFHS handles the major vector as an SNA resource. If either of these subvectors is present and the hardware monitor resource hierarchy contains either a service point resource type (SP or PUGW), or a transaction program resource type (TP or PUGA), the resource must be a non-SNA resource. GMFHS uses the "First Method" on page 169 to process this non-SNA resource. If either of these subvectors is present and neither a service point type (SP or PUGW), or a transaction program resource type (TP or PUGA) is contained in the hardware monitor hierarchy, the resource being reported on can still be either a SNA or a non-SNA resource. GMFHS uses the method described in "Second Method" on page 170.

If GMFHS determines that the resource being reported on is a non-SNA resource, GMFHS takes action according to procedures described in "Objects in RODM Representing Non-SNA Resources" on page 171. The remainder of this section describes the actions GMFHS takes if it determines that the resource being reported on is an SNA resource.

GMFHS tries to find an object in the SNA_Domain_Class with a name that matches the original SNA domain name for the major vector. If it does not find this object, GMFHS drops the major vector. If this object is found, GMFHS tries to find an object in the GMFHS_Shadow_Objects_Class with a name that is the concatenation of the SNA network (SNANet) field of the SNA_Domain_Class object, a period (.) delimiter, and the resource name farthest to the right in the hardware monitor resource hierarchy.

For example, suppose the following object is defined in the SNA_Domain_Class:

```
MyName : A10NV
SNANet : NETA
```

If GMFHS receives an alert with an origin SNA domain name of A10NV and that alert has NT69I073 as the name farthest to the right in the hardware monitor resource hierarchy, the name of the object searched for in the GMFHS_Shadow_Objects_Class follows:

```
NETA.NT69I073
```

If GMFHS finds this object in the GMFHS_Shadow_Objects_Class, it turns on the event notification bit in the UserStatus field of this object, creates an event report protocol data unit, and logs it.

When you create objects in the SNA_Domain_Class and GMFHS_Shadow_Objects_Class, you need to coordinate the names of these objects with the names of your SNA networks, SNA domains, and SNA resources in those domains.

Objects in RODM Representing NMGs

GMFHS uses NMG objects during alert processing if it has determined that the second method is necessary to resolve the alert. The way in which the NMG object is used is defined under the "Second Method" on page 170.

Objects in RODM Representing Non-SNA Domains

When GMFHS receives an alert or resolution for a non-SNA resource, it first determines the identity of the non-SNA domain containing the non-SNA resource being reported on. Next GMFHS tries to identify the resource itself. GMFHS does this by using hardware monitor resource hierarchy information as a search argument to compare against the names of objects you defined in the Non_SNA_Domain_Class. Knowing how this search is accomplished can help you understand how to set up a plan to name your Non_SNA_Domain_Class objects with information contained in the hardware monitor resource hierarchy.

GMFHS uses two methods mentioned previously to determine the identity of the non-SNA domain. These methods are described in detail in this chapter. In the first method, the resource is assumed to be a non-SNA resource. If, after applying this method, GMFHS cannot identify the non-SNA domain of the resource being reported on, it drops the major vector because it cannot identify the non-SNA resource. In the second method, alerts that are not for non-SNA resources are assumed to be for SNA resources, and the steps described in "Objects in RODM Representing SNA Resources" on page 168 are used. When you define objects in the Non_SNA_Domain_Class, be sure your plan includes information GMFHS looks for in the hardware monitor resource hierarchy.

First Method

As described previously, it has been determined that either a Hierarchy Resource List or a Hierarchy Name List subvector is present in the alert, and the alert contains a service point or a transaction program or both upon entrance to this method.

Beginning with the hierarchy element defined as a service point (if found), or beginning with the hierarchy element defined as a transaction program if a service point is not found, GMFHS builds a concatenation of all names remaining in the resource hierarchy. In this concatenation, the names are separated from one another by a period (.).

GMFHS next compares this concatenation with the names of all objects in the Non_SNA_Domain_Class. All of the objects in this class have been sorted in descending order based on the length of their names. If GMFHS cannot find a non-SNA domain object that matches the current concatenation list, then the rightmost object is removed from concatenation and the Non_SNA_Domain_Class is searched once again for this new name. This process continues until either a Non_SNA_Domain_Class object matches, or the concatenation list contains no more elements.

For example, suppose the hardware monitor resource hierarchy contains the following resource name and type pairs:

```
Name        Type

NMGPU5       PU
SP010        SP
RING010      RING
PRINTER1     PRTR
```

There is an object in the Non_SNA_Domain_Class named SP010.RING010. GMFHS looks for a Non_SNA_Domain_Class object with one of these names, exactly as shown, and in the order shown:

```
SP010.RING010.PRINTER1
SP010.RING010
SP010
```

GMFHS acts on the first object that matches with the current concatenation list. In this example, there is no non-SNA domain object named `SP010.RING010.PRINTER1`, but there is an object named `SP010.RING010`. GMFHS handles the object named `SP010.RING010` as though it represents the domain of the resource reported on.

There might also be a non-SNA domain object named `SP010` in this example. However, the match will occur on the first non-SNA domain object in the sorted list; therefore, the match will occur on `SP010.RING010` before `SP010`. Also, the names must match exactly; a concatenation name of `SP010.RING01` will not match a non-SNA domain name of `SP010.RING010`.

Second Method

If the alert hierarchy does not have a service point or a transaction program, GMFHS compares each name in the resource hierarchy, beginning with the rightmost resource in the hierarchy, to each NMG_Class object name.

Note: This is not a concatenation list as used in the first method, but rather each individual resource name. If a match is not found, the alert is treated as an alert for a SNA object.

If a match is found, all Non_SNA_Domain_Class objects are searched for a match on the same name. If a match is not found, the alert is treated as an alert for a SNA object. Otherwise, a match has been found on a non-SNA domain object.

For example, suppose the hardware monitor resource hierarchy contains the following resource name and type pairs:

```
Name        Type

NMGPU5       PU
PRINTER2     DEV
```

There is an object in the NMG_Class named `NMGPU5`, and an object in the Non_SNA_Domain_Class named `NMGPU5`. GMFHS looks for an NMG_Class object with one of these names, exactly as shown, and in the order shown:

```
PRINTER2
NMGPU5
```

As soon as a match is found with an NMG_Class object (in this case, with the object named `NMGPU5`), a check is made for the same object name in the Non_SNA_Domain_Class. If a match is found there, then this domain contains the object being reported on.

It is important to note that if the Non_SNA_Domain_Class name does not match, the search will not continue with the next name in the resource list and the NMG_Class. The first time the NMG_Class is matched, either the Non_SNA_Domain name also matches the resource hierarchy element, or the alert is treated as a SNA resource alert.

Objects in RODM Representing Non-SNA Resources

If GMFHS finds the non-SNA domain as described in "Objects in RODM Representing Non-SNA Domains" on page 169, it tries to identify the non-SNA resource. GMFHS does this by invoking the load module named in the AlertProc field of the Non_SNA_Domain_Class object. Refer to the *IBM Tivoli NetView for z/OS Data Model Reference* for more information about the AlertProc field.

The default value for the AlertProc field is DUIFEDEF. A sample DUIFEDEF is shipped with the NetView program. DUIFEDEF can return the following:
- A list of zero or more possible resource names to GMFHS
- A feedback indicator that specifies whether the names are for a single non-SNA resource or for multiple non-SNA resources
- The name of the RODM class containing these non-SNA resources
- The value for DisplayStatus

Single Non-SNA Resource

When the DUIFEDEF feedback indicator specifies that the names are for a single non-SNA resource, then, for each name in this list, GMFHS tries to find an object in the class returned by DUIFEDEF, until either an object is found or the end of the list is reached.

For the first object found (and only this object), GMFHS:
- Determines the DisplayStatus returned by DUIFEDEF or, if not present, translates the status reported in the alert or resolution into a GMFHS DisplayStatus. Refer to the *IBM Tivoli NetView for z/OS Data Model Reference* for more information about the DisplayStatus field.
- Relates this status to the object in the class returned by DUIFEDEF.
- Builds an event report protocol data unit.
- Logs this protocol data unit in the Dbserver database.

Multiple Non-SNA Resources

When the DUIFEDEF feedback indicator specifies that the names are for multiple non-SNA resources, GMFHS tries to find an object in the class returned by DUIFEDEF for each name in the list. For each object found, GMFHS:
- Determines the DisplayStatus returned by DUIFEDEF or, if not present, translates the status reported in the alert or resolution into a GMFHS DisplayStatus.
- Relates the status reported to the object in the class returned by DUIFEDEF.
- Builds an event report protocol data unit.
- Logs this protocol data unit.

All alerts and resolutions that report on resources in a non-SNA domain are processed by the same AlertProc module. Be sure that the alerts and resolutions for any non-SNA domain where you have made modifications are always formatted so that the AlertProc module for that domain produces the expected results.

DUIFEDEF Alert Processing

If no value is present for AlertProc or if DUIFEDEF is named in the AlertProc field, DUIFEDEF provides the possible name of the non-SNA resource or resources described in an alert or resolution, and the name of the class containing these resources. The sample DUIFEDEF provided with the NetView program also looks for alerts from LANs that can report on single or multiple resources.

Parameters

GMFHS runs DUIFEDEF (or any other load module named in the AlertProc field) with the following parameters:

Pointer to a reentrant work area

The AlertProc module is reentrant and uses this work area. The same work area is shared among all AlertProc modules. An AlertProc module cannot assume that information the module stores in this work area is still intact at a later call of the module. The work area format is as follows:

- Fullword representing the length of the work area set by GMFHS. This must not be modified by the AlertProc module.
- Fullword containing the following fields:
  - One byte containing the DisplayStatus value set by the AlertProc module before returning to GMFHS. The DisplayStatus value and its meanings are as follows:

Value	**Meaning**
0	DisplayStatus has not been determined. Use the status mapping table.
Non-0	The DisplayStatus value that is to be used.

  - Two bytes reserved.
  - One byte containing the binary feedback indicator set by the AlertProc module before returning to GMFHS. The feedback indicator value and its meanings are as follows:

Value	**Meaning**
0	Each possible name identifies only one non-SNA resource. GMFHS queries RODM for each name until it finds a match, and relates the status to only this resource.
Non-0	Each possible name identifies a separate non-SNA resource. GMFHS queries RODM for each name, and for each name found, applies the status to the resource.

    Note: Prior to NetView V3R1, the binary feedback indicator was four bytes. For migration purposes, two of these bytes are now reserved and one is used for the DisplayStatus value. Set the binary feedback indicator to 0 or 1.
- Fullword containing the offset from the start of the work area to the first possible name.

- The name of the RODM class which contains the possible resource names. The class name is formatted as follows:
  - Halfword, not boundary aligned, containing the length of the class name.
  - Character string containing the RODM class name.
- The list of possible resource names is formatted as follows:
  - Halfword, not boundary aligned, containing the length of the resource name.
  - Character string containing the resource name.

  When more than one name is returned, names are concatenated without any boundary alignment. The list of possible names ends with a halfword containing binary zero, also not boundary-aligned. GMFHS accepts a list where the length of the first possible name is zero.

Pointer to a second reentrant work area

This work area is a separate work area supplied to each AlertProc module, and is 4088 (X'FF8') bytes in length. If an AlertProc module needs to retain information unaltered across calls, that information can be stored in this work area.

Value of the EMDomain field

The EMDomain field of the Non_SNA_Domain_Class object is a value representing the domain ID. It can be used by the AlertProc module to build candidate name lists. For more information about the EMDomain field refer to the *IBM Tivoli NetView for z/OS Data Model Reference*.

Value of the DomainCharacteristics field

The DomainCharacteristics field of the Non_SNA_Domain_Class represents the features supported by the domain. Refer to the *IBM Tivoli NetView for z/OS Data Model Reference* for more information about this field.

Pointer to an array of structures

Each structure describes a subvector within the major vector. Each structure has the following format:

- Fullword containing the pointer to a subvector. The leftmost bit is turned on in the fullword pointer that points to the last subvector in the major vector.
- Fullword integrity validation flag. If this fullword is all zeros, the subvector length is validated (in other words, not zero, and contained within the length of the major vector); if the subvector contains subfields, the subfield lengths are not validated. If this fullword is not all zeros, the subvector length is validated; if the subvector contains subfields, the subfield lengths are also validated.

There is a separate structure for each embedded product ID subvector (X'11') immediately following the structure for the product set ID subvector (X'10').

Pointer to hardware monitor resource hierarchy

This is a list, supplied by the hardware monitor, containing a text representation of the resource name and type pairs contained in the H/RL or HNL subvector. Each name and type pair contains an 8-character resource name, left-justified and right-padded with blanks, and a 4-character resource type, left-justified and right-padded with blanks. GMFHS supplies the portion of the hardware monitor resource hierarchy that follows the names which make up the name of the Non_SNA_Domain_Class object.

In the example, “First Method” on page 169 GMFHS supplies a list containing one name and type pair:

```
PRINTER1PRTR
```

Pointer to the length of the hardware monitor resource hierarchy

In the example, GMFHS supplies a pointer to a fullword containing the decimal value 12.

Register 15 Conventions

DUIFEDEF (or any other AlertProc module) returns a value in register 15 as follows:

Value	Meaning
0	The first reentrant workarea provided by GMFHS contains a list of zero or more possible resource names, formatted as described previously, the name of the RODM class containing the resource or resources, and optionally, a value for DisplayStatus for the resources. If there are zero names, the AlertProc module completed successfully but did not identify any non-SNA resources. GMFHS processes the name list and status according to the fullword feedback indicator in the work area.
Greater than 0	The first reentrant workarea provided by GMFHS is not large enough to hold all of the possible names and the RODM class name. The value in register 15 is the length of a work area required to contain all of the possible names and the RODM class name. If this is the first time the AlertProc module requested a larger work area for this alert, GMFHS acquires more space to satisfy the request and calls the AlertProc module again. Otherwise, GMFHS logs the error in a system error synopsis and issues console message DUI3913E.
Less than 0	The AlertProc module detected a calling parameter error. GMFHS logs the error in a system error synopsis and issues console message DUI3913E.

Default DUIFEDEF Actions

If neither subvector X'51' nor subvector X'5D' is present in the major vector, the alert or resolution reports status on only one non-SNA resource. DUIFEDEF follows these steps.

- Builds a list of either one or two possible names.
  - The first name is a concatenation of:
    - The EMDomain field supplied in the third calling parameter, not including trailing blanks.
    - A period (.) delimiter.
    - All resource names in the hardware monitor resource hierarchy, not including trailing blanks, delimited by periods (.), if indicated by the value of the DomainCharacteristics field. Refer to the *IBM Tivoli NetView for z/OS Data Model Reference* for information about this value in the DomainCharacteristics field.
  - The second name is a concatenation of:
    - The EMDomain field supplied in the third calling parameter, not including trailing blanks.
    - A period (.) delimiter.
    - The last resource name in the hardware monitor resource hierarchy.

  If the second name is identical to the first, only the first is returned to GMFHS.
- Returns a value of 0 in the binary feedback indicator.
- Returns a value of GMFHS_Managed_Real_Objects_Class in the RODM class name.

If either subvector X'51' or subvector X'5D' are present in the major vector, the alert or resolution reports status on one or more non-SNA resources. DUIFEDEF follows these steps:

- Builds a list of zero or more possible names.
  - Searches for the following subfields:
    X'03' - Local Individual MAC Address
    X'04' - Remote Individual MAC Address
    X'06' - Ring Fault Domain Description
    X'08' - Single MAC Address
    X'23' - Local Individual MAC Name
    X'24' - Remote Individual MAC Name
    X'26' - Fault Domain Names
    X'28' - Single MAC Name
  - Creates, for each subfield found, either one possible name:
    ```
    X'03', X'04', X'08', X'23',
    X'24', X'28'
    ```
    or two possible names:
    ```
    X'06', X'26'
    ```
  - Translates addresses to display hexadecimal. Each possible name is a concatenation of:
    - The EMDomain field supplied in the third calling parameter including trailing blanks.
    - A period (.) delimiter.
    - The name or address in the subfield. All resource names in the candidate name list can be delimited with a period if so requested in the DomainCharacteristics field. Refer to the*IBM Tivoli NetView for z/OS Data Model Reference* for information about this value in the DomainCharacteristics field.
    - If any resulting name is a duplicate of a name already in the list, it is not added to the list.
    - If any resulting object name is longer than 254 maximum characters RODM permits, the name is not added to the list.
    - If any name in subfields X'23', X'24', X'26', or X'28' is all blanks, GMFHS does not build a possible name.
    - Trailing blanks in subfields X'23', X'24', X'26', and X'28' are not included in possible names. Embedded blanks in these subfields are included in possible names. Since RODM does not currently permit object names with embedded blanks, GMFHS is not successful when it attempts to find objects with such names in RODM.
- Returns a value of 1 in the binary feedback indicator.
- Returns a value of GMFHS_Managed_Real_Objects_Class in the RODM class name.

  To illustrate, suppose the value of the EMDomain field of this Non_SNA_Domain_Class object is DOMAIN1. If there is no subvector X'51' or subvector X'5D', DUIFEDEF returns a feedback indicator value of 0 and one possible name:

```
DOMAIN1.PRINTER1
```

If, however, there is a subvector X'51' or subvector X'5D', which contains a Ring Fault Domain Description subfield, and the addresses in the subfield are X'00101AF1CE74' and X'00101AF1CE0B', then, DUIFEDEF returns a feedback indicator value of 1 and two possible names:

```
DOMAIN1.00101AF1CE74
DOMAIN1.00100AF1CE0B
```

Alert Translation Tables

DUIFEUSR and DUIFEIBM are alert translation tables contained in non-reentrant and non-reusable load modules. DUIFEIBM is supplied to you as a load module only. DUIFEUSR is supplied to you as a load module, an assembler source module, and an assembler macro named DUIFEDST.

DUIFEIBM contains the default code point translations supplied by IBM. DUIFEUSR is supplied to the you as an empty table. You can add code point translations to DUIFEUSR which override matching code point translations contained in DUIFEIBM.

One or more DUIFEDST macros can be added to DUIFEUSR to define alert code point translation. The macro format is as follows:

DUIFEDST

```
►►──DUIFEDST STATUS=DisplayStatus_value──┬──────────────────┬──────────────────►
                                         └─,ALERT=alert_type─┘

►──┬─────────────────┬──┬──────────────────────┬──────────────────────────────►◄
   └─,CLASS=class_name─┘  └─,MYNAME=resource_name─┘
```

Where:

STATUS=*DisplayStatus_value*

The NetView DisplayStatus value for this table entry. For example, to assign a DisplayStatus value of UNSATISFACTORY, code STATUS=UNSATISFACTORY. The STATUS keyword is required. Valid values are:

- SATISFACTORY
- UNSATISFACTORY
- INTERMEDIATE
- UNKNOWN
- DS136 (User Positive 1)
- DS137 (User Positive 2)
- DS138 (User Positive 3)
- DS139 (User Positive 4)
- DS140 (User Positive 5)
- DS141 (User Positive 6)
- DS142 (User Positive 7)
- DS143 (User Positive 8)
- MEDSA (Medium Satisfactory)
- LOWSA (Low Satisfactory)
- DS152 (User Negative 1)
- DS153 (User Negative 2)
- DS154 (User Negative 3)
- DS155 (User Negative 4)
- DS156 (User Negative 5)

- DS157 (User Negative 6)
- DS158 (User Negative 7)
- DS159 (User Negative 8)
- MEDUN (Medium Unsatisfactory)
- LOWUN (Low Unsatisfactory)

ALERT=*alert_type*
Is any valid alert type from the basic alert or generic alert. The ALERT keyword is optional.

CLASS=*class_name*
The name of the RODM class that applies to this table entry. The CLASS keyword is optional.

MYNAME=*resource_name*
The MyName of the resource or groups of resources that applies to this table entry. The wildcard character (*) can be used as a suffix to specify groups of resources. The MYNAME keyword is optional.

GMFHS sequentially searches the table to find the first match for an alert. Therefore, place your DUIFEDST macros in most-specific to least-specific order to ensure your desired status processing occurs.

To specify that alert_type X'03' (Performance) is to result in a DisplayStatus_value of UNSATISFACTORY for all resources that begin with 'A.B.C', code the following statement:

```
DUIFEDST MYNAME=A.B.C*,ALERT=03,STATUS=UNSATISFACTORY
```

The last statement in DUIFEDST must be as follows:

DUIFEDST END

This statement must appear immediately before the END statement in your assembler source file.

Table 16 contains the default alert translations that exist in DUIFEIBM.

Table 16. Default Alert Translations in DUIFEIBM

Alert Type	DisplayStatus Value
01	UNSATISFACTORY
02	UNSATISFACTORY
03	UNSATISFACTORY
04	INTERMEDIATE
0A	INTERMEDIATE
0F	SATISFACTORY
10	UNSATISFACTORY
11	INTERMEDIATE
12	RESERVED
14	INTERMEDIATE
15	INTERMEDIATE

Part 3. Using RODM for Network Automation

Chapter 7. Writing Automation Code 181
Advantages of Using the Supplied Data Models for Automation 181
Notifying Your Application about Changes in GMFHS Fields 181
Accessing and Changing GMFHS-Defined Fields 182
Using GMFHS Methods 183
  DUIFCCAN: Clear All Notes 183
  DUIFCATC: Aggregation Threshold Change . . 183
  DUIFCLRT: Link Resource Type 183
  DUIFCUAP: Update Aggregation Path 183
  DUIFCUUS: Update User Status 184
  DUIFECDS: Change Display Status 184
  DUIFFAWS: Aggregation Warm Start 184
  DUIFFIRS: Set Initial Resource Status 184
  DUIFFRAS: Recalculate Aggregate Status . . . 184
  DUIFFSUS: Set Unknown Status 184
  DUIFRFDS: Refresh DisplayStatus Change Method DUIFCRDC 185
  DUIFVCFT: Change Exception State 185
  DUIFVINS: Install View Notification Granularity Method 185
  GMFHS Methods That Cannot Be Used . . . 185
GMFHS Automation Example 185
  Sample Automation Application and Method 186

Chapter 8. Using the RODM Automation Platform 189
RODM Automation Platform Services 189
  Sample Automation Code 190

Chapter 7. Writing Automation Code

This chapter describes how you can write automation applications and methods to interface with the data models that are supplied with the NetView program, including the GMFHS data model and the SNA topology manager data model. It also describes the rules and considerations involved in extending a data model that is supplied by the NetView program to meet your automation needs. When you design automation applications involving RODM, you can either design your own data model or use the data models that are supplied with the NetView program.

Advantages of Using the Supplied Data Models for Automation

Though you can create your own data model instead of using the data models that are supplied with the NetView program, consider the following advantages of designing your automation routines around the data models that are supplied with the NetView program:

- The supplied data models are designed to model networks, and if you use them, you avoid the extra step of having to design and implement your own data model, which can be time consuming and costly.
- The supplied data models provide many fields and objects that your automation routines can use, such as the DisplayStatus field. After objects are defined in RODM using the supplied data models, these fields are maintained by NetView code. Because you do not have to write the code to keep the fields up to date, you save resources.
- The NetView management console uses the information in the supplied data models to dynamically construct views of the network for workstation operators who are monitoring the network. Operators make inferences as to the cause of problems, based on the relationships of resources shown in the views and issue commands to initiate corrective action. If you are using the same data model for your automation that operators are using, you can correlate your automation with the people involved in operating and maintaining your network, as well as design automation routines for the network operators' tasks.

The GMFHS data model that is supplied with the NetView product as a RODM load file might not meet all of your automation needs. For example, your automation code might require a line speed field on link objects that is not currently provided by the GMFHS data model. You can modify the shipped source data to meet your needs. Refer to *IBM Tivoli NetView for z/OS Data Model Reference*, which describes all of the classes and fields in the data model, for information about modifying the GMFHS data model.

Notifying Your Application about Changes in GMFHS Fields

RODM can notify user applications when the value of a field in the data model changes. See “RODM Notification Process” on page 318 for a description of how to set up this notification. You can create notification subscriptions for fields on individual objects or for fields on classes. If you create a notification subscription for a field on a class, your user application is notified when that field changes on any object of the class.

The NetView product supplies general purpose notification methods for use with RODM. You can use these notification methods to notify your user application of changes to fields in the data model. Methods are supplied to notify when any change to a field or to notify only when the value of a field exceeds or equals a specified value or values. You first define the notification method on the field of the object or class. Then your application subscribes to the notification queue of that notification method. See “Supplied Methods” on page 480 for a description of these methods. You can also write your own notification methods if the methods that are supplied with the NetView program do not meet your needs.

One useful field for automation is the DisplayStatus field. This field indicates the status of the resource. If you register your automation code on this field, your code is notified by RODM when the status of a resource changes. For example, if the status of a resource changes from satisfactory to unsatisfactory, your code can check the relationship of this object and its status to other objects connected to it in order to determine whether this is a new problem or the symptom of a higher-level problem. The example program in “GMFHS Automation Example” on page 185 performs this task.

Because RODM notifies your automation code when specified fields change, your automation code can focus on analyzing the information provided by the notification and taking appropriate action.

Accessing and Changing GMFHS-Defined Fields

Your automation code can access all fields defined in the GMFHS Data Model to determine the values of these fields. Your automation code can also change some fields. The code must reflect the following rules:

- Do not change the values of class fields. Change values of object fields only. The exceptions to this rule are the CodePage field of the Global_NLS_Parameters_Class and the UnknownThreshold field of the Global_Aggregation_Parameters_Class class
- Do not change the value of the fields of any object that is a descendant of one these classes:
  - Agent_Parent_Class
  - Domain_Parent_Class
  - View_Information_Reference_Class
  - View_Information_Object_Class
- Do not change the value of the DefaultAggregationPriorityCopy field on any objects.
- Do not change the value of the following fields of the GMFHS_Aggregate_Objects_Class:
  - SuspendedCount
  - TotalRealResourceCount
  - StatusGroupCounts
  - PriorityXCPTCount
  - XCPTCount
  - NOXCPTCount
  - UnknownCount
- For GMFHS data model fields on which change methods are installed, your automation code must use the functions which trigger methods. For example, use the EKG_ChangeField or EKG_ChangeMultipleFields functions instead of the EKG_ChangeSubfield function. If the change method is not triggered, operations such as aggregation calculations are not performed.
- GMFHS installs a notification method on all fields used by GMFHS to construct graphical workstation views. Your automation code must use the functions that

trigger methods when it changes fields in the GMFHS data model on which notification methods are installed. For example, use the EKG_LinkTrigger function instead of the EKG_LinkNoTrigger function. If the notification method is not triggered, GMFHS cannot notify operators monitoring views of the change. See the specific field description to determine if GMFHS installs a notification method on the field.

- Some fields must be changed only by using the methods that are supplied with the NetView program and that are designed to change those fields. The methods that can change these fields are described in "Using GMFHS Methods."
- Do not add query methods to fields in the GMFHS data model.
- Do not add change methods to any fields that are created by IBM in the GMFHS data model. You can add change methods to fields you add to the data model.

Using GMFHS Methods

This section briefly describes the GMFHS methods that your automation applications and methods can access. See "GMFHS Methods" on page 488 for more information including the input and output parameters for each method.

DUIFCCAN: Clear All Notes

Use the DUIFCCAN method to clear all note fields without going through the topology console for each real and aggregate object. An operator ID of DUIFCCAN is set to indicate that the note was cleared by this method, instead of an operator.

DUIFCATC: Aggregation Threshold Change

This is a change method installed on the aggregation threshold field of the GMFHS_Aggregate_Objects_Class and is triggered if any of these field's values are changed. Your application does not directly run this method. However, when you design your application, consider that if more than one threshold value is being changed for an object, use the non-triggering (subfield) form of the change request for all but the last change. This eliminates unnecessary triggering of the aggregation calculation method.

DUIFCLRT: Link Resource Type

This object-independent method links Display_Resource_Type_Class objects with real and aggregate objects. This method is intended to be triggered using the RODM INVOKED_WITH load function primitive statement when you create your network definition statements for GMFHS. Use this method for any application that links or unlinks objects of the Display_Resource_Type_Class class with objects of the GMFHS_Managed_Real_Objects_Class class, or its child classes, or with objects of the GMFHS_Aggregate_Objects_Class class. The DUIFCLRT method ensures that the DisplayStatus of aggregate resources is recalculated if necessary because of the link or unlink. See "DUIFCLRT: Link Resource Type Method" on page 489 for a description of the parameters for this method.

DUIFCUAP: Update Aggregation Path

This object-independent method is intended to be run using the INVOKED WITH primitive of the RODM load function. Use this method for any application that is changing the aggregation hierarchy. Use of this method ensures that the count fields and DisplayStatus of aggregate resources is recalculated as required by the change. Note that running the DUIFFAWS method (aggregation warm start) after such a change accomplishes the same thing, but it is more expensive and is intended to be an initialization method.

DUIFCUUS: Update User Status

This named method can be used by an application to update the UserStatus field of objects within the GMFHS_Displayable_Objects_Parent_Class class. While the UserStatus field value can be changed directly, use the DUIFCUUS method to prevent changes that are irrelevant or incorrect, such as suspending aggregation for a shadow object.

DUIFECDS: Change Display Status

This named method can be used by an application to update the DisplayStatus field of objects within GMFHS_Managed_Real_Objects_Class class. This method offers the advantage of checking the SourceStatusUpdateTime field value in the target object against one provided by the invoker to ensure that updates are not applied if the status provided is older than that in the object.

DUIFFAWS: Aggregation Warm Start

Run this object-independent method by any application that needs to ensure that the count and DisplayStatus values of aggregate resources are correct before proceeding. It requires no short-lived parameters.

You might need to run this method if you receive the DUI4020A message with method name DUIFCUAC. This indicates a problem with status being propagated through the aggregation hierarchy. You trigger the DUIFFAWS method when you use the GMFHS CONFIG NETWORK command to reinitialize GMFHS.

You can also trigger this method with the following RODM load function primitive statement: OP DUIFFAWS INVOKED_WITH.

DUIFFIRS: Set Initial Resource Status

This object-independent method is used by GMFHS to set the DisplayStatus of all of the real resource objects linked to the ContainsResource field of a Non_SNA_Domain_Class object to the InitialResourceStatus value of that domain object. You might find this method useful for an application that is initializing and maintaining its own real resource DisplayStatus (in place of GMFHS).

DUIFFRAS: Recalculate Aggregate Status

This object-independent method can be run by any application to cause the DisplayStatus value of all the GMFHS_Aggregate_Objects_Class objects to be recalculated. This method is useful if it is believed that the count fields of the aggregate objects are correct but that the DisplayStatus might be incorrect. The DUIFFRAS method requires no input parameters. If fields other than DisplayStatus might be corrupted, use the DUIFFAWS method instead.

This method can also be triggered with the following RODM load function primitive statement: OP DUIFFRAS INVOKED_WITH.

DUIFFSUS: Set Unknown Status

This object-independent method is used by GMFHS to set the DisplayStatus of all of the real resource objects linked to the ContainsResource field of a Non_SNA_Domain_Class object to the unknown value. You might find this method useful for an application that is initializing and maintaining its own real resource DisplayStatus (in place of GMFHS).

DUIFRFDS: Refresh DisplayStatus Change Method DUIFCRDC

This object-independent method can be called by any application to change the DisplayStatus field to the current DisplayStatus value for every real and aggregate resource defined in RODM. This method is useful when the DisplayStatus mapping table (DUIFSMT) has been changed. Instead of waiting on a status change from the network to trigger an exception view update, method DUIFRFDS can be run to cause the status change which recalculates the exception state for the objects. The appropriate exception views are then updated. For more information, see “Customizing the DisplayStatus Mapping Table for Exception Views” on page 104.

DUIFVCFT: Change Exception State

This object-independent method can be called by a user method to change the exception state of an object. The user method is specified by the USRXMETH keyword in DisplayStatus mapping table DUIFSMT. Sample user methods DUIFCUXM and DUIFCUX2 run method DUIFVCFT to set either value XCPT or NOXCPT in the ResourceTraits field the same way a real DisplayStatus change is processed. DUIFVCFT then triggers a method to determine whether the change in exception state will cause the object to be added to or deleted from any open exception views.

DUIFVINS: Install View Notification Granularity Method

This object-independent method is used by GMFHS to install the view notification granularity method, DUIFVNOT, on a field. See “DUIFVINS: Install View Granularity Method (DUIFVNOT)” on page 499 for a description of this method.

GMFHS Methods That Cannot Be Used

In addition to the GMFHS methods described in this section, GMFHS uses other methods that cannot be used by your programs. See “GMFHS Methods” on page 488 for a list of GMFHS methods that you cannot use.

GMFHS Automation Example

This section presents an automation example, which consists of an application and a method. It is intended to describe how you might set up your own application for automating a complex task. Though this example uses a DisplayStatus field that is defined on the GMFHS_Managed_Real_Objects_Class class, this example applies to any object class that has a DisplayStatus field defined.

In this example, the automation application runs under the NetView product, but an application can also run in its own address space. This example connects to RODM and requests to be notified when the DisplayStatus field of a GMFHS_Managed_Real_Objects_Class object changes in value. This change occurs as a result of an alert coming in for the object that is analyzed by GMFHS.

In this example, the application is registered to be notified if the status changes for either of the two minicomputers contained in the sample network described in Chapter 2, “Defining Your Network to GMFHS,” on page 17 and illustrated in Figure 3 on page 20. When the application determines that the status of one of these resources has changed to unsatisfactory, it runs an object-independent method running under RODM. This method queries the ParentAccess field of the resource whose status has changed and its parents, until it either encounters a resource with Unsatisfactory status or encounters a resource with no ParentAccess link. The method then informs the running application whether or not it has found an ancestor resource that is in an unsatisfactory state.

If the method finds an ancestor resource in an unsatisfactory state, the running application assumes that the alert is a symptom of a higher-level problem and does nothing further. If the method does not find an ancestor resource in an unsatisfactory state, the running application assumes that the alert represents a new problem. In this case, the application might open a problem report for the new problem using the NetView Bridge or issue appropriate commands to bypass the problem. The action taken depends upon the installation, and so is not shown in the code.

The GMFHS automation example is intended to illustrate a possible use of RODM automation and to demonstrate how to write code that uses the RODM interface; do not view this as a solution to a particular automation problem. The program does not check for loops in the parent-child path. The logic of the program is based on the assumption that if a higher-level resource is down, the alert for a lower-level resource is a symptom of that problem, or at least represents a problem that cannot be attended to until the higher-level problem is solved. This assumption is not always valid; its validity depends upon the installation and network resources involved. The example illustrates an automation of the work of GMFHS operators and their inferences and actions as they monitor configuration and status information on workstations.

Sample Automation Application and Method

The CNMSNIFF sample application program accepts a RODM name, a RODM user name, and a RODM password from the NetView command line. The application then uses the three parameters to perform the following functions:

1. Sends a connect request to the specified RODM.
2. Subscribes to the DisplayStatus fields of objects that represent resources in the SYSPLEX network.
3. Issues EKGWAIT and waits for the DisplayStatus fields of objects that represent resources in the SYSPLEX network to change.
4. Triggers the EKGSNIFF sample object-independent method when one or more DisplayStatus fields change.
5. The sample code does no processing at this step. If you were creating a working automation application, you might create appropriate code for your system to correct the problem or to log a problem record based on the return and reason code returned by the EKGSNIFF method after the EKGSNIFF method finishes processing.
6. Issues EKGWAIT and waits until either a problem occurs or RODM ends.

The CNMSNIFF application is written in C and runs in the NetView address space. The source code for this example application is shipped as a NetView sample. The sample name is CNMS4402 (alias CNMSNIFF) in data set CNMSAMP.

The EKGSNIFF sample object-independent method is triggered by the CNMSNIFF sample automation application program. The EKGSNIFF method accepts an ObjectID of the target object as a parameter. When triggered, the EKGSNIFF method queries the DisplayStatus fields of the target object and the object's parent. The method then returns a return and reason code, based on the values of the

DisplayStatus fields of the target object and its parent, to the CNMSNIFF automation program that is in the transaction information block.

The source code for the EKGSNIFF method is shipped as a NetView sample. The sample name is CNMS4403 (alias EKGSNIFF) in data set CNMSAMP.

Chapter 8. Using the RODM Automation Platform

This chapter is an overview of the RODM automation platform. The *RODM automation platform* is a set of NetView services that make automation using RODM easier.

Additional information about the RODM automation platform is contained in the *IBM Tivoli NetView for z/OS Automation Guide*. This book also contains an extensive RODM automation scenario which shows how the automation platform can be used.

RODM Automation Platform Services

The following services make up the RODM automation platform:
- DSIQTSK task
- ORCONV command
- EKGSPPI method
- CNMQAPI service routine
- DSINOR service routine
- ORCNTL command

The DSIQTSK task is dedicated to communicating with the RODM address space. It receives command requests from EKGSPPI and dispatches the commands to an autotask. Each RODM that you want to manage from the NetView address space must be defined to DSIQTSK.

The ORCONV command enables the NetView automation table, command lists, and applications to issue requests to RODM that change values of fields and trigger methods. The ORCONV command requires that the DSIQTSK task is running in the NetView from which the commands are issued, and that RODM is defined to the DSIQTSK task.

The EKGSPPI method that is supplied with the NetView program sends commands from RODM to the DSIQTSK task in the NetView product using the program-to-program interface. See “EKGSPPI: Send a command to NetView” on page 485 for a description of the EKGSPPI method.

The CNMQAPI service routine is an enhanced API that enables applications in the NetView address space to issue RODM functions with less programming effort. CNMQAPI can be used with the PL/I and C high-level languages. CNMQAPI enables an application to issue requests while RODM is processing a checkpoint request. CNMQAPI queues the requests and sends them to RODM when the checkpoint process is complete. Refer to the *IBM Tivoli NetView for z/OS Programming: PL/I and C* for the syntax of CNMQAPI.

The DSINOR assembler-language macro provides an API like CNMQAPI for assembler applications running in the NetView address space. Refer to the *IBM Tivoli NetView for z/OS Programming: Assembler* for the syntax of DSINOR.

The ORCNTL command manages the administrative details about the RODMs defined to the DSIQTSK task. See the ORCNTL command in NetView online help for more information.

Sample Automation Code

The NetView product supplies sample code that you can use to learn how to use some of the RODM automation platform services. This sample code is found in the NETVIEW.V6R1M0.CNMSAMP sample library as follows:

CNMS4230
This sample shows you how to use the CNMQAPI service routine when programming with the PL/I language.

CNMS4260
This sample shows you how to use the CNMQAPI service routine when programming with the C language.

CNMS4290
This sample shows you how to use the DSINOR assembler-language macro.

Part 4. Application Programming Using RODM

Chapter 9. Understanding RODM Concepts . . 195
RODM Classes 195
  Class Names 195
    Class Name Characteristics with CHARACTER_VALIDATION(YES) 195
    Class Name Characteristics with CHARACTER_VALIDATION(NO) 196
  System-Defined Classes 196
    UniversalClass 197
    EKG_SystemDataParent Class 198
    EKG_System Class 198
    EKG_User Class 201
    EKG_NotificationQueue Class 204
    EKG_Method Class 206
RODM Objects 208
  Object Names 208
    Object Name Characteristics with CHARACTER_VALIDATION(YES) 209
    Object Name Characteristics with CHARACTER_VALIDATION(NO) 209
  Object Identifiers 210
RODM Fields 210
  Field Names. 210
    Field Name Characteristics with CHARACTER_VALIDATION(YES) 210
    Field Name Characteristics with CHARACTER_VALIDATION(NO) 210
  Field Identifiers. 211
  System-Defined Fields 211
RODM Subfields 213
  Data Types for Subfields. 215
Multivalued Fields and Links between Objects . . 216
  Link and Unlink Action Functions 218
  Subfields Associated with Fields 219
Indexed Fields 220
Object and Class Locking in RODM 220
Using the Application Program Interfaces 220
  User Application Program Interface (API) . . . 220
  Method Application Program Interface (API) 221
RODM Abstract Data Types 221
  Null Values of Data Type 222
  Data Type Identifiers 222
  Types of Data in Fields 222
  Abstract Data Type Reference 223
    Anonymous(N) (Reserved) 223
    AnonymousVar. 223
    ApplicationID (Reserved) 224
    BERVar 224
    CharVar 226
    CharVarAddr (Reserved) 227
    ClassID (Reserved) 227
    ClassIDList (Reserved) 227
    ClassLinkList (Reserved) 228
    ECBAddress (Reserved) 228
    FieldID 228
    Floating 229
    GraphicVar 229
    Integer 229
    IndexList 230
    MethodName (Reserved) 230
    method_parameter_list (Reserved) 231
    MethodSpec 231
    ObjectID (Reserved) 231
    ObjectIDList (Reserved) 232
    ObjectLink 232
    ObjectLinkList 232
    ObjectName (Reserved) 233
    RecipientSpec (Reserved) 233
    SelfDefining 234
    ShortName (Reserved) 235
    Smallint 235
    SubscribeID (Reserved) 235
    SubscriptSpec (Reserved) 236
    SubscriptSpecList (Reserved) 236
    TimeStamp 236
    TransID (Reserved) 237

Chapter 10. Using the RODM Load Function 239
Considerations When Designing a Data Model . . 239
Introduction to the RODM Load Function 240
  Load Function Statements 240
  Load Function Operations 240
  Loading the RODM Data Cache 241
Using Load Function Statements 241
  High-Level Load Function Statements 242
  Load Function Primitive Statements 242
  When to Use High-Level or Primitive Load Function Statements 243
Process for Loading the RODM Data Cache . . . 244
  Identifying the Methods to Install 245
  Creating the Class Structure and Object Definitions 245
    Data Definition Statement Labels 245
    Concatenation of Data Sets 246
    Definition Examples 246
  Deciding on the Type of Load 246
    Initialization Load 246
    Structure Load Only 247
    Object Load Only 248
  Running the RODM Load Function 248
    The Load Function as an Initialization Method 248
    Invoking the Load Function As a Batch Job 250
    Calling the Load Function from a Module 251
    Considerations When Running the RODM Load Function 252
  Checking the Output Listings 253
    RODM Load Function Output Listing . . . 253
    RODM Load Function Output Format . . . 254
Load Function Reference 258
  Understanding the Verify Operation. 258
  Using CLASSID and OBJECTID Data Types . . 259

    - CLASSID . . . 259
    - OBJECTID . . . 259
  - Null Values for RODM Load Function Data Types . . . 260
  - Control Table—EKGCTABL. . . . 260
    - Relationships to Other Tables and DD Names 260
  - Method Name Table . . . 261
    - Associated DD Statements and Control Table 262
  - Parameter Mapping Table . . . 262
  - RODM Data Definition (DD) Statements . . . 264
    - Data Definitions Necessary for Initialization 266
    - Data Definitions Necessary for Structure Load Only . . . 266
    - Data Definitions Necessary for Object Load Only . . . 266
  - z/OS Linkage Conventions . . . 266
    - Parameter Structure . . . 267
    - DD List Structure . . . 267
    - Access Block . . . 268
    - Calling the RODM Load Function . . . 268
  - RODM Load Function Parameter Syntax . . . 269
    - CODEPAGE . . . 270
    - LISTLEVEL . . . 270
    - LOAD . . . 270
    - NAME . . . 271
    - OPERATION . . . 271
    - ROUTECODE . . . 272
    - SEVERITY . . . 273
  - Coding RODM High-Level Load Function Statements . . . 273
    - Syntax Rules for High-Level Load Function Statements . . . 273
    - Syntax for High-Level Load Function Statements . . . 275
  - Coding RODM Load Function Primitive Statements . . . 282
    - Global Character . . . 282
    - Syntax Rules for Load Function Primitives 282
    - Syntax and Processing Logic for Load Function Primitives . . . 282
  - Common Syntactic Elements . . . 291
    - Syntax for Common Syntactic Elements . . . 291

Chapter 11. Writing Applications that Use RODM . . . 301

- Tasks Best Performed with User Applications. . . 301
- Using the User Application Program Interface . . . 302
  - Register Conventions . . . 302
  - Usage Notes. . . . 302
  - Compiling and Link-Editing . . . 303
    - Compiling C Modules that Call EKGUAPI 303
    - Compiling PL/I Modules that Call EKGUAPI 303
    - Linking Modules that Call EKGUAPI Directly . . . 304
    - Linking Modules that Load and then Call EKGUAPI . . . 304
- Using Control Blocks . . . 304
  - Access Block . . . 305
  - Transaction Information Block . . . 307
  - Function Block . . . 308
  - Entity Access Information Block . . . 309
  - Field Access Information Block . . . 312
  - Response Block. . . . 314
- Error Conditions in Transactions . . . 317
- RODM Notification Process . . . 318
  - Setup . . . 319
  - Wait . . . 321
    - Calling EKGWAIT . . . 321
    - PL/I Coding Example . . . 322
    - C Coding Example . . . 323
    - EKGWAIT Usage Notes . . . 323
  - Notification . . . 324
  - Clean Up. . . . 325
- Asynchronous Error Notification . . . 325
- Object Deletion Notification . . . 326
  - Setup for Object-Deletion Notification . . . 326
  - Wait for Object-Deletion Notification . . . 327
  - Notification for Object-Deletion Notification . . 327
  - Cleanup for Object-Deletion Notification . . . 327
- Connecting to RODM . . . 327
- Disconnecting from RODM. . . . 328

Chapter 12. Topology Object Correlation . . . 329

- Enabling the Correlation Function . . . 329
  - Enabling SNA Topology Manager Object Correlation . . . 329
  - Enabling GMFHS Object Correlation . . . 330
- Correlation Concepts . . . 330
  - Correlation Methods . . . 330
    - FLCMCONI Method . . . 330
    - FLCMCON Method . . . 330
    - FLCMCOR Method . . . 331
  - Objects Enabled for Correlation . . . 331
  - Types of Correlation . . . 331
    - Network Address Correlation . . . 331
    - Free-Form Correlation . . . 331
  - Correlated Aggregate Object Classes and Names 333
  - Correlated Object Relationships . . . 333
  - Correlated Aggregate Object Display Labels . . 333
  - Correlated Aggregate Object Field Values . . . 334
- Using Correlation for Objects You Create . . . 335
- Extending Correlation of Objects Created by MultiSystem Manager and SNA Topology Manager 335
  - How to Determine Object Names. . . . 336
  - Correlating MultiSystem Manager Objects . . . 336
  - Correlating SNA Topology Manager Objects . . 336
- Customizing the Correlation Function . . . 336
  - Changing the Display Name Priority . . . 337
  - Disabling Correlation for Specific Resources . . 338

Chapter 13. Writing RODM Methods . . . 339

- Tasks Best Performed with Methods. . . . 339
- Types of Methods . . . 340
  - Object-Independent Methods . . . 340
    - Initialization Method . . . 341
  - Object-Specific Methods . . . 342
    - Change Methods . . . 342
    - Query Methods. . . . 344
    - Notify Methods . . . 346
    - Named Methods . . . 349
    - Inheritance in Object-Specific Methods . . . 350
  - Null Method . . . 352

Deciding Which Method Type to Use 352
When to Use an Object-Independent Method 352
When to Use an Object-Specific Method . . . 352
Query Method 353
Change Method 353
Notify Method 353
Named Method 353
Using the Method API 353
Register Conventions 354
Usage Notes. 355
Method Parameters 355
Long-Lived Parameters 355
Short-Lived Parameters 356
Installing and Freeing Methods 356
Synchronous and Asynchronous Execution of Functions. 357
Method Anchor Service 357
Coding Your RODM Method 358
Installation Written Methods 358
NetView Methods 358
Programming Language Specific Preprocessor Statements 359
Compiling IBM C Methods 359
Compiling IBM PL/I Methods. 360
Linking Methods that Call EKGMAPI Directly 360
Restrictions on Methods 360
PL/I Language Restrictions. 360
C Language Restrictions 362
Restrictions in General 362
RODM Method Services 363
Services Available to both Object-Specific and Object-Independent Methods 363
Other Services Available to Object-Independent Methods 364
Other Services Available to Object-Specific Methods 364
Services Available to the Initialization Method 364
RODM Method Library 365

Chapter 14. Application Programming Reference 367
Summarizing RODM Functions 367
Access Functions 367
Control Functions 367
Administrative Functions 367
Action Functions 368
Query Functions 369
RODM User API Services 370
RODM Method API Services 370
Function Reference 371
Function Reference Format 371
Purpose 371
Function block format 371
Examples. 371
Summary. 372
Usage Notes. 372
EKG_AddNotifySubscription — Add Notification Subscription 373
EKG_AddObjDelSubs — Add Object Deletion Subscription 374
EKG_ChangeField — Change a Field 376
EKG_ChangeMultipleFields — Change Multiple Fields 377
EKG_ChangeSubfield — Change a Subfield . . 378
EKG_Checkpoint — Checkpoint RODM to DASD 380
EKG_Connect — Connect to RODM. 383
EKG_ConnectLong — Connect to RODM . . . 384
EKG_CreateClass — Create a Class 386
EKG_CreateField — Create a Field 387
EKG_CreateObject — Create an Object 388
EKG_CreateSubfield — Create a Subfield . . . 390
EKG_DeleteClass — Delete a Class 391
EKG_DeleteField — Delete a Field 392
EKG_DeleteNotifySubscription — Delete Notification Subscription 393
EKG_DeleteObject — Delete an Object 395
EKG_DeleteSubfield — Delete a Subfield . . . 396
EKG_DelObjDelSubs — Delete Object Deletion Subscription 397
EKG_Disconnect — Disconnect from RODM 399
EKG_ExecuteFunctionList — Execute a List of Functions. 400
EKG_LinkNoTrigger, EKG_LinkTrigger — Link Two Objects 402
EKG_Locate—Locate Objects Using Public Indexed Field 404
EKG_LockObjectList — Lock List of Objects . . 405
EKG_MessageTriggeredAction — Trigger an Action by a Message 406
EKG_OutputToLog — Output to Log 408
EKG_QueryEntityStructure — Query Structure of an Entity 409
EKG_QueryField — Query a Field 411
EKG_QueryFieldID — Query Field Identifier 412
EKG_QueryFieldName — Query a Field Name 413
EKG_QueryFieldStructure — Query Structure of a Field. 415
EKG_QueryFunctionBlockContents — Query Function Block Contents. 416
EKG_QueryMultipleSubfields — Query Multiple Value Subfields. 418
EKG_QueryNotifyQueue — Query Notification Queue. 421
EKG_QueryObjectName — Query Object Name 423
EKG_QueryResponseBlockOverflow — Query for Response Block Overflow 424
EKG_QuerySubfield — Query a Subfield . . . 426
EKG_ResponseBlock — Output to Response Block 428
EKG_RevertToInherited — Revert to Inherited Value 429
EKG_SendNotification — Send a Notification 431
EKG_SetReturnCode — Set Return and Reason Codes 432
EKG_Stop — Stop RODM 434
EKG_SwapField — Swap a Field 435
EKG_SwapSubfield — Swap a Subfield. . . . 437
EKG_TriggerNamedMethod — Trigger a Named Method 438

EKG_TriggerOIMethod — Trigger an Object-Independent Method 440
EKG_UnlinkNoTrigger, EKG_UnlinkTrigger — Unlink Two Objects 441
EKG_UnlockAll — Unlock All Held Entities . . 443
EKG_WhereAmI — Where Am I 444
Function Parameter Descriptions 445
RODM Return and Reason Codes 452
Reason Codes for Return Code 0 453
Reason Codes for Return Code 4 453
Reason Codes for Return Code 8 457
Reason Codes for Return Code 12 467
List of Reason Codes for Each Function . . . 470
List of Functions for Each Reason Code . . . 472
List of Function Names by Function ID. . . . 478
List of Reason Codes from Supplied Methods 479
Maximizing RODM Performance 480
Data Model Structure and Size 480
Method Design 480
User Application Design. 480
Customization Parameters and System Fields 480
Indexed Fields 480
Supplied Methods 480
RODM Notification Methods 481
EKGNOTF: General Notification 482
EKGNEQL: Notify If Equal. 482
EKGNLST: Notify if Equal to List 483
EKGNTHD: Notify If Outside Threshold . . 483
RODM Change Methods 484
EKGCTIM: Trigger Object-Independent Method 484
RODM Named Methods. 485
EKGMIMV: Increment Value 485
EKGCTIM: Trigger Object-Independent Method 485
RODM Object-Independent Methods 485
EKGSPPI: Send a command to NetView . . 485
GMFHS Methods 488
DUIFCCAN: Clear All Notes 489
DUIFCLRT: Link Resource Type Method . . 489
DUIFCUAP: Update Aggregation Path Method 491
DUIFCUUS: Update User Status Method . . 492
DUIFECDS: Change Display Status Method 494
DUIFFAWS: Aggregation Warm Start Method 495
DUIFFIRS: Set Initial Resource Status Method 496
DUIFFRAS: Recalculate Aggregate Status Method 497
DUIFFSUS: Set Unknown Status Method . . 497
DUIFRFDS: Refresh DisplayStatus Change Method DUIFCRDC 498
DUIFVCFT: Change Exception State 498
DUIFVINS: Install View Granularity Method (DUIFVNOT) 499

Chapter 9. Understanding RODM Concepts

This chapter describes the structure of the RODM data cache, methods, and applications. This chapter will help you understand RODM concepts so that you can create your own data models and associated methods and applications.

This chapter explains the RODM abstract data types. These data types, such as Integer and MethodSpec define the format of data stored in RODM.

RODM Classes

The ability to group objects and the ability to group or arrange groups of objects is useful in network management. RODM implements this concept of grouping through the use of *classes*. Classes define the data structure of the data cache.

A class represents a grouping and defines fields for all classes and objects below that class. If you view the RODM data cache as a tree structure, classes represent the branches of the tree with the UniversalClass as the top-most class. Figure 37 on page 196 shows an example of the tree structure.

RODM classes:

- Can have:
  - No children
  - Class children only
  - Object children only
  - Both class and object children
- Define the complete data organization for their class children or for their object children.
- Consist of public fields that contain data for the object.
- Include private fields that are not inherited.
- Define the inheritance structure.

Class Names

Each RODM class has a character string in its MyName field called the *class name*. RODM system-defined class names are reserved by RODM and cannot be deleted. All system-defined names, except for UniversalClass, begin with `EKG_`.

The CHARACTER_VALIDATION keyword in EKGCUST specifies what degree of validity checking RODM performs for characters used in object names (see “Object Names” on page 208), field names (see “Field Names” on page 210), and class names.

Class Name Characteristics with CHARACTER_VALIDATION(YES)

When CHARACTER_VALIDATION(YES), which is the default, is coded in EKGCUST, valid class names have the following characteristics:

- The name consists of 1 to 64 characters that conform to the ShortName data type with the PL/I syntax of `CHAR(64) VARYING`.
- The first character of the string must be alphabetic or numeric. The others, if any, can be alphabetic, numeric, the break character (_), the commercial “at” sign (@), the number sign (#), or the period (.).

- The `EKG_` prefix is reserved for RODM created classes. Do not use this prefix in the names of classes that you create.
- Both uppercase and lowercase alphabetic characters are permitted, and names are case-sensitive.
- Each class name in the RODM data cache is unique. RODM supports a maximum of 4,079 classes.

Class Name Characteristics with CHARACTER_VALIDATION(NO)

When CHARACTER_VALIDATION(NO) is coded in EKGCUST, valid class names have the following characteristics:

- The name consists of 1 to 64 characters that conform to the ShortName data type with the PL/I syntax of `CHAR(64) VARYING`.
- The first character cannot be the number sign (#) because it is reserved for MultiSystem Manager.
- Blank characters are not valid.
- Null characters are not valid.
- The `EKG_` prefix is reserved for RODM created classes. Do not use this prefix in the names of classes that you create.
- Both uppercase and lowercase alphabetic characters are permitted, and names are case-sensitive.
- Each class name in the RODM data cache is unique. RODM supports a maximum of 4,079 classes.

System-Defined Classes

When RODM is cold started, RODM initialization occurs and the class definitions are created. This data model provides the starting point for all RODM classes and objects. These *system-defined classes* enable users to access information about their application and about RODM itself. Figure 37 shows the RODM system-defined classes and their hierarchy.

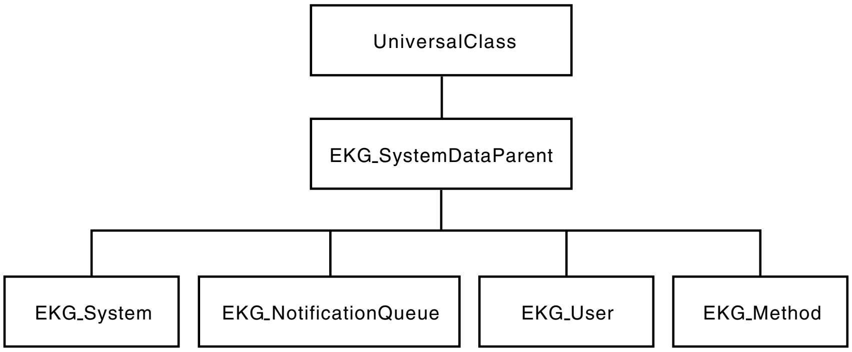


Figure 37. RODM System-Defined Classes

RODM has the following system-defined classes:

UniversalClass
The root of the inheritance tree structure of the RODM data cache

EKG_SystemDataParent
The system data parent class, the parent of all RODM predefined system classes

EKG_System
The system object class, all the RODM system data created by RODM when you start RODM

EKG_User
The user object class, the fields and methods that RODM creates when an application connects to RODM

EKG_NotificationQueue
The notification queue object class, the fields and methods that RODM creates when an application creates a notification queue

EKG_Method
The method object class, the fields and methods that RODM creates when you install a method

The following six sections describe the six RODM system-defined classes. Information, which is common to all six classes, includes the following:

- The fields that are created by RODM and can be accessed by application programs and methods.
- The subfields that are created by RODM on system-defined fields. User applications cannot add subfields to fields of system-defined classes. You can add notification subscriptions to the specified fields using the EKG_AddNotifySubscription function.
- The specification of the notify subfield identifies the fields to which an application can subscribe for notification. RODM notifies each application which has subscribed to a field when the value of the field changes.
- Applications can change write-access fields only.
- Applications can change values in the fields of objects only.

UniversalClass

UniversalClass is the RODM universal class, the root of the hierarchy of RODM classes. All classes and objects are descendents of the universal class. Each class and object in RODM inherits the fields of the UniversalClass. The contents of these fields are not inherited, just the field definitions.

The UniversalClass has no parent.

Table 17 describes the fields of UniversalClass, the access for each field, the data type of the field, and the subfields defined on each field.

Table 17. UniversalClass Fields

Field Name	Access	Data Type	Query	Change	Notify	Time stamp
MyName	Read	ObjectName or ShortName	X			
MyID	Read	ObjectID or ClassID	X			
MyPrimaryParentName	Read	ShortName	X			
MyPrimaryParentID	Read	ClassID	X			
WhatIAm	Read	Enumerated Integer	X		X	
MyClassChildren	Read	ClassIDList	X		X	
MyObjectChildren	Read	ObjectIDList	X		X	

The UniversalClass fields are as follows:

MyName
The name of the object or class. The data type of this field is ObjectName when the field is created for an object, and ShortName when the field is created for a class. You supply the class name or object name when you create the class or object.

MyID
The numerical identifier of the object or class assigned by RODM. When you create a class or object in RODM, you supply RODM with the name of the class or object. RODM then assigns a numerical identifier to the class or object. It is more efficient to refer to a class by its class ID and to refer to an object by its object ID than it is to refer to them by their names.

MyPrimaryParentName
The name of the class of this object, or the name of the parent class of this class

MyPrimaryParentID
The ID of the class of this object, or the ID of the parent class of this class

WhatIAm
This field indicates the type of object or class. The values that are valid follow:

Value	Meaning
1	Object
2	Class with no children
3	Class with object children
4	Class with class children
5	Class with class and object children

MyClassChildren
A list of the class children of this class, which is valid when the value of the WhatIAm field is 4 or 5. This field is set to the null value when the class has no class children.

MyObjectChildren
A list of the object children of this class, which is valid when the value of the WhatIAm field is 3 or 5. This field is set to the null value when the class has no object children.

EKG_SystemDataParent Class

EKG_SystemDataParent is the parent class of all RODM system data.

The EKG_SystemDataParent class provides a named parent for all of the system data classes and objects that RODM creates. It separates the system-defined classes from all other classes defined under the UniversalClass.

The parent of the EKG_SystemDataParent is the UniversalClass.

SystemDataParent inherits all of its fields from the UniversalClass. All fields in EKG_SystemDataParent are read access only.

EKG_System Class

The EKG_System class is a child of the EKG_SystemDataParent class and contains all of RODM's system data.

At cold start, RODM creates the EKG_System class and one object of the EKG_System class. The object contains system data for this RODM.

When RODM is warm started, RODM updates most of the EKG_System fields. The EKG_TransSegment and EKG_WindowSize fields retain the values they contained at the last checkpoint. Any user-defined fields or subscriptions you add to this class also retain their values from the last checkpoint.

Initial values for some of the fields in EKG_System are read from the RODM customization file when RODM is started. Refer to the *IBM Tivoli NetView for z/OS Administration Reference* for information about the RODM customization file.

Table 18 describes the fields of the EKG_System class, the access for each field, the data type, and the subfields for each field.

Table 18. EKG_System Fields

Field Name	Access	Data Type	Query	Change	Notify	Time stamp
EKG_Name	Read	CharVar				X
EKG_APIVersion	Read	Integer				
EKG_ReleaseID	Read	CharVar				
EKG_ExternalLogState	Write	Enumerated Integer			X	X
EKG_LastCheckpointID	Read	TransID			X	X
EKG_LastCheckpointResult	Read	SelfDefining			X	X
EKG_LastAsyncError	Read	AnonymousVar			X	
EKG_AsyncTasks	Read	Integer				
EKG_ConcurrentUsers	Read	Integer				
EKG_PLI_ISA	Read	Integer				
EKG_SSBChain	Read	Integer				
EKG_TransSegment	Read	Integer				
EKG_WindowSize	Read	Integer				

The field definitions are as follows:

`EKG_Name`
RODM name. This field contains the name of this RODM. RODM sets the timestamp subfield of this field to the time at which RODM was started.

`EKG_APIVersion`
The API version. This field contains the latest API level supported by this RODM.

`EKG_ReleaseID`
The release level. For service, RODM generates a string that identifies the version and release in the form *product_acronym version release*. The current value of this field is `RODMN610`. A value of `RODMN610` indicates Tivoli NetView for z/OS V6R1.

`EKG_ExternalLogState`
The administrative state (log or no log) for external logging. You can dynamically control logging to the RODM log by changing this field. The following values are valid:

Value	Meaning
1	Log
2	No log

This logging applies only to the external file data set. When the external log is full, RODM automatically switches to the secondary log if one was allocated. Otherwise, RODM overwrites the primary log.

EKG_LastCheckpointID
The transaction ID of the last successful checkpoint operation. User applications can subscribe to this field for successful checkpoint notification because this field is only updated on a successful checkpoint. Applications can query the timestamp subfield of this field for the time of the last successful checkpoint. During warm start operation, RODM initializes this field to the last transaction ID contained in the checkpoint files from before the warm start.

EKG_LastCheckpointResult
A SelfDefining value as shown in Table 19 that indicates the status and a transaction ID for the last checkpoint attempt, including canceled checkpoints.

If the checkpoint is requested by a checkpoint MODIFY command, RODM updates this field with the current transaction ID. Otherwise, the transaction ID is that of the requesting User API.

User applications can subscribe to the EKG_LastCheckpointResult system field for the notification of checkpoint attempt completions. Applications can query the field for the return_code and reason_code to determine success, and if unsuccessful the reason for failure. Applications can also query the timestamp subfield of this field for the time of the last checkpoint attempt.

Table 19. EKG_LastCheckpointResult System Field

Offset	Length	Type	Use	Parameter
000	2	Integer	—	Length of SelfDefining
002	2	Integer	—	Data type identifier
004	4	Integer	Out	Return_code
008	2	Integer	—	Data type identifier
010	4	Integer	Out	Reason_code
014	2	Integer	—	Data type identifier
016	8	TransID	Out	Transaction_ID

EKG_LastAsyncError
The last asynchronous error that occurred in RODM. Applications can subscribe to this field for notification of any asynchronous error occurring within RODM. When an asynchronous error occurs, RODM puts a copy of the log record created for the error into this field. RODM might or might not actually write the record to the RODM log.

An asynchronous error is an error in a RODM function or method which is running asynchronously. Functions which are executed using the EKG_MessageTriggeredAction function run asynchronously. Methods can also run asynchronously.

RODM also defines an EKG_LastAsyncError field on the EKG_User class. EKG_LastAsyncError on EKG_System contains the last error for any user of RODM. EKG_LastAsyncError on EKG_User contains the last error for the user of RODM defined by a particular object under EKG_User.

EKG_AsyncTasks
Maximum number of asynchronous tasks. This field specifies the maximum number of asynchronous tasks that can be active concurrently.

This field is filled in from the ASYNC_TASKS operand in the RODM customization file at warm start and at cold start.

`EKG_ConcurrentUsers`
Maximum number of concurrent users. This field specifies the maximum number of users that can have an active transaction concurrently executing within the RODM address space.

This field is filled in from the CONCURRENT_USERS operand in the RODM customization file at warm start and at cold start.

`EKG_PLI_ISA`
PL/I initial storage area. This field specifies the size of the initial storage area preallocated for each PL/I environment.

This field is filled in from the PLI_ISA operand in the RODM customization file at warm start and at cold start.

`EKG_SSBChain`
SSB chain size. This field specifies the number of same-name system status blocks (SSBs) that can exist concurrently. These entries contain RODM activation records.

This field is filled in from the SSB_CHAIN operand in the RODM customization file at warm start and at cold start.

`EKG_TransSegment`
Translation segment size. This field specifies the size of the RODM translation segment in millions of bytes. The translation segment is used to store internal RODM tables.

This field is filled in from the TRANS_SEGMENT operand in the RODM customization file at cold start only.

`EKG_WindowSize`
Data window size. This field specifies the size of the RODM data windows. The data windows are used for storing RODM data.

This field is filled in from the WINDOW_SIZE operand in the RODM customization file at cold start only.

EKG_User Class

EKG_User is the class of application programs that use RODM. This class defines the fields of the objects that represent application programs connected to RODM. An application can query its EKG_User object to get information about itself.

The parent of EKG_User is EKG_SystemDataParent.

When an application connects to RODM, RODM creates an object of the EKG_User class to represent that application. When the application disconnects from RODM, RODM deletes the object. If an application has notification queues or subscriptions defined, RODM deletes the object in EKG_User based on the value of the EKG_StopMode field of that object.

Initial values for some of the fields in EKG_User are read from the RODM customization file when RODM is started. Refer to the *IBM Tivoli NetView for z/OS Administration Reference* for information about the RODM customization file.

At warm start, RODM sets the status of all EKG_User objects to disconnected. RODM then deletes any objects that do not have notification queues.

An EKG_User object inherits the fields of the UniversalClass through the EKG_SystemDataParent class and the EKG_User class. Query the MyObjectChildren field of the EKG_User class to get a list of applications connected to RODM.

Table 20 describes the fields of EKG_User class, the access for each field, the data type, and the subfields defined for each field.

Table 20. EKG_User Fields

Field Name	Access	Data Type	Query	Change	Notify	Time stamp
EKG_Status	Read	Enumerated Integer			X	X
EKG_StopMode	Write	Enumerated Integer			X	
EKG_LastAsyncError	Read	AnonymousVar			X	
EKG_Uses_Q	Read	ObjectLinkList				
EKG_RBOverflowAction	Write	Enumerated Integer			X	
EKG_LogLevel	Write	Integer				
EKG_MLogLevel	Write	Integer				
EKG_MTraceType	Write	4-Byte Integer				

The field definitions are as follows:

EKG_Status

The current user application status. RODM updates the timestamp subfield of EKG_Status each time status changes. Query the timestamp subfield to determine the time of connection to RODM. Valid values are as follows:

Value	Meaning
1	Connected
2	Disconnected
3	Unknown

EKG_StopMode

The stop mode. This field specifies the processing that RODM does for a user application when the user application disconnects. The default action is to purge all notification queues and all subscriptions. Your application programs can change the setting of this field to specify that RODM purge only the notification queues or to purge nothing. Valid values are as follows:

Value	Meaning
1	Purge notification queues and subscriptions
2	Purge notification queue elements only
3	Do not purge notification queues or subscriptions

If one of your applications disconnects with a setting that preserves queues, subscriptions, or both, and then some event changes this field while your application is disconnected, the new setting of the field has immediate effect. But if the new setting is to preserve the queues, the subscriptions, or both, the new setting cannot take effect until your application reconnects and establishes new queues and subscriptions.

Purging queues without purging subscriptions causes RODM to purge only the data associated with notification queues. RODM retains the

EKG_NotificationQueue object. If your application or RODM purges all of the subscriptions for a specified queue, RODM also purges the EKG_NotificationQueue object for that queue.

`EKG_LastAsyncError`

Last asynchronous error. Users can subscribe to this field for notification of any asynchronous error associated with transactions that this user ID has initiated. When RODM logs an error, it writes a copy of the error record into this field, even if it does not write the error record to the RODM log. RODM then notifies the users subscribed to this field.

RODM also defines an EKG_LastAsyncError field on the EKG_System class. EKG_LastAsyncError on EKG_System contains the last error for any user of RODM. EKG_LastAsyncError on EKG_User contains the last error for the user of RODM defined by a particular object under EKG_User.

`EKG_Uses_Q`

A list of links to notification queue objects. This list contains a link for each queue specified by a notification subscription for this user. RODM creates the links in this list in response to subscription requests. The link is between the EKG_Uses_Q field of the User object and the EKG_UsedBy field of the EKG_NotificationQueue object.

`EKG_RBOverflowAction`

Response block overflow action control. Valid values are as follows:

Value	Meaning
1	Save
2	Discard

If your application sets the value of this field to save, RODM automatically collects response block overflow data in a buffer. Your application then must get the overflow data from the buffer before it can query other data. If your application sets the value of this field to discard, RODM discards any overflow data. If the value of this field is changed from save to discard, RODM immediately discards all collected overflow data associated with the User_appl_ID. The default value for this field is save.

If a single user is running concurrent transactions through multitasking and one thread causes a response block overflow and a different thread changes this field to discard, the transaction causing the overflow might receive a return code indicating the overflow. However, the overflow data is discarded.

`EKG_LogLevel`

Logging level control for user API functions. After the processing of a transaction is complete, this parameter determines whether or not to write a log record to record this transaction. The basis of the log control is the transaction return code. If the transaction return code is greater than or equal to EKG_LogLevel, RODM writes a log record. Your application can override the default value for the class by specifying a new value in this field. If your application specifies a value of 0, RODM writes for that application a log record of all transactions across the user API.

RODM reads the customization file to determine the default value to assign to the class level field. If the customization file contains a LOG_LEVEL parameter, the value of that parameter determines the class default value. If the customization file does not contain a value for LOG_LEVEL, the default value of 8 is used.

`EKG_MLogLevel`

Specifies the log level for tracing method API function calls. RODM generates a log record when the return code from a method API function call is greater than or equal to the value of EKG_MLogLevel.

This field is filled in from the MLOG_LEVEL operand in the RODM customization file at warm start and at cold start.

`EKG_MTraceType`

Specifies whether RODM traces method entry and exit and specifies the type of methods RODM traces. This field is filled in from the MTRACE_TYPE operand in the RODM customization file at warm start and at cold start.

The first three bytes of EKG_MTraceType are always X'000000'. The right-hand byte is used as seven flag bits:

Bit	**Meaning if bit is set**
1...	Trace object deletion methods
.1..	Trace object independent methods
..1.	Trace named methods
...1	Trace notify methods
.... 1...	Trace change methods
.... .1..	Trace query methods
.... ..1.	Trace method exit and storage
....1	Trace method entry and storage

You can set any combination of these 7 bits. If the trace method entry and trace method exit bits are both zero, method tracing is inactive. If all bits are zero, all tracing is inactive.

RODM generates a log record when method entry or method exit tracing is specified.

The EKG_MTraceFlag field for each method object, in addition to the corresponding method-type bit in EKG_MTraceType, specifies whether a method is enabled for tracing. If either the corresponding method-type bit in EKG_MTraceType is set or the EKG_MTraceFlag field in the associated method object is one, the method is traced.

EKG_NotificationQueue Class

EKG_NotificationQueue is the class of notification queues. Notification queues are used for the RODM notification process. See "RODM Notification Process" on page 318 for more information about notification.

The parent is EKG_SystemDataParent.

An application or method creates a notification queue by creating an object of the EKG_NotificationQueue class. The EKG_CreateObject function directs RODM to create the notification queue object and assign a user specified event control block (ECB) to the queue object. Once the queue is created, notification methods can place notification blocks on the queue. Applications and methods can delete notification queues by deleting the EKG_NotificationQueue object using the EKG_DeleteObject function. When it creates the queue, RODM automatically qualifies the name of any notification queue with the User_appl_ID from the access block. Each notification queue created with a particular User_appl_ID must be unique.

Table 21 on page 205 describes the fields of the EKG_NotificationQueue class, the access for each field, the data type, and the subfields defined for each field.

Table 21. EKG_NotificationQueue Fields

Field Name	Access	Data Type	Query	Change	Notify	Time stamp
EKG_Status	Write	Enumerated Smallint			X	X
EKG_ECBAddress	Write	ECBAddress				X
EKG_ECBPostedStatus	Read	Enumerated Smallint			X	
EKG_UsedBy	Read	ObjectLink				
EKG_SubscribedFromClass	Read	ClassLinkList				
EKG_SubscribedFromObject	Read	ObjectLinkList				
EKG_Maximum_Q_Entries	Write	Integer			X	
EKG_MessagesOnQueue	Read	Integer				
EKG_SubscribedForDelete	Read	ObjectIDList				

The field definitions are as follows:

EKG_Status
The status of the notification queue. The following values are valid:

Value	**Meaning**
0	Inactive
1	Active

Active status causes RODM to attach notifications to this queue regardless of the ECB value. If a queue accumulates entries when no ECB has been established, RODM posts the ECB as soon as the application sets an ECB value.

Inactive status causes RODM to not attach notifications even if the ECB is already set. This field has a default value of active except in the following situation. User_A creates a notification queue for User_B and there is no user object for User_B. RODM creates the required objects, sets EKG_Status in the NotificationQueue object to inactive, and sets the EKG_Status of the user object to disconnected.

EKG_ECBAddress
The address of an ECB. This is the address of the optional ECB that is posted when a notification block is added to this notification queue. The ECB is created in the address space of the user application that is using this notification queue.

EKG_ECBPostedStatus
Posted status. Valid values are as follows:

Value	**Meaning**
0	False
1	True

This field is set to true if the application has been posted and the queue is not empty. This field is set to false when the queue is empty.

EKG_UsedBy
This field specifies the user that created this notification queue.

EKG_SubscribedFromClass
This field is a list of classes that have a subscription to this notification queue. The field is a one-way link.

The field has a data type of ClassLinkList; each list item consists of a ClassID and a FieldID. The field referenced by the FieldID contains subscription information in the form of a RecipientSpec data type. The RecipientSpec data type contains an 8-byte SubscribeID that your application can use to locate the notification queue object. For information about these data types, see "Abstract Data Type Reference" on page 223.

`EKG_SubscribedFromObject`

This field is a list of objects that have a subscription to this notification queue. The field is a one-way link.

The field has a data type of ObjectLinkList; each list item consists of an ObjectID and a FieldID. The field referenced by the FieldID contains subscription information in the form of the RecipientSpec data type. The RecipientSpec data type contains an 8-byte SubscribeID that your application can use to locate the notification queue object. For information about these data types, see "Abstract Data Type Reference" on page 223.

`EKG_MessagesOnQueue`

The number of messages currently on the EKG_NotificationQueue.

`EKG_Maximum_Q_Entries`

The maximum number of entries permitted on the EKG_NotificationQueue. You can use this field to limit the amount of RODM storage used for unread notifications. When the number of messages on the EKG_NotificationQueue reaches the value of EKG_Maximum_Q_Entries, RODM does not place any more messages on the queue. RODM issues return code 4 with reason code 158 to the notification method which explains that the message cannot be placed on the queue.

The default setting of this field is -1, which indicates no limit.

`EKG_SubscribedForDelete`

This field is a list of objects that have an object-deletion subscription to this notification queue.

The field has a data type of ObjectIDList; each list item consists of an ObjectID. For information about these data types, see "Abstract Data Type Reference" on page 223.

EKG_Method Class

EKG_Method is the class of all RODM methods.

The parent of EKG_Method class is EKG_SystemDataParent class.

Before your application program can refer to a method in a function request or trigger a method, the method must:

- Have an object of the EKG_Method class that represents it
- Be present in memory or you must load it into memory through a method installation process

If RODM cannot find or load the method, it generates an error return code. For more information about installing methods, see "Installing and Freeing Methods" on page 356.

When a method object is created, that method name is made executable for both user API and method API functions. A method has different available functions or different abilities to access data depending on whether it is an object-specific method or an object-independent method. You can write a method that is both an object-specific method and an object-independent method.

The object name of the EKG_Method object you create is the same as the name of the method you are installing. You can identify all installed methods by querying the EKG_Method class using the EKG_QueryEntityStructure function.

The null method that is supplied with the NetView program, NullMeth, is not installed by user creation of an object. This method is built into RODM.

You also use an object of the RODM Method class during the refreshing of the method. Refreshing is accomplished by using the EKG_TriggerNamedMethod function to invoke the method indicated by the EKG_Refresh field in the method object of the method which is to be refreshed. Refreshing deletes the old copy of the method from memory and loads a new copy of the method for all future references.

You can create or delete all fields of EKG_Method.

Table 22 describes the fields of EKG_Method class, the access for each field, the data type, and the applicable operations.

Table 22. EKG_Method Fields

Field Name	Access	Data Type	Query	Change	Notify	Time stamp
EKG_InstallerID	Read	CharVar				X
EKG_UsageCount	Read	Integer				
EKG_Refresh	Read	MethodSpec				
EKG_MTraceFlag	Write	Integer				X

The field definitions are as follows:

EKG_InstallerID
The user ID associated with the installation of the method. The timestamp subfield indicates when the method was installed.

EKG_UsageCount
The current number of references of this method from notify, change, and query subfields, and from value subfields used for named methods. When you delete an object of the EKG_Method class, the usage count, EKG_UsageCount, must be zero. When you refresh an object of the EKG_Method class, there is no restriction on value of EKG_UsageCount.

EKG_Refresh
The name of an internal RODM refresh method that must be invoked to refresh the method represented by the method object. If an application queries the EKG_Refresh value subfield, RODM returns a null value for the Object_ID field of the MethodSpec data.

When the refresh method is triggered using the EKG_TriggerNamedMethod API, RODM loads a new copy of the method from the method library. The Method_parms field of the EKG_TriggerNamedMethod function block is not used by the refresh method.

A method can be refreshed even though it is currently referenced in notify, change, or query subfields. The refresh operation will wait until the method is not executing before loading the new copy of the method. Subsequent executions of the method are suspended until the new copy has been loaded.

`EKG_MTraceFlag`

Specific method trace enable flag. This field specifies if the method is enabled for tracing. Valid values are as follows:

Value	**Meaning**
0	Defers the trace decision to EKG_MTraceType.
1	Ensures tracing.

The initial value is 0.

Tracing must also be enabled by the EKG_MTraceType field in the EKG_User class before RODM can trace this method.

Deleting an Object of the EKG_Method Class: Deleting a method object checks whether the specified method is assigned to any field or subfield as a named, change, query, or notify method. If not, the method is removed from RODM's active methods and the corresponding load module can be freed from memory.

If the method is an object-specific method and is referenced by one or more fields, then it cannot be deleted until all such references are first removed. To remove these references to an object-specific method prior to deleting a method:

- Change the fields that have a data type of MethodSpec and reference the object-specific method to the null value (NullMeth) using the EKG_ChangeField or EKG_ChangeMultipleFields functions.
- Change all subfield that have a data type of MethodSpec and reference the object-specific method to the null value (NullMeth) using the EKG_ChangeSubfield function.
- Remove the notification subscriptions for the notification method using the EKG_DeleteNotifySubscription function.

RODM Objects

Objects are the basic units of data in RODM. They are organized by class and represented by a name containing up to 254 characters. Objects can represent real-world objects, such as DASD devices or printers. Objects can also represent management objects, such as a view on a graphical display, operator access authority, or an application program. Objects can contain locally defined data or inherit data from a class.

User applications and object-independent methods can create objects using the EKG_CreateObject function. You can also create objects using the RODM load function. When you create an object, you specify the name of the object and the class to which the object belongs. RODM returns the numerical object identifier of the new object. The object inherits the public fields that are defined on the class to which the object belongs.

Object Names

Each RODM object has a character string name in its MyName field called the ***object name***.

Two objects, each in a separate class, can have the same object name. Each object can be accessed with the combination of its class name and object name in the form Class_Name.Object_Name.

RODM system-defined object names are reserved by RODM and cannot be deleted by the user.

RODM assigns an object name to any object you create if you do not specify a name when you create the object. RODM assigns names of the form EKG*ddddddd*, where *ddddddd* ranges from 0000000 to 9999999, starting with EKG0000001. Note that values in this range are for RODM use only.

If you are creating an object of the EKG_Method class or the EKG_NotificationQueue class, the object name is limited to 8 characters. For the EKG_NotificationQueue class, if the user ID and object name are combined to produce a fully qualified notification queue name in the form User_appl_ID.object_name, the resulting fully qualified notification queue name is limited to 17 characters, including the separating period.

The CHARACTER_VALIDATION keyword in EKGCUST specifies what degree of validity checking RODM performs for characters used in class names (see "Class Names" on page 195), field names (see "Field Names" on page 210), and object names.

Object Name Characteristics with CHARACTER_VALIDATION(YES)

When CHARACTER_VALIDATION(YES), which is the default, is coded in EKGCUST, valid object names have the following characteristics:

- The name consists of 1 to 254 characters with an abstract data type of ObjectName that conforms to the PL/I syntax of CHAR(254) VARYING.
- The first character of the string must be alphabetic or numeric. The others, if any, can be alphabetic, numeric, or any of the special characters: # @ . , : ; ? ( ) ' " - _ & + % * = < > /
- Both uppercase and lowercase alphabetic characters are permitted, and names are case-sensitive.
- The `EKG_` prefix is reserved for RODM-created classes and objects. Do not use this prefix in the names of classes or objects that you create.
- `EKG`*xxxxxxx* (`EKG` followed by seven digits) is reserved for RODM use only. Do not use this format for the names of objects that you create.
- Each object in a class must have a unique object name.
- RODM supports a maximum of 2097135 objects.

Object Name Characteristics with CHARACTER_VALIDATION(NO)

When CHARACTER_VALIDATION(NO) is coded in EKGCUST, valid object names have the following characteristics:

- The name consists of 1 to 254 characters with an abstract data type of ObjectName that conforms to the PL/I syntax of CHAR(254) VARYING.
- The first character cannot be the number sign (#) because it is reserved for MultiSystem Manager.
- Blank characters are not valid.
- Null characters are not valid.
- Both uppercase and lowercase alphabetic characters are permitted, and names are case-sensitive.
- The `EKG_` prefix is reserved for RODM-created classes and objects. Do not use this prefix in the names of classes or objects that you create.
- `EKG`*xxxxxxx* (`EKG` followed by seven digits) is reserved for RODM use only. Do not use this format for the names of objects that you create.
- Each object in a class must have a unique object name.
- RODM supports a maximum of 2097135 objects.

Object Identifiers

To minimize access time, RODM supports another approach to accessing an object. Any object in any class can be accessed in RODM based solely on the ObjectID of the object. RODM provides functions that convert the fully qualified "class name.object name" to an ObjectID, and convert the ObjectID to the fully qualified "class name.object name".

You can locate objects using any one of the specifications listed below. These specifications are listed in decreasing order of search performance.
1. ObjectID
2. ClassID plus ObjectName
3. ClassName plus ObjectName

RODM Fields

All classes consist of fields that are either public or private, but not both. They must have a field name, and RODM assigns a field identifier. RODM supports a maximum of 4079 fields.

Fields within objects can contain information about the relationships among objects defined in RODM. You can determine these relationships by examining RODM classes and objects.

Field Names

Each RODM field has a character string name, called the *field name*. RODM system-defined field names are reserved by RODM and cannot be deleted by the user. See “System-Defined Fields” on page 211 for a list of the RODM system-defined fields.

The CHARACTER_VALIDATION keyword in EKGCUST specifies what degree of validity checking RODM performs for characters used in object names (see “Object Names” on page 208), class names (see “Class Names” on page 195), and field names.

Field Name Characteristics with CHARACTER_VALIDATION(YES)

When CHARACTER_VALIDATION(YES), which is the default, is coded in EKGCUST, valid field names have the following characteristics:

- The name consists of 1 to 64 characters with a data type of ShortName that conforms to the PL/I syntax of `CHAR(64) VARYING`.
- The first character of the string must be alphabetic or numeric. The others, if any, can be alphabetic, numeric, the break character (_), the commercial at sign (@), the number sign (#), or the period (.).
- You can use both uppercase and lowercase alphabetic characters. Field names are case-sensitive under RODM, regardless of whether your application translates them into a single case.

Field Name Characteristics with CHARACTER_VALIDATION(NO)

When CHARACTER_VALIDATION(NO) is coded in EKGCUST, valid field names have the following characteristics:

- The name consists of 1 to 64 characters with a data type of ShortName that conforms to the PL/I syntax of `CHAR(64) VARYING`.
- The first character cannot be the number sign (#) because it is reserved for MultiSystem Manager.
- Blank characters are not valid.

- Null characters are not valid.
- You can use both uppercase and lowercase alphabetic characters. Field names are case-sensitive, regardless of whether your application translates them into a single case.

Field Identifiers

RODM assigns a 4-byte field identifier to each field. A field identifier is a symbolic representation of the name of a field. You can assign it and compare it to other field IDs. You can use a field ID instead of a field name to address the field through the user API. Using a field ID to address a field through the API is more efficient than using the field name. RODM includes the EKG_QueryFieldName function to convert a FieldID to a field name and the EKG_QueryFieldID function to convert a field name to a FieldID.

RODM-generated internal identifiers exist because they are faster to process than are character string names. These identifiers are always given preference over character string names in resolving which field is to be addressed.

For example, if both the Field_ID and the Field_name_length parameters are not null in a field access information block, the Field_ID is used, and the Field_name_ptr parameter is ignored. RODM does not check that a supplied Field_ID is consistent with a supplied field name. See Table 31 on page 313 for the format and parameters in a field access information block.

Field identifiers differentiate field names from each other without regard to the class or object where the field is located, a field identifier obtained for a field of one class or object can be reused for any field with the identical name regardless of the class or object. A field name does not contain any information about the class or object with which it is associated; however, the classes and objects include the information of what fields they contain.

System-Defined Fields

System-defined fields are fields that are predefined by RODM and must exist for every class and object. These fields and their values are never inherited; RODM creates the fields and sets their values when it creates or changes the object or class to which they belong. Application programs and methods cannot change the contents of these fields through the user API or the method API.

The names of the system-defined fields are reserved names in RODM. You cannot define other fields in classes using these same names.

Of the system-defined fields, only the MyClassChildren, MyObjectChildren and WhatIAm fields change during RODM execution. Therefore, these are the only system-defined fields for which a notify subfield can be created.

Note: Notification methods assigned to these fields to detect deletions of class or object children cannot access the deleted class or object. RODM executes the notification method after it completes the delete process.

Every RODM class and object contains the following system-defined fields:

MyPrimaryParentID
The class ID of the parent class in the primary hierarchy. For objects, this field contains the class ID of the class of the object. For classes (other than the universal-class), this field contains the class ID of the parent class in

the primary hierarchy. The universal-class is the only class that has no parent, and therefore, a null MyPrimaryParentID field.

The data type of this field is ClassID.

MyPrimaryParentName
The name of the parent class in the primary hierarchy.

The data type of this field is ShortName.

MyID The ID of the object or class upon which the field resides. For objects, the contents of MyID is the object ID. For classes, the contents of MyID is the class ID.

The data type of this field is ObjectID for objects and ClassID for classes.

MyName
The full name of the current object or class. For objects, this field contains the object name. For classes, this field contains the class name.

The data type of this field is ObjectName for objects and ShortName for classes.

WhatIAm
The object or class type.

The data type for this field is Integer and has the following values:

1	An object
2	A class with no children
3	A class with object children
4	A class with class children
5	A class with both class children and object children

Every RODM class contains the following additional system-defined fields:

MyClassChildren
A list of class IDs of the class children of this class. Each entry in the list is the class ID of one child class.

The data type of this field is ClassIDList.

When a class is created, the value of this field is set to null. Thereafter, entries are added, set, and deleted from this list by the creation and deletion of classes that are specified at creation as having this class as primary parent.

MyObjectChildren
A list of object IDs of the object children of this class. Each entry in the list is the object ID of one child object.

Data type is ObjectIDList.

When a class is created, the value of this field is set to null. Thereafter, entries are added, set, and deleted from this list by the creation and deletion of objects that are specified at creation as having this class as primary parent.

The MyClassChildren and MyObjectChildren fields are never created for objects.

RODM Subfields

The RODM data types, defined in “Abstract Data Type Reference” on page 223, restrict the values that RODM considers valid for a field. But network management applications require more information about a field than just its value. A field must contain several pieces of data or logic to be useful in a data cache that stores both persistent and volatile information.

When a field is created, RODM automatically creates a value subfield for the field. If no other subfields are explicitly defined for the field, any reference to the field is the same as a reference to the value subfield of the field.

Suppose that the dominant value to be preserved in the *number_of_waiting_print_jobs* field of a printer object is the number of print jobs waiting to be printed. This value is volatile and the contents of this field are of little use if the value is several hours old. Suppose also that you can save the number of jobs waiting to be printed and also the time at which the value was obtained. You can now use this timestamp to invalidate the data that is old and indicate that current data is required.

A time stamp alone does not solve the problem. When an application requests the contents of the *number_of_waiting_print_jobs* field, there must be some logic in place to compare the contents of the timestamp with the current time and take an appropriate action based on the age of the data in the field. The design of RODM permits a field to be composed of several subfields. These subfields can refer to methods that can be set to automatically do such things as check time stamps before responding to a query.

There is a fixed list of subfields that can appear in a field. All subfields are optional except for the value subfield, which contains the data stored in the field and so must exist if the field exists. The following list contains each kind of subfield and its intended use.

The value and prev_val subfields have the same data type as the corresponding field. All other subfields have predetermined data types that are set based on the kind of subfield. The data type of each subfield is specified in the following list along with a description of each subfield. When a subfield is created, RODM assigns it a null value based on the subfield data type requirements.

RODM defines the following subfields:

Value (Required)

The actual data associated with the field. The value is defined in terms of RODM abstract data types, such as Integer, CharVar, or Floating.

The data type must be one of those defined in “Abstract Data Type Reference” on page 223 and is identical to the data type of the field. The value subfield is the only system-defined subfield of a field. All other subfields are optional with their presence obtained by a transaction against the field of the class through the user API.

Query

A method specification (data type MethodSpec) for a query method.

- Querying a field invokes a query method if this subfield has a value.
- A query method can modify the queried data from a field.

The query subfield contains a method that is invoked before the field contents are returned to a caller in response to a query of the field. If a query method is defined, the query method is responsible for returning a value in response to the query. If a query method does not return a value in response to the query, RODM returns one.

The data type of a query subfield is MethodSpec. The MethodSpec type includes the object identifier of the method to be invoked, plus a list of parameters to be passed to the method.

The parameters indicate fields of the object that the user has set up to be used by the method. The parameters in those fields are most frequently set when the method is installed in the subfield. However, some or all of those parameters can be set by assigning values to the corresponding fields immediately before the query transaction that triggers the query method is requested.

Change

A method specification for a change method.

- A change field request invokes a change method if this subfield has a value.
- A change method modifies the data in the field on which it is defined.

The change subfield is a method that is invoked to change the contents of a field as requested by an EKG_ChangeField or EKG_ChangeMultipleFields function request, either from a user outside of RODM, or by another method. If a field receives a change request and has a change subfield, the change method must make the change to the value of field; RODM does not change the value of a field that has a change subfield defined.

The data type of a change subfield is MethodSpec. The subfield includes the ID of a method and the locations in fields of the object where parameters for the method are to be found.

The change subfield cannot exist for any system-defined field, such as MyName, MyID, MyPrimaryParentID, MyPrimaryParentName, WhatIAm, MyClassChildren, and MyObjectChildren.

Notify

A method specification for one or a list of notification methods.

- Changing a field invokes a notification method if this subfield has value. RODM invokes the notification method after the change in the field is complete.
- A notify method can notify subscribed users of changes to fields.

The notify subfield contains a list of methods and associated parameters. Each method in the list is invoked one at a time after every change in the value of the field as requested by a change request from a user. Methods in the list are intended to notify other objects or to notify RODM users when changes in state take place. The data type of each entry in the list is SubscriptSpec.

The data type of the subfield is SubscriptSpecList. A method name, parameters for the method from object fields, and a description of who is to be notified are included in each entry. When the method is invoked, the logic in the method decides, based on the data in the object, whether to notify anyone. The method can notify the original subscriber or it can be programmed to notify another application or to submit transactions to other RODM objects. Notification methods can submit transactions, other than the EKG_QueryObjectName function, to other RODM objects only through the EKG_MessageTriggeredAction method API function.

`Timestamp`

The time at which the value subfield of the field was last changed. RODM manages this subfield. This subfield is read-only. The data type of the subfield is TimeStamp.

The timestamp subfield is created and deleted using the EKG_CreateSubfield and EKG_DeleteSubfield functions. When it is defined, RODM updates the timestamp subfield for every successful change transaction against the field, including when the new value is the same as the old value. The timestamp subfield is always associated with the value subfield of the same field. A change transaction against the value subfield, rather than against the field, does not cause the timestamp subfield to be updated. If you issue the EKG_RevertToInherited function and the field contains a local value and corresponding time-stamp, the time-stamp subfield is also reverted to its inherited value.

`Prev_val`

A copy of the previous contents of the value subfield. RODM manages this subfield. This subfield is read-only. The data type of this subfield is the same as the data type of the value subfield. You cannot create a prev_val subfield for system-defined fields. See “Data Types for Subfields” for a list of abstract data types that the prev_val field can contain.

The prev_val subfield is created and deleted using the EKG_CreateSubfield and EKG_DeleteSubfield functions. When it is defined, RODM updates the prev_val subfield for every successful change transaction against the field, including when the new value is the same as the old value. The prev_val subfield is always associated with the value subfield of the same field. A change transaction against the value subfield, rather than against the field, does not cause the prev_val subfield to be updated. If you issue the EKG_RevertToInherited function and the field contains a local value and corresponding prev_val, the prev_val subfield is also reverted to its inherited value.

Data Types for Subfields

Certain RODM abstract data types can be used for each subfield. The abstract data types are defined in “Abstract Data Type Reference” on page 223.

Subfield	Valid Abstract Data Types
Value	AnonymousVar BERVar CharVar FieldID Floating GraphicVar IndexList Integer MethodSpec ObjectLink ObjectLinkList SelfDefining Smallint TimeStamp
Query	MethodSpec

Change
- MethodSpec

Notify
- SubscriptSpecList

Time Stamp
- TimeStamp

Prev_val
- AnonymousVar
- BERVar
- CharVar
- FieldID
- Floating
- GraphicVar
- IndexList
- Integer
- MethodSpec
- SelfDefining
- Smallint
- TimeStamp

Multivalued Fields and Links between Objects

RODM permits the use of multivalued fields to establish the relationships between objects. Multivalued fields support the creation of one-to-one, one-to-many, many-to-one, and many-to-many relationships between objects.

Note: The links described in this section are RODM-defined relational links. These links are defined between two objects in the RODM data cache and must not be confused with physical links, such as network links, which are represented by GMFHS-defined link objects.

The EKG_LinkNoTrigger and EKG_LinkTrigger functions enable user applications and methods to create links between two objects. The EKG_UnlinkNoTrigger and EKG_UnlinkTrigger functions enable user applications and methods to delete links between two objects. Use an ObjectLink type field to link to one object. Use an ObjectLinkList type field to link to one or more objects. An ObjectLink field of one object always links to an ObjectLink or ObjectLinkList field of another object. An ObjectLinkList field of one object always links to ObjectLink or ObjectLinkList fields of other objects.

The reserved data types ObjectID and ObjectIDList are used by RODM for links between system-defined fields. These system-defined fields, such as the MyObjectChildren field, are managed by RODM and cannot be changed directly by user applications or methods.

Figure 38 on page 217 shows single-value links using fields of data type ObjectLink and a multivalue link using a field of data type ObjectLinkList.

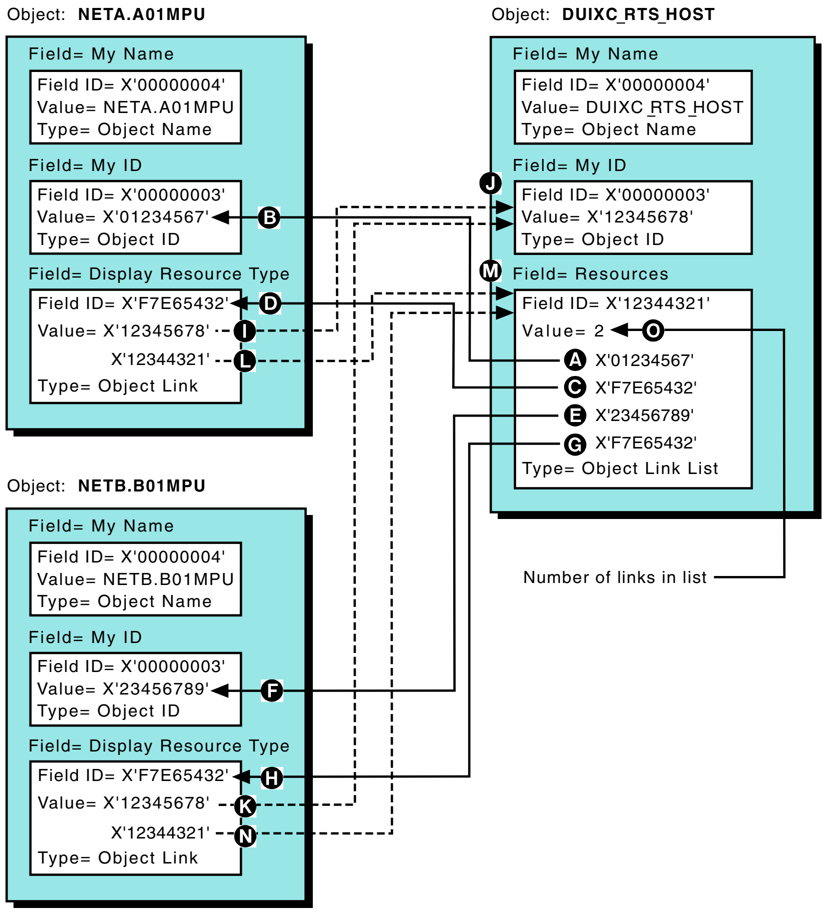


Figure 38. Examples of Links between Objects in RODM

Figure 38 contains three RODM objects. Two of the objects represent host processors in a network, and the third object is a resource type object which is used to identify types of objects. Each of the two host objects, NETA.A01MPU and NETV.B01MPU, has a single-value link to the resource type object. The resource type object, DUIXC_RTS_HOST, has a multivalue link to each of the two host objects.

The object NETA.A01MPU has a field named DisplayResourceType, which is data type ObjectLink. The DisplayResourceType field contains the ObjectID (I) of the object being linked to (J), and the FieldID (L) of the field being linked to (M).

The object NETB.B01MPU also has a field named DisplayResourceType linked to the field Resource of object DUIXC_RTS_HOST. DisplayResourceType contains the ObjectID (K) of DUIXC_RTS_HOST (J) and the FieldID (N) of Resources (M).

The object DUIXC_RTS_HOST has the field Resources that is linked to both of the host objects. The ObjectLinkList field Resources contains the number of objects it is linked to (O). The first list element of Resources contains the ObjectID (A) of object NETA.A01MPU (B) and the FieldID (C) of field DisplayResourceType (D). The second list element of Resources contains the ObjectID (E) of object NETB.B01MPU (F) and the FieldID (G) of field DisplayResourceType (H).

When you create links using the EKG_LinkNoTrigger or EKG_LinkTrigger functions, you specify the pair of objects and fields to be linked, and RODM fills in the ObjectID and FieldID values in both objects. Both objects must exist in RODM before they can be linked.

Link and Unlink Action Functions

The link and unlink action functions can be invoked by users through the method API and user API. The EKG_LinkNoTrigger function and the EKG_LinkTrigger function are used to establish a link between two fields on two objects. The EKG_UnlinkNoTrigger function and the EKG_UnlinkTrigger function delete a link between two objects. Each of these functions require two objects and two fields specified through the Entity_access_info_ptr and Field_access_info_ptr parameters. The fields must be of data type ObjectLinkList or ObjectLink. See "EKG_LinkNoTrigger, EKG_LinkTrigger — Link Two Objects" on page 402 and "EKG_UnlinkNoTrigger, EKG_UnlinkTrigger — Unlink Two Objects" on page 441 for function block formats and additional details.

Fields that are lists or of type ObjectLink are changed only by link and unlink actions. For these actions, there are always two fields involved, one at each end of the link. Change methods can be defined to these fields. These change methods are triggered by the EKG_LinkTrigger or EKG_UnlinkTrigger functions. The change methods must set a return code with EKG_SetReturnCode to indicate whether the link or unlink can proceed.

- A nonzero return code prevents the link or unlink.
- If no change method exists on one (or both) of the fields, RODM assumes the return code is zero and the link or unlink proceeds.
- If a change method exists, but it does not set the return code explicitly, RODM assumes the return code is zero and the link or unlink proceeds.

The change methods are triggered in the order in which the fields appear in the function block.

To be symmetric, the RODM program invokes the appropriate notify methods at both ends of a link when a link or unlink action is requested and the subfields exist at both ends of the link. If two methods are invoked, the one invoked first is the top field specified in the function block that specifies the desired action. For notify methods, first one list is processed, then the other list is processed. If the link or unlink is prevented by the nonzero return code, the notify methods are not triggered.

Link and unlink action functions are applicable only in linking two objects together. It is not possible, using the link action function, to link a class to another class or object. An object inherits the existence of fields of type ObjectLink from its class, but an object can only inherit the null value from its class for these fields. Likewise, in the hierarchy of classes, the existence of fields of type ObjectLink is inherited by children classes, but values in all such fields are null.

If the type of a field to be linked is ObjectLinkList, the link action creates a new entry in the list and sets that entry to contain the ObjectID and FieldID of the other object-field pair. Links constructed for fields of data type ObjectLinkList are not guaranteed to be ordered within the field according to any particular algorithm like FIFO or LIFO. If the type is a simple ObjectLink, the value of that field is set to contain the ObjectID and FieldID of the other object-field pair. Because the link applies to each object-field pair, it establishes a two-way link between the two objects. Unlink removes such links. Link and unlink actions are the only actions available to RODM users that change fields of type ObjectLink.

If a field is a single ObjectLink, a query of that field yields a response of type ObjectLink, which is an 8-byte ObjectID followed immediately by a 4-byte FieldID for a total of twelve bytes. If a field is an ObjectLinkList, a query of the field through either the user API or method API causes an array of ObjectLink entries to be returned to the user. In other words, each element in the array is a 12-byte pair of ObjectID and FieldID. RODM users cannot query the entries of an ObjectLinkList, individually.

The same principle applies to queries of a MyObjectChildren field. A query of such a field yields an array where each element in the array is of data type ObjectID for MyObjectChildren field. The length of the array is identical to the length of the list in the queried field.

Links between objects established with the link action function are used to represent both peer-to-peer relationships and to represent secondary parent-child relationships. Primary parent-child relationships are required and are embodied in the system-defined fields MyClassChildren, and MyObjectChildren of objects and classes.

Subfields Associated with Fields

You cannot create a query subfield for fields that are of data types ObjectLink or ObjectLinkList. For fields that are not of data types ObjectLink or ObjectLinkList, the value subfield is the single field entry and can be queried and manipulated without triggering methods. For fields that are of data types ObjectLink or ObjectLinkList, the value subfield consists of an entire list of entries, and the value subfield can only be queried without triggering a query method.

Change transactions are not applicable to fields of data types ObjectLink or ObjectLinkList, and similarly, change transactions are not applicable to the value subfield of a field that is of data types ObjectLink or ObjectLinkList. Only link and unlink functions exist for changing the values in fields of type ObjectLinkList, and only creation and deletion of children changes a MyObjectChildren field.

To perform the link and unlink action functions, without triggering notify methods, the RODM program supports the EKG_LinkNoTrigger function and the EKG_UnlinkNoTrigger function.

The subfields possible for fields that are of type ObjectLink are query, notify, and timestamp subfields. For fields of type ObjectLink and ObjectLinkList, change subfields are enabled. However, the RODM program supports only one subfield for the entire list; separate subfields are not supported for each entry in the list. Any change to any entry of the list is considered a change to the entire list. Therefore, if there is a notify list, any change to any entry in the list of links (the field) results in all the methods in the notify list being invoked.

If a child object inherits the existence of a field that is of data types ObjectLink or ObjectLinkList, the child object also sees the field as a data type ObjectLink or ObjectLinkList field. But the RODM program does not support the inheritance of values in fields of data types ObjectLink or ObjectLinkList. The entries in fields of data types ObjectLink or ObjectLinkList are independent of the entries in any other fields of data types ObjectLink or ObjectLinkList. They are created one at a time by the EKG_LinkNoTrigger function or the EKG_CreateObject function, and they are deleted one at a time by the EKG_UnlinkNoTrigger function or the EKG_DeleteObject function.

Indexed Fields

The EKG_Locate function retrieves a list of Object IDs of objects having a specified value in a specified field. This function makes it easier for an application to retrieve the list of Object IDs. Rather than scanning the user's entire data model using the query field functions (looking for the specified field and value), the application invokes the EKG_Locate function with the desired field and field value.

For a field to be located by the EKG_Locate function, that field must have been created as a public_indexed field. For public_indexed fields, RODM maintains tables of Object IDs by field name and field value. Because additional processing is required to maintain these tables, users must create public_indexed fields only for fields that exploit the EKG_Locate function. An example of this is a data model with Employees as a class, each employee name as an object under that class, and EmployeePhoneNumber as an indexed field. In this example, an application can locate all of the objects that have a specified phone number in field EmployeePhoneNumber without performing a query on every object in the data model.

Indexed Fields can be of CharVar or IndexList data type. IndexList fields generate multiple ObjectID table entries - one for each value in the list. For both CharVar and IndexList, EKG_Locate accepts one character string (maximum length 254 bytes) for comparison, pointed to by Indexed_data_ptr.

See “Indexed Fields” on page 480 for performance-related information about defining public_indexed fields.

Object and Class Locking in RODM

RODM now controls locking automatically. The following functions are no longer necessary, but remain available for compatibility with existing applications.
- EKG_LockObjectList function
- EKG_UnlockAll function

No changes to existing applications that use these functions are required.

Using the Application Program Interfaces

This section briefly explains the two RODM application program interfaces.

User Application Program Interface (API)

A RODM user application is an external program that accesses RODM data through the user API to perform a task. This RODM user application can be coded in any language that enables you to meet the parameter passing conventions of RODM. However, RODM supplies control block structures only for PL/I and C.

Figure 39 on page 221 illustrates how user applications access RODM data in a z/OS environment using EKGUAPI, the user API module. The steps for coding a full RODM application are described in Chapter 11, “Writing Applications that Use RODM,” on page 301.

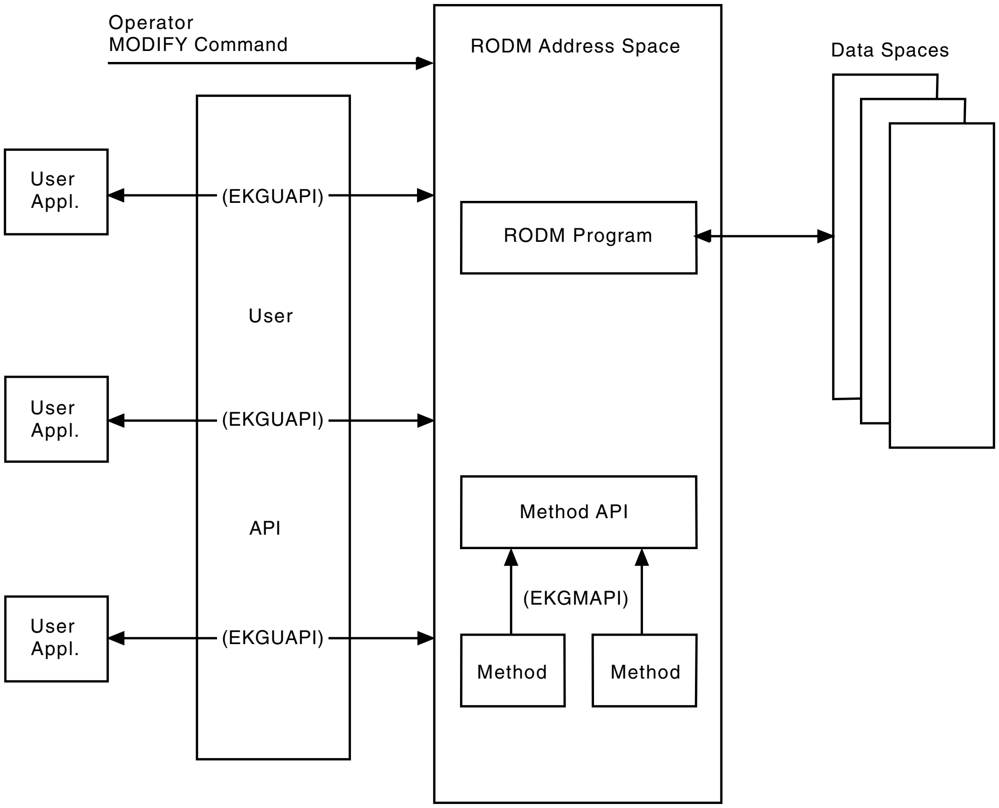


Figure 39. RODM System Structure (z/OS)

Method Application Program Interface (API)

Methods are small executable programs that reside in the RODM address space. Methods can be invoked by user applications, by changes to fields in RODM, by other methods, and at RODM initialization.

The NetView program supplies several general-purpose methods that might meet your needs; if not, you can write your own using PL/I or C.

Figure 39 illustrates how methods access RODM data in a z/OS environment using EKGMAPI, the method API module. The steps and information associated with coding a RODM method are described in Chapter 13, “Writing RODM Methods,” on page 339.

RODM Abstract Data Types

This section describes how to use the RODM data types. Different data types can be used in different contexts, such as the types of data in fields, subfields, fields of the user API or method API, or parameters passed to methods.

Several of the RODM data types are compound data types; they correspond to structures in programming languages. PL/I macro declarations and C typedef statements are provided for these compound data types. Ensure that there is no compiler-generated padding when you map these declarations to storage. You can do this in PL/I by adding the UNALIGNED attribute to each declaration, and, in C, by using the _Packed qualifier.

Null Values of Data Type

The RODM program specifies a null value for each data type. Typically, you use null values for:

- Locator types

  Locator types are data that locates or points to other data. A null value means that the data is *pointing to nothing*.
- Types that contain non-locator information

  For types that contain non-locator information, such as numbers, counts, or flags, the null value always implies *no information here* or *not yet set to a value*.

The RODM program sets the value of a field or a subfield to the null value for the type of field or subfield whenever it first creates it on a class. When a class or object inherits a field from its parent class, the value of the field is set to the value on the parent class.

See “Abstract Data Type Reference” on page 223 for a specification of the null value for each data type.

Data Type Identifiers

When user applications pass data to the RODM program, the RODM program usually requires that they also pass the data type of the data along with the data. When the RODM program passes data to an application, the RODM program usually includes the data type of the data along with the data. To efficiently identify data types, there is a decimal data type identifier for each RODM data type.

To find the data type identifier for a particular data type, see “Abstract Data Type Reference” on page 223.

Types of Data in Fields

Your application programs and methods must assign a data type to each field in a class when they issue an API call to create a field. After the API has created the field, you cannot change the data type during the life of the field.

List abstract data types are specified for fields that are to contain lists of information instead of a single value. The list data type is available to form lists of type IndexList, ObjectLink, ObjectID, and ClassID. This field type enables the specification of multiple-to-single relationships and multiple-to-multiple relationships of classes and objects.

Some data types that can be specified for fields are restricted, depending on the nature of the field. The RODM program limits the possible relationships of objects and classes in order to assure that incorrect identifiers are not left in RODM after an object or a class is deleted. For example, the following conceptually feasible relationships are prohibited by RODM:

- Relationships between an object and classes other than the parent child relationships in the primary hierarchy. Class relationships *must* be inheritance relationships.
- Relationships between two objects other than those that are represented by ObjectLinks, using the EKG_LinkNoTrigger and EKG_LinkTrigger functions.

Abstract Data Type Reference

This section describes the abstract data types defined by the RODM program. Include the macro EKG1IADT for PL/I or EKG3CADT for C in your user applications and methods. Including this macro enables you to declare the variables in your programs to be the data types needed to use RODM functions.

For example, if you need to specify the name of a method in a RODM function block, the parameter you pass must be declared as the MethodName abstract data type. To declare a variable named ThisMethodName in PL/I, use the statements:

```
%include EKGLIB(ekg1iadt);           /* Abstract data declaration  */
DCL ThisMethodName      MethodName;  /* 8-byte char                */
```

To declare the same variable in C, use the statements:

```
#include "ekg3cadt.h"                /* Abstract data declaration  */
MethodName       ThisMethodName;     /* 8-byte char                */
```

Examples of declaring variables of each type are provided in the file EKG5VDCL for PL/I and in the file EKG6VDCL for C.

In the data type definitions that follow, some of the data types are specified as being reserved. You cannot specify these data types when you create a field definition; these data types are reserved for fields created by the RODM program.

Anonymous(N) (Reserved)

Data Type Identifier: 29

Description: A variable length sequence of data bytes in which only the creator of the data knows the value of the data contents. The maximum length of the string is 254 bytes. The actual length is implicit and based on where a variable of this type has been defined for use. The format of the variable contents is unknown at the user API level. Only the application program or method that is using RODM and that set the value understands this type. This abstract data type cannot be used in a SelfDefining data string.

Null Value: Unknown

PL/I Declaration:

```
% Anonymous = 'CHAR';
```

C Declaration:

```
typedef char Anonymous;
```

AnonymousVar

Data Type Identifier: 30

Description: A variable length string of data that consists of up through 32767 bytes. Constructed as a 2-byte length field followed by the number of data bytes specified by the length field. This data string can be binary data bytes of any value.

The format of the variable contents is unknown at the user API level. Only the application program or method that set the value can understand the format.

Null Value: Length field is zero.

PL/I Declaration:

```
% AnonymousVar = 'CHAR(32767) VARYING';
```

C Declaration:

```
typedef    _Packed struct {
             Smallint  Length;
             Anonymous Text[1];
           } AnonymousVar;
```

ApplicationID (Reserved)

Data Type Identifier: 3

Description: An 8-byte token containing the user application name. This application ID is verified by your system authorization facility. Characters are positioned left-justified within the 8 bytes and padded with blanks on the right. The host system code page defines the blank; for S/370, the assumed code page is code page 00500, on which a blank is X'40'.

Null Value: All bytes are blank (for code page 00500, X'40').

PL/I Declaration:

```
% ApplicationID = 'CHAR(8)';
```

C Declaration:

```
typedef    _Packed struct {
             char     Data_char[8];
           } ApplicationID;
```

BERVar

Data Type Identifier: 31

Description: The BERVar data type specifies BER data to the RODM load function. RODM verifies part of the BER data format but does not interpret any of it. The following description identifies the information verified by RODM.

The maximum length of the BER data type (including the identifier, length and contents bytes) must not exceed 32767. Figure 40 shows the format of BER data.

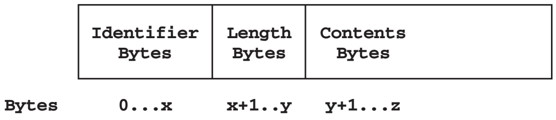


Figure 40. Format of BER Data

RODM verifies the following BER data:

- **Identifier bytes.** Identifier bytes can take two forms, short or long. The form is determined by the tag number (bits 5 to 1) in the first byte.
  - If the tag number is less than or equal to 30 ('11110'b), the identifier byte is in the short form and only a single identifier byte is needed.

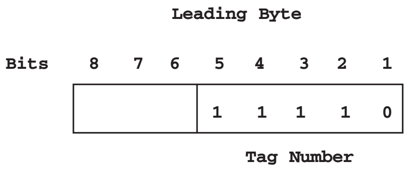


Figure 41. Identifier Byte in Short Form

- If the tag number in the first byte is equal to 31 ('11111'b), the identifier byte is long. For the long form, more than one identifier byte exists. In each byte following the leading byte, bit 8 is set to 1 until the last identifier byte. In the last identifier byte bit 8 is set to 0 (zero).

  Figure 42 shows the long form with three identifier bytes.

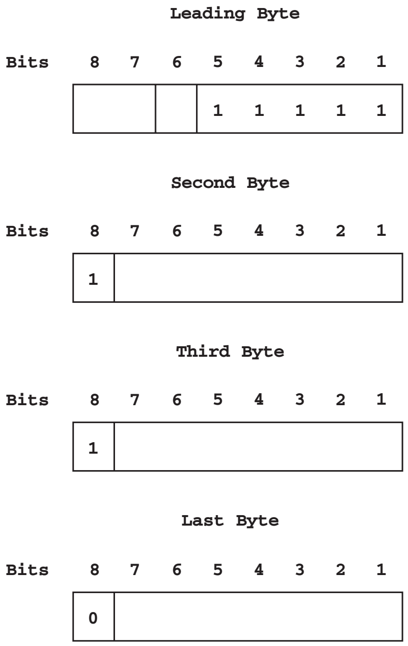


Figure 42. Identifier Byte in Long Form

- **Length bytes.** The length byte specifies the length of the contents bytes and can take 2 forms, short or long.
  - If bit 8 equals 0, the length byte is short. In this form, bits 7 to 1 represent the length of the contents bytes as an unsigned binary integer. The contents bytes can only be less than or equal to 127 bytes with the short form.

    Figure 43 shows the short form of a length byte with the value of 86 bytes.

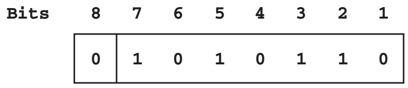


Figure 43. Length Byte in Short Form

  - If bit 8 equals 1, the length byte is long. For this form, bits 7 to 1 represent the length of the contents bytes as an unsigned binary integer. Each subsequent byte is an unsigned binary integer, and when added together, represents the length of the contents bytes. If the contents bytes are greater than 127 bytes, you must use the long form.

Figure 44 shows the long form of a length byte with the value of 357 bytes. Two length bytes are needed to represent 357.

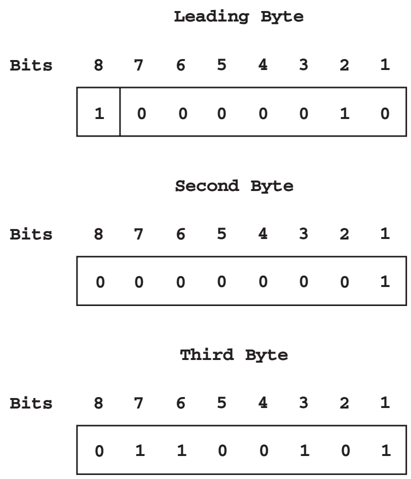


Figure 44. Length Byte in Long Form

Null Value: Length field is zero.

PL/I Declaration:

```
% BERVar = 'CHAR(32767) VARYING';
```

C Declaration:

```
typedef    _Packed struct {
                Smallint  Length;
                Anonymous Text[1];
           } BERVar;
```

CharVar

Data Type Identifier: 4

Description: Variable-length character string of up through 32767 bytes. The structure of this data type is a 2-byte length field followed by the characters in the string. CharVar data can be optionally terminated with a null byte with value X'00' by the user for C string support. When RODM formats character strings, it always adds the null terminator. For example, a CharVar field specified with the null byte that contains the string “RODM” has the value X'0004D9D6C4D400'. Note that the null terminator byte is not included in the length field of the CharVar data.

For information about specifying a CharVar string in a SelfDefining data string, see “SelfDefining” on page 234.

For DBCS (double-byte character set) support, the special control character shift-out (X'0E') can begin a DBCS string, and the control character shift-in (X'0F') can end a DBCS string. When embedded between the shift-out and shift-in control characters, each double-byte character is counted as two bytes. In addition, the shift-out and shift-in characters are included in the length of the DBCS string. The valid double-byte characters are the same as those for the GraphicVar data type; see “GraphicVar” on page 229.

Null Value: Length field is zero.

PL/I Declaration:

```
% CharVar = 'CHAR(32767) VARYING';
```

C Declaration:

```
typedef   _Packed struct {
            Smallint Length;
            char     Text[1];
          } CharVar;
```

CharVarAddr (Reserved)

Data Type Identifier: 7

Description: Pointer to any variable-length character string. The pointer does not imply any maximum length requirements.

Null Value: NULL pointer.

PL/I Declaration:

```
% CharVarAddr = 'POINTER';
```

C Declaration:

```
typedef  Pointer  CharVarAddr;
```

ClassID (Reserved)

Data Type Identifier: 1

Description: A full-word integer that identifies a class to RODM. ClassID is the data type only of the MyID field on a class and the MyPrimaryParentID field on classes and objects.

Null Value: All bits are zero.

PL/I Declaration:

```
% ClassID = 'FIXED BINARY(31)';
```

C Declaration:

```
typedef  long  ClassID;
```

ClassIDList (Reserved)

Data Type Identifier: 2

Description: A list of Class IDs. This is the data type only of the MyClassChildren field of a class. The Length field of ClassIDList is the number of elements in the list, not the length in bytes.

Null Value: Length field is zero.

PL/I Declaration:

```
DCL
  1 ClassIDList EKG_BOUNDARY,
    3 Len      Integer,
    3 List(1)  ClassID;
```

Note: EKG_BOUNDARY is a character substitution for the UNALIGNED and BASED PL/I attributes and is used with all abstract data type Pl/I definitions using DCL statements.

C Declaration:

```
typedef    _Packed struct {
             Integer  Length;
             ClassID  List[1];
           } ClassIDList;
```

ClassLinkList (Reserved)

Data Type Identifier: 6

Description: A 4-byte length field followed by a list in which each entry is a concatenated Class ID and Field ID. The Length field of ClassLinkList is the number of elements in the list, not the length in bytes. Each entry specifies a link to some field of a class, required for a system-class definition of the MyClassChildren field of a class.

Null Value: Length field is zero.

PL/I Declaration:

```
DCL
  1 ClassLinkList EKG_BOUNDARY,
    3 Len      Integer,
    3 List(1),
      5 ClassIdentifier  ClassID,
      5 FieldIdentifier  FieldID;
```

C Declaration:

```
 typedef    _Packed struct {
              Integer    Length;
              ClassLink  List[1];
            } ClassLinkList;
```

ECBAddress (Reserved)

Data Type Identifier: 8

Description: The 4-byte address of an ECB that the RODM program uses to post an application when an event occurs. The EKG_NotificationQueue class requires this data type.

Null Value: Null pointer

PL/I Declaration:

```
% ECBAddress = 'POINTER';
```

C Declaration:

```
typedef  void  *ECBAddress;
```

FieldID

Data Type Identifier: 26

Description: A full-word integer for field identifiers. This data type is used for fields that contain the identifier of other fields.

Null Value: All bits are zero.

PL/I Declaration:

```
% FieldID = 'FIXED BINARY(31)';
```

C Declaration:

```
typedef  long  FieldID;
```

Floating

Data Type Identifier: 9

Description: A floating point number for general use. The number is represented in eight bytes.

Null Value: All bits zero

PL/I Declaration:

```
% Floating = 'FLOAT BINARY(53)'
```

C Declaration:

```
typedef  double  Floating;
```

GraphicVar

Data Type Identifier: 5

Description: A sequence of data constructed as a 2-byte length field followed by a set of double-byte characters. The value of the length field must be no more than 16,383 double-byte units. One 16-bit double-byte character has a length of one double-byte unit. Valid characters must have both the first and second byte of data defined in the range X'41' through X'FE'. The characters X'4040' are also valid. GraphicVar data is terminated by two null bytes with value X'0000'. The null terminator bytes are not included in the length field of the GraphicVar data.

Null Value: Length field is zero.

PL/I Declaration:

```
DCL
  1 GraphicVar EKG_BOUNDARY,
    3 Len     Smallint,
    3 Text    CHAR(1);
```

C Declaration:

```
typedef   _Packed struct {
            Smallint Length;
            Smallint Text[1];
          } GraphicVar
```

Integer

Data Type Identifier: 10

Description: Full-word integer intended for general use.

Null Value: All bits are zero.

PL/I Declaration:

```
% Integer = 'FIXED BINARY(31)';
```

C Declaration:

```
typedef  long  Integer;
```

IndexList

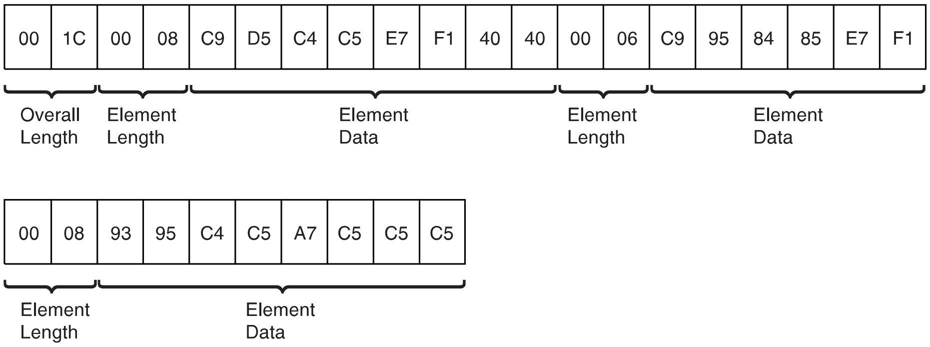
Data Type Identifier: 32

Description: A variable-length string of data that is composed of multiple values up through a maximum of 32767 bytes. The data is a list of AnonymousVar data values, and each individual data value in the list has the following characteristics:

- Must be unique within the field.
- Has a maximum length of 254 bytes
- Is composed of a 2-byte length field followed by the number of data bytes specified by the length field. The AnonymousVar data type identifier is not part of the value.

Figure 45 shows an example Indexlist string that contains three AnonymousVar values:

- `00 08 C9 D5 C4 C5 E7 F1 40 40`
- `00 06 C9 95 84 85 E7 F1`
- `00 08 93 95 C4 C5 A7 C5 C5 C5`



00	1C	00	08	C9	D5	C4	C5	E7	F1	40	40	00	06	C9	95	84	85	E7	F1

00	08	93	95	C4	C5	A7	C5	C5	C5

Figure 45. Example IndexList Field

Null Value: Length field is zero.

PL/I Declaration:

```
% IndexList = 'CHAR(32767) VARYING';
```

C Declaration:

```
typedef    _Packed struct {
             Smallint Length;
             char    Text[1];
           } IndexList;
```

MethodName (Reserved)

Data Type Identifier: 11

Description: An 8-character data type for the name of a method.

Null Value: NullMeth.

PL/I Declaration:

```
% MethodName = 'CHAR(8)';
```

C Declaration:

```
typedef    _Packed struct {
             char     Data_char[8];
           } MethodName;
```

method_parameter_list (Reserved)

Data Type Identifier: 12

Description: Long-lived parameters retained by RODM and passed to a method. The maximum length is 254 bytes, excluding the 2-byte header of X'000C'.

Null Value: Length field is zero.

PL/I Declaration:

```
% method_parameter_list =  'SelfDefining';
```

C Declaration:

```
typedef  SelfDefining  method_parameter_list
```

MethodSpec

Data Type Identifier: 13

Description: A method object ID plus a method parameter list that specify an object-specific method and the parameters that it has when you trigger it.

Null Value: Method object ID for the reserved method named *NullMeth* concatenated with a null method parameter list.

PL/I Declaration:

```
DCL
  1 MethodSpec EKG_BOUNDARY,
    3 ObjectIdentifier   ObjectID,
    3 MthdParmList       SelfDefining;
```

C Declaration:

```
typedef    _Packed struct {
             ObjectID      ObjectIdentifier;
             SelfDefining  MthdParmList;
           } MethodSpec;
```

ObjectID (Reserved)

Data Type Identifier: 14

Description: Double word for an object ID, required on the MyID field of an object.

Null Value: All bits are zero.

PL/I Declaration:

```
% ObjectID = 'BIT(64)';
```

C Declaration:

```
typedef    _Packed struct {
             Smallint  Collision_number;
             Smallint  Class_identifier;
             Integer   Object_identifier;
           } ObjectID;
```

ObjectIDList (Reserved)

Data Type Identifier: 15

Description: A list in which the entries are Object IDs. The data type of the MyObjectChildren field on a class. A sequence of data constructed as a 4-byte length field followed by a concatenation of the ObjectIDs that are the entries in the list. The Length field of ObjectIDList is the number of elements in the list, not the length in bytes. All object IDs in the list are concatenated and contiguous.

Null Value: Length field is zero

PL/I Declaration:

```
DCL
  1 ObjectIDList EKG_BOUNDARY,
    3 Len      Integer,
    3 List(1)  ObjectID;
```

C Declaration:

```
typedef    _Packed struct {
             Integer   Length;
             ObjectID  List[1];
           } ObjectIDList;
```

ObjectLink

Data Type Identifier: 16

Description: Double-word object ID plus field ID for specifying a link to a field in another object.

Null Value: A NULL Object ID concatenated with a NULL field ID.

PL/I Declaration:

```
DCL
  1 ObjectLink EKG_BOUNDARY,
    3 ObjectIdentifier   ObjectID,
    3 FieldIdentifier    FieldID;
```

C Declaration:

```
typedef    _Packed struct {
             ObjectID ObjectIdentifier;
             FieldID  FieldIdentifier;
           } ObjectLink;
```

ObjectLinkList

Data Type Identifier: 17

Description: A list of Object Links. A sequence of data constructed as a 4-byte length field followed by the concatenation of the Object Links that are the entries in the list. The Length field of ObjectLinkList is the number of elements in the list, not the length in bytes. All object IDs in the list are concatenated and contiguous.

Null Value: Length field is zero

PL/I Declaration:

```
DCL
  1 ObjectLinkList EKG_BOUNDARY,
    3 Len      Integer,
    3 List(1),
      5 ObjectIdentifier   ObjectID,
      5 FieldIdentifier    FieldID;
```

C Declaration:

```
typedef    _Packed struct {
             Integer     Length;
             ObjectLink  List[1];
           } ObjectLinkList;
```

ObjectName (Reserved)

Data Type Identifier: 18

Description: The data type of the MyName field of an object. The name consists of no more than 254 characters, terminated by one byte of X'00'. The structure of ObjectName data is a 2-byte length field followed by the characters in the string. The null terminating character is not included in the length field. See "Object Names" on page 208 for information about valid object names.

Null Value: Length field is zero; in PL/I, set with *string* = '

PL/I Declaration:

```
% ObjectName = 'CHAR(254) VARYING';
```

C Declaration:

```
typedef    _Packed struct {
             Smallint Name_length;
             char     Name_content[255];
           } ObjectName;
```

RecipientSpec (Reserved)

Data Type Identifier: 20

Description: Information that notification methods require to notify an application program. A sequence of data including an 8-byte ApplicationID, an 8-byte notification-queue SubscribeID, and an 8-byte user word of data type Anonymous.

Null Value: Concatenation of a null Application ID, a null SubscribeID, and a null Anonymous(8) string.

PL/I Declaration:

```
DCL
  1 RecipientSpec EKG_BOUNDARY,
    3 User_appl_ID         ApplicationID,
    3 Notification_queue   SubscribeID,
    3 User_word            Anonymous(8);
```

C Declaration:

```
typedef    _Packed struct {
             ApplicationID  User_appl_ID;
             SubscribeID    Notification_queue;
             Anonymous      User_Word[8];
           } RecipientSpec;
```

SelfDefining

Data Type Identifier: 19

Description: A string of no more than 32767 bytes that is a concatenation of tagged data items; each tagged data item represents a RODM abstract data-type ID followed by its corresponding data. All reserved abstract data types can be used in SelfDefining data strings except the Anonymous(N) data type.

Figure 46 shows the format of SelfDefining data.

Self_Defining

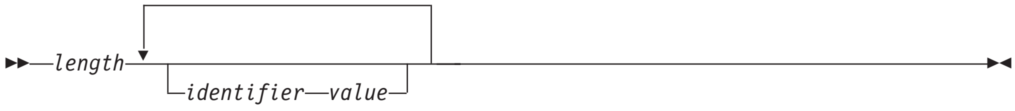


Figure 46. SelfDefining Data Type Syntax

The following variables are used in the SelfDefining syntax:

`length`
A 2-byte integer that specifies the total length of the SelfDefining data string excluding the 2-byte length field itself.

`identifier`
A 2-byte unsigned integer that specifies the RODM data type of the data that immediately follows the identifier in the SelfDefining data string. Data type identifiers are specified in the RODM data type definitions in “Abstract Data Type Reference” on page 223.

`value`
The value of the data that is specified by *identifier*. For values that are of data type ObjectName and ShortName, the null terminator is not included in the SelfDefining data string.

When specifying a CharVar inside a SelfDefining data string, you must include the 1-byte null terminator in the length field of the SelfDefining data string, but do not include it in the length field of the CharVar specification within the SelfDefining data string.

Figure 47 on page 235 shows an example SelfDefining string that contains a Smallint with a 2-byte (short integer) hexadecimal representation of the value `2010`, a CharVar with the value `RODM`, and an 8-byte application ID value of `NETV61`, which is padded on the right with two blanks.

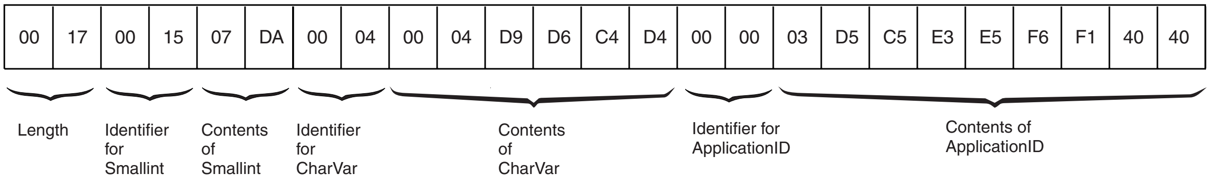


Figure 47. Example SelfDefining Field

Null Value: Length field is zero.

PL/I Declaration:

```
% SelfDefining = 'CHAR(32767) VARYING';
```

C Declaration:

```
typedef    _Packed struct {
             Smallint  Data_length;
             Anonymous Data_content;
           } SelfDefining;
```

ShortName (Reserved)

Data Type Identifier: 23

Description: Data type of the MyName field on a class and MyPrimaryParentName field on any object or class. The name consists of no more than 64 characters, terminated by one byte of X'00'. The structure of ShortName data is a 2-byte length field followed by the characters in the string. For information about constructing field names, see “RODM Fields” on page 210.

Null Value: Length field is zero; in PL/I, set with *string* = '.

PL/I Declaration:

```
% ShortName = 'CHAR(64) VARYING';
```

C Declaration:

```
typedef    _Packed struct {
             short  Name_length;
             char   Name_content[65];
           } ShortName;
```

Smallint

Data Type Identifier: 21

Description: A 2-byte (half-word) signed integer for general use.

Null Value: All bits are zero.

PL/I Declaration:

```
% Smallint = 'FIXED BINARY(15)';
```

C Declaration:

```
typedef  short  Smallint;
```

SubscribeID (Reserved)

Data Type Identifier: 22

Description: The 8-character notification queue name that is used to associate a field with a notification queue when the field is subscribed to. The association is established during the subscription process. The characters are positioned left-justified within the eight bytes and padded with blanks (for code page 00500, X'40') on the right.

Null Value: All bytes are blank (X'40' for code page 00500).

PL/I Declaration:

```
% SubscribeID = 'CHAR(8)';
```

C Declaration:

```
typedef    _Packed struct {
             char     Data_char[8];
           } SubscribeID;
```

SubscriptSpec (Reserved)

Data Type Identifier: 24

Description: A method specification plus a recipient specification used to record a notification request in the RODM program. The SubscriptSpec includes information about the method, the method parameters, and the intended recipient of the notification.

Null Value: Concatenation of a null MethodSpec and a null RecipientSpec

Note: The MethodSpec data type, a part of the SubscriptSpec data type, consists of an ObjectID and a method parameter list. The method parameter list is self-defining and is, in PL/I syntax, CHAR(254) VARYING.

SubscriptSpecList (Reserved)

Data Type Identifier: 25

Description: The data type of a notify subfield. This data type contains a list of SubscriptSpec elements, where each SubscriptSpec element represents a notification subscription. The length field of SubscriptSpecList is the number of elements in the list, not the length in bytes. All SubscriptSpec elements in the list are concatenated and contiguous.

Null Value: All bits are zero.

PL/I Declaration:

```
DCL
  1 SubscriptSpecList EKG_BOUNDARY,
    3 Len      Integer,
    3 Text     CHAR(1);
```

C Declaration:

```
typedef    _Packed struct {
             Integer Length;
             char    Text[1];
           } SubscriptSpecList;
```

TimeStamp

Data Type Identifier: 27

Description: The time value represented in Lilian milliseconds (eight bytes). Lilian milliseconds is the number of milliseconds since midnight 14 October 1582, which marks the beginning of the use of the Gregorian calendar. The time range provided is from 14 October 1582 through 31 December 9999. This is similar to the time format that is supported by the Common Execution Library for IBM compilers. To use this time with the Common Execution Library routines, divide the value by 1000.

Generation of this time format assumes that the Time-of-day (TOD) clock is set to Greenwich Mean Time (GMT) and based on the standard epoch.

Null Value: All bits are zero.

PL/I Declaration:

```
% TimeStamp = 'FLOAT BINARY(53)';
```

C Declaration:

```
typedef  double  TimeStamp;
```

TransID (Reserved)

Data Type Identifier: 28

Description: The transaction ID is a unique identifier of a RODM transaction.

Null Value: All bits are zero.

PL/I Declaration:

```
% TransID = 'CHAR(8)';
```

C Declaration:

```
typedef    _Packed struct {
             char   Content[8];
           } TransID;
```

Chapter 10. Using the RODM Load Function

This chapter describes how to create your own data model and load object definitions using the RODM load function. You create a data model as part of creating a new RODM application that does not use a data model that is supplied by IBM. This can be done by modifying an existing model or creating an entirely new data model using RODM load function statements.

The RODM load function enables you to create a data model and define its initial data values. It enables you to create, modify, and delete RODM classes and objects while the RODM program is running. You create sequential data sets that contain the load function statements. The load function reads the input data sets and loads the information into the RODM data cache.

This chapter contains five sections:
- Considerations when designing a data model
- Introduction to the RODM load function
- Using load function statements
- Process for loading the data cache
- Load function reference

You can use the load function to update an existing data model while RODM is running. You can run the load function using an initialization method so that it runs before RODM accepts any other transactions.

Considerations When Designing a Data Model

RODM classes can have objects as children, other classes as children, or both objects and other classes as children. You can add a new class or a new object to a parent class, as shown in Figure 48.

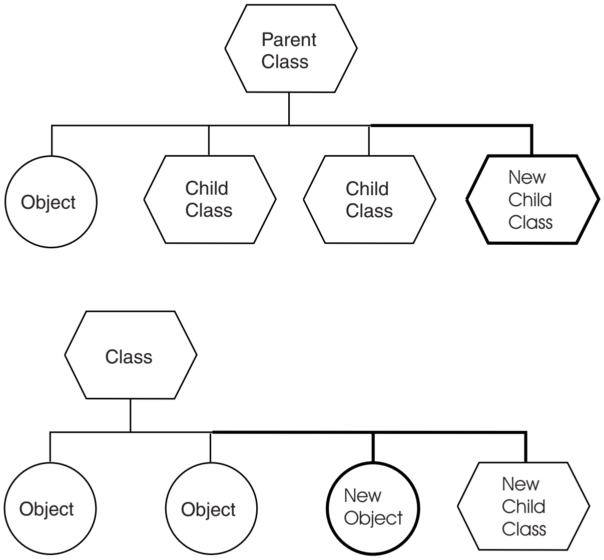


Figure 48. Adding Objects and Classes

Introduction to the RODM Load Function

The RODM load function is a part of RODM that shares libraries with RODM, but operates like an application program through the RODM user application program interface (API). It performs operations on the RODM data cache using load function statements. You code these statements in sequential files which are used as input to the RODM load function.

Load Function Statements

Two different levels of load function statements are processed by the RODM load function:
- High-level load function statements
- Load function primitive statements

RODM ***high-level*** load function statements are the statements most commonly used when defining your data model hierarchy. During RODM load function processing each of these statements is parsed into one or more RODM load function primitive statements. These primitive statements are then processed for syntax and action.

RODM load function ***primitive*** statements are the low-level syntax statements. They are either generated by the RODM load function from processing high-level statements or used directly as input to the RODM load function for loading and managing the RODM data cache. Each primitive statement corresponds closely to a user API call, but in some cases can include more than one user API call.

In addition, there are ***common syntactic elements*** which are a group of described variables used in RODM high-level load function syntax and RODM load function primitive syntax.

Load Function Operations

The RODM load function provides three different operations that enable you to load, update, and validate the contents of the RODM data cache. These three operations are:
- Parse
- Load
- Verify

The ***parse*** operation processes the load function input files and tests the syntax of all of the statements. No changes are made to the data cache, and RODM does not need to be running when you use the parse operation. This operation returns error messages for any statements in the load function input files that contain syntax errors. However, it cannot generate errors for problems such as assigning a value to a field that does not exist.

The ***load*** operation parses the load function input files and updates the contents of the RODM data cache. The load function input files can contain both high-level load function statements and load function primitive statements.

The RODM load function returns error messages for any statements in the load function input files that contain syntax errors. The load function also returns error messages for any request that does not complete successfully, even if the syntax was correct. For example, if you try to assign a value to a field which does not exist, the load function returns an error. Because the load function converts each high-level load function statement into several load function primitive statements as part of its processing, you might receive error messages which describe problems with load function primitives when you code a high-level load function statement.

Before you run the load operation, run the parse operation and correct any syntax errors. Then, use the load operation to create or update the contents of the data cache. You can update the data cache using the load function any time RODM is running.

The ***verify*** operation parses the load function input files and compares the statements with the contents of the data cache. No changes are made to the data cache, but RODM must be running to use the verify operation. The verify operation enables you to determine if specified classes, objects, and fields exist in the data cache. You can also determine if a field has a specified value. See "Understanding the Verify Operation" on page 258 for a more detailed description of the verify operation.

Loading the RODM Data Cache

After you create the RODM load function input files, you need to run the load function to load the RODM data cache. You call the RODM load function either as:
- An initialization method run at RODM start
- A module call from a program
- A JCL batch job

You have different types of loads from which to choose:

Initialization	You load the methods, the class structure, and the object definitions at RODM start.
Structure only	You load only the methods and the class structure definitions—a structure load.
Object only	You load only the object definitions—an object load.

The RODM load function loads the RODM data cache with a data model based on definitions in the load function input data sets. These data sets are identified to the RODM load function by the JCL data definition (DD) statements labeled:

EKGIN1	Class structure definitions
EKGIN2	Method name table
EKGIN3	Object definitions

For more information about loading the RODM data cache, see "Process for Loading the RODM Data Cache" on page 244.

Using Load Function Statements

This section describes the RODM high-level load function statements and the RODM load function primitive statements, and when to use them. The RODM load function uses these statements to issue RODM user API calls that cause RODM to:
- Create classes, objects, fields, and subfields
- Delete classes, objects, fields, and subfields
- Set fields to initial values
- Establish the parent-child relations that define the hierarchy
- Set the values of fields
- Trigger methods

High-Level Load Function Statements

This topic describes the RODM high-level load function statements. For information about coding these statements, see "Coding RODM High-Level Load Function Statements" on page 273.

The four RODM high-level load function statements are:

MANAGED OBJECT CLASS
The RODM high-level load function class structure syntax you use to build the hierarchy of the data model in the RODM data cache by adding class definitions and setting initial values.

CREATE
The RODM high-level load function object syntax you use to create an object of a class in the RODM data cache.

DELETE
The RODM high-level load function object syntax you use to delete an object from the RODM data cache.

SET The RODM high-level load function object syntax you use to set the values of fields of objects in the RODM data cache.

When RODM high-level load function statements are processed, each RODM high-level load function statement is first converted to RODM load function primitive statements. For example, the following MANAGED OBJECT CLASS high-level load function statement defines a child class named SNA_Domain_Class with a field named SNANet under the class named Domain_Parent_Class:

```
SNA_Domain_Class            MANAGED OBJECT CLASS;
  PARENT IS Domain_Parent_Class;
  ATTRLIST
   SNANet                              CHARVAR;
END;
```

The high-level statement is parsed by the RODM load function and results in the following RODM load function primitive statements:

```
OP SNA_Domain_Class HAS_PARENT Domain_Parent_Class;
OP SNA_Domain_Class HAS_FIELD (CHARVAR) SNANet;
```

Each RODM load function primitive statement is then processed for syntax and action. See "Load Function Primitive Statements" for more information about RODM load function primitive statements.

If any of the RODM load function primitive statements generated for a RODM high-level load function statement encounters an error, any subsequent RODM load function primitive statements for that RODM high-level load function statement will be ignored. That means any syntax errors following the detected error within the bounds of the RODM high-level load function statement being processed will not be detected.

Load Function Primitive Statements

The RODM load function primitives are an external interface that is at a lower level than the RODM high-level load function statements described in "High-Level Load Function Statements." For information about how to code RODM load function primitive statements, see "Coding RODM Load Function Primitive Statements" on page 282.

RODM load function primitives come directly from user-generated input files or are generated by the RODM load function from RODM high-level load function statements within the input files. Both RODM load function primitives and RODM high-level load function statements can be used in the same RODM load function input file, but load function primitives cannot be coded within a high-level statement.

The load function processes primitive statements sequentially, one primitive statement at a time. The RODM load function interprets each of them according to their processing options and issues the appropriate user API calls to perform RODM functions. The primitives correspond very closely to the user API calls, but in some cases they can include more than one user API call.

When to Use High-Level or Primitive Load Function Statements

Use RODM high-level load function statements when you are:
- Performing the initial loading of a data model
- Making changes to the structure of the data model
- Adding a large number of classes or objects into the RODM data cache, where using RODM load function primitives is cumbersome

Use RODM load function primitives to define class structure changes that involve the deletion of classes, the modification of classes, the modification of the hierarchy, or when a desired function cannot be performed by a high-level statement.

The following RODM load function primitives perform functions that cannot be performed by RODM high-level load function statements for objects or classes:

FORCE_HAS_NO_INSTANCE
Unconditionally, deletes an object after unlinking any links the object has.

FORCE_NOT_A_CLASS
Unconditionally, deletes a class and any children of the class, regardless of links.

HAS_NO_FIELD
Deletes a field within a class.

HAS_NO_SUBFIELD
Deletes a subfield within a field.

INVOKED_WITH
Triggers a named or object-independent method.

NOT_A_CLASS
Conditionally deletes a childless class.

The following RODM load function primitives perform functions that cannot be performed on classes by RODM high-level load function statements:

Note: RODM high-level load function statements can perform these functions on objects.

HAS_VALUE
Defines a value for a field within a class.

The RODM high-level load function statement MANAGED OBJECT CLASS can define an initial value for the field of a specific class, but it cannot be used to change the value.

INHERITS
Removes the locally defined value for the specified class field and reverts the field value to the value that it inherited from its parent.

SUBFIELD_HAS_VALUE
Defines a value for a subfield within a class.

Only the value subfield can be initialized for the class by the RODM high-level load function statement MANAGED OBJECT CLASS.

SUBFIELD_INHERITS
Removes the locally-defined value for the specified class subfield and reverts the subfield value to the value that it inherited from its parent.

You can code the primitives for either a structure load or an object load, but you must define all of the structure first and then define the objects because you must ensure that parent classes are created before their class children or their object children are created.

When it is easier to perform an operation with a RODM load function primitive than with a RODM high-level load function statement, use a RODM load function primitive. For example, the field value of the field named SNANet of the object named CNM01 under the class named SNA_Domain_Class can be set to a new value with the SET high-level statement, but you need several lines of SET statement syntax:

```
SET  INVOKER   ::= 0001;
     MODE      ::= non-confirmed;
     OBJCLASS  ::= SNA_Domain_Class;
     OBJINST   ::= MyName = (CHARVAR) 'CNM01';
     MODLIST   SNANet ::= (CHARVAR) 'NETC';
END;
```

Whereas, you can use the HAS_VALUE primitive to set the field value of the object with only one line of syntax:

```
OP SNA_Domain_Class.CNM01.SNANet HAS_VALUE (CHARVAR) 'NETC';
```

Process for Loading the RODM Data Cache

This section describes the process used to load the RODM data cache using the RODM load function. The process steps are first listed in order and described in the same order.

To load the RODM data cache:

1. Identify the methods to install
2. Create the class structure and object definitions
3. Decide on the type of load
4. Run the RODM load function
5. Check the output listings

There are also optional steps which enable you to change member names and parameter mapping:
- Modify the control table
- Modify the parameter mapping table

Identifying the Methods to Install

When you load the class structure as part of an initial load or a class structure change, you can also install the methods. You identify the methods to be installed in the RODM address space in the method name table (EKGINMTB). The table is a member of the partitioned data set identified by the EKGIN2 DD statement. See “Method Name Table” on page 261 for information about the format of the table and other associated DD statements.

When you run the RODM load function and specify LOAD=STRUCTURE, the RODM load function performs the following steps for each method name specified in the method name table:

1. Searches STEPLIB DD data sets to ensure method is available
2. Creates a method object
3. Installs the method

If the method is already installed or is specified twice in the method name table, the RODM load function will issue the error message:

```
EKG8568W -

THE METHOD method_name HAS NOT BEEN INSTALLED AS IT ALREADY EXISTS
```

You must have an EKGIN2 file. If you are installing no methods, the EKGIN2 file is an empty file. The methods must reside in one of the data sets identified by the STEPLIB DD statement in the target RODM start up JCL.

Creating the Class Structure and Object Definitions

Create sequential files that contain your class structure and object definitions, when you are:

- Performing the initial load of the class structure and object definitions into the RODM data cache
- Making changes to the structure of the data model or defined objects in the data cache

These definitions consist of RODM high-level load function statements and RODM load function primitives. See “Using Load Function Statements” on page 241 for more information about using RODM high-level load function statements and RODM load function primitives.

Data Definition Statement Labels

The RODM load function expects to find the DD statements that declare the sequential data set or the concatenation of sequential data sets that contain the load function input definitions to be labeled:

- EKGIN1 for the class structure definitions
- EKGIN3 for the object definitions

Although this is the load function's expectation, practically, you can put all your definitions into a single sequential data set or concatenation of sequential data sets. You choose either EKGIN1 or EKGIN3 as the DD name of the DD statement that identifies the data set depending on the type of load. See “Deciding on the Type of Load” on page 246 for information about the type of load dependency.

This technique is especially useful for incremental data cache changes, but it is very important that you observe the concatenation caveats described in “Concatenation of Data Sets” on page 246.

Concatenation of Data Sets

You can divide the class structure and object definitions into several sequential data sets and then concatenate the data sets that contain these definitions. The order of the data sets in the concatenation is important. Whether you use RODM high-level load function statements or RODM load function primitives, you must arrange the files containing the definitions so that:

- RODM load function creates any parent class before it creates its children
- Class structure definitions precede any associated object definitions
- The statements that create objects are processed before the statements that create links between objects

You can concatenate object definitions so that each data set contains one or more object definitions, and a data set can represent a domain, a subarea, or whatever makes sense. By structuring your data sets in this way, you can facilitate adding or refreshing information for a domain.

Definition Examples

RODM provides two sample files in the samples library partitioned data set named CNMSAMP.

Member	Contents
EKGIN1	An example of load function statements designed to: • Create a class under the UniversalClass • Create fields for all data types supported • Set initial values for the fields
EKGIN3	An example of load function statements designed to: • Create 3 objects • Set initial values

Deciding on the Type of Load

The steps in the loading process differ, depending on how you intend to run the RODM load function and on what type of load you are performing. You can run the RODM load function as an initialization method during a cold start of RODM or during a warm start of RODM. You can run the RODM load function by means of a JCL job. You can run the RODM load function by means of a module call from an application. The RODM load function offers the following of load types:

- Initialization load
- Structure load only
- Object load only

Initialization Load

In an initialization load, you can load the class structure, the names of the methods to install, and the object definitions. This is done at RODM cold start by invoking EKGLISLM.

Initialization requires three DD statements for input data with the following labels:

EKGIN1
Class structure definitions

EKGIN2
Method name table

EKGIN3
Object definitions

When RODM initialization takes place, the RODM load function (EKGLISLM), is triggered to create the RODM structure. This initial load method runs an object-independent method that sets the values of the objects in the RODM data cache. After completion of the initial load, further changes are usually modifications of defined objects or the addition of new object definitions.

In an initial load, you cannot directly specify the RODM load function parameters. RODM uses a parameter mapping table (EKGPTENU). If you want to change the default values of the parameters, change the default values in the parameter mapping table. When the load function is initially run, the load function parameters get their default values from the parameter mapping table. However, the load function ignores any abbreviations or string substitutions in the table. See "Parameter Mapping Table" on page 262 for information about creating your own parameter mapping table or modifying the table copied during RODM installation. For a display of the parameter mapping table that EKGPTENU supplied with RODM, see Figure 60 on page 264.

Structure Load Only

A ***structure load*** is a load in which you load only the methods and the class structure into RODM. This is generally done as a job containing JCL or a module call while RODM is running.

EKGIN2 Data Definition: RODM load function first processes the data definition statement with the label EKGIN2, which specifies the partitioned data set that contains the method name table in one of its members. The name of the member that contains the method name table is found by RODM in the control table EKGCTABL. For information about control table EKGCTABL and how to optionally modify or create a new table, see "Control Table—EKGCTABL" on page 260.

For each entry in the method name table, the RODM load function:

1. Searches the data sets identified by the STEPLIB DD statement in the RODM start up JCL to see if the method is installed. If the method is not installed, a return code of 8 and a reason code of 81 is returned and the load function issues an error message.
2. Converts into RODM user API calls the load function primitives that associate the entries in the method name table with the MethodName fields of the appropriate classes. In other words, adds an object to the RODM EKG_Method class.
3. Loads the method into the RODM address space.

EKGIN1 Data Definition: During a structure load, whether an initial structure load or a structure change, the RODM load function processes the EKGIN1 data definition statement after the EKGIN2 data definition statement processing is complete.

EKGIN1 identifies the sequential data set or concatenation of sequential data sets that contain the load function input statements that specify the classes and their parents.

The RODM load function reads this input as a stream of class definitions in sequential order, and parses all RODM high-level load function statements into RODM load function primitives. The RODM load function then converts the load function primitives to a succession of RODM user API calls, which create the classes in your RODM data cache.

When concatenating data sets, the order of the data sets in the EKGIN1 DD statement is important. Load the data sets that contain parent classes before those that contain their children. Figure 49 shows a concatenation of data sets for the EKGIN1 DD statement.

```
//EKGIN1   DD   DSN=parent.class.input.dataset1,DISP=SHR    (All parent classes)
//         DD   DSN=child.class.input.dataset1,DISP=SHR     (Domain 1 children )
//         DD   DSN=child.class.input.dataset2,DISP=SHR     (Domain 2 children )
//         DD   DSN=child.class.input.dataset3,DISP=SHR     (Domain 3 children )
```

Figure 49. Data Set Concatenation for EKGIN1

Object Load Only

In an object load, you can load only the object definitions. You can load object definitions as a job or as a module call while RODM is running. The object load uses one DD statement labeled EKGIN3 to identify the sequential data set or concatenation of sequential data sets that contain the object definitions for the load.

When you concatenate data sets, be sure that the statements that create objects are processed before the statements that create links between objects. Both objects being linked must be in RODM when the link statement is processed. Concatenation takes the standard z/OS format for concatenated data sets. Figure 50 shows a concatenation of data sets for the EKGIN3 DD statement.

```
//EKGIN3   DD   DSN=object.instance.input.dataset1,DISP=SHR    (Domain 1)
//         DD   DSN=object.instance.input.dataset2,DISP=SHR    (Domain 2)
//         DD   DSN=object.instance.input.dataset3,DISP=SHR    (Domain 3)
```

Figure 50. Data Set Concatenation for EKGIN3

Running the RODM Load Function

This topic contains a description of invoking the RODM load function, plus considerations when running the load function, in the following order:
- The load function as an initialization method
- Invoking the load function as a batch job
- Running the load function from a module
- Considerations when running the load function

You can run the RODM load function by running it as an initialization method, as a job, or as a module call. A RODM load function job can parse the data model, load the data model into the RODM data cache, or verify the data model.

A good practice is to parse your data model definition before you attempt to load it. This can reduce the number of errors that occur during the load. This practice enables you to identify and correct errors in your load function input statement syntax prior to loading these definitions into your RODM data cache.

The Load Function as an Initialization Method

Use the initialization method provided with the NetView program or you can write one. In either case, before the initialization method can be triggered, an object with the name of the method must be created in the EKG_Method class by the user or by the RODM load function.

The initialization method that is supplied by the NetView program has two parts:

EKGLISLM

Loads the methods defined in the method name table identified by the EKGIN2 DD statement; loads the class structure definitions in the sequential data set or concatenation of sequential data sets identified by the EKGIN1 DD statement; and then triggers EKGLIILM.

EKGLIILM
Loads the object definitions in the sequential data set or concatenation of sequential data sets identified by the EKGIN3 DD statement.

EKGLISLM and EKGLIILM run as methods in the RODM address space. These methods use the environment that RODM passes to them and operate as object-independent methods.

Cold Start (Initialization): To initialize RODM and load the data cache from a cold start, you specify the name of the initialization method using the INIT= parameter of the RODM start up command. You run a program (EKGTC000), which triggers EKGLISLM, the load function initialization method, which in turn triggers EKGLIILM. Because a cold start requires a structure load, you do not specify INIT=EKGLIILM as a parameter of the RODM start up command for a cold start.

NetView provides an example of a RODM start up procedure named EKGXRODM. This procedure performs an initialization load, but before running this start up procedure, make the following modifications to the start up procedure JCL:

- Change the specification of *USER.METHODS* for DSN= parameter on the STEPLIB DD statement to reflect the name of the partitioned data set containing your user-written methods. If there are none, comment out or delete this statement.
- Ensure that EKGIN1 and EKGIN3 DD statements identify your class structure and object definitions. The supplied procedure identifies data sets that contain examples of how to code the definitions.
- Remove the comment delimiters from all other JCL statements.

You run the procedure by entering:

```
S EKGXRODM,TYPE=C,INIT=EKGLISLM
```

In this example:

- EKGXRODM is the procedure name
- TYPE=C specifies a cold start operation
- INIT=EKGLISLM specifies the name of the method to trigger

Warm Start: Although you can use EKGLISLM to load the class structure and object definitions into the data cache at warm start, just like a cold start, you normally specify EKGLIILM for the INIT= parameter to load only the object definitions. Usually you are warm starting to change the network configuration or as a result of an error.

NetView provides an example RODM start up procedure named EKGXRODM. Use it to perform the object definition load. Before running the procedure, make the following modifications to the sample procedure's JCL to load only the object definitions:

- Comment out the C Library in the STEPLIB DD, if necessary, as described in the notes in the procedure heading.
- Ensure that the EKGIN3 DD statement identifies your definitions. The supplied procedure identifies the data set that contains examples of how to code the object definitions.

- Remove the comment delimiters from only the EKGLUTB, EKGPRINT and EKGIN3 DD statements.

Run the procedure by entering:

```
S EKGXRODM,TYPE=W,INIT=EKGLIILM
```

where:
- EKGXRODM is the procedure name
- TYPE=W specifies a warm start operation
- INIT=EKGLIILM specifies the name of the method to trigger

Invoking the Load Function As a Batch Job

You can run the RODM load function as a batch job. The RODM load function uses the verified user ID of the job submitter as the User_appl_ID to connect to RODM. The verified user ID is obtained from the system authorization facility. This user ID must have a minimum RODM authorization level of 3 or 5, depending on the load function statements used. See “Authorization and Authorization Levels” on page 252 for the required authorization level.

Your job can load:
- The object definitions only
- The methods and class structure definitions
- The methods and all the definitions

NetView supplies a sample job and procedure to run the RODM load function as a batch job. The sample job EKGLLOAD calls the procedure EKGLOADP and passes the parameters you specify. The following sections show how to update the EKGLLOAD sample job for each of the three ways you can load RODM.

Loading Object Definitions Only: Copy the sample job EKGLLOAD and update it to load object definitions into RODM. Update the system level qualifier in the EKGLOADP procedure if you do not use NETVIEW.V6R1M0 as the high-level qualifiers of the RODM data sets on your system. The following steps give example values for the parameters passed by the EKGLLOAD job to the EKGLOADP procedure. Provide your own values for each parameter.

1. Update the JOB statement with your accounting information.
2. Fill in RODMNAME with the name of your RODM.
3. Fill in EKGIN3 with the name of the data set that contains your object definitions.
4. Ensure RODM is running and submit the EKGLLOAD job.

Figure 51 shows the lines in EKGLLOAD updated with example values.

```
//STEP01   EXEC EKGLOADP,
//              RODMNAME=EKGXRODM,
//              EKGIN3=NETVIEW.V6R1M0.CNMSAMP(EKGIN3)
```

Figure 51. Object Load Batch Job Using EKGLLOAD Sample

Loading Method Names and Class Structure: Copy the sample job EKGLLOAD and update it to load class and method definitions into RODM. Update the system level qualifier in the EKGLOADP procedure if you do not use NETVIEW.V6R1M0 on your system. The following steps give example values for the parameters passed by the EKGLLOAD job to the EKGLOADP procedure. Provide your own values for each parameter:

1. Update the JOB statement with your accounting information.

2. Fill in RODMNAME with the name of your RODM.
3. Fill in EKGIN1 with the name of the data set that contains your class definitions.
4. Specify `LOAD=STRUCTURE` for a class and method load.
5. Ensure RODM is running and submit the EKGLLOAD job.

Your methods are defined in the method table in NETVIEW.V6R1M0.CNMSAMP. You do not need to specify this data set name. Figure 52 shows the lines in EKGLLOAD updated with example values.

```
//STEP01   EXEC EKGLOADP,
//            RODMNAME=EKGXRODM,
//            EKGIN1=NETVIEW.V6R1M0.CNMSAMP(EKGIN1),
//            LOAD=STRUCTURE
```

Figure 52. Class and Method Load Batch Job Using EKGLLOAD Sample

Loading Method Names and All Definitions: You have two options to load the classes, methods, and objects using the EKGLLOAD sample job:

- Load the classes and methods first, following the steps in "Loading Method Names and Class Structure" on page 250 and then load the objects, following the steps in "Loading Object Definitions Only" on page 250.
- Put all of the class, method, and object definitions in a single data set and load that data set by following the steps in "Loading Object Definitions Only" on page 250.

Instead of putting all of the definitions in a single data set, you can concatenate separate data sets. This requires updating the EKGLOADP procedure, because the EKGLLOAD job can pass only one data set as a parameter.

Calling the Load Function from a Module

To run the RODM load function from a module, call the appropriate entry point for the language that you are using. The RODM load function uses the verified user ID, associated with the calling program at execution time, as the User_appl_ID to connect to RODM. The verified user ID is obtained from the system authorization facility. This user ID must have a minimum RODM authorization level of 3 or 5, depending on the load function statements used. See "Authorization and Authorization Levels" on page 252 for the required authorization level. If a listing is requested, the listing and other information are written to the specified data set for use by the calling module.

You must specify RMODE=24 when you link-edit the RODM load function module.

From Modules Written in PL/I and C: User application programs written in PL/I or C that call the RODM load function directly must call the EKGLJOB entry point. The linkage to EKGLJOB must adhere to z/OS conventions as described in "z/OS Linkage Conventions" on page 266. The RODM load function runs all load functions in the user application program task control area environment.

From Modules Not Written in PL/I or C: User application programs not written in PL/I or C that call the RODM load function directly must call the EKGLOTLM entry point. The EKGLOTLM entry point creates a task control area environment in which all load functions are run. Use the same linking conventions as for EKGLJOB. See "z/OS Linkage Conventions" on page 266.

Considerations When Running the RODM Load Function

The RODM Load Function: When running the RODM load function, you can run only one RODM load function job per address space. Ensure that the PL/I runtime libraries are installed or available prior to submitting or running a job. The RODM load function sets the value of the EKG_StopMode field to 3 before disconnecting. (Do not purge notification queues or subscriptions.) This value enables the RODM load function to disconnect without purging any notification subscriptions, notification queues, or notification methods that are created as the result of methods triggered by the RODM load function.

The RODM Program: The RODM program must be running for OPERATION=LOAD and for OPERATION=VERIFY because the RODM load function issues a connect request to RODM to access the data cache. If RODM is not running, an error message is issued.

RODM does not need to be running for OPERATION=PARSE. With OPERATION=PARSE, the RODM load function reads the load function input files and parses them to find syntax errors. The RODM load function issues the connect function to RODM and queries the RODM version and release. Errors found in the connect and query function are logged in the Job log and RODM log. However, these errors are not considered as errors of the RODM load Parse operation. For more information about OPERATION=, see "OPERATION" on page 271.

Ensure that the name you use to run the RODM load function is the same as the name of the RODM program that is running. The specification for the NAME= parameter must equal the name of the running RODM program. For information about parameter NAME=, see "NAME" on page 271.

Authorization and Authorization Levels: The TSO ID and TSO password that you use to run the RODM load function and user application programs that run the RODM load function must be authorized by your system authorization facility to access RODM, unless the SEC_CLASS keyword is set to *TSTRODM in customization file EKGCUST.

The ID that runs the load function must have an authorization level of at least 3 or 5, depending on the load function statements used. Table 23 shows the load function statement, the statement type, the minimum authorization level, and a reference to additional information about the statement.

Table 23. Load Function Statements and Minimum Authorization Levels

Statement	Statement Type	Minimum Authorization Level	See Page
CREATE	High-level	3	278
DELETE	High-level	3	279
FORCE_HAS_NO_INSTANCE	Primitive	3	283
FORCE_NOT_A_CLASS	Primitive	5	283
HAS_FIELD	Primitive	5	284
HAS_INSTANCE	Primitive	3	284
HAS_NO_FIELD	Primitive	5	285
HAS_NO_INSTANCE	Primitive	3	285
HAS_NO_SUBFIELD	Primitive	5	286
HAS_PARENT	Primitive	5	286

Table 23. Load Function Statements and Minimum Authorization Levels (continued)

Statement	Statement Type	Minimum Authorization Level	See Page
HAS_PRV_FIELD	Primitive	5	286
HAS_SUBFIELD	Primitive	5	287
HAS_VALUE	Primitive	3	287
INHERITS	Primitive	3	288
INVOKED_WITH	Primitive	3	288
IS_LINKED_TO	Primitive	3	289
IS_NOT_LINKED_TO	Primitive	3	289
MANAGED OBJECT CLASS	High-level	5	276
NOT_A_CLASS	Primitive	5	290
SET	High-level	3	280
SUBFIELD_HAS_VALUE	Primitive	3	290
SUBFIELD_INHERITS	Primitive	3	291

Checking the Output Listings

To understand the output listings, you must understand the format of the output messages and the contents of the output listing.

Note: Refer to the NetView online help for a description of the messages issued by the RODM load function. All RODM load function messages start with EKG8.

Two output listings consisting of different types of information are created when you run the RODM load function. One listing is created by the RODM load function and is written to the data set specified by the EKGPRINT DD statement. The other is system-generated output and is directed to SYSOUT. If the EKGPRINT DD statement specifies SYSOUT as the output data set, the separate listings appear as one report.

RODM Load Function Output Listing

The listing created by the RODM load function contains the date, the name of the function with its current level, a list of the options used when the load function was run, load function input, actions taken by the function, echoed syntax when an error occurs, and messages including an END OF JOB message. See Figure 55 on page 257 for an example of the load function output listing for an object load.

When displaying the contents of the data set identified by the EKGPRINT DD statement, ensure that the software and hardware used can do so in mixed case. RODM data is case sensitive, and to display the data in other than mixed case hinders your verification of the RODM load.

All syntax can be echoed, interleaved with messages, where appropriate, indicating the success or failure of the primitive that was performed, or only syntax errors can be echoed, with messages indicating where errors are detected. The LISTLEVEL parameter as described on page 270 defines which level of syntax echoing occurs.

RODM Load Function Output Format

Formats differ slightly for the RODM load function output, depending on the following:

- Type of operation: PARSE, LOAD, or VERIFY
- Type of load: STRUCTURE or INSTANCE
- LISTLEVEL option: ERRORSYNTAX or ALLSYNTAX

For more information about these parameters, see “RODM Load Function Parameter Syntax” on page 269.

Compare the following figures for format differences:

- Figure 53 on page 255, a PARSE operation output example
- Figure 54 on page 256, a structure load output example
- Figure 55 on page 257, an object load output example

```
OPTIONS USED
------------
OPERATION:PARSE
NAME:RODMNAME
SEV:WARNING
LISTLEVEL:ALLSYNTAX
CODEP:EKGCP500
LOAD:INSTANCE
ROUTECODE:1
INSTANCE  ELEMENTS PROCESSED
   .
   .
   .
--*  DESCRIPTION: SAMPLE STRUCTURE LOAD INPUT FILE                  *--
   .
   .
   .
SUPERCLASS                           MANAGED OBJECT CLASS;
PARENT IS                            UNIVERSALCLASS;
ATTRLIST
   FIELD_ANONYMOUSVAR    ANONYMOUSVAR  INITIAL (X'4040'),
   FIELD_BERVAR          BERVAR        INIT(X'810499FF88FF'),
   FIELD_CHARVAR         CHARVAR       INIT ('ANYCHARACTER'),
   FIELD_INDEXCHAR1      CHARVAR       INIT ('INDEXNAME') PUBLIC_INDEXED,
   FIELD_CLASSID         CLASSID,
   FIELD_FIELDID         FIELDID       INIT (SUPERCLASS.FIELD_CHARVAR),
   FIELD_FLOATING        FLOATING      INIT (50.00),
   FIELD_GRAPHICVAR      GRAPHICVAR    INIT ( DBCSDATA ) PRIVATE,
   FIELD_INTEGER         INTEGER       INIT(50) PUBLIC,
   FIELD_OBJECTID        OBJECTID,
   FIELD_OBJECTLINK      OBJECTLINK,
   FIELD_OBJECTLINKLIST  OBJECTLINKLIST,
   FIELD_SMALLINT        SMALLINT      INIT(50),
   FIELD_TIMESTAMP       TIMESTAMP     INIT(X'41B8CCCCCCCCCCCD'),
   FIELD_METHODSPEC      METHODSPEC    INIT('EKGNOTF' ((INTEGER) 50)),
   FIELD_SELFDEFINING    SELFDEFINING,
   FIELD_INDEXLIST1      INDEXLIST,
   FIELD_INDEXINDEXLIST1 INDEXLIST     PUBLIC_INDEXED;
END;
BEGIN CLASS SUPERCLASS;* HAS_PARENT UNIVERSALCLASS;* HAS_FIELD (ANONYMOUSVAR)
HAS_VALUE (INTEGER) 50;* HAS_FIELD (OBJECTID) FIELD_OBJECTID;* HAS_FIELD
(OBJECTLINK) FIELD_OBJECTLINK;* HAS_FIELD (OBJECTLINKLIST)
HAS_VALUE (METHODSPEC) ('EKGNOTF' ((INTEGER) 50));* HAS_FIELD (SELFDEFINING)
FIELD_SELFDEFINING;* HAS_FIELD (INDEXLIST) FIELD_INDEXLIST1;*
HAS_INDEXED_FIELD (INDEXLIST) FIELD_INDEXINDEXLIST1;END CLASS *;
   .
   .
   .
END OF JOB    OVERALL RETURN CODE: 00    11:17:15
```

Figure 53. Example of PARSE Operation Output to EKGPRINT

```
OPTIONS USED
------------
OPERATION:LOAD
NAME:RODMNAME
SEV:WARNING
LISTLEVEL:ALLSYNTAX
CODEP:EKGCP500
LOAD:STR
ROUTECODE:1
STRUCTURE ELEMENTS PROCESSED
  .
  .
  .
--*  DESCRIPTION: SAMPLE STRUCTURE LOAD INPUT FILE                  *--
  .
  .
  .
SUPERCLASS                           MANAGED OBJECT CLASS;
PARENT IS                            UNIVERSALCLASS;
ATTRLIST
   FIELD_ANONYMOUSVAR    ANONYMOUSVAR  INITIAL (X'4040'),
   FIELD_BERVAR          BERVAR        INIT(X'810499FF88FF'),
   FIELD_CHARVAR         CHARVAR       INIT ('ANYCHARACTER'),
   FIELD_INDEXCHAR1      CHARVAR       INIT ('INDEXNAME') PUBLIC_INDEXED,
   FIELD_CLASSID         CLASSID,
   FIELD_FIELDID         FIELDID       INIT (SUPERCLASS.FIELD_CHARVAR),
   FIELD_FLOATING        FLOATING      INIT (50.00),
   FIELD_GRAPHICVAR      GRAPHICVAR    INIT ( DBCSDATA ) PRIVATE,
   FIELD_INTEGER         INTEGER       INIT(50) PUBLIC,
   FIELD_OBJECTID        OBJECTID,
   FIELD_OBJECTLINK      OBJECTLINK,
   FIELD_OBJECTLINKLIST  OBJECTLINKLIST,
   FIELD_SMALLINT        SMALLINT      INIT(50),
   FIELD_TIMESTAMP       TIMESTAMP     INIT(X'41B8CCCCCCCCCCCD'),
   FIELD_METHODSPEC      METHODSPEC    INIT('EKGNOTF' ((INTEGER) 50)),
   FIELD_SELFDEFINING    SELFDEFINING,
   FIELD_INDEXLIST1      INDEXLIST,
   FIELD_INDEXINDEXLIST1 INDEXLIST     PUBLIC_INDEXED;
END;
* HAS_PARENT UNIVERSALCLASS;
EKG8258I - THE HAS_PARENT PRIMITIVE STATEMENT COMPLETED SUCCESSFULLY.
* HAS_FIELD (ANONYMOUSVAR) FIELD_ANONYMOUSVAR;
EKG8258I - THE HAS_FIELD PRIMITIVE STATEMENT COMPLETED SUCCESSFULLY.
EKG8258I - THE HAS_VALUE PRIMITIVE STATEMENT COMPLETED SUCCESSFULLY.
* HAS_FIELD (BERVAR) FIELD_BERVAR;
EKG8258I - THE HAS_FIELD PRIMITIVE STATEMENT COMPLETED SUCCESSFULLY.
EKG8258I - THE HAS_VALUE PRIMITIVE STATEMENT COMPLETED SUCCESSFULLY.
* HAS_FIELD (CHARVAR) FIELD_CHARVAR;
EKG8258I - THE HAS_FIELD PRIMITIVE STATEMENT COMPLETED SUCCESSFULLY.
EKG8258I - THE HAS_VALUE PRIMITIVE STATEMENT COMPLETED SUCCESSFULLY.
* HAS_INDEXED_FIELD (CHARVAR) FIELD_INDEXCHAR1;
EKG8258I - THE HAS_INDEXED_FIELD PRIMITIVE STATEMENT COMPLETED SUCCESSFULLY.
  .
  .
  .
EKG8258I - THE SUBFIELD_HAS_VALUE PRIMITIVE STATEMENT COMPLETED SUCCESSFULLY.
END OF JOB    OVERALL RETURN CODE: 00    13:58:29
```

Figure 54. Example of Structure Load Output to EKGPRINT

```
OPTIONS USED
------------
OPERATION:LOAD
NAME:RODMNAME
SEV:WARNING
LISTLEVEL:ALLSYNTAX
CODEP:EKGCP500
LOAD:INSTANCE
ROUTECODE:1
INSTANCE  ELEMENTS PROCESSED
   .
   .
   .
--*  DESCRIPTION: SAMPLE INSTANCE LOAD INPUT FILE                   *--
   .
   .
   .
CREATE INVOKER  ::= 1;
       OBJCLASS ::= SUBCLASS_2;
       OBJINST  ::= MYNAME = (CHARVAR) 'INSTANCE_4';
       ATTRLIST
           FIELD_ANONYMOUSVAR ::= (ANONYMOUSVAR) X'ABCD',
           FIELD_BERVAR ::= (BERVAR) X'810499FF88FF',
           FIELD_CHARVAR ::= (CHARVAR) 'CHARTEST',
           FIELD_FIELDID ::= (FIELDID) SUPERCLASS.FIELD_INTEGER,
           FIELD_FLOATING ::= (FLOATING) 100.00,
           FIELD_INTEGER  ::= (INTEGER) 100,
           FIELD_SMALLINT ::= (SMALLINT) 100,
           FIELD_TIMESTAMP ::= (TIMESTAMP) X'41B8CCCCCCCCCCCD',
           FIELD_METHODSPEC ::= (METHODSPEC) ('EKGNOTF' ((INTEGER) 100));
END;
SUBCLASS_2 HAS_INSTANCE *;
EKG8258I - THE HAS_INSTANCE PRIMITIVE STATEMENT COMPLETED SUCCESSFULLY.
EKG8258I - THE HAS_VALUE PRIMITIVE STATEMENT COMPLETED SUCCESSFULLY.
EKG8258I - THE HAS_VALUE PRIMITIVE STATEMENT COMPLETED SUCCESSFULLY.
EKG8258I - THE HAS_VALUE PRIMITIVE STATEMENT COMPLETED SUCCESSFULLY.
EKG8258I - THE HAS_VALUE PRIMITIVE STATEMENT COMPLETED SUCCESSFULLY.
EKG8258I - THE HAS_VALUE PRIMITIVE STATEMENT COMPLETED SUCCESSFULLY.
EKG8258I - THE HAS_VALUE PRIMITIVE STATEMENT COMPLETED SUCCESSFULLY.
EKG8258I - THE HAS_VALUE PRIMITIVE STATEMENT COMPLETED SUCCESSFULLY.
EKG8258I - THE HAS_VALUE PRIMITIVE STATEMENT COMPLETED SUCCESSFULLY.
EKG8258I - THE HAS_VALUE PRIMITIVE STATEMENT COMPLETED SUCCESSFULLY.
   .
   .
   .
-- DELETE SUBFIELDS USING THE HAS_NO_SUBFIELD PRIMITIVE --

OP SUPERCLASS.FIELD_CHARVAR HAS_NO_SUBFIELD NOTIFY;
EKG8258I - THE HAS_NO_SUBFIELD PRIMITIVE STATEMENT COMPLETED SUCCESSFULLY.
OP SUPERCLASS.FIELD_CHARVAR HAS_NO_SUBFIELD PREV_VALUE;
EKG8258I - THE HAS_NO_SUBFIELD PRIMITIVE STATEMENT COMPLETED SUCCESSFULLY.
END OF JOB    OVERALL RETURN CODE: 00    13:58:46
```

Figure 55. Example of Object Load Output to EKGPRINT

Load Function Reference

This section contains additional reference information for the RODM load function. It describes the following:

- Verify operation of the load function
- Usage of data types
- Null values for load function data types
- RODM tables:
  - Control table—EKGCTABL
  - Method name table
  - Parameter mapping table
- Required and optional data definition names
- z/OS linkage conventions for the load function
- Syntax for RODM load function:
  - Parameters used to run the load function
  - High-level statements
  - Primitives
  - Common syntactic elements

Understanding the Verify Operation

The verify operation parses the RODM load function input files and compares the statements with the contents of the data cache. No changes are made to the data cache. The verify operation parses both high-level load function statements and load function primitive statements. The load function primitive statements are easier to understand, so they are described first.

Each load function primitive statement description in "Syntax and Processing Logic for Load Function Primitives" on page 282 includes an explanation of the verify operation logic for that statement. The verify operation logic describes how the load function compares the statement to the contents of the data cache. If the comparison is true, the load function issues a return code of zero. If the comparison is not true, the load function returns an error message.

For example, if you want to ensure that one class in the data cache is the parent of another class, you can use the verify operation with the HAS_PARENT load function primitive statement. The verify operation logic for the HAS_PARENT load function primitive statement directs the load function to check if the specified child class and parent class exist in the data cache. The load function then checks if the MyPrimaryParentID field of the child class points to the parent class. RODM must be running when you use the verify operation of the load function.

The RODM load function processes high-level load function statements by first converting them to load function primitive statements. The load function primitive statements are then processed as in the previous example.

For example, the following high-level load function statement can be processed by the load function.

```
ClassA       MANAGED OBJECT CLASS;
PARENT IS    UniversalClass;
ATTRLIST
  Field_1    CHARVAR  INIT('abc'),
  Field_2    CHARVAR  PRIVATE INIT('gsb'),
  Field_3    CHARVAR;
END;
```

When you run the verify operation, the load function converts the statement to load function primitive statements. The first two lines of the statement are converted to the following:

```
OP ClassA HAS_PARENT UniversalClass;
```

This load function primitive statement is processed as in the first example.

Each line of the field definition list is converted to one statement to create the field and a second statement to assign the initial value if one is supplied. The first field definition in this example is converted to the following:

```
OP ClassA HAS_FIELD (CHARVAR) Field_1;
OP ClassA..Field_1 HAS_VALUE (CHARVAR) 'abc';
```

Each of the load function primitive statements is then processed as described in "Syntax and Processing Logic for Load Function Primitives" on page 282.

When you use the verify operation with load function statements that specify values for fields, be careful because values often change. Only test for a specific value when you are interested in that value. In the high-level load function statement example, the initial value of Field_1 caused the load function to generate a statement to test Field_1 for the value `abc`. Remove the initial values from field definitions before using the verify operation if all you need to test for is the structure of the data cache.

Using CLASSID and OBJECTID Data Types

The RODM load function enables you to specify the CLASSID and OBJECTID data types for fields. However, the corresponding ClassID and ObjectID abstract data types in RODM are reserved; you cannot create fields with these data types, except within a SELFDEFINING variable.

CLASSID

If you create a field of type CLASSID using the RODM load function, the field is created in the RODM data cache with the Integer abstract data type. The RODM load function gets the class ID for the class name you specify and puts the class ID value in the target field in the RODM data cache which must be of type Integer.

When you assign a value of type CLASSID using the RODM load function, you supply a class name, but be sure the class name specified already exists. If you create a field of type CLASSID using the RODM load function, but do not assign an initial value, the field is created with a null value.

OBJECTID

If you create a field of type OBJECTID using the RODM load function, the field is created in the RODM data cache with the AnonymousVar abstract data type. The RODM load function gets the object ID for the object name you specify and puts the object ID value in the target field in the RODM data cache which must be of type AnonymousVar.

When you assign a value of type OBJECTID using the RODM load function, you supply a class name and an object name, but be sure the object name and class name you specify already exist. If you create a field of type OBJECTID using the RODM load function, but do not assign an initial value, the field is created with a null value.

Null Values for RODM Load Function Data Types

You can specify null values for some of the data types used in RODM load function primitives and RODM high-level load function statements. This enables you to set the value of a field to its null value as defined by RODM. The following list shows how to specify each null value:

```
(ANONYMOUSVAR) X''
(BERVAR) X''
(APPLICATIONID) ''
(CHARVAR) ''
(CHARVARADDR) X'00000000'
(ECBADDRESS) X'00000000'
(GRAPHICVAR) ''
(INDEXLIST) ()
(METHODNAME) 'NullMeth'
(METHODPARAMETERLIST) ()
(OBJECTNAME) ''
(SELFDEFINING) ()
(SHORTNAME) ''
(SUBSCRIBEID) ''
```

Control Table—EKGCTABL

You can modify the member names contained in this required control table called EKGCTABL. This table is a member of the partitioned data set identified by the EKGLUTB DD statement which is a required DD statement. RODM expects the member name to remain EKGCTABL and to be contained in the data set identified by the EKGLUTB DD statement.

The EKGCTABL control table contains two entries:

PARAMETER_MAPPING_MEMBER
Specifies the name of the member of the partitioned data set identified by the EKGLUTB DD statement that contains the parameter mapping table.

INSTALL_METHOD_MEMBER
Specifies the name of the member of the partitioned data set identified by the EKGIN2 DD statement that contains the method name table.

Figure 56 shows an example control table. The column scale is inserted for explanation purposes and is not part of the control table.

```
         1         2         3          4        5
1...+....0....+....0....+....0....+.....1...+....0...

PARAMETER_MAPPING_MEMBER:               EKGPTENU
INSTALL_METHOD_MEMBER:                  EKGINMTB
```

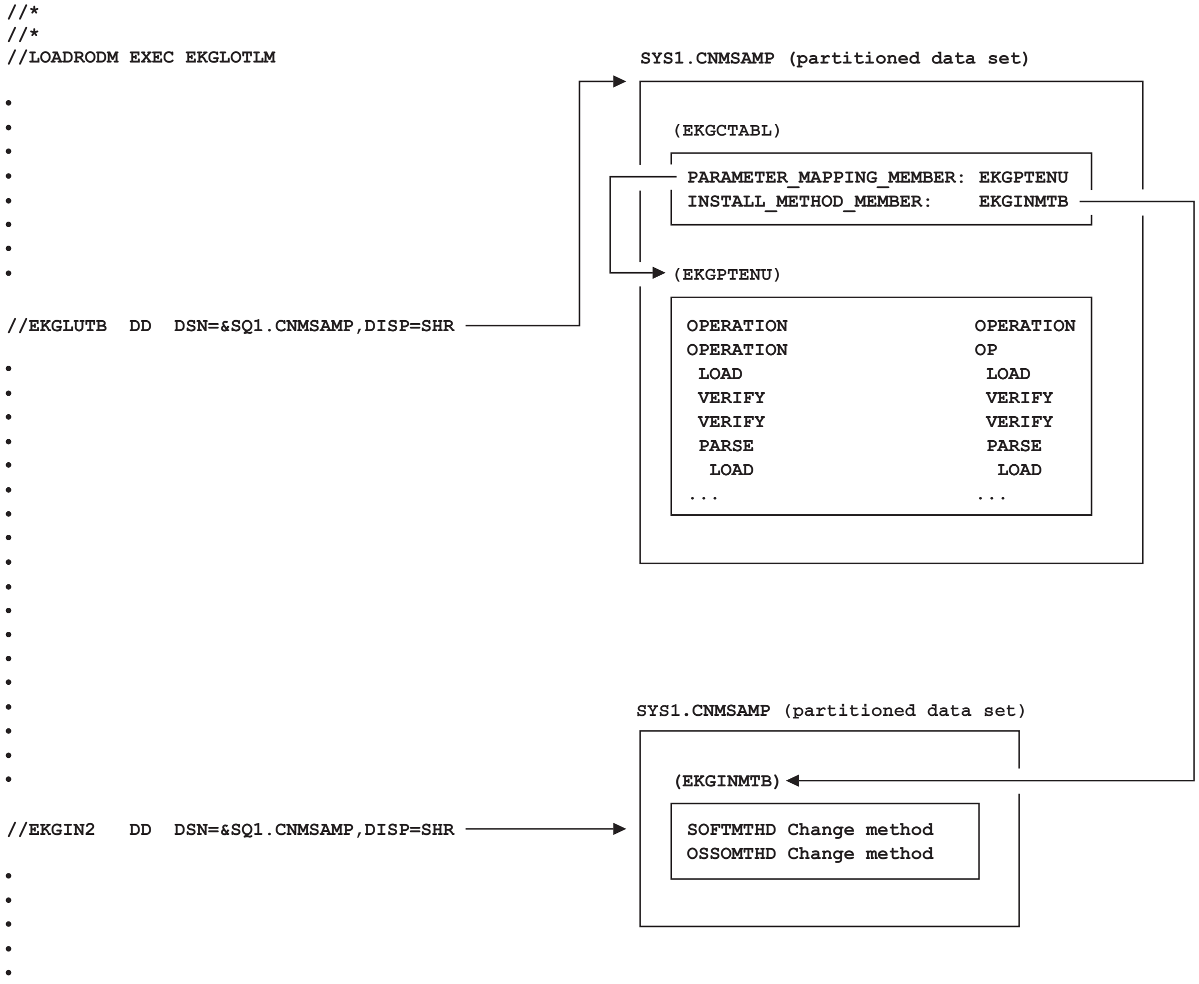
Figure 56. Sample Control Table EKGCTABL with Column Scale

The required symbols PARAMETER_MAPPING_MEMBER and INSTALL_METHOD_MEMBER must start in column 1. The member names, EKGPTENU and EKGINMTB in this example, must start in column 41.

Relationships to Other Tables and DD Names

Figure 57 on page 261 shows the relationship between the control table EKGCTABL, the parameter mapping table EKGPTENU, the method name table EKGINMTB, and the DD names EKGLUTB and EKGIN2.

In the figure, the job stream to verify the structure of a RODM named RODMNAME has DD statements EKGLUTB and EKGIN2. The DD statement labeled EKGLUTB identifies the partitioned data set NETVIEW.V6R1M0.CNMSAMP containing the members EKGCTABL and EKGPTENU. The DD statement labeled EKGIN2 identifies the partitioned data set NETVIEW.V6R1M0.CNMSAMP containing the member EKGINMTB. RODM uses the control table EKGCTABL to obtain the member names of the parameter mapping table and method name table.



```
//*
//*
//LOADRODM EXEC EKGLOTLM
•
•
•
•
•
•
•
•

//EKGLUTB  DD  DSN=&SQ1.CNMSAMP,DISP=SHR
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•

//EKGIN2   DD  DSN=&SQ1.CNMSAMP,DISP=SHR
•
•
•
•
•
```

```
PARAMETER_MAPPING_MEMBER: EKGPTENU
INSTALL_METHOD_MEMBER:    EKGINMTB
```

```
OPERATION                 OPERATION
OPERATION                 OP
 LOAD                      LOAD
 VERIFY                    VERIFY
 VERIFY                    VERIFY
 PARSE                     PARSE
  LOAD                      LOAD
...                       ...
```

```
SOFTMTHD Change method
OSSOMTHD Change method
```

Figure 57. Relationship between EKGCTABL, EKGINMTB, EKGPTENU and JCL

Method Name Table

The method name table contains the names of the methods you want installed by the RODM load function. A sample file named EKGINMTB that contains only one entry (EKGNOTF) is shipped in the samples library NETVIEW.V6R1M0.CNMSAMP. You can either copy that file and make modifications or create your own.

You do not have to use the name of EKGINMTB for your method name table, but if you use a different name you must modify the control table EKGCTABL because in the control table that is supplied by IBM the member name specified for the method name table is EKGINMTB. For more information about control table EKGCTABL, see “Control Table—EKGCTABL” on page 260.

Figure 58 shows a method name table (EKGINMTB) that declares two user-written methods and seven methods that are supplied with the NetView program. The column scale is inserted for explanation purposes and is not part of the method name table.

```
         1        2         3         4         5
1...+..8..1...+....0....+....0....+....0....+....0...

EKGNOTF   NOTIFICATION
EKGNLST   Notify
EKGNEQL   Notify
EKGNTHD   Notify
EKGCTIM   Change method to trigger an OI method
EKGMIMV   Named method to increment a value
EKGSPPI   Object-Independent method
SOFTMTHD  Change Method - (user written)
OSSOMTHD  Change Method - (user written)
```

Figure 58. Method Name Table Format with Column Scale

Each entry in a method name table consists of one row. Columns 1–8 contain the name of the method, and columns 11–80 can optionally contain a comment, such as the type of method.

To bypass the RODM method name table load, replace EKGINMTB with *NONE in control table EKGCTABL as shown in Figure 59. The column scale is inserted for explanation purposes and is not part of the method name table.

```
        1         2         3          4        5
1...+....0....+....0....+....0....+.....1...+....0...

INSTALL_METHOD_MEMBER:                 *NONE
```

Figure 59. Sample Control Table EKGCTABL with Column Scale

Associated DD Statements and Control Table

The DD statement that declares the partitioned data set containing the method name table as one of its members is labeled EKGIN2. The member name for the method name table is in control table EKGCTABL which is in the partitioned data set identified by the DD statement labeled EKGLUTB. See Figure 57 on page 261 for a pictorial of this relationship.

Parameter Mapping Table

When you run the RODM load function, you must supply parameters, such as NAME, OPERATION, CODEPAGE, and LOAD. According to JCL conventions, these parameters go in parentheses on the PARM= part of the EXEC statement. They take the form:

PARM=('*keyword1=keyword_value1,keyword2=keyword_value2***,...')**

The parameter mapping table is a fixed-block table with an LRECL of 80. The table enables string substitutions to be used for the syntax known by the RODM load function (internal syntax). These string substitutions can be abbreviations, a mapping to a national language, or both. This enables the RODM load function to use other syntax formats.

The parameter mapping table (EKGPTENU) is a member of the partitioned data set identified by the EKGLUTB DD statement. The EKGCTABL control block contains the member name of the parameter mapping table. See Figure 57 on page 261 for a pictorial of this relationship.

Table EKGPTENU has a one-to-one relationship between the internal syntax in columns 1–30 and the substitution string in columns 31–80. See “RODM Load Function Parameter Syntax” on page 269 for information about the load function parameter data (internal syntax) in columns 1–30.

The syntax rules are:

- Internal *keyword entries* must start in column 1 and each related substitution string entry must start in column 31.
- Internal *keyword values* must start in column 2 and each related substitution string value must start in column 32.
- The internal *keyword default* value must start in column 3 and the substitution string default value must start in column 33.
- For each keyword, the keyword entry is followed by the value entries for that keyword, which are in turn followed by the default value entry for that keyword.

Figure 60 documents the format of this table and shows examples of abbreviation substitution strings. The column scale is inserted for explanation purposes and is not part of the parameter mapping table.

```
         1         2         3          4        5
1...+....0....+....0....+....0....+.....1...+....0...

OPERATION                     OPERATION
OPERATION                     OP
 LOAD                          LOAD
 VERIFY                        VERIFY
 VERIFY                        VER
 PARSE                         PARSE
 PARSE                         PARS
  LOAD                          LOAD
NAME                          NAME
SEVERITY                      SEVERITY
SEVERITY                      SEV
 WARNING                       WARNING
 WARNING                       WARN
 ERROR                         ERROR
 ERROR                         ERR
  WARNING                       WARNING
LISTLEVEL                     LISTLEVEL
LISTLEVEL                     LISTLVL
 ERRORSYNTAX                   ERRORSYNTAX
 ERRORSYNTAX                   ERRORSNTX
 ALLSYNTAX                     ALLSYNTAX
 ALLSYNTAX                     ALLSNTX
  ERRORSYNTAX                   ERRORSYNTAX
CODEPAGE                      CODEPAGE
CODEPAGE                      CODEP
 EKGCP500                      EKGCP500
  EKGCP500                      EKGCP500
LOAD                          LOAD
 STRUCTURE                     STRUCTURE
 STRUCTURE                     STR
 INSTANCE                      INSTANCE
 INSTANCE                      INS
  INSTANCE                      INSTANCE
ROUTECODE                     ROUTECODE
```

Figure 60. Sample Parameter Table EKGPTENU with Column Scale

You can modify an existing mapping table or create a new table. A sample load function parameter mapping table can be found in member EKGPTENU of data set CNMSAMP in the samples library supplied with RODM. Copy the sample and make any updates to the copy. If you change the name of the parameter table, be sure to update the EKGCTABL control table.

RODM Data Definition (DD) Statements

The DD statements that are used to run the load function declare the data sets. Ensure that the data sets appropriate to the type of load you are running are present. Ensure that the contents of the data sets are valid.

You can change DD names to match your needs by using the DD list structure, which you can pass to RODM using a parameter list when the load function is run. The DD list structure is described in “z/OS Linkage Conventions” on page 266.

STEPLIB (Required If You Do Not Use LNKLIST)

The data set identified as STEPLIB must be a partitioned data set that contains the RODM load function code. STEPLIB is a required DD statement when the RODM load function code is not in the z/OS LNKLIST. Another DD statement must be concatenated to the STEPLIB DD statement that identifies the Language Environment® runtime library. The format of STEPLIB is the standard DCB (data control block) format for any link-edited data set.

EKGLANG (Required)
The EKGLANG DD statement identifies the partitioned data set that contains the message file for the RODM load function.

EKGLUTB (Required)
The EKGLUTB data definition identifies the partitioned data set that contains the EKGCTABL control table file as one of its members. This required control table contains the member name of the parameter mapping table and the member name of the method name table. For more information about modifying the EKGCTABL control table and its relationship with the parameter mapping table and the method name table, see "Control Table—EKGCTABL" on page 260.

The data control block for the DD statement labeled EKGLUTB specifies `LRECL=80` and `RECFM=FB` for the data set. The block size must be a multiple of 80.

EKGPRINT (Required)
The EKGPRINT data definition identifies the data set containing the RODM load function output listing. This listing contains the load function input, echoed syntax, a report of primitive success or failure, messages and codes, and other information.

You can direct the print to SYSOUT, to a sequential file, or to a member of a partitioned data set. The data set or file must specify `LRECL=80` and `RECFM=FB`. The block size must be a multiple of 80.

EKGIN1 (Required for Class Structure Definition)
EKGIN1 identifies the sequential data set or concatenation of sequential data sets that contain the class structure definitions. The data sets that define the class structure must be sequential data sets with a data control block that specifies `LRECL=80` and `RECFM=FB`. The block size must be a multiple of 80. The class structure definitions which represent the GMFHS data model are contained in member DUIFSTRC of the CNMSAMP data set in the samples library.

EKGIN2 (Required for Class Structure Definition)
EKGIN2 identifies the partitioned data set that contains the method name table file as one of its members. EKGIN2 must be a partitioned data set with a data control block that specifies `LRECL=80` and `RECFM=FB`. The block size must be a multiple of 80. The method name table that is supplied by IBM which has one entry of EKGNOTF (notify method) is contained in member EKGINMTB of the CNMSAMP data set in the samples library.

EKGIN3 (Required for Object Definition)
EKGIN3 identifies the sequential data set or concatenation of sequential data sets that contain the object definitions. You create these definitions to define your network. The data control block of each of the data sets concatenated as EKGIN3 must specify `LRECL=80` and `RECFM=FB`. The block size must be a multiple of 80. The object definitions which define the network described in Chapter 2, "Defining Your Network to GMFHS," on page 17 are contained in member DUIFSNET of the CNMSAMP data set in the samples library as an example.

Data Definitions Necessary for Initialization

If you are running an initialization method, either during a cold start or a warm start of RODM, you need data sets for the following data definition names:

EKGIN1
EKGIN2
EKGIN3
EKGLANG
EKGPRINT
EKGLUTB

Data Definitions Necessary for Structure Load Only

When running the RODM load function either through job posting or through a module call to load only the class structure and install methods, you need data sets for the following data definition names:

EKGIN1
EKGIN2
EKGLANG
EKGPRINT
EKGLUTB

Data Definitions Necessary for Object Load Only

When running the RODM load function either through job posting or through a module call to load only the object definitions, you need data sets for the following data definition names:

EKGIN3
EKGLANG
EKGPRINT
EKGLUTB

z/OS Linkage Conventions

Figure 61 on page 267 shows the z/OS linkage requirements for running the RODM load function by means of a module call to EKGLJOB.

Register 1 points to the parameter list, which contains up to three parameter addresses. The first parameter address points to a *parameter structure* that you use to specify the RODM load function parameters. The second parameter address is optional unless the third parameter address is supplied. If it is supplied, it points to a *DD list structure* that you use to change the default RODM load function DD names. The third parameter address is optional. If it is supplied, it points to the *access block* that was used to connect to RODM. The last address in this parameter list must have the high-order bit set ON.

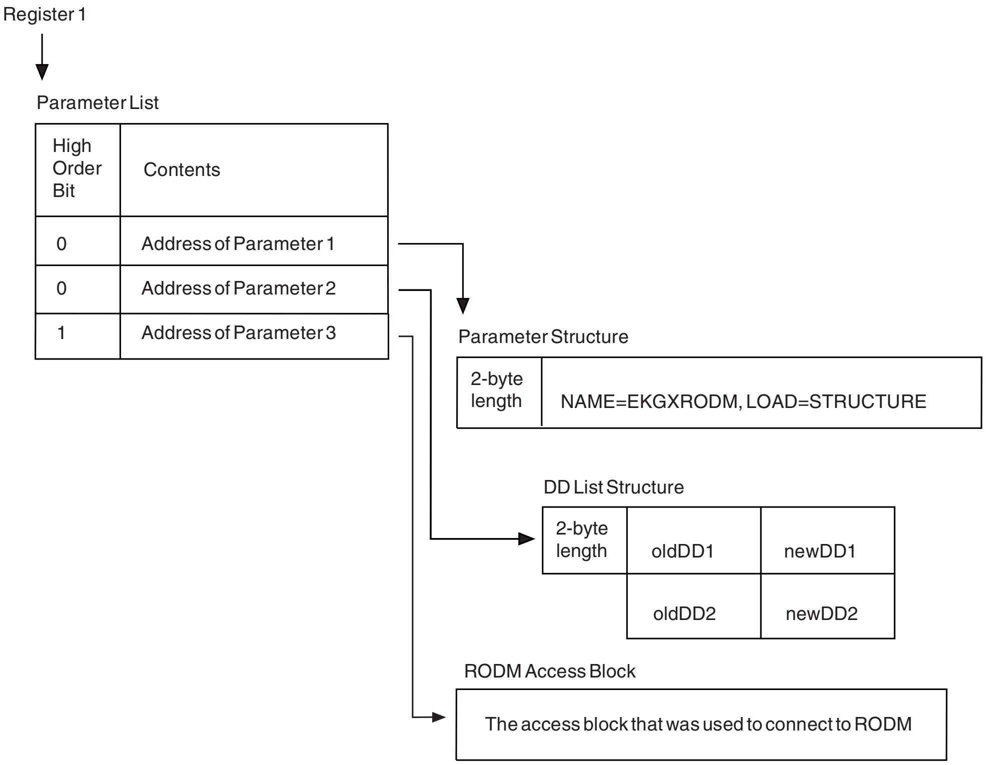


Figure 61. z/OS Linkage Conventions Required for Module Call to EKGLJOB

Parameter Structure

The parameters passed to the load function are the same as the ones specified in the JCL except that you must provide the length of the parameter. The only required parameter is NAME; all of the parameters that are not specified, default to the values specified in the parameter mapping table.

The NAME parameter is ignored if the access block is specified.

The parameter structure consists of a 2-byte fixed field followed by a character field. The fixed field must contain the length of the following character field. The restrictions on JCL when running the load function require that the character field to be no more than 100 bytes in length. The character field can contain any valid combination of input parameter values.

The following is an example of the parameter structure in hextype format (hexadecimal representation in the first line, EBCDIC in the second):

```
001CD5C1D4C57EC5D2C7E7D9D6C4D46BD3D6C1C47EE2E3D9E4C3E3E4D9C5

     N A M E = E K G X R O D M , L O A D = S T R U C T U R E
```

This parameter specifies that the character field has a length of X'1C' bytes. The character field contains the required NAME parameter and the LOAD=STRUCTURE parameter. The remaining load function parameters will default to the default values specified in the parameter mapping table.

DD List Structure

The DD list structure, if specified, consists of a two-byte fixed field followed by a character field with no maximum length restriction, although the length of the character field must be a multiple of 16. The DD list structure is used to specify DD names only, not data set names or member names.

The character field consists of an array of DD name pairs in which each element is 16 (X'10') bytes in length. The first eight bytes is the default or old DD name used in the RODM load function, and the second eight bytes is the new DD name to be used in the RODM load function. This array of DD name pairs can be in any order. If no new DD names are provided, the default required DD names specified in “RODM Data Definition (DD) Statements” on page 264 are used.

The following is an example of the DD list structure in hextype format (hexadecimal representation in the first line, EBCDIC in the second):

```
0020C5D2C7C9D5F14040E2E3D9E4C3E34040C5D2C7C9D5F34040D6C2D1C5C3E34040

    E K G I N 1     S T R U C T     E K G I N 3     O B J E C T
```

This parameter specifies that there are two DD name pairs and that the RODM load function is to use the new DD name STRUCT instead of EKGIN1 and the new DD name OBJECT instead of EKGIN3.

Access Block

The access block, if specified, is the access block that the user application used when it connected to RODM. This allows a user application that is already connected to RODM to use the RODM load function without first disconnecting from RODM.

If the access block parameter is specified, the DD list structure must also be specified. However, if you do not want to change the DD names, you can specify a null string.

Calling the RODM Load Function

When you call the RODM load function, follow the linkage convention shown in Figure 61 on page 267. The RODM load function linking convention follows a standard z/OS approach. Use the ASM and INTER options when you define the linkage of your modules to the RODM load function. Refer to Figure 62 and locate the statement:

```
DCL EKGLJOB       OPTIONS(ASM INTER) ENTRY EXTERNAL;
```

Figure 62 is an example of how to call the RODM load function from a PL/I program.

```
/*******************************************************************/
/* Local Variables                                                 */
/*******************************************************************/
%DECLARE PL1_OR_C FIXED;            /* Flag indicates whether this */
%PL1_OR_C = 1;                      /*  module is IBM PL/1 or C*/
DCL MODULETYPE FIXED INIT(1);       /* Input parm                  */

/*******************************************************************/
/* Declare the parms to pass to RODM LOAD function                 */
/*******************************************************************/
                                     /* Keyword parms for load     */
DCL PARM_STRING    CHAR(100) VARYING ALIGNED;
                                     /* Load DD name mapping       */
DCL DD_STRING      CHAR(160) VARYING ALIGNED;
```

Figure 62. Calling the RODM Load Function from a PL/I Program (Part 1 of 4)

```
/*******************************************************************/
/* Declare the external entry                                      */
/*******************************************************************/
                                      /* This entry is used when    */
                                      /*  calling C or IBM PLI  */
                                      /* modules                    */
DCL EKGLJOB        OPTIONS(ASM INTER) ENTRY EXTERNAL;
                                      /* This entry is used         */
                                      /*  otherwise                 */
DCL EKGLTOLM       OPTIONS(ASM INTER) ENTRY EXTERNAL;
```

Figure 62. Calling the RODM Load Function from a PL/I Program (Part 2 of 4)

```
/*******************************************************************/
/* Assign the value for the parms                                  */
/*******************************************************************/
                                      /* Load function input parms  */
PARM_STRING = 'OPERATION=LOAD,LOAD=INSTANCE,NAME=EKGXRODM';
                                      /* DD name mapping            */
                                      /* Must be multiple of 16     */
                                      /* First 8 bytes specific RODM*/
                                      /*  DD name, and the second 8 */
                                      /*  bytes specifics the DD    */
                                      /*  name user want to use     */
                                      /*  instead.                  */
                                      /* Use OBJECT1 DD name instead*/
                                      /*  EKGIN3 DD name            */
DD_STRING = 'EKGIN3    OBJECT1 ';
                                      /* Use SYSPRINT DD for load   */
                                      /*  messages.                 */
DD_STRING = DD_STRING || 'EKGPRINTSYSPRINT';
```

Figure 62. Calling the RODM Load Function from a PL/I Program (Part 3 of 4)

```
/*******************************************************************/
/* Call load function.                                             */
/*******************************************************************/
IF MODULE_TYPE = PL1_OR_C THEN        /* If it is IBM PL/1 or C */
DO;                                   /* Check DD name mapping      */
  IF LENGTH(DD_STRING) > 0 THEN       /* If yes, pass both parms    */
                                      /* If yes, pass both parms    */
      CALL EKGLJOB(PARM_STRING,DD_STRING);
  ELSE                                /* If no,pass only PARM_STRING*/
      CALL EKGLJOB(PARM_STRING);
END;                                  /* End check DD name mapping  */
ELSE                                  /* Use EKGLOTLM entry point   */
DO;                                   /* Check DD name mapping      */
  IF LENGTH(DD_STRING) > 0 THEN       /* If yes, pass both parms    */
                                      /* If yes, pass both parms    */
      CALL EKGLOTLM(PARM_STRING,DD_STRING);
  ELSE                                /* If no,pass only PARM_STRING*/
      CALL EKGLOTLM(PARM_STRING);
END;                                  /* End check DD name mapping  */
```

Figure 62. Calling the RODM Load Function from a PL/I Program (Part 4 of 4)

RODM Load Function Parameter Syntax

The following are descriptions and syntax for RODM load function parameters in alphabetical order.

The syntax is shown in syntax diagrams.

CODEPAGE

Description: The code page for input scanning.

Syntax:

CODEPAGE

```
   ┌─CODEPAGE=EKGCP500─┐
►►─┴───────────────────┴──────────────────────────────────────────►◄
```

Usage Notes: To indicate code page 500 (U.S. English) for input scanning, you code: `CODEPAGE=EKGCP500`

Note: RODM load function supports only code page 500.

LISTLEVEL

Description: The level of the listing to generate. You can list only the syntax that is in error or list all syntax used as input to the RODM load function.

Syntax:

LISTLEVEL

```
   ┌─LISTLEVEL=ERRORSYNTAX──────┐
►►─┼────────────────────────────┼─────────────────────────────────►◄
   └─LISTLEVEL=──┬─ERRORSYNTAX─┬┘
                 └─ALLSYNTAX───┘
```

Usage Notes: When you specify:

`LISTLEVEL=ALLSYNTAX`
All syntax, including generated primitive statements, is listed with messages indicating the success or failure of the high-level statements and primitives that were performed interleaved where appropriate.

`LISTLEVEL=ERRORSYNTAX`
Only the statements in error, *excluding primitive statements generated from high-level statements*, are listed with their error messages. Error messages for generated primitive statements appear after their associated high-level statement. *The generated primitive statement that caused the error is not listed*.

LOAD

Description: The type of load. A structure load or an object load.

Syntax:

LOAD

```
   ┌─LOAD=INSTANCE──────────┐
►►─┼────────────────────────┼─────────────────────────────────────►◄
   └─LOAD=──┬─INSTANCE──┬───┘
            └─STRUCTURE─┘
```

Usage Notes: When you specify:

LOAD=STRUCTURE
Only the input statements from the data sets identified by the EKGIN1 and EKGIN2 data definition statements are used. Used for structure load.

LOAD=INSTANCE
Only the input statements from the data sets identified by the EKGIN3 data definition statement are used. Used for object load.

You can also use the LOAD=STRUCTURE specification to load object definitions as well as class structure definitions. Concatenate the data sets that contain the object definitions, normally identified by the EKGIN3 DD statement, to the EKGIN1 DD statement.

You can also include class structure definition with object definitions when specifying LOAD=INSTANCE. Using concatenation of data sets, arrange the JCL statements for the EKGIN3 DD so that the class structure definitions, usually identified by the EKGIN1 DD, are processed first with the object definitions following.

NAME

Description: The name of the RODM on which the load is to be performed. This is a required parameter for structure loads and object loads.

Syntax:

NAME

```
►►──NAME=rodm_name──────────────────────────────────────────────────►◄
```

Usage Notes: To specify a RODM name of `MYRODM` code: `NAME=MYRODM`

The NAME parameter is required for load and verify operations. If you specify NAME for a parse operation, the RODM load function connects to the named RODM, but this is not required.

The NAME parameter is not required for an initialization method load. Because a particular RODM has run the RODM load function, the RODM name is known by the load function.

OPERATION

Description: The operation the RODM load function is to perform. The operation parameter can specify that the RODM load function parse the load function input statement syntax for validity, load the RODM data cache, or verify that defined contents exist prior to performing another operation.

Syntax:

OPERATION

```
    ┌─OPERATION=LOAD──────────┐
►►──┤                         ├──────────────────────────────────────────────────►◄
    └─OPERATION=──┬─LOAD────┬─┘
                  ├─PARSE───┤
                  └─VERIFY──┘
```

Usage Notes: You code:

`OPERATION=PARSE`
To parse the syntax of the data sets that contain your RODM load function input parameters. RODM does not need to be running for OPERATION=PARSE. With OPERATION=PARSE, the RODM load function reads the load function input files and parses them to find syntax errors. The RODM load function issues the connect function to RODM and queries the RODM version and Release. Any errors found in the connect and query function are logged in the Job log and RODM log. However, these errors are not considered as errors of the RODM load Parse operation.

`OPERATION=LOAD`
To parse the input statements and then load the data cache.

`OPERATION=VERIFY`
To parse and verify the contents of the RODM data cache.

Neither PARSE nor VERIFY performs the LOAD operation.

If you want to assign values to objects and wish to see which of the objects actually exist instead of having them fail, use the VERIFY operation. For more information about VERIFY see "Understanding the Verify Operation" on page 258.

If `LOAD=STRUCTURE`, the input statements from the data sets identified by the DD labeled EKGIN1 is parsed, but the data identified by the DD labeled EKGIN2 is not. If `LOAD=INSTANCE`, only the input statements from the data sets identified by the DD labeled EKGIN3 are parsed. This occurs for LOAD, PARSE, or VERIFY operations.

ROUTECODE

Description: Defines the route code to be used when the loader issues messages to a console by way of the WTO or WTOR macros. Valid values are in the range 1 – 128. The default value is 1.

Messages that can be issued before this parameter is processed will use the default route code 1, regardless of the value set here.

Syntax:

ROUTECODE

```
    ┌─ROUTECODE=1───┐
►►──┤               ├────────────────────────────────────────────────────────────►◄
    └─ROUTECODE=nnn─┘
```

SEVERITY

Description: The way that the application is to treat an error (return code 8) in the processing of a class structure definition or an object definition: as an error (return code 8) or as a warning (return code 4).

For SEVERITY=ERROR, when the RODM load function encounters an error in a load function input statement, it ends processing at that statement and issues a return code of 8. For SEVERITY=WARNING, when the RODM load function encounters an error in a load function input statement, it continues processing and issues a return code of 4 upon completion.

Syntax:

SEVERITY

```
   ┌─SEVERITY=WARNING─────────┐
►►─┤                          ├──────────────────────────────────────────────────►◄
   └─SEVERITY=──┬─WARNING─┬───┘
                └─ERROR───┘
```

Usage Notes: If the application is to treat an error in the processing of a class structure definition or an object definition as an error, you code: `SEVERITY=ERROR`

If the application is to treat an error in the processing of a class structure definition or an object definition as a warning, you code: `SEVERITY=WARNING`

Use the WARNING option when you are parsing the syntax; use the ERROR option when you are loading.

Coding RODM High-Level Load Function Statements

This topic of the reference section describes how to code RODM high-level load function statements. It provides the syntax and associated rules for high-level load function statements.

The syntax is shown in syntax diagrams.

Syntax Rules for High-Level Load Function Statements

This topic addresses syntax rules that apply to RODM high-level load function statements.

Input Columns: The RODM load function reads all columns of an input record as data. Do not use columns 73 to 80 for sequence or line numbers. You can use sequence or line numbers if you mark them as comments using the comment (--) characters.

Delimiters: Table 24 describes valid syntax delimiters for RODM high-level load function statements.

Table 24. Syntax Delimiters for RODM High-Level Load Function Statements

Delimiter	Function
' '	Used to enclose a character string.
X'0E' (Shift-out)	Marks the start of a DBCS mixed string data type.
X'0F' (Shift-in)	Marks the end of a DBCS mixed string data type.

Table 24. Syntax Delimiters for RODM High-Level Load Function Statements (continued)

Delimiter	Function
-- (two hyphens)	Marks the beginning or end of a comment.

The RODM load function allows free-form syntax. Spaces can be used to improve the readability of your load function input data because the RODM load function allows one or more spaces between parts of a RODM high-level load function statement. For example, the following MANAGED OBJECT CLASS high-level load function statement is a valid use of spaces to improve readability:

```
Software                                   MANAGED OBJECT CLASS;
  PARENT IS UniversalClass;
  ATTRLIST;
END;
```

Quoted Strings: A quoted string must begin and end on the same line. To create a string longer than a single line, break it into separately quoted parts on multiple lines. Multiple parts are concatenated by the RODM load function. For example, the following two lines results in a single quoted string:

```
INIT(' This is the first line of two lines '
     ' that results in one quoted string ' );
```

A quotation mark contained within quotation marks is represented by two single quotation marks, for example:

```
INIT('This is '' a quote '' within a quote. ');
```

Quotation marks are used to enclose the entire string, including any keywords or separators as a portion of the string. For example:

```
INIT(' Create the "MANAGED OBJECT CLASS" now ');
```

Double-Byte Character Strings: All data values between a X'0E' shift-out character and a X'0F' shift-in character are treated by the RODM load function as double-byte character string (DBCS) data. This means that any hexadecimal codes that normally denote delimiters are treated as data within the double-byte character string. The valid double-byte characters are the same as those for the GraphicVar data type; see “GraphicVar” on page 229.

Field Definition Lists: When specifying a field definition list with the ATTRLIST or MODLIST keyword, separate each member of the list with a comma and end the list with a semicolon. Otherwise, the RODM load function treats each member of the list as a separate statement.

Enabled data types and data type values for high-level statements are all those enabled by RODM. For more information about these data types, see “Abstract Data Type Reference” on page 223. For a list of these data type values and a syntax diagram of the `typed_value` load function common syntactic element, see “typed_value” on page 299.

Comments: Comments are delimited by two hyphens (--) at the beginning and at the end. An example is:

```
   -- This is a comment --
```

If the end of comment delimiter is not specified, the end of the comment is assumed to be at the end of the input line. The RODM load function ignores all text between comment delimiters.

Syntax for High-Level Load Function Statements

This is a syntax reference for your use in coding the RODM high-level load function statements for the data model definition to be created in your RODM data cache. Each RODM high-level load function statement has a description containing the name, purpose, external syntax, parameter descriptions, and an example of use.

Note: RODM high-level load function statement syntax is case sensitive.

The examples of use for the RODM high-level load function statements in this section are subsets of the load function input statement stream as shown in Figure 63. These statements create and use the hierarchical pseudo-structure shown in Figure 64 on page 276. This structure and the associated fields are an example for explanation purposes only, they are not part of RODM.

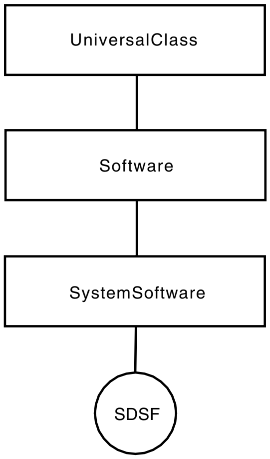


Figure 63. Hierarchical Pseudo-Structure for Examples

```
Software                                       MANAGED OBJECT CLASS;
  PARENT IS UniversalClass;
  ATTRLIST;
END;
SystemSoftware                                 MANAGED OBJECT CLASS;
  PARENT IS Software;
  ATTRLIST           -- Field List --
    ProductName                       CHARVAR,
    ProgramNumber                     CHARVAR    INIT('None'),
    LatestPTFNumber                   CHARVAR    INIT('UY12345'),
    CorrespondingAPARNumber           CHARVAR,
    DateApplied                       CHARVAR,
    Priority                          INTEGER    INIT(3),
    UseInHost                         OBJECTLINKLIST;
END;
CREATE    INVOKER    ::= 0000003;
          OBJCLASS   ::= SystemSoftware;
          OBJINST    ::= MyName = (CHARVAR) 'SDSF';
          ATTRLIST
                   ProductName             ::= (CHARVAR)  'SDSF',
                   ProgramNumber           ::= (CHARVAR)  '5697-B82',
                   LatestPTFNumber         ::= (CHARVAR)  'UY12903',
                   CorrespondingAPARNumber ::= (CHARVAR)  'PL45419',
                   DateApplied             ::= (CHARVAR)  '03/01/97',
                   UseInHost               ::= (OBJECTLINKLIST)
                    ('Host_Class'.'HostA'.'UseSystemSoftware')
                    ('Host_Class'.'HostC'.'UseSystemSoftware');
END;
SET       INVOKER    ::= 0000004;
          MODE       ::= non-confirmed;
          OBJCLASS   ::= SystemSoftware;
          OBJINST    ::= MyName = (CHARVAR) 'SDSF';
          MODLIST
                   ProductName             ::= (CHARVAR) 'SDSF V2', REPLACE,
                   ProgramNumber           ::= (CHARVAR) '5697-B82',
                   LatestPTFNumber         ::= (CHARVAR), SET TO DEFAULT,
                   CorrespondingAPARNumber ::= (CHARVAR) ' ',
                   DateApplied             ::= (CHARVAR) '03/01/97',
                   UseInHost               ::= (OBJECTLINKLIST)
                             ('Host_Class'.'HostA'.'UseSystemSoftware'),
                                                      REMOVE VALUE;
END;
DELETE    INVOKER    ::= 0000005;
          OBJCLASS   ::= SystemSoftware;
          OBJINST    ::= MyName = (CHARVAR) 'SDSF';
END;
```

Figure 64. High-Level Input Statements for Pseudo-Structure

MANAGED OBJECT CLASS:

Purpose: Use the MANAGED OBJECT CLASS high-level load function statement to define the hierarchy and create the data model class structure in the RODM data cache.

The following syntax declares class structure that the RODM load function adds to the RODM data cache. It does not contain keywords for resetting values, modifying, or deleting part or all of the class structure.

Syntax:

```
►►──class MANAGED OBJECT CLASS──;── PARENT IS parent_name──;─────────────────────────────────────►

        ┌─,─────────────────────────────────────────────────────┐
        ▼                                                       │
►── ATTRLIST──┬─────────────────────────────────────────────────┬─┴──;── END──;──────────────►◄
              │              ┌─PUBLIC─────────┐                 │
              └─ field type ─┼────────────────┼──┬────────────────────┬─┘
                             ├─PRIVATE────────┤  └─ INIT (init_value)─┘
                             └─PUBLIC_INDEXED─┘
```

Keyword and Parameter Descriptions:

class The name or label of the class that you are defining.

PARENT IS *parent_name*
The name of the parent class of the class being created.

field type
Creates a field with name *field* of data type *type* for the class being created. For a list of valid data types for this field, see “type” on page 298.

PUBLIC|PRIVATE|PUBLIC_INDEXED
Specifies if the field is a public, a public indexed, or a private field. Public fields are inherited by children of this class, private fields are not inherited. For more information about public indexed fields, see “Indexed Fields” on page 220.

INIT (*init_value***)**
An initial value setting for the field. INITIAL can be used instead of INIT.

Example: Consider the specification of a class named SystemSoftware that is a child of the class named Software and has the following fields:

ProductName
ProgramNumber
LatestPTFNumber
CorrespondingAPARNumber
DateApplied
Priority
UseInHost

Suppose that the initial value for the field named ProgramNumber is `None`, the initial value for the field named LatestPTFNumber is `UY12345`, and the initial value for the field named Priority is 3. The following MANAGED OBJECT CLASS statement defines the class named SystemSoftware:

```
SystemSoftware                      MANAGED OBJECT CLASS;
  PARENT IS Software;
  ATTRLIST          -- Field List --
    ProductName                     CHARVAR,
    ProgramNumber                   CHARVAR    INIT('None'),
    LatestPTFNumber                 CHARVAR    INIT('UY12345'),
    CorrespondingAPARNumber         CHARVAR,
    DateApplied                     CHARVAR,
    Priority                        INTEGER    INIT(3),
    UseInHost                       OBJECTLINKLIST;
END;
```

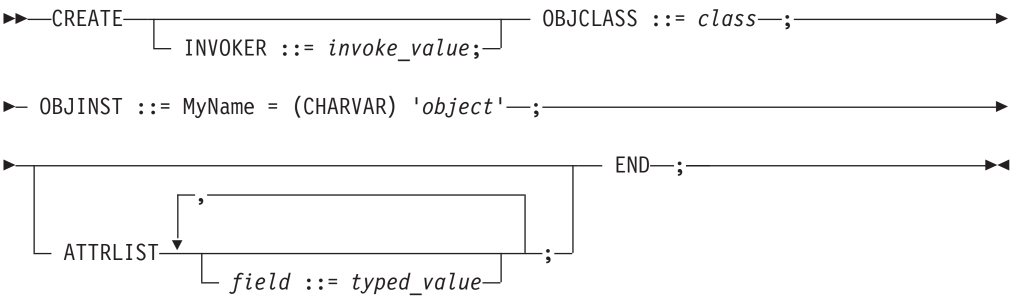
Usage Notes: Observe the following rules when you specify the *init_value* associated with the INIT or INITIAL keyword in a field definition list:

- Enclose all values in parentheses.
- Enclose character values in single quotation marks within the parentheses.
- Do not add additional parentheses to values for data types, such as METHODSPEC and SELFDEFINING, that are already bound by parentheses.
- Enclose non-null GRAPHICVAR values in shift-out and shift-in characters within the parentheses.
- Enclose a null GRAPHICVAR value in single quotation marks within the parentheses.

CREATE:

Purpose: Use the CREATE high-level load function statement to create an object of a specific class in the RODM data cache.

Syntax:



Keyword and Parameter Descriptions:

INVOKER ::= *invoke_value*
The identifier value. The value is ignored by the RODM load function, but can be used to number high-level load function statements in your definition files.

OBJCLASS ::= *class*
The name of the parent class of the object being created.

OBJINST ::= MyName = (CHARVAR) *object*
The name of the object being created.

field **::=** *typed_value*
Sets the field named *field* to the value *typed_value*. For a list of valid data types and values, see “typed_value” on page 299.

Example: Consider the specifications necessary for creating an object to represent system software called SDSF. SDSF is a child of the class named SystemSoftware and has the following fields and values:

- ProductName with a value of `SDSF`
- ProgramNumber with a value of `5697-B82`
- LatestPTFNumber with a value of `UY12903`
- CorrespondingAPARNumber with a value of `PL45419`
- DateApplied with a value of `03/01/97`
- UseInHost field that links this object to HostA and HostC

Note: HostA and HostC must already exist for the links to be successful.

The following is the statement needed to create the object SDSF:

```
CREATE    INVOKER    ::= 0000003;
          OBJCLASS   ::= SystemSoftware;
          OBJINST    ::= MyName = (CHARVAR) 'SDSF';
          ATTRLIST
                     ProductName             ::= (CHARVAR)  'SDSF',
                     ProgramNumber           ::= (CHARVAR)  '5697-B82',
                     LatestPTFNumber         ::= (CHARVAR)  'UY12903',
                     CorrespondingAPARNumber ::= (CHARVAR)  'PL45419',
                     DateApplied             ::= (CHARVAR)  '03/01/97',
                     UseInHost               ::= (OBJECTLINKLIST)
                      ('Host_Class'.'HostA'.'UseSystemSoftware')
                      ('Host_Class'.'HostC'.'UseSystemSoftware');
END;
```

Figure 65. Create Object Example

Usage Notes: When specifying the parameters of the OBJINST keyword of the CREATE high-level statement you normally specify `MyName` as the name of the field because the MyName field always represents the name of the object. For example:

```
OBJINST ::= MyName = (CHARVAR) 'SDSF';
```

But if you want another of the object's fields to also have the object name as its value, you specify that field name instead of `MyName` in the OBJINST definition. The MyName field and that field are then assigned the same value. For example, if you want the object name of `SDSF` assigned as the value of both the MyName and ProductName fields of the object, you specify:

```
OBJINST ::= ProductName = (CHARVAR) 'SDSF';
```

Do not repeat ProductName as a field in the ATTRLIST.

DELETE:

Purpose: Use the high-level load function DELETE statement to delete an object from the RODM data cache.

Syntax:

```
►►──DELETE──┬──────────────────────────┬──;── OBJCLASS ::= class──;──────────────►
            └─ INVOKER ::= invoke_value─┘

►── OBJINST ::= MyName=(CHARVAR) 'object'──;── END──;───────────────────────────►◄
```

Keyword and Parameter Descriptions:

INVOKER ::= *invoke_value*
The identifier value. The value is ignored by the RODM load function, but can be used to number high-level load function statements in your load function input files.

OBJCLASS ::= *class*
The name of the parent class of the object being deleted.

OBJINST ::= MyName = (CHARVAR) *object*
The name of the object being deleted.

Example: Figure 66 shows a DELETE statement that deletes an object from the data model.

```
DELETE    INVOKER    ::= 0000005;
          OBJCLASS   ::= SystemSoftware;
          OBJINST    ::= MyName = (CHARVAR) 'SDSF';
END;
```

Figure 66. Delete Object Example

The object to be deleted, *SDSF*, is specified as a parameter of the OBJINST keyword, and the parent class of the object, *SystemSoftware*, is specified as a parameter of the OBJCLASS keyword.

SET:

Purpose: Use the SET high-level load function statement to set the values of fields within an object in the RODM data cache.

Syntax:

```
►►──SET──┬──────────────────────────┬─;── MODE ::= mode_value──;─────────────►
         └─ INVOKER ::= invoke_value─┘


►─ OBJCLASS ::= class──;── OBJINST ::= MyName = (CHARVAR) 'object'──;──────────►


              ┌─,──────────────────────────────────┐
              ▼                                    │
►─ MODLIST──── field ::= typed_value─┬─────────────┬┴─;── END──;────────────►◄
                                     └─,──modifier─┘
```

Keyword and Parameter Descriptions:

INVOKER ::= *invoke_value*
The identifier value. The value is ignored by the RODM load function, but can be used to number high-level load function statements in your load function input files.

MODE ::= *mode_value*
This value is ignored by the RODM load function, and is assumed to always be `non-confirmed`.

OBJCLASS ::= *class*
The name of the parent class of the object for which field values are being set.

OBJINST ::= MyName = (CHARVAR) *object*
The name of the object for which field values are being set.

field **::=** *typed_value*
The field named *field* is set to the value *typed_value*. For a list of valid data types and values, see "typed_value" on page 299.

modifier
Use this parameter to specify the type of modification. The possible values of *modifier* are:

Value Description

ADD VALUE
Use only for data types of OBJECTLINK or OBJECTLINKLIST to create a new link.

REMOVE VALUE
Use only for data types of OBJECTLINK or OBJECTLINKLIST to delete an existing link.

REPLACE
Use for data types other than OBJECTLINK or OBJECTLINKLIST to change the value subfield of the specified field to a new value.

SET TO DEFAULT
Use for data types other than OBJECTLINK or OBJECTLINKLIST to change the value subfield of the specified field to the default value. The default value is the value of the field for the parent class.

If the data type is OBJECTLINK or OBJECTLINKLIST, the default is ADD VALUE. For all other data types, the default is REPLACE.

END The required keyword that identifies the end of the SET high-level load function statement.

Example: Consider a SET high-level load function statement where you want to change the values of the SDSF object, which is a child of the class named SystemSoftware. In particular, you want to make the following changes to the fields of SDSF:

- Change the ProductName field value to `SDSF V2`.
- Change the ProgramNumber field value to `5697-B82`.
- Change the LatestPTFNnumber field value to the default value.
- Reset the CorrespondingAPARNumber field value to a blank string.
- Change the DateApplied field value to `03/01/97`.
- Unlink the UseSystemSoftware field in the HostA object of Host_Class from the UseInHost field.

The statement to set the values of the fields of the SDSF object is shown in Figure 67.

```
SET       INVOKER    ::= 0000004;
          MODE       ::= non-confirmed;
          OBJCLASS   ::= SystemSoftware;
          OBJINST    ::= MyName = (CHARVAR) 'SDSF';
          MODLIST
                   ProductName             ::= (CHARVAR) 'SDSF V2', REPLACE,
                   ProgramNumber           ::= (CHARVAR) '5697-B82',
                   LatestPTFNumber         ::= (CHARVAR), SET TO DEFAULT,
                   CorrespondingAPARNumber ::= (CHARVAR) ' ',
                   DateApplied             ::= (CHARVAR) '03/01/97',
                   UseInHost               ::= (OBJECTLINKLIST)
                           ('Host_Class'.'HostA'.'UseSystemSoftware'),REMOVE VALUE;
END;
```

Figure 67. Set Value of Fields in an Object Example

Usage Notes: For definitions of OBJECTLINK and OBJECTLINKLIST fields, the RODM load function creates a link if the modification is ADD VALUE and deletes a link if the modification is REMOVE VALUE. Additionally, enclose in parentheses the value of any fields that specify a data type of either OBJECTLINK or OBJECTLINKLIST.

Coding RODM Load Function Primitive Statements

This topic of the reference section describes how to code RODM load function primitive statements. It provides the syntax and processing logic along with the associated syntax rules. It also describes the use of the global character with RODM load function primitives.

The syntax is shown in syntax diagrams.

Global Character

You can use an asterisk (*) as a *global character* to replace one or more values in RODM primitive statements. Each global character is used to substitute for one name, class, object, field, or subfield within a RODM primitive statement. When the primitive statement is converted to a RODM function, each global character is replaced with a corresponding value from the previous primitive on which the name, class, object, field, or subfield was explicitly specified. However, the global character can not be used to specify a method name.

When more than one global character is used, it substitutes values from previous primitive statements using the same relative position. For example:

```
OP ClassA  HAS_PARENT    UniversalClass;
OP *       HAS_FIELD     (INTEGER) FieldA_Integer;
OP ClassB  HAS_PARENT    *;
OP *       HAS_FIELD     (CHARVAR) FieldB_CharVar;
```

The global character in the second primitive statement is substituted with *ClassA* from the first primitive. The global character in the third primitive statement is substituted with *UniversalClass* from the first primitive. The global character in the fourth primitive statement is substituted with *ClassB* from the third primitive. Finally, the two global characters in the fifth primitive statement are substituted with *ClassB* and *FieldB_CharVar*, respectively, from the third and fourth primitives.

The global character is intended as a shorthand way of specifying RODM load function primitive statements. The RODM processing logic is not changed by use of the global character. The global character does not imply grouping of primitive statements.

Syntax Rules for Load Function Primitives

Like RODM high-level load function statement syntax, one or more spaces can separate parts of a RODM load function primitive.

Note: RODM load function primitive syntax is case sensitive.

Syntax rules applying to input columns, quoted strings, double-byte character strings, and comments are the same for RODM load function primitive syntax as those specified for RODM high-level load function syntax. See “Syntax Rules for High-Level Load Function Statements” on page 273.

Syntax and Processing Logic for Load Function Primitives

This is a reference to the syntax and processing logic for the RODM load function primitives. The RODM load function primitives are in alphabetical order, and each RODM load function primitive has a description containing its name, meaning, external syntax, and the implementation logic.

FORCE_HAS_NO_INSTANCE:

Description: FORCE_HAS_NO_INSTANCE ensures that there is no object existing under the specified class with the specified name. If links to the object exist, they are unlinked, and then the object itself is deleted.

This statement might fail to delete an object after failed retries of deleting all the links in a class object or all the objects.

Syntax:

```
►►──OP ──class FORCE_HAS_NO_INSTANCE object──;─────────────────────────────────►◄
```

object of *class* is deleted if it exists.

Syntax Logic for PARSE, LOAD, and VERIFY: Carry out the following syntax checks:
1. Check that *class* is a valid RODM class name.
2. Check that *object* is a valid RODM object name.

LOAD Logic: Perform the following:
1. Delete *object* from *class*.
2. If the object cannot be deleted because of links:
   a. Query the structure of the class.
   b. Query all link fields.
   c. For each field with links, delete the links.
   d. Retry the delete object request.

VERIFY Logic: Check that *object* of *class* does not exist.

FORCE_NOT_A_CLASS:

Description: FORCE_NOT_A_CLASS ensures that there is no class existing with the specified name. If objects of the class exist, they are deleted, meaning that all links to the objects are dropped, that the objects themselves are deleted, and that the class itself is deleted.

Syntax:

```
►►──OP ──class FORCE_NOT_A_CLASS──;────────────────────────────────────────────►◄
```

class is deleted if it exists.

Syntax Logic for PARSE, LOAD, and VERIFY: Check that *class* is a valid RODM class name.

LOAD Logic: Perform the following:
1. Delete *class*.
2. If the class cannot be deleted because of children, delete the children and retry the delete request.
3. If the class cannot be deleted because of objects, delete the objects and retry the delete request.

VERIFY Logic: Check that *class* does not exist.

HAS_FIELD:

Description: HAS_FIELD ensures that a class *defines* a specified public field.

Syntax:

```
►►──OP ──class HAS_FIELD (type)field──;──────────────────────────────────────►◄
```

class locally defines a field named *field* of type *type*.

Syntax Logic for PARSE, LOAD, and VERIFY: Carry out the following syntax checks:
1. Check that *class* is a valid RODM class name.
2. Check that *field* is a valid RODM field name.
3. Check that *type* is a valid RODM load function data type.

LOAD Logic: Check that the *class* exists, and create *field* of *type* for *class*.

VERIFY Logic: Check that *class* exists, that it locally defines *field*, and that the type of this field matches *type*.

HAS_INDEXED_FIELD:

Description: HAS_INDEXED_FIELD ensures that a class defines a specified public indexed field.

Syntax:

```
►►──OP ──class HAS_INDEXED_FIELD (CHARVAR)field──;───────────────────────────►◄
```

class locally defines a field named *field* of type CHARVAR.

Syntax Logic for PARSE, LOAD, and VERIFY: Carry out the following syntax checks:
1. Check that *class* is a valid RODM class name.
2. Check that *field* is a valid RODM field name.
3. Check that CHARVAR is a valid RODM load function data type. Only CHARVAR fields can be public indexed.

LOAD Logic: Check that the *class* exists, and create *field* of CHARVAR for *class*.

VERIFY Logic: Check that *class* exists, that it locally defines *field*, and that the type of this field is CHARVAR.

HAS_INSTANCE:

Description: HAS_INSTANCE ensures that a specific object of the specified class exists.

Syntax:

```
►►──OP ──class HAS_INSTANCE object──;────────────────────────────────────────►◄
```

class has an object named *object*.

Syntax Logic for PARSE, LOAD, and VERIFY: Carry out the following syntax checks:
1. Check that *class* is a valid RODM class name.
2. Check that *object* is a valid RODM object name.

LOAD Logic: Check that the *class* exists, and create *object* of *class*.

VERIFY Logic: Check that *class* exists and that it has an object *object*.

HAS_NO_FIELD:

Description: HAS_NO_FIELD deletes the specified field from the specified class. Fields cannot be deleted from classes that have class or object children. Also, inherited fields cannot be deleted.

Syntax:

```
►►──OP ──class HAS_NO_FIELD field──;────────────────────────────────────────►◄
```

field is deleted from the definition of *class* if it exists and the class has no object children.

Syntax Logic for PARSE, LOAD, and VERIFY: Carry out the following syntax checks:
1. Check that *class* is a valid RODM class name.
2. Check that *field* is a valid RODM field name.

LOAD Logic: Delete *field* from *class*.

VERIFY Logic: Check that *field* is not defined by *class*.

HAS_NO_INSTANCE:

Description: HAS_NO_INSTANCE ensures that a specific object of a specific class does not exist. The only imperative used to implement this specification is a simple delete.

If the object is linked to other objects, it cannot be deleted by this primitive alone; in that case, see "FORCE_HAS_NO_INSTANCE" on page 283.

Syntax:

```
►►──OP ──class HAS_NO_INSTANCE object──;────────────────────────────────────►◄
```

object of *class* is deleted if it exists and has no links to other objects.

Syntax Logic for PARSE, LOAD, and VERIFY: Carry out the following syntax checks:
1. Check that *class* is a valid RODM class name.
2. Check that *object* is a valid RODM object name.

LOAD Logic: Delete *object* from *class*.

VERIFY Logic: Check that *object* does not exist in *class*.

HAS_NO_SUBFIELD:

Description: HAS_NO_SUBFIELD ensures that a specific subfield does not exist for the specified field. Subfields cannot be deleted from classes that have objects. Also, subfields on inherited fields cannot be deleted.

Syntax:

```
►►──OP ──class.field HAS_NO_SUBFIELD subfield──;──────────────────────────────►◄
```

subfield is deleted from *field* of *class* if it exists and the class has no object children.

Syntax Logic for PARSE, LOAD, and VERIFY: Carry out the following syntax checks:
1. Check that *class* is a valid RODM class name.
2. Check that *field* is a valid RODM field name.
3. Check that *subfield* is a valid RODM subfield name.

LOAD Logic: Delete *subfield* from *field* of *class*.

VERIFY Logic: Check that *subfield* is not defined for *field* of *class*.

HAS_PARENT:

Description: HAS_PARENT ensures that a class exists under the specified parent.

Syntax:

Has_Parent

```
►►──OP ──child_class HAS_PARENT parent_class──;───────────────────────────────►◄
```

child_class must be a child of *parent_class*.

Syntax Logic for PARSE, LOAD, and VERIFY: Check that the class names follow the rules for class names in RODM.

LOAD Logic: Create *child_class* as a child of *parent_class*.

VERIFY Logic: Check that both *child_class* and *parent_class* exist and that the parent field of *child_class* points to *parent_class*.

HAS_PRV_FIELD:

Description: HAS_PRV_FIELD ensures that a class defines a specified private field.

Syntax:

```
►►──OP ──class HAS_PRV_FIELD (type)field──;───────────────────────────────────►◄
```

class locally defines a field named *field* of type *type*.

Syntax Logic for PARSE, LOAD, and VERIFY: Carry out the following syntax checks:

1. Check that *class* is a valid RODM class name.
2. Check that *field* is a valid RODM field name.
3. Check that *type* is a valid RODM load function data type.

LOAD Logic: Check that the *class* exists, and create *field* of *type* for *class*.

VERIFY Logic: Check that *class* exists, that it defines *field* as private, and that the type of this field matches *type*.

HAS_SUBFIELD:

Description: HAS_SUBFIELD ensures that a field of a class has a specified subfield.

Syntax:

```
►►──OP ──class.field HAS_SUBFIELD subfield──;──────────────────────────────►◄
```

field of *class* has *subfield*.

Syntax Logic for PARSE, LOAD, and VERIFY: Carry out the following syntax checks:
1. Check that *class* is a valid RODM class name.
2. Check that *field* is a valid RODM field name.
3. Check that *subfield* is a valid RODM subfield name.

LOAD Logic: Check that the *class* exists, that the *field* exists on the class, and create *subfield* of *type* for the *field* on that *class*.

VERIFY Logic: Check that *class* exists, that it locally defines *field*, and that this field has *subfield* defined.

HAS_VALUE:

Description: HAS_VALUE ensures that a field of a specific object or class has the specified value.

Syntax:

```
►►──OP ──class.──┬────────┬─.field HAS_VALUE typed_value──;─────────────────►◄
                 └─object─┘
```

field of *object* of *class* has value *typed_value*.

field of *class* has value *typed_value*.

Syntax Logic for PARSE, LOAD, and VERIFY: Carry out the following syntax checks:
1. Check that *class* is a valid RODM class name.
2. Check that *object*, if specified, is a valid RODM object name.
3. Check that *field* is a valid RODM field name.
4. Check that *typed_value* is a valid RODM typed value.

LOAD Logic: Check that the *class*, *object*, and *field* exist, set *field* of *class.object* to the type and value specified by *typed_value*, or set *field* of *class* to the type and value specified by *typed_value*.

VERIFY Logic: Check that *field* of *class.object* has the type and value specified by *typed_value* or check that *field* of *class* has the type and value specified by *typed_value*.

INHERITS:

Description: INHERITS ensures that a specific field of the specified object or class is not locally defined.

Syntax:

```
►►──OP ──class──┬─────────┬── INHERITS field──;────────────────────────────►◄
                └─.object─┘
```

field of *object* of *class* is reverted to its inherited value.

field of *class* is reverted to its inherited value.

Syntax Logic for PARSE, LOAD, and VERIFY: Carry out the following syntax checks:
1. Check that *class* is a valid RODM class name.
2. Check that *object*, if specified, is a valid RODM object name.
3. Check that *field* is a valid RODM field name.

LOAD Logic: Revert *field*. If a local value is present, it is deleted.

VERIFY Logic: Check that the value of *field* is inherited.

INVOKED_WITH:

Description: INVOKED_WITH runs a named object-specific method or an object-independent method.

A maximum of 8 parameters can be specified with *sd_parm*.

Syntax:

Invoked_With

```
►►──OP ──┬─method_name────────────────┬── INVOKED_WITH─────────────────────►
         └─class.─┬────────┬─.field─┘
                  └─object─┘

►─┬─────────────────────────┬─;────────────────────────────────────────►◄
  └─(SELFDEFINING)sd_parm─┘
```

class.object.field named object-specific method is run with *sd_parm* parameters.

class..field named object-specific method is run with *sd_parm* parameters.

method_name object-independent method is run with *sd_parm* parameters.

Syntax Logic for PARSE, LOAD, and VERIFY: Carry out the following syntax checks:

For a named object-specific method:

1. Check that *class* is a valid RODM class name.
2. Check that *object*, if specified, is a valid RODM object name.
3. Check that *field* is a valid RODM field name.
4. Check that *sd_parm* is a valid SELFDEFINING value.

For an object-independent method:

1. Check that *method_name* is a valid RODM method name.
2. Check that *sd_parm* is a valid SELFDEFINING value.

LOAD Logic:

For a named object-specific method, trigger the method specified by *class.object.field* or by *class..field* with the parameters specified in *sd_parm*. The data type of the field must be MethodSpec.

For an object-independent method, trigger the *method_name* with the parameters specified in *sd_parm*. The *method_name* must be the name of an object of the EKG_Method class.

VERIFY Logic: None.

IS_LINKED_TO:

Description: IS_LINKED_TO ensures that two objects are linked by the specified fields. The fields must be of type OBJECTLINK or OBJECTLINKLIST.

Syntax:

```
►►──OP ──class_1.object_1.field_1 IS_LINKED_TO class_2.object_2.field_2──;────────►◄
```

field_1 of *class_1.object_1* is linked to *field_2* of *class_2.object_2*.

Syntax Logic for PARSE, LOAD, and VERIFY: Carry out the following syntax checks:

1. Check that *class_1* is a valid RODM class name.
2. Check that *class_2* is a valid RODM class name.
3. Check that *object_1* is a valid RODM object name.
4. Check that *object_2* is a valid RODM object name.
5. Check that *field_1* is a valid RODM field name.
6. Check that *field_2* is a valid RODM field name.

LOAD Logic: Link *field_1* of *class_1.object_1* to *field_2* of *class_2.object_2*.

VERIFY Logic: Query *field_1* of *class_1.object_1* and check that *field_2* of *class_2.object_2* is in the list of linked fields that is returned by the query.

IS_NOT_LINKED_TO:

Description: IS_NOT_LINKED_TO ensures that two objects are not linked by the specified fields.

Syntax:

```
►►──OP ──class_1.object_1.field_1 IS_NOT_LINKED_TO class_2.object_2.field_2──;────►◄
```

field_1 of *class_1.object_1* is not linked to *field_2* of *class_2.object_2*.

Syntax Logic for PARSE, LOAD, and VERIFY: Carry out the following syntax checks:
1. Check that *class_1* is a valid RODM class name.
2. Check that *class_2* is a valid RODM class name.
3. Check that *object_1* is a valid RODM object name.
4. Check that *object_2* is a valid RODM object name.
5. Check that *field_1* is a valid RODM field name.
6. Check that *field_2* is a valid RODM field name.

LOAD Logic: Unlink *field_1* of *class_1.object_1* to *field_2* of *class_2.object_2*.

VERIFY Logic: Query *field_1* of *class_1.object_1* and check that *field_2* of *class_2.object_2* is not in the list of linked fields that is returned by the query.

NOT_A_CLASS:

Description: NOT_A_CLASS ensures that there is no class existing with the specified name. The only imperative used to implement this specification is a simple delete; if a class has objects, it cannot be deleted with this primitive alone. Instead, FORCE_NOT_A_CLASS must be used or the objects must first be deleted.

Syntax:

```
►►──OP ──class NOT_A_CLASS──;──────────────────────────────────────────────►◄
```

class is deleted if it exists and has no objects or children.

Syntax Logic for PARSE, LOAD, and VERIFY: Check that *class* is a valid RODM class name.

LOAD Logic: Delete *class*.

VERIFY Logic: Check that *class* does not exist.

SUBFIELD_HAS_VALUE:

Description: SUBFIELD_HAS_VALUE ensures that a subfield has the specified value.

Syntax:

```
►►──OP ──class.─┬────────┬─.field.subfield── SUBFIELD_HAS_VALUE typed_value──;───►◄
                └─object─┘
```

subfield of *field* of *object* of *class* has value *typed_value*.

subfield of *field* of *class* has value *typed_value*.

Syntax Logic for PARSE, LOAD, and VERIFY: Carry out the following syntax checks:
1. Check that *class* is a valid RODM class name.
2. Check that *object*, if specified, is a valid RODM object name.
3. Check that *field* is a valid RODM field name.

4. Check that *subfield* is a valid RODM subfield name.
5. Check that *typed_value* is a valid RODM typed value.

LOAD Logic: Set *subfield* of *field* of *class* to the type and value of *typed_value* or set *subfield* of *field* of *class.object* to the type and value of *typed_value*.

VERIFY Logic: Check that *subfield* of *field* of *class* has the type and value of *typed_value* or check that *subfield* of *field* of *class.object* has the type and value of *typed_value*.

SUBFIELD_INHERITS:

Description: SUBFIELD_INHERITS ensures that a specific subfield of the specified object or class is not locally defined.

Syntax:

```
►►──OP ──class.──────────┬─.field── SUBFIELD_INHERITS subfield──;─────────────►◄
                └─object─┘
```

subfield_name reverted to its inherited value. If a local value is present, it is deleted.

Syntax Logic for PARSE, LOAD, and VERIFY: Carry out the following syntax checks:
1. Check that *class* is a valid RODM class name.
2. Check that *object*, if specified, is a valid RODM object name.
3. Check that *field* is a valid RODM field name.
4. Check that *subfield* is a valid RODM subfield name.

LOAD Logic: Revert *subfield_name*.

VERIFY Logic: Check that the value of *subfield_name* is inherited.

Common Syntactic Elements

The RODM load function primitive and RODM high-level load function statements use common syntactic elements such as *class*, which is a class name. These simple common elements are described here along with descriptions of common text and numeric character strings.

These elements and character strings are described using syntax diagrams.

Syntax for Common Syntactic Elements

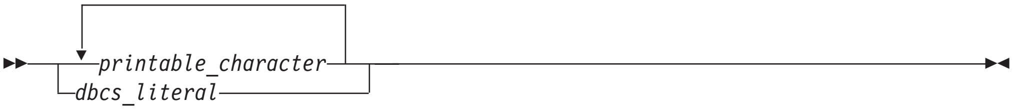
The following is a description for each common syntactic element for the RODM load function.

chars:

Purpose: A character string, which can be one or more printable single-byte or double-byte characters.

Format:

Chars



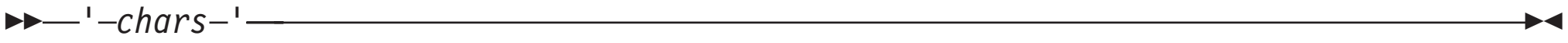
Usage Notes: A double-byte character string must be preceded by a shift-out character and ended with a shift-in character.

char_literal:

Purpose: A character string within single quotation marks.

Format:

Char_Literal



Usage Notes: To indicate a single quotation mark (') within a *char_literal*, use two immediately adjacent single quotation marks with no spaces or new lines between the two single quotation marks. This is the traditional *doubled quote* rule.

You can continue *char_literal* primitives across lines of input by enclosing the pieces on each line within single quotation marks.

class:

Purpose: A valid RODM class name.

Format:

class

Usage Notes: If the class name contains any non-alphanumeric character, enclose the class name in single quotation marks.

class_list:

Purpose: A list of RODM class names, separated by commas.

Format:

class_list

classlink_list:

Purpose: A list of class links separated by commas. Each class link is a concatenation of a class name, a period, and a field name.

Format:

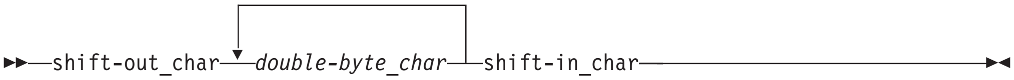
classlink_list

dbcs_literal:

Purpose: A concatenation of a shift-out character, one or more valid double-byte characters, and a shift-in character.

Format:

DBCS_Literal



Parameter Descriptions:

shift-out_char
A value of X'0E'.

double-byte_char
Four hexadecimal characters (two bytes) representing one printable character.

shift-in_char
A value of X'0F'.

Usage Notes: Double-byte text must begin with shift-out and end with shift-in. If the text continues for multiple lines, the double-byte text on each line must be within the shift-out and shift-in pair. The valid double-byte characters are the same as those for the GraphicVar data type; see "GraphicVar" on page 229.

digits:

Purpose: The concatenation of any of the decimal digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9.

Format:

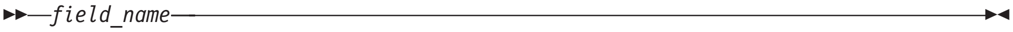
Digits

field:

Purpose: A valid RODM field name.

Format:

field



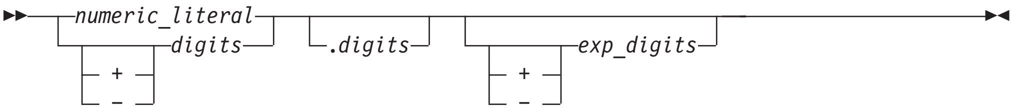
Usage Notes: If the field name contains any non-alphanumeric character, enclose the field name in single quotation marks.

float_constant:

Purpose: A floating-point constant is a concatenation of a numeric literal, an optional decimal fraction, and an optional signed floating-point exponent digit.

Format:

Float_Constant



hex_chars:

Purpose: The concatenation of hexadecimal character pairs, where each pair represents one byte.

Format:

Hex_Chars

```
     ┌──────────────┐
     ▼              │
►►───┬─0─┬──┬─0─┬───┴──────────────────────────────────────────────►◄
     ├─1─┤  ├─1─┤
     ├─2─┤  ├─2─┤
     ├─3─┤  ├─3─┤
     ├─4─┤  ├─4─┤
     ├─5─┤  ├─5─┤
     ├─6─┤  ├─6─┤
     ├─7─┤  ├─7─┤
     ├─8─┤  ├─8─┤
     ├─9─┤  ├─9─┤
     ├─A─┤  ├─A─┤
     ├─B─┤  ├─B─┤
     ├─C─┤  ├─C─┤
     ├─D─┤  ├─D─┤
     ├─E─┤  ├─E─┤
     ├─F─┤  ├─F─┤
     ├─a─┤  ├─a─┤
     ├─b─┤  ├─b─┤
     ├─c─┤  ├─c─┤
     ├─d─┤  ├─d─┤
     ├─e─┤  ├─e─┤
     └─f─┘  └─f─┘
```

hex_literal:

Purpose: One or more pairs of hexadecimal characters, within the hex delimiters.

Format:

Hex_Literal

```
►►──X'──hex_chars──'──────────────────────────────────────────────►◄
```

il_parm:

Purpose: An INDEXLIST parameter is a list of typed values. Each typed value can be either an ANONYMOUSVAR data type value or a CHARVAR data type value. However, CHARVAR values are converted to ANONYMOUSVAR values by the RODM load function.

Format:

Il_Parm

```
►►──(typed_value)─────────────────────────────────────────────────►◄
```

method_spec:

Purpose: A method specification is a concatenation of a method name and a SELFDEFINING parameter within parentheses.

Format:

method_spec

```
►►──(─method_name ─┬─────────┬─)─────────────────────────────────────►◄
                   └─sd_parm─┘
```

numeric_literal:

Purpose: A signed string of numeric digits.

Format:

Numeric

```
►►──┬─────┬──digits──────────────────────────────────────────────►◄
    ├─ + ─┤
    └─ – ─┘
```

object:

Purpose: A valid RODM object name.

Format:

object

```
►►──object_name──────────────────────────────────────────────────►◄
```

Usage Notes: If the object name contains any non-alphanumeric character, enclose the object name in single quotation marks.

objectid_list:

Purpose: A list of object IDs separated by commas. An object ID is a concatenation of a class name, a period, and an object name.

Format:

objectid_list

```
      ┌─,────────────┐
      ▼              │
►►────class.object───┴───────────────────────────────────────────►◄
```

objectlink_list:

Purpose: An *objectlink_list* is a list of object links separated by spaces. An object link is a concatenation of a class name, a period, an object name, a period, and a field name within parentheses.

Format:

objectlink_list

```
     ┌─────────────────────┐
     ▼                     │
►►───(class.object.field)──┴─────────────────────────────────────►◄
```

recipient_spec:

Purpose: A recipient specification is a concatenation of two character literals and a literal, all of which must be exactly eight bytes in length.

Format:

recipient_spec

```
►►──'appl_id','subscribe_id',X'hex_chars'────────────────────────►◄
```

Usage Notes: The first character literal is an *application_id*. The second character literal is a *subscribe_id*. If either character literal is less than eight bytes long, the literal will be left-justified and padded with blanks on the right by the RODM load function to make them eight bytes long. There must be sixteen hex digits for the hex data to be eight bytes long.

sd_parm:

Purpose: A SELFDEFINING parameter is a list of typed values, optionally separated by blanks, within parentheses.

Format:

sd_parm

```
        ┌──────────────┐
        │ └─ ─┘        │
        ▼              │
►►──(───typed_value────┴──)──────────────────────────────────────►◄
```

subfield:

Purpose: A predefined subfield name.

Format:

subfield

```
►►──┬─CHANGE─────┬───────────────────────────────────────────────►◄
    ├─NOTIFY─────┤
    ├─PREV_VALUE─┤
    ├─QUERY──────┤
    ├─TIMESTAMP──┤
    └─VALUE──────┘
```

Usage Notes: The subfield name definitions are:

CHANGE
The method specification of the change method

NOTIFY
A subscription specification list representing notification subscriptions

PREV_VALUE
The previous value of the field

QUERY
The method specification of the query method

TIMESTAMP
The time stamp of the last change to the field

VALUE
The value of the field

subs_spec:

Purpose: A *subs_spec* is a notification subscription specification which consists of a method specification followed by a recipient specification, separated by a comma.

Format:

subs_spec

```
►►──method_spec─,─recipient_spec──────────────────────────────────────────►◄
```

subs_spec_list:

Purpose: A *subs_spec_list* is a list of subscript specifications.

Format:

sub_spec&list

```
     ┌─,────────────────────────┐
     ▼                          │
►►────method_spec,recipient_spec┴─────────────────────────────────────────►◄
```

type:

Purpose: A predefined data type keyword.

Format:

type

```
►►──┬─ANONYMOUSVAR────────┬──────────────────────────────────────────────────────►◄
    ├─APPLICATIONID───────┤
    ├─BERVAR──────────────┤
    ├─CHARVAR─────────────┤
    ├─CHARVARADDR─────────┤
    ├─CLASSID─────────────┤
    ├─CLASSIDLIST─────────┤
    ├─CLASSLINKLIST───────┤
    ├─ECBADDRESS──────────┤
    ├─FIELDID─────────────┤
    ├─FLOATING────────────┤
    ├─GRAPHICVAR──────────┤
    ├─INTEGER─────────────┤
    ├─INDEXLIST───────────┤
    ├─METHODNAME──────────┤
    ├─METHODPARAMETERLIST─┤
    ├─METHODSPEC──────────┤
    ├─OBJECTID────────────┤
    ├─OBJECTIDLIST────────┤
    ├─OBJECTLINK──────────┤
    ├─OBJECTLINKLIST──────┤
    ├─OBJECTNAME──────────┤
    ├─RECIPIENTSPEC───────┤
    ├─SELFDEFINING────────┤
    ├─SHORTNAME───────────┤
    ├─SMALLINT────────────┤
    ├─SUBSCRIBEID─────────┤
    ├─SUBSCRIPTSPEC───────┤
    ├─SUBSCRIPTSPECLIST───┤
    ├─TIMESTAMP───────────┤
    └─TRANSID─────────────┘
```

Notes:

1. These data types are valid only within SELFDEFINING data:

APPLICATIONID
CHARVARADDR
CLASSIDLIST
CLASSLINKLIST
ECBADDRESS
METHODNAME
METHODPARAMETERLIST
OBJECTIDLIST
OBJECTNAME
RECIPIENTSPEC
SHORTNAME
SUBSCRIBEID
SUBSCRIPTSPEC
SUBSCRIPTSPECLIST
TRANSID

2. For limitations in CLASSID and OBJECTID, see “Using CLASSID and OBJECTID Data Types” on page 259.

typed_value:

Purpose: A *typed_value* is a concatenation of a left parenthesis, a type keyword, a right parenthesis, and a value to match the data type of the type keyword.

Format:

typed_value

```
►►──┬─(ANONYMOUSVAR)X'hex_chars'────────┬────────────────────────────────────►◄
    ├─(APPLICATIONID)'chars'────────────┤
    ├─(BERVAR)X'hex_chars'──────────────┤
    ├─(CHARVAR)'chars'──────────────────┤
    ├─(CHARVARADDR)X'hex_chars'─────────┤
    ├─(CLASSID)class────────────────────┤
    ├─(CLASSIDLIST)class_list───────────┤
    ├─(CLASSLINKLIST)classlink_list─────┤
    ├─(ECBADDRESS)X'hex_chars'──────────┤
    ├─(FIELDID)class.field──────────────┤
    ├─(FLOATING)float_constant──────────┤
    ├─(GRAPHICVAR)dbcs_literal──────────┤
    ├─(INTEGER)numeric_literal──────────┤
    ├─(INDEXLIST)il_parm────────────────┤
    ├─(METHODNAME)method_name───────────┤
    ├─(METHODPARAMETERLIST)sd_parm──────┤
    ├─(METHODSPEC)method_spec───────────┤
    ├─(OBJECTID)class.object────────────┤
    ├─(OBJECTIDLIST)objectid_list───────┤
    ├─(OBJECTLINK)(class.object.field)──┤
    ├─(OBJECTLINKLIST)objectlink_list───┤
    ├─(OBJECTNAME)object────────────────┤
    ├─(RECIPIENTSPEC)recipient_spec─────┤
    ├─(SELFDEFINING)sd_parm─────────────┤
    ├─(SHORTNAME)'chars'────────────────┤
    ├─(SMALLINT)numeric_literal─────────┤
    ├─(SUBSCRIBEID)'chars'──────────────┤
    ├─(SUBSCRIPTSPEC)subs_spec──────────┤
    ├─(SUBSCRIPTSPECLIST)subs_spec_list─┤
    │                          (1)      │
    ├─(TIMESTAMP)X'hex_chars'───────────┤
    │                        (2)        │
    └─(TRANSID)X'hex_chars'─────────────┘
```

Notes:

1 TIMESTAMP must be exactly 8 bytes.

2 TRANSID must be exactly 8 bytes.

Usage Notes: You can specify null values for some of the data types. See “Null Values for RODM Load Function Data Types” on page 260.

Chapter 11. Writing Applications that Use RODM

RODM provides a *user application programming interface* (user API). This user API allows a properly authorized address space to access the data contained in the RODM address space and data spaces. Through this user API, objects can be created, organized into hierarchies, or deleted. The user API can also be used to query the value of a field associated with an object or to alter the value in that field. The user API can be called from NetView command processors and from applications written in any programming language that meets the parameter passing conventions of RODM. While RODM provides control block mappings in PL/I and C, you can write applications in any programming language that uses the interface described in “Register Conventions” on page 302.

RODM also provides a method API, which shares many functions with the user API. The method API is described in Chapter 13, “Writing RODM Methods,” on page 339.

The NetView program supplies a set of general-purpose methods. For a description of these methods, see “Supplied Methods” on page 480.

Tasks Best Performed with User Applications

This section describes which tasks are best performed with user applications.

Use an application program to do the following:

- Supply status changes of resources to the RODM data cache.

  The RODM data cache is viewed as a model of real-world resources; therefore, ensure that resource objects in the data cache are updated as actual resources change status.
- Subscribe for notification of data changes.

  Before a user application program can receive notification of RODM data cache changes, a notification subscription to the necessary fields in the relevant objects or classes is required.
- Wait for and process data change notifications.

  The user application is responsible for waiting for and processing the notifications from the objects or classes to which it is subscribed.
- Query data for operator view, displays, and queries.

  Application programs that communicate with users through a user interface and require access to data in the RODM data cache and must query that data through RODM.
- Add or delete resources.

  Application programs requiring data cache hierarchy modification can do so by calling RODM to manipulate objects and classes.
- Communicate with NetView applications.

  NetView applications can query and change RODM data through the user API. You can use either RODMView or the MultiSystem Manager Access facility to query and change RODM data.

Using the User Application Program Interface

User API calls to RODM must pass the following four parameters to module EKGUAPI:
- Access block
- Transaction information block
- Function block
- Response block

The function block can point to additional parameters, such as entity access information blocks and field access information blocks, which identify the target of the function.

Figure 68 shows typical user API invocations, first in C and then in PL/I.

```
#include <EKG3CEEP.H>                      /* EKGUAPI declaration for C    /*
EKGUAPI( &access_block,                      /* address of access block      /*
         &transaction_info_block,            /* address of trans info block  /*
         &function_block,                    /* address of function block    /*
         &response_block);                   /* address of response block    /*


%include syslib (EKG1IEEP);                  /* EKGUAPI declaration for PL/I /*
CALL EKGUAPI( access_block,                  /* access block                 /*
              transaction_info_block,        /* transaction info block       /*
              function_block,                /* function block               /*
              response_block);               /* response block               /*
```

Figure 68. Typical User API Invocation in C and PL/I

The call statement transfers control to the code segment identified as EKGUAPI. The user can include EKGUAPI module during the link-edit of the application.

Register Conventions

The generated code must follow these conventions.

Register 1
Points to a four-entry parameter list that contains the addresses of the access_block, transaction_info_block, function_block, and response_block, respectively. These control blocks are shown in Figure 69 on page 305.

Register 13
Contains the address for the calling program's 72-byte save area.

Register 14
Contains the return address for the calling program.

Register 15
Contains the entry address for the EKGUAPI module.

Usage Notes

Within this programming guide the term *null pointer* is used. The value of a null pointer is defined as X'00000000'. Using PL/I, this value is provided by the built-in SYSNULL function. Do not use the built-in NULL function; it generates the value X'FF000000'.

If the call is made from a high-level language where the parameter list is built by the compiler and a null response_block value cannot be passed, a pointer to a dummy response_block must be specified. The dummy response_block must be in the correct format and specify a length of at least 8. See “Response Block” on page 314 for additional information about response blocks.

User API calls are synchronous. The EKG_ExecuteFunctionList function can specify a list of other functions that are to be run. If the list of functions contains two adjacent functions that affect the same object, the lock on that object is not released during the time interval between the processing of the two functions.

RODM applications must be running in key 8 at the time EKGUAPI is called. All parameter lists, control blocks, and other data areas that are passed to RODM must reside in storage that is accessible in key 8.

Compiling and Link-Editing

The application can link-edit the EKGUAPI module during the link-edit step or dynamically load the module during execution.

Compiling C Modules that Call EKGUAPI

If any RODM control blocks are referenced in the modules, include the EKG3CINC.H file in your source file. This file includes all of the RODM function and response blocks, and the function prototype statements for the RODM entry points EKGUAPI, EKGMAPI, and EKGWAIT.

If no RODM control blocks are referenced in the modules, but the modules call EKGUAPI or EKGWAIT, include the EKG3CEEP.H file in your source file.

Example:

```
#include "EKG3CINC.H"
  /* or */
#include "EKG3CEEP.H"

void thisproc (void arg)
{
   /* code */
}
```

Compiling PL/I Modules that Call EKGUAPI

If any RODM control blocks are referenced in the modules, include the EKG1IINC file in your source file. This file includes all of the RODM function and response blocks, and the function prototype statements for the RODM entry points EKGUAPI, EKGMAPI, and EKGWAIT.

If no RODM control blocks are referenced in the modules but the modules call EKGUAPI or EKGWAIT, include the EKG1IEEP file in your source file.

Specify the MACRO preprocessor compiler option if you include RODM macros in your user application, for example, as follows:

```
*PROCESS MACRO;
 thisproc: proc;

%include ekglib(EKG1IINC);
    or
%include ekglib(EKG1IEEP);
```

```
 /* code */

end thisproc;
```

Linking Modules that Call EKGUAPI Directly

The INCLUDE SYSLIB(EKGUAPI) link-edit control statement must be specified before the ENTRY statement in your source file.

The AMODE=31 link-edit option must be specified.

The RMODE=ANY or RMODE=24 link-edit option must be specified.

The following ENTRY CEESTART statement must be specified:

```
<module code>

INCLUDE SYSLIB(EKGUAPI)
ENTRY CEESTART
NAME module_name(R)
```

Linking Modules that Load and then Call EKGUAPI

Because EKGUAPI is a load module, modules that first load and then call EKGUAPI do not need special link-edit control statements. However, the EKGUAPI load module must be accessible to the module that loads it (through STEPLIB, JOBLIB, or z/OS linklist).

Using Control Blocks

All user API calls to RODM pass four parameters as shown in Figure 69 on page 305. The figure is an example of the relationships between the user API call and the control blocks for a RODM query function request. The control block relationships are similar for other RODM function requests from the user application.

The parameters passed are pointers to the following control blocks:

Access Block
Contains the user information needed to process the user API request.

Transaction Information Block
Contains transaction information and status about the API request.

Function Block
Contains the details of the requested transaction against RODM data. The content of this control block varies depending on the transaction requested. For some requested transactions it includes pointers to two information blocks:
Entity Access Information Block
Field Access Information Block

Response Block
Contains the output data from the transaction requested. The format and specific content of the response block depends on the type of transaction requested.

In Figure 69 on page 305, the PL/I-like syntax describes the four passed control blocks and the two associated access information blocks. Equivalently organized blocks can be represented in C. The actual order and offset position within the control blocks are specified in the tables referenced within each of the following control block descriptions.

```
Call EKGUAPI( access_block,
              transaction_info_block,
              query_function_specific_data,
              response_block
            );
```

```
Declare
  1 Access_block,
    2 RODM_name,
    2 Sign_on_token,
    2 User_appl_ID;
```

```
Declare
  1 Transaction_info_block,
    2 API_version,
    2 EPL_blk_len,
    2 Transaction_ID,
    2 Return Code,
    2 Reason Code,
    2 EPL_info,
      3 Lock_level;
```

```
Declare
  1 Query_function_specific_data,
    2 Function_specific_data,
      3 Function_ID,
      3 Entity_access_info_ptr,
      3 Field_access_information_block,
        4 Field_access_info_ptr,
        4 Subfield,
      3 Reserved,
      3 Method_parms;
```

```
Declare
  1 Entity_access_info,
    2 Reserved,
    2 Naming_count,
    2 Class_information,
      3 Class_ID,
      3 Class_name_length,
      3 Class_name_ptr,
    2 Object_information,
      3 Object_ID,
      3 Object_name_length,
      3 Object_name_ptr;
```

```
Declare
  1 Response_block,
    2 Response_block_length,
    2 Response_block_used,
    2 Requested_data,
      3 Data_type,
      3 Data;
```

```
Declare
  1 Field_access_info,
    2 Reserved,
    2 Naming_count,
    2 Field_information,
      3 Field_ID,
      3 Field_name_length,
      3 Field_name_ptr;
```

Figure 69. API Query Function Control Block Example

Access Block

Description

The access block contains user information that RODM needs to process user API requests.

Function Block Format

Table 25 on page 306 describes the format of the access block. The table headings have the following meanings:

Offset Specifies the offset to the beginning of the parameter in decimal bytes.

Length
Specifies the length of the parameter in decimal bytes.

Type Specifies the RODM data type of the parameter. See “Abstract Data Type Reference” on page 223 for more information.

Use Specifies whether the parameter is used for data input to a function or for data output by a function.

Parameter Name
Specifies the name of the parameter.

Table 25. RODM Access Block

Offset	Length	Type	Use	Parameter Name
000	8	character(8)	In	RODM_name
008	16	Anonymous(16)	In/Out	Sign_on_token
024	8	ApplicationID	In	User_appl_ID

Function Block Field Descriptions

`RODM_name`
The name of the RODM that is to receive this request to connect must be placed by the caller in the RODM_name field. Because the access block is usually reused on successive calls, the RODM_name field is set only once by a user, just before the connection request is issued. This is the name that you specify when you start RODM. To determine the RODM name, refer to NetView online help.

`Sign_on_token`
The token that RODM uses to uniquely identify the user. The data structure that RODM sets at completion of the connection is returned in the sign_on_token parameter.

The sign_on_token is set by RODM each time a user connects to RODM.

`User_appl_ID`
The identifier that the user application program specifies to identify itself. For an APF (authorized program facility) authorized program, the User_appl_ID alone identifies the user to RODM and determines the user's capabilities. For application programs that are not APF authorized, the User_appl_ID is combined with the password or password phrase from the connect function block to identify the user to RODM and determine the user's capabilities. This field is a maximum of 8 bytes with shorter values left-justified in the field and padded on the right with blanks. Valid characters for this string are the same as for object names.

Examples

Sample control blocks for PL/I and C are supplied with RODM. Include these control blocks in your programs.

Table 26. Sample Names for Access Block

Example	Name
PL/I access block	EKG1ACCB
C access block	EKG3ACCB

Usage

RODM needs a fully initialized access block to successfully complete user API calls that are issued after the Connect request. You must reference or define an access control block with every call to the RODM User Interface (EKGUAPI).

Several applications can access the RODM data cache at the same time and trigger methods appropriate to each application's function. The sign_on_token field of the access block is used to identify the user for each transaction.

RODM verifies the authorization level of the user application. Each RODM function requires a particular authorization level.

The fields in the access block set by the caller are the RODM_name and User_appl_ID fields. These fields are set once, by the application, just before the user API is called. The EKG_Connect or EKG_ConnectLong user API fills in a value for the sign_on_token field. After the access block is established by a connect request, the application does not modify the information in that block.

More details about connection to RODM are provided in "Connecting to RODM" on page 327.

Transaction Information Block

Description

The transaction information block contains transaction-status information about each API request. The transaction information block is required for every RODM function request.

Function Block Format

Table 27 describes the format of the transaction information block. The table headings have the following meanings:

Offset Specifies the offset to the beginning of the parameter, in decimal bytes.

Length
Specifies the length of the parameter, in decimal bytes.

Type Specifies the RODM data type of the parameter.

Use Specifies whether the parameter is used for data input to a function or for data output by a function. A dash (—) indicates that the parameter is not used by functions or is reserved.

Parameter Name
Specifies the name of the parameter.

Table 27. RODM Transaction Information Block

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	API_version
004	4	Integer	In	EPL_blk_len
008	8	TransID	Out	Transaction_ID
016	4	Integer	Out	Return_code
020	4	Integer	Out	Reason_code
024	0	Structure	—	EPL_info
024	4	Integer	In	Lock_level

Function Block Field Descriptions

`API_version`

The API_version field specifies the version of the API that RODM is to use for the API request. The valid values for this field are:

0 RODM is to use the most recent API version

1 RODM is to use version 1 API

`EPL_blk_len`
Not used, but retained for compatibility.

`Transaction_ID`
Every RODM transaction initiated by a user application is assigned a unique transaction ID by RODM. Synchronous method transactions that are triggered by a user application transaction have the same transaction ID as the user application. The transaction_ID field controls the order of this transaction relative to all other transactions. The transaction ID is also used in journaling all transactions against RODM between checkpoints. These are described in detail in the section of this document on Registering for Checkpoint Notification. See "Coding Checkpoint Control" on page 382.

`Return_code`
Return code from RODM. See "RODM Return and Reason Codes" on page 452 for a list of return codes.

`Reason_code`
Reason code from RODM. See "RODM Return and Reason Codes" on page 452 for a list of reason codes.

`EPL_info`
Not used, but retained for compatibility.

`Lock_level`
Not used, but retained for compatibility.

Examples

Sample control blocks for PL/I and C are supplied with RODM. Include these control blocks in your programs.

Table 28. Sample Names for Transaction Information Block

Example	Name
PL/I transaction information block	EKG1TRAB
C transaction information block	EKG3TRAB

Usage

The return code and reason code fields are used for RODM to communicate with the user application about the status of the requested function.

Function Block

Description

The details of all transactions against RODM data are specified in function blocks. A user builds a function block and passes it to RODM to request a desired transaction.

Function Block Format

The format of each function block is listed in "Function Reference" on page 371.

Function Block Field Descriptions

A description of each parameter used in the function blocks is listed in "Function Parameter Descriptions" on page 445.

Usage

The first field in every function block contains a 4-byte integer that specifies which function is being requested. The format of the remainder of the function block is dependent upon the four-byte function ID.

One common format for a function block includes the specification of a class, an object, and a field. Sometimes there are also fields in the function block used to specify a subfield in RODM. Sometimes only a class and an object can be specified in a function block. Sometimes, only a class can be specified.

Entity Access Information Block

Description

The entity access information block (EAIB) contains information used by the API to access a class or object. The EAIB is separate from the function block so that it can be reused on subsequent API calls. A pointer to the EAIB is stored in the function block.

The access information is available in two different forms:

- Symbolic names provided by the application.
- IDs generated by RODM when symbolic names are used to create a class or object. This form provides the fastest access to the information.

Function Block Format

Table 29 describes the format of the entity access information block. The table headings have the following meanings:

Offset Specifies the offset to the beginning of the parameter, in decimal bytes.

Length
Specifies the length of the parameter, in decimal bytes.

Type Specifies the RODM data type of the parameter.

Use Specifies whether the parameter is used for data input to a function or for data output by a function. A dash (—) indicates that the parameter is not used by functions or is reserved.

Parameter Name
Specifies the name of the parameter.

Table 29. RODM Entity Access Information Block

Offset	Length	Type	Use	Parameter Name
000	4	Anonymous(4)	—	Reserved
004	4	Integer	In	Naming_count
008	4	ClassID	In/Out	Class_ID
012	4	Integer	In	Class_name_length
016	4	Pointer	In	Class_name_ptr
020	8	ObjectID	In/Out	Object_ID
028	4	Integer	In	Object_name_length
032	4	Pointer	In	Object_name_ptr

Function Block Field Descriptions

`Naming_count`
The Naming_count field in the entity access information block specifies which data in the block is valid. Valid values are:

Value	Meaning
0,2	Specifies that the target of the function is either a class or an object and that both the object access information and the class access information are valid.
1	Specifies that the target of the function is a class and that only the class access information is valid.

Interpretation of all this information is subject to the rules in "Usage."

`Class_ID`
Class identifier.

`Class_name_length`
Class name length.

`Class_name_ptr`
This is the pointer to the class name. With a variable declared in PL/I as a varying length string, for example, CLASS1 CHAR(64) VARYING, the class name pointer is specified using the PL/I Pointeradd built-in function. To point directly at the character data rather than at the PL/I 2-byte length prefix, code `class_name_ptr = POINTERADD(ADDR(CLASS1) ,2 )`

`Object_ID`
Object identifier.

`Object_name_length`
Object name length.

`Object_name_ptr`
This is the pointer to the object name. With a variable declared in PL/I as a varying length string, for example, OBJECT1 CHAR(255) VARYING, the object name pointer is specified using the PL/I Pointeradd built-in function. To point directly at the character data rather than at the PL/I 2-byte length prefix, code `object_name_ptr = POINTERADD(ADDR(OBJECT1) ,2 )`

Examples

Sample control blocks for PL/I and C are supplied with RODM. Include these control blocks in your programs.

Table 30. Sample Names for Entity Access Information Block

Sample	Name
PL/I entity access information block	EKG1ENTB
C entity access information block	EKG3ENTB

Usage

The function_ID in the function block specifies the function block used. The function block specifies whether or not the entity access information block is used for that function.

A null length value for a corresponding pointer indicates a null string, regardless of the value of the pointer. Similarly, a null pointer value also indicates a null string, regardless of the value of the corresponding length. A null string is indicated by either a null length or a null pointer.

Pointers to names, if used, point to variable-length character strings. The length of the character string is specified as a parameter in the entity access information block, and the pointer in the entity access information block directly points to the first byte of the character data.

Identifiers (RODM-generated internal IDs) exist in RODM because they are faster to process than are character string names. Identifiers are always given preference over character string names in resolving which class or object is to be addressed. The following apply:

- If both the Class_ID and the Class_name_length are not null values in an entity access information block, the Class_ID is used, and the Class_Name_Ptr is ignored. RODM does not check to determine if a Class_ID is consistent with a class name where both are supplied by the caller.
- If both the Object_ID and the Object_name_length are not null and the Naming_count is not 1, the Object_ID is used, and the Object_Name_Ptr is ignored. RODM does not check to determine if a supplied Object_ID is consistent with a supplied object name.
- If the Naming_count is 1, only class information is used by RODM.

An object identifier is sufficient to locate an object; it includes the identification of the class that contains the object. When an object identifier is given, RODM ignores all other object and class information.

If no Object_ID is provided and an object is required in the specification of the target of the intended transaction, an Object_Name must be provided. In that case, either the Class_ID must specify the class of the object, or the Class_Name_Ptr must point to the name of the class. An error results if the specified class has no object with that name.

For transactions that address a field of a class, no object is involved. The same format is used for object and class access information blocks. Set the Object_ID and the Object_name_length fields to null values to alert RODM that the target of the transaction is on a class instead of on an object. The target class is the one specified with either a Class_ID or by the Class_Name_Ptr. Alternatively, the user can set the Naming_count field to a value of 1 and limit the scope of information analyzed by RODM.

Control blocks are designed to be used repeatedly. For improved performance, reuse control blocks. During the execution of an application that uses RODM, similar transactions might be repeatedly requested with changes in the targets of those transactions. The following actions are taken by RODM to simplify repeated use of an entity access information block.

- If the Class_ID field is null when RODM is called, and the Class_Name_Ptr field is not null, and the requested transaction completes successfully (a return code less than or equal to 4), RODM fills in the Class_ID field with the class-identifier of the target class. RODM also fills in the Class_ID when an error prevents the successful completion of the transaction if the target is accessed before the error is detected.
- If the Object_ID field is null when RODM is called, and the Object_Name_Ptr is not null, and the naming count is not equal to 1 (which specifies that only class information is used), and the requested transaction completes successfully (a return code less than or equal to 4), RODM fills in the Object_ID field with the

Object-identifier of the target Object. RODM also fills in the Object_ID when an error prevents the successful completion of the transaction if the target is accessed before the error is detected.

If names are used to specify the targets in a transaction request and the request is then repeated, reusing the same entity access information block, the identifier fields are already filled in from the first transaction. The second transaction, therefore, runs more quickly.

This increase in performance of a second transaction occurs to a lesser degree in each of several circumstances where the second transaction is similar to but not the same as the first transaction. For example, a performance increase of a lesser degree on a second transaction is obtained when:

- The second transaction specifies the same field as the first transaction, regardless of the class and object fields.
- The first and second transactions have the same object as a target, but the first transaction uses a character string name to specify the object.
- The second transaction specifies the same class as the first transaction (in the class fields), but each transaction specifies a different object using a character string name. When entity access information blocks are repeatedly used in this way, the ObjectID must be set to null after each use of that block. Otherwise, on reuse, the rule that identifiers are given preference over character string names applies, and the second transaction is routed to the same target object, as that of the first transaction.

When a function block is reused and the Class_name or Object_name field (or pointer) is updated, the corresponding identifier fields (Class_ID, Object_ID) must be reset to null. This is necessary because the character string name has significance only if the identifier field is set to 0.

Field Access Information Block

Description

The field access information block (FAIB) contains information used by the API to access a field. The FAIB is separate from the function block so that it can be reused on subsequent API calls. A pointer to the FAIB is stored in the function block.

The access information is available in two different forms:

- Symbolic names provided by the application.
- IDs generated by RODM when symbolic names are used to create a field. This form provides the fastest access to the information.

Function Block Format

Table 31 on page 313 describes the format of the field access information block. The table headings have the following meanings:

Offset Specifies the offset to the beginning of the parameter, in decimal bytes.

Length
Specifies the length of the parameter, in decimal bytes.

Type Specifies the RODM data type of the parameter.

Use Specifies whether the parameter is used for data input to a function or for data output by a function. A dash (—) indicates that the parameter is not used by functions or is reserved.

Parameter Name
Specifies the name of the parameter.

Table 31. RODM Field Access Information Block

Offset	Length	Type	Use	Parameter Name
000	4	Anonymous(4)	—	Reserved
004	4	Integer	In	Naming_count
008	4	FieldID	In/Out	Field_ID
012	4	Integer	In	Field_name_length
016	4	Pointer	In	Field_name_ptr

Function Block Field Descriptions

`Naming_count`
The naming_count field in the field_access_info block specifies if the field access information is valid. The valid values are:

Value	Meaning
0	The information is valid
1	Reserved

Always set Naming_count to 0 (zero).

`Field_ID`
Field identifier.

`Field_name_length`
Field name length.

`Field_name_ptr`
This is the pointer to the field name.

Examples

Sample control blocks for PL/I and C are supplied with RODM. Include these control blocks in your programs.

Table 32. Sample Names for Field Access Information Block

Example	Name
PL/I field access information block	EKG1FLDB
C field access information block	EKG3FLDB

Usage

The function_ID in the function block specifies the function block used. The function block specifies whether the field access information block is used for that function.

A null length value for a corresponding pointer indicates a null string, regardless of the value of the pointer. Similarly, a null pointer value also indicates a null string, regardless of the value of the corresponding length. A null string is indicated by either a null length or a null pointer.

Pointers to names, if used, point to variable-length character strings. The length of the character string is specified as a parameter in the field access information block along with the pointer that points directly to the first byte of the character data.

Identifiers (RODM-generated internal IDs) exist in RODM because they are faster to process than are character string names. Identifiers are always given preference over character string names in resolving which field is to be addressed. If both the Field_ID and the Field_name_length are not null in a field access information block, the Field_ID is used, and the Field_Name_Ptr is ignored. RODM does not check that a supplied Field_ID is consistent with a supplied field name.

If a field is the target of the desired transaction, the specification of a field must be provided by a Field_ID or Field field that is not null. The specified field is associated with the entity (object or class) specified in the corresponding entity access information block.

If names are used to specify the targets in a transaction request and the request is then repeated, reusing the same entity access information block, the identifier fields are already filled in from the first transaction. The second transaction, therefore, runs more quickly.

Control blocks are designed to be used repeatedly. For improved performance, reuse control blocks. During the execution of an application that uses RODM, similar transactions might be repeatedly requested with changes in the targets of those transactions. RODM takes the following action to simplify repeated use of a field access information block:

- If the Field_ID field is null when RODM is called, and the Field_name_Ptr is not null, and the target of the transaction requires a field, and the requested transaction completes successfully, RODM fills in the Field_ID field with the Field-identifier of the target field.
- RODM also fills in the Field_ID when an error prevents the successful completion of the transaction if the target is accessed before the error is detected.

When a function block is reused and the Class_name or Object_name field (or pointer) is updated, the corresponding identifier fields (Class_ID, Object_ID) must be reset to null. This is necessary because the character string name has significance only if the identifier field is set to 0.

Response Block

Description

The output from RODM query requests, query methods, named methods, and object-independent methods is returned in response blocks. The format of the response block and the data that the response block contains are dependent on the kind of transaction that generated the response.

Function Block Format

The format of each response block is listed with its associated function. Table 33 contains a page reference to each response block format by function.

Table 33. Functions with Response Blocks

Function with Response Block	See Page
EKG_Locate	404
EKG_QueryEntityStructure	409
EKG_QueryField	411
EKG_QueryFieldID	412
EKG_QueryFieldName	414

Table 33. Functions with Response Blocks (continued)

Function with Response Block	See Page
EKG_QueryFieldStructure	415
EKG_QueryMultipleSubfields	419
EKG_QueryFunctionBlockContents	417
EKG_QueryNotifyQueue	421
EKG_QueryObjectName	423
EKG_QueryResponseBlockOverflow	424
EKG_QuerySubfield	426
EKG_TriggerNamedMethod	439
EKG_TriggerOIMethod	440
EKG_WhereAmI	444

Function Block Field Descriptions

A description of each parameter used in the response blocks is listed in “Function Parameter Descriptions” on page 445.

Usage

All response blocks have the same basic format:

- A Response_block_length field set by the method or application indicates the length in bytes of the response block that is supplied.
- A Response_block_used field set by RODM indicates the amount of storage used in the response block or the amount needed if the block is too small.
- A block of storage whose format and contents depend on the transaction type but that typically contains:
  - A Data_type field providing the data type ID of the returned data
  - The data returned by the function or by a method triggered by the function

If the response block provided by the caller is too small to hold a complete response, one of the following happens:

- If the supplied response block has fewer than 8 bytes, the transaction is immediately ended with an error return code.
- If the supplied response block has 8 or more bytes, the transaction is run by RODM.
- The data type and lengths of the returned values and the volume of the output that is generated determine the total number of bytes needed in a response block.
- If there is insufficient room in the response block for the normal return of information after RODM has completed the transaction, RODM sets the Response_Block_Used field of the response block to show the total size of the generated response. RODM stores that portion of the data in the response block equal to the number of bytes specified in the Response_Block_Length field.

  RODM can take one of two actions depending on the setting of the EKG_RBOverflowAction field in the user object:
  - If that field specifies discard, any overflow data is lost.
  - If that field specifies to save overflow information, RODM saves the response block overflow data for the user to retrieve on a later call.

See "EKG_QueryResponseBlockOverflow — Query for Response Block Overflow" on page 424.

The overflow data is identified by the Transaction ID in the transaction information block of the transaction that caused the overflow. The Transaction ID must be specified in the Correlation_ID parameter of the EKG_QueryResponseBlockOverflow function to retrieve the data that did not fit into the original response block. The return and reason codes that are passed to RODM in the function block are set to show the error (response block is too small).

Note: With the exception of the EKG_QueryResponseBlockOverflow function and the EKG_Disconnect function, additional transactions associated with the same access block as this transaction are rejected by RODM until the response block overflow data is retrieved by the user.

- If the transaction causing a response block overflow is run from a list of transactions, remaining transactions in the list are run with all results going into the overflow block for later retrieval.
- All overflow data is placed into an overflow buffer. It is the responsibility of the application to concatenate the data in the response block and this overflow data.

Following the response_block_used field, the remainder of the block depends on transaction type, data types, and lengths of lists of data.

When named and object-independent methods are triggered by transactions against RODM, those methods can generate SelfDefining data strings (variable length strings of type SelfDefining) that return to the task running the transaction through the response block. When named and object-independent methods are triggered, the variable portion of the response block is dedicated to delivering these strings to the calling task.

If a named or object-independent method causes an overflow in the response block, the method itself receives a return code and reason code for the overflow. However, the method might not pass this return code and reason code back to the program that triggered the method. Always compare the Response_block_length parameter with the Response_block_used parameter returned in the response block if a named or object-independent method is triggered. If the value of the Response_block_used parameter is larger than the value of the Response_block_length parameter, an overflow occurred.

If multiple transactions are running simultaneously on a single user application ID, any or all of them can cause a response block overflow. After an overflow occurs, no further user API functions are enabled from EKGUAPI (with the exception of the EKG_Disconnect function) until the EKG_QueryResponseBlockOverflow function is called.

All overflow response blocks must be retrieved by the EKG_QueryResponseBlockOverflow function before any other user API request (with the exception of the EKG_Disconnect function) is enabled from EKGUAPI. Each call to the EKG_QueryResponseBlockOverflow function must specify a correlation ID, which is the transaction ID of the transaction that caused the response block overflow. The correlation ID allows the correct overflow response block to be returned.

Additional details on various kinds of response blocks are provided with many of the descriptions of individual RODM functions.

Error Conditions in Transactions

If an error condition occurs during the execution of a transaction, RODM issues a return code and reason code in the transaction information block. Errors can also be recorded in the RODM log, depending on the values of LOG_LEVEL and MLOG_LEVEL that are set in the customization file. Unless a method abends, the decision to continue execution is left to the method.

Methods can issue return codes to RODM using the EKG_SetReturnCode function. See "EKG_SetReturnCode — Set Return and Reason Codes" on page 432. The error can be recorded in the RODM log, and the return and reason code in the call to RODM are set to show that the transaction did not complete successfully.

The return code and reason code issued to methods and user applications are determined by RODM as follows:

- The initial return code and reason code for all user API and method API transactions are set to 0.
- The return code and reason code returned to the user application are determined by a synchronous method if one is triggered during the processing of the user API request. If a synchronous method does not set the return code, it is set by RODM if RODM detects an error during the execution of the user API transaction.
- A method can set the return code and reason code that are returned to the caller. The current return and reason codes for a method are initially set to 0. The method can change the return and reason codes using the EKG_SetReturnCode function. The current return and reason codes are returned to the method that triggered this method or to RODM, if RODM triggered this method.

  If the method sets a new return code and reason code using the EKG_SetReturnCode function, RODM determines the return code and reason code that are returned to the caller as follows:

  - If the new return code is greater than the current return code, the new return code and reason code replace the current return and reason code for the method.
  - If the new return code is less than or equal to the current return code, the current return and reason code for the method are not changed.
- If the return code and reason code set by a method are returned to the method that called it, the calling method's return code and reason code are determined exactly as was the called method's.

In addition to issuing return and reason codes, RODM can also write log records that provide additional diagnostic information about errors. Transactions that pass through the user API are each given a unique Transaction_ID, which RODM returns to the caller in the access block. If errors occur in methods or elsewhere in a transaction, the Transaction_ID is written in the RODM log record for the error. Transactions that pass through the method API are each given the Transaction_ID of the parent transaction that was submitted across the method API.

- If a method calls the EKG_SetReturnCode function and the return code and reason code are changed, RODM writes a type-3 log record (for object-specific methods) or a type-4 log record (for object-independent methods) only if the following are true:
  - If the method is a synchronous method, the return code must be greater than the value of the EKG_LogLevel field in the application program's EKG_User object, and logging must be enabled. For information about the EKG_LogLevel field, see "EKG_User Class" on page 201.

  - If the method is asynchronous, the return code must be greater than the LOG_LEVEL parameter in the RODM customization file. Refer to the *IBM Tivoli NetView for z/OS Administration Reference* for more information about the RODM customization file.
- The final return code and reason code returned from the level-1 method (that is, the first asynchronous method that is triggered by a EKG_MessageTriggeredAction function) determines the following:
  - If the final return code is greater than or equal to the value in the EKG_LogLevel field of the user object that represents the application program that triggered the asynchronous method, a log record is written.
- For user application programs that call EKGUAPI:
  - If the final return code is greater than or equal to the value in the EKG_LogLevel field of the application program's object, RODM writes a type-2 log record to the log.

Following is an example of return and reason code propagation:

1. User application program UA1 calls EKGUAPI to query a field.
2. Query method QM1 is triggered because the queried field has a query method subfield. The initial return code and reason code for QM1 are both 0.
3. QM1 triggers a named method, NM1, to perform some processing on the target object. The initial return code and reason code for NM1 are both 0.
4. NM1 sets the return code and reason code, using the EKG_SetReturnCode function, to 4 and 2000, respectively.
5. QM1 receives return code and reason code 4 and 2000 from the named method but does not want to return these return and reason codes to the user application program. Instead, it sets the return code and reason code to 0 and 3000, respectively, using the EKG_SetReturnCode function. Had QM1 not set the return code and reason code with the EKG_SetReturnCode function, RODM returns the return and reason codes of 0 to the user application program.
6. The user application program receives the return and reason codes of 0 and 3000.

Method writers must be aware of the implications of issuing return and reason codes from methods. An application might interpret a return and reason code returned by the method as being related to the success or failure of the function, when it might only relate to the success or failure of the method. For example, a notification subscription is assigned to a field that is successfully changed by the EKG_ChangeField function, but the notification method fails and sets a return and reason code. In this case, the application might interpret the return and reason code as a failure of the EKG_ChangeField function and not a failure of the notification method.

RODM Notification Process

The RODM notification process enables your user application to be notified when a specified field in RODM changes value. You can use the notification process to automate any process that needs to take place when the value of a field changes. For example, you can automate the recovery of certain network resources when they go down.

The RODM notification process can also be used to notify user applications of:

- Asynchronous errors and checkpoints. "Asynchronous Error Notification" on page 325 describes notification for errors and checkpointing. User applications must set up any required notifications as soon as possible after connecting to RODM.
- Deleted objects. "Object Deletion Notification" on page 326 describes notification for deleted objects. Instead of installing your own notification methods, your applications use the EKG_AddObjDelSubs function (described on page 374) to subscribe to notification of deleted objects.

This section describes the RODM notification process, using an example of an automated recovery application. For this example, assume that you have resources named NETRES1, NETRES2, NETRES3, and so on, represented by objects in the RODM data cache. A field of the object named DisplayStatus represents the status of the resource; the value of this field is maintained by another application. Assume also that you have written a user application named RECOVER that can recover one of these resources when it goes down. Set up RODM so that your RECOVER application is notified each time a resource goes down.

The RODM notification process has four overall steps:
1. Setup
2. Wait
3. Notification
4. Clean up

Each overall step is described using the RECOVER example. Some steps can be done in different ways; this example follows the simplest way and describes the other ways as well.

The RODM notification process has five elements:
- Notification queue
- Notification method
- Notify subfield
- Event control block (ECB)
- User application

Setup

The first step in the RODM notification process is setup. Setup includes:
- Connecting the user application to RODM
- Creating the notify subfield
- Installing the notification method
- Creating the notification queue
- Subscribing to the field

This example assumes that RODM is running and the objects and application that maintains them are defined. You can complete the setup steps for each field on each object for which you want to be notified, or you can set up notification at the class level. If you set up notification at the class level, the notification process is defined for every object of that class.

1. The first step in working with RODM is connecting to RODM. The RECOVER application connects to RODM using the EKG_Connect or EKG_ConnectLong function. RODM creates an object of the EKG_User class that represents the RECOVER application.
2. If the DisplayStatus field does not have a notify subfield, the RECOVER application creates one using the EKG_CreateSubfield function. The subfield is created on the same class as the DisplayStatus field.

3. Methods must be installed before they can be used. You install a method by placing it in the specified library for RODM and by creating an object of the EKG_Method class that represents the method. "Installing and Freeing Methods" on page 356 describes how to install a method.

   In this example, one of the notification methods supplied with RODM is being used. The EKGNTHD notification method is triggered when the value of the field falls outside the specified thresholds. The thresholds are passed to EKGNTHD in the Long_lived_parm that is specified on the EKG_AddNotifySubscription function.

   The EKGNTHD notification method is described in "RODM Notification Methods" on page 481. If the methods supplied with the NetView program do not meet your needs, you can write your own notification method.
4. Create a notification queue and its associated event control block (ECB). You need only one notification queue for all objects that are to notify your user application RECOVER. A notification queue is associated with a single user application, but a user application can have many notification queues. The notification queue is an object of the EKG_NotificationQueue class.
   a. RECOVER creates an object of the EKG_NotificationQueue class using the EKG_CreateObject function. Notification queue names must be unique within a user application. For this example, specify the queue name RECOVQ as the object name in the entity access block of this transaction. RODM concatenates the User_appl_ID of the user application with the queue name specified to create the MyName field of the EKG_NotificationQueue object; in this example, MyName is set to RECOVER.RECOVQ. RODM links the EKG_UsedBy field of the EKG_NotificationQueue object to the EKG_Uses_Q field of the EKG_User object that represents the user application.
   b. Set the value of the ECB to 0 (zero).
   c. Set the EKG_ECBAddress field to the address of the ECB you use for this queue. RECOVER uses the EKG_ChangeField function to set the value of this field. The ECB is created in the address space of the user application. Many notification queues can use the same ECB.
   d. Set the EKG_Status field of the notification queue object you created in Step 4a to 1 (active). RECOVER uses the EKG_ChangeField function to set the value of this field.

   You do not have to associate an ECB with a notification queue. Your application can simply query the notification queue from time to time to see if any notifications have been added. However, this is not as useful as the asynchronous notification provided by the ECB.
5. The last step in setup is to subscribe to the field for each object. The RECOVER application issues the EKG_AddNotifySubscription function. This function puts the notification method name EKGNTHD, the method parameters, the notification queue name RECOVQ, and the user application ID of RECOVER in the notify subfield. Specify the parameters of this function call as follows:

   Entity_access_info_ptr
   A pointer to the entity access block that specifies the class and object for which you are creating the notification subscription.

   Field_access_info_ptr
   A pointer to the field access block that specifies the DisplayStatus field.

   User_appl_ID
   Set this to the null value. RODM fills in the value that corresponds to the RECOVER application that is issuing this function call.

Notification_queue
Specify the name of the notification queue you created in Step 4 on page 320. For this example, enter the name as RECOVQ, not as RECOVER.RECOVQ. The User_appl_ID part of the name is supplied by RODM.

User_word
You can leave this optional field blank.

Notify_method
Specify the object ID of the object of the EKG_Method class that represents the notification method EKGNTHD. If this is an installed method, this is the value that was returned in the Object_ID field of the entity access block when you created the object for EKGNTHD in Step 3 on page 320. If this is a pre-installed method, the object ID is obtained by querying the MyName field of the method.

Long_lived_parm
Specify the parameters that are to be passed to EKGNTHD when it is triggered. This is where you specify the thresholds that cause this method to be triggered. These parameters are described in "RODM Notification Methods" on page 481.

Repeat Step 5 on page 320 once for each field you subscribe to. The setup for the notification process is complete when the EKG_AddNotifySubscription function has run successfully for each object.

Although this example describes notifying one user application when a field changes, any number of applications can be notified. The notify subfield can contain a list of notification subscriptions. Repeat the entire notification process for each user application that is to be notified.

Instead of creating a notification subscription for each object, you can create a notification subscription for a class. RODM triggers a notification method defined for a field of a class when that field is changed on any object of the class. The notification method needs to use the Where Am I (2007) function to identify the particular object that caused the method to be triggered.

Wait

After you have set up the notification process, your application suspends processing until RODM notifies it of a change. Calling EKGWAIT enables your application to wait until a specified ECB or any ECB in a list of ECBs is posted by RODM.

EKGWAIT is an interface module that provides the WAIT facilities. Your application calls EKGWAIT with a parameter list containing ECB information.

For this example, RECOVER issues a call to EKGWAIT specifying an ECB. When the ECB is posted, EKGWAIT returns control to RECOVER. RECOVER then processes the notification.

Calling EKGWAIT

RODM supplies sample code that shows how to call EKGWAIT. The PL/I sample is EKG5WAIT and the C sample is EKG6WAIT.

Only user applications can use EKGWAIT. The format of the call to EKGWAIT is as follows:

```
EKGWAIT(Num_ECBs, ECB_Array, Return_code, Reason_code)
```

The following is an explanation of each parameter in the list of parameters specified in a call to the EKGWAIT interface module. This parameter list is also used by EKGWAIT to pass information back to the user application when EKGWAIT returns control.

Parameter Name	Description
Num_ECBs (In)	A 2-byte Smallint which specifies the number of ECBs in the event list.
ECB_Array (In)	An array of Pointers where each pointer contains the address of an ECB.
Return_code (Out)	A 4-byte Integer containing the return code.
Reason_code (Out)	A 4-byte Integer containing the reason code. If Return_code is 0, then this field contains the index into ECB_Array for which the ECB was posted.

PL/I Coding Example

Figure 70 is an example for calling EKGWAIT from a PL/I user application:

```
%Include SYSLIB(EKG1IEEP); /* EKGWAIT declaration               */

%Dcl n fixed;
%n=3;                      /* Arbitrary max number of ECBs in list*/
```

Figure 70. PL/I Coding Example (Part 1 of 4)

```
Dcl
  ECB_Array(n) Pointer,    /* Array of ECB pointers             */
  Return_code  fixed bin(31), /* Return code from EKGWAIT        */
  Reason_code  fixed bin(31), /* Reason code from EKGWAIT        */
  Num_ECBs     fixed bin(15), /* Number of ECBs                  */
  POSTED_ECB   fixed bin(31) based,  /* ECB which was posted     */
```

Figure 70. PL/I Coding Example (Part 2 of 4)

```
  ECB1         fixed bin(31),  /* First ECB                      */
  ECB2         fixed bin(31),  /* Second ECB                     */
  ECBn         fixed bin(31);  /* Nth ECB                        */

ECB_Array(1)=addr(ECB1);   /* Address of ECB1                   */
ECB_Array(2)=addr(ECB2);   /* Address of ECB1                   */
ECB_Array(n)=addr(ECBn);   /* Address of ECBn                   */
Num_ECBs=n;                /* Number of ECBs in list            */
```

Figure 70. PL/I Coding Example (Part 3 of 4)

```
CALL EKGWAIT(Num_ECBs,ECB_Array,Return_code,Reason_code); /* Wait
                              on list of ECBs                      */

If Return_code = 0 then    /* No errors in WAIT                    */
  Do;
    /***********************************************************/
    /* ECB_Array(Reason_code) is a pointer to the posted ECB.    */
    /***********************************************************/
    ECB_Array(Reason_code)->POSTED_ECB=0;
  End;
```

Figure 70. PL/I Coding Example (Part 4 of 4)

C Coding Example

Figure 71 is an example for calling EKGWAIT from a C user application:

```
#include "EKG3CEEP.H"      /* EKGWAIT declaration                  */
#define n 3                /* Arbitrary max number of ECBs in list*/

int*    ECB_Array[n];      /* Array of ECB pointers                */
int     Return_code;       /* Return code from EKGWAIT             */
int     Reason_code;       /* Reason code from EKGWAIT             */
int     Num_ECBs;          /* Number of ECBs                       */
```

Figure 71. C Coding Example (Part 1 of 3)

```
int     ECB1;              /* First ECB                            */
int     ECB2;              /* Second ECB                           */
int     ECBn;              /* Nth ECB                              */

ECB_Array[0]=&ECB1;        /* Address of ECB1                      */
ECB_Array[1]=&ECB2;        /* Address of ECB2                      */
ECB_Array[n-1]=&ECBn;      /* Address of ECBn                      */
Num_ECBs=n;                /* Number of ECBs in list               */
```

Figure 71. C Coding Example (Part 2 of 3)

```
EKGWAIT(&Num_ECBs,ECB_Array,&Return_code,&Reason_code); /* Wait
                              on list of ECBs                      */

if (Return_code == 0) {    /* No errors in WAIT                    */
  /************************************************************/
  /* ECB_Array[Reason_code-1] is a pointer to the posted ECB.    */
  /************************************************************/
  *ECB_Array[Reason_code-1]=0;
}
```

Figure 71. C Coding Example (Part 3 of 3)

EKGWAIT Usage Notes

The purpose of the ECB_Array is to contain the ECB addresses being set to the EKG_ECBAddress fields in the EKG_NotificationQueue objects. However, always include in the ECB_Array the Stop_ECB identified to RODM at connect time. This can prevent a user from waiting indefinitely, if RODM is stopped.

On a successful return, where Return_code equals 0, the Reason_code is set to an integer value indicating the index (1 to N) within the ECB_Array of the ECB that was posted. Clear the ECB being posted immediately after a successful return from this function call.

An ECB address of 0 passed to this function call causes an immediate return with a warning return code. But, an ECB address that is not valid can cause an abend or an indefinite wait.

Notification

When the field to which your application has subscribed changes value, its notification method is triggered. In this example, if the DisplayStatus field of object NETRES3 changes, RODM triggers notification method EKGNTHD. EKGNTHD then compares the new value of DisplayStatus to the thresholds you specified in the Long_lived_parm parameter of the EKG_AddNotifySubscription function.

If the new value exceeds the specified thresholds, EKGNTHD places a notification block on notification queue RECOVQ and RODM posts the ECB for the RECOVER application. Notification methods use the EKG_SendNotification function to place the notification block on the queue. When the ECB is posted, EKGWAIT returns control to RECOVER.

RODM posts the ECB for a notification queue when all of the following conditions are met:
- The notification queue exists.
- A notification block is added to a previously empty queue.
- The ECB pointer for the queue points to a valid ECB.

After RODM posts an ECB for a particular notification queue, RODM does not post the ECB for that queue again until the queue has been completely drained and a new block added or until the EKG_ECBAddress field in the notification queue object is changed.

If you reconnect to RODM and notification subscriptions and notification queue objects for your user application still exist, the ECB cannot be posted. You must reset the EKG_ECBAddress field in each notification queue object to a current ECB address to enable RODM to post the ECBs.

The remaining processing is done by your application.

1. The user application clears the ECB by setting it equal to 0. This enables RODM to post additional notifications.
2. The application gets the notification blocks from the notification queue using the EKG_QueryNotifyQueue function. The notification block contains a Notification_block_type field that indicates the type of event that caused the notification.

   One block is removed for each function call. The response block for this function indicates the number of notification blocks on the queue in the Notification_queue_count parameter. The application processes each block on the notification queue. The EKG_QueryNotifyQueue function must be issued from the address space that the user application connected from.

   In our example, RECOVER calls the EKG_QueryNotifyQueue function once, specifying the notification queue name RECOVQ.
3. The application uses the notification block information returned in the response block to initiate its processing. In our example, RECOVER uses the Object_ID

parameter to identify the resource that changed its DisplayStatus. RECOVER can use the EKG_QueryField function to get the new DisplayStatus value from the RODM data cache. RECOVER then issues the appropriate commands to reactivate the failing resource NETRES3.

4. When it finishes processing the notification queue, the user application calls EKGWAIT to wait until the next notification takes place.

Clean Up

Notification processing uses system resources including memory and processor cycles. When a notification is no longer needed for an object, delete the notification.

There are two ways to delete a notification:
- Delete the notification queue.
- Delete the notification subscription.

If you want to delete all notification subscriptions that use a notification queue, delete the object of the EKG_NotificationQueue class that represents the notification queue. Use the EKG_DeleteObject function. RODM deletes the notification queue and all notification subscriptions that specify that queue. RODM also deletes any notification blocks that are still on the notification queue.

If you have more than one notification subscription that uses a notification queue, and you do not want to delete all of the subscriptions, use the EKG_DeleteNotifySubscription function for each subscription you want to delete.

In this example, you want to shut down NETRES2 for maintenance. To prevent RECOVER from trying to restart NETRES2, issue the EKG_DeleteNotifySubscription function and specify NETRES2 with the Entity_access_info_ptr parameter. The other notification subscriptions are not affected.

RODM deletes the links between the EKG_User object and the EKG_NotificationQueue object when you delete a notification queue. When a user application disconnects from RODM or ends without disconnecting, RODM can delete the notification queues and subscriptions associated with the user application. The EKG_StopMode field in the EKG_User object that represents the object specifies what action RODM takes. See “EKG_User Class” on page 201 for information about the EKG_StopMode field.

Asynchronous Error Notification

Your user applications can be notified about asynchronous errors and checkpoints by subscribing to fields in RODM system-defined objects. Subscribe to the EKG_LastAsyncError field in the EKG_System object to be notified about asynchronous errors that occur during the execution of asynchronous API requests, asynchronous methods, or RODM internal processing. Subscribe to the EKG_LastAsyncError field in the EKG_User object for a user application to receive notifications only about errors in transactions initiated by that user application.

The EKGNOTF method that is supplied with the NetView program can be used for these notification subscriptions. See “RODM Notification Methods” on page 481 for a description of this method. The log record is assigned to the EKG_LastAsyncError field. This log record information is placed in the user_data field of notification queue blocks created because of a subscription to the

EKG_LastAsyncError field. User application programs can obtain this information by calling the EKG_QueryNotifyQueue function.

When an error occurs, the specified notification method is triggered. All user applications that subscribed to the EKG_LastAsyncError field are notified.

The EKG_LastAsyncError field is changed and any notification methods are triggered when an error message is written to the log as the result of a method running asynchronously to a user application. RODM writes error log entries when a method sets its return code to a value greater than or equal to either the user's EKG_LogLevel or the Log_level customization parameter specified for an asynchronous method.

Object Deletion Notification

If your application needs to be notified when certain objects are deleted, the application can subscribe to those objects with an *object-deletion subscription*. If the object is deleted, RODM places a notification block on a notification queue and posts the ECB for the application.

For the format of the notification block, refer to the description of the EKG_QueryNotifyQueue response block on page 421.

The four steps of the RODM notification process (setup, wait, notification, and cleanup) apply to object-deletion notification, with some differences.

Setup for Object-Deletion Notification

For object-deletion notification, setup differs from the normal RODM notification process described on page 319.

1. Connect to RODM. Do not create a notify subfield, install a notification method, or subscribe to the field.
2. Create a notification queue and its ECB, as described in Step 4 on page 320.
3. The last step in setup is to subscribe to the object. Your application issues the EKG_AddObjDelSubs function to create an object-deletion subscription for the object. This function specifies an object, a user application, and a notification queue. If the object is deleted, RODM places a notification block on the specified notification queue and posts the ECB for the user application. Specify the parameters of this function call as follows:

   Entity_access_info_ptr
   A pointer to the entity access block that specifies the class and object for which you are creating the object-deletion subscription

   User_appl_ID
   Set this to the null value. RODM fills in the value that corresponds to the user application that is issuing this function call.

   Notification_queue
   Specify the name of the notification queue you created in Step 4 on page 320. The User_appl_ID part of the name is supplied by RODM.

   User_word
   You can leave this optional field blank.

   Long_lived_parm
   When the object is deleted, RODM puts the value of this optional parameter in the user_area parameter of the response block

Repeat Step 3 on page 326 once for each object you subscribe to. The setup for the deletion-notification process is complete when the EKG_AddObjDelSubs function has run successfully for each object.

Wait for Object-Deletion Notification

This step is the same as “Wait” on page 321.

Notification for Object-Deletion Notification

When the object to which your application has subscribed is deleted, RODM places a notification block on the application's notification queue and posts the ECB for the application.

The rest of this step is the same as described in “Notification” on page 324.

Cleanup for Object-Deletion Notification

To delete an object-deletion subscription, use the EKG_DelObjDelSubs function described in “EKG_DelObjDelSubs — Delete Object Deletion Subscription” on page 397.

Connecting to RODM

Before you can run any user API functions, you must connect to RODM using the EKG_Connect or EKG_ConnectLong API function. When you connect to RODM, specify an access block containing your user application ID and the name of the RODM to which you want to connect. RODM sets the Sign_on_token field in your access block after a successful connect. This value represents your connection to RODM and must not be changed. If RODM detects that the value in the Sign_on_token field in your access block is not valid when you request an API function other than EKG_Connect or EKG_ConnectLong, RODM rejects your API function request and returns the appropriate reason code.

RODM permits only one connection for each application user ID. Attempts to connect with a user application ID that is already connected fail, and the appropriate reason code is returned.

Applications that are cancelled by the operator or are otherwise abended while they are connected to RODM, are disconnected.

If you chose to disconnect from RODM without purging the subscription notification queue, upon subsequent connection, all ECB addresses associated with the notification subscriptions must be reset to point to the new address space ID.

Your application cannot connect to RODM if your application is running in cross-memory mode. RODM checks for this condition and returns an error reason code.

After successfully connecting to RODM, RODM creates a user object in the EKG_User class representing your user application. This user object contains your application environment and is preserved until your application disconnects. While you can have multiple concurrent API requests executing in RODM for the same user application ID, each request uses and possibly modifies the information in the user object.

For more information about connecting to RODM, see "EKG_Connect — Connect to RODM" on page 383 or "EKG_ConnectLong — Connect to RODM" on page 384.

Disconnecting from RODM

When an application completes all of its tasks and has no further API function requests to perform, it disconnects using the RODM EKG_Disconnect API function. After disconnecting, the sign-on-token is no longer valid. RODM returns an error reason code if your application subsequently attempts to run another API function request, unless the API function request is an EKG_Connect or EKG_ConnectLong function request.

When your application disconnects, RODM performs clean-up of notification queues, depending on the value of EKG_StopMode in your user object. RODM might purge all of your user application ID-owned notification queues, queue elements, and subscriptions, purge only notification queue elements and retain all notification queues and subscriptions, or purge nothing and retain all notification queues, queue elements, and subscriptions. If RODM purges all notification queues, queue elements, and subscriptions, RODM also purges your user object.

Note: Applications that end while they are connected to RODM, are disconnected.

For more information about disconnecting from RODM, see "EKG_Disconnect — Disconnect from RODM" on page 399.

Chapter 12. Topology Object Correlation

This chapter describes the object correlation function. It includes the following information:
- Enabling the correlation function
- Correlation concepts
- Including your objects in correlation
- Correlating SNA topology manager and MultiSystem Manager objects
- Customizing the correlation function

Using correlated aggregate objects, a NetView management console (NetView management console) operator can perform the following tasks:
- Navigate between correlated resources
- View consolidated data about the correlated resources
- Monitor aggregate status of the correlated resources

For more information about using correlated objects, see the *IBM Tivoli NetView for z/OS User's Guide: NetView Management Console*.

Enabling the Correlation Function

Object correlation is enabled by loading the FLCSDM8 input file into RODM. To load FLCSDM8, remove the asterisk (*) from the following line in the CNMSJH12 job:

```
//* DD DSN=NETVIEW.V6R1M0.CNMSAMP(FLCSDM8),DISP=SHR  <-CORRELATE SAMPL
```

Correlation occurs when an application sets a valid value in a field of a RODM object that is enabled for correlation. Objects are enabled for correlation by loading the FLCSDM8 file. Because MultiSystem Manager and SNA topology manager automatically set the value of these fields, which causes correlation to occur, and the views are displayed on a NetView management console.

Enabling SNA Topology Manager Object Correlation

To enable correlation for resources managed by SNA topology manager, edit initialization file FLBSYSD and change the value of the following statement to `YES`:

```
  WRITE_CORRELATABLE_FIELDS=NO
```

SNA correlation occurs on PU resources. PU resources are excluded from TOPOSNA commands that do not include the *LOCAL* parameter. Use the *LOCAL* parameter on any TOPOSNA command issued to resources you want included in correlation.

The resources on which SNA topology manager provides a correlator value are PU 2.1 workstations. If SNA topology manager does not monitor any PU 2.1 workstations, none of your SNA resources can be correlated. If you know the LAN MAC address of your SNA resources, you can include them in correlation. Refer to "Extending Correlation of Objects Created by MultiSystem Manager and SNA Topology Manager" on page 335.

Enabling GMFHS Object Correlation

To enable correlation for GMFHS resources, set a value on one or more of the following fields on the GMFHS_Managed_Real_Objects_Class:
- aIndMACAddress
- Correlater
- iPAddress

The FLCSDM8 RODM load input file creates these fields on the GMFHS_Managed_Real_Objects_Class when it is loaded.

Correlation Concepts

The correlation function is triggered when the value of a field on which the FLCMCON method is installed changes. Correlation automatically associates resources managed by different agents. The correlation function runs dynamically and is implemented using RODM methods. Correlated objects have a common correlater value, and a *correlated aggregate object* is used to represent these objects. When correlation is by IP address or MAC address, the correlated aggregate object is represented in RODM using aggregateSystem class objects. When correlation is by a value in the Correlater field, the correlated aggregate object is represented in RODM using GMFHS_Aggregate_Object_Class objects.

A *correlated object* is an object of any correlation-enabled class that has a value in one of the following fields:
- aIndMACAddress
- iPAddress
- Correlater

This value is the *correlater value*.

The term *cross-correlation* is used to describe the relationship between two or more real objects that have an identical correlater value. For example, assume the following conditions:
- The correlation function is enabled.
- A system can be part of an internet and part of a sysplex.
- The resources are represented by objects in RODM, and, for each object, the iPAddress field has the value 9.37.65.43.

Because these two objects have identical values for the same field, the objects are cross-correlated.

Correlation Methods

The following RODM methods implement the correlation function.

FLCMCONI Method

The FLCMCONI method is an initialization method that loads the FLCMCON method on classes that support correlation. FLCMCONI is used instead of the DUIFSTRC RODM load input file because FLCMCONI passes parameters to the FLCMCON method.

FLCMCON Method

The FLCMCON method is a notification method that is loaded on certain fields of classes for which correlation is enabled. To determine which classes are enabled for correlation and the fields on which the FLCMCON method is loaded, browse the FLCSDM8 RODM load file. FLCMCON runs FLCMCOR.

FLCMCOR Method

The FLCMCOR method is an object-independent method that creates and updates correlated aggregate objects.

The load and customization of these methods is accomplished using the FLCSDM8 RODM load file. For more information, refer to “Enabling the Correlation Function” on page 329 and “Customizing the Correlation Function” on page 336.

Objects Enabled for Correlation

Loading the FLCSDM8 sample automatically enables correlation for resources that are managed by MultiSystem Manager, SNA topology manager, and customer applications that use the GMFHS data model. To determine which classes are automatically enabled, browse the FLCSDM8 RODM load file. All classes on which the FLCMCON method is loaded are automatically enabled.

For example, the following code enables correlation by IP address on objects of the OperatingSystem class, which are created by the MultiSystem Manager IBM Tivoli Network Manager feature:

```
OP FLCMCONI INVOKED_WITH (SELFDEFINING)
(
 (OBJECTID) EKG_Method.FLCMCON
 (CLASSID) OperatingSystem
 (FIELDID) '1.3.18.0.0.3330'.'iPAddress'
 (CLASSID) '1.3.18.0.0.6464'
 (CLASSID) 'GMFHS_Managed_Real_Objects_Class'
);
```

Types of Correlation

There are two types of correlation:
- Network address correlation
- Free-form correlation

Network Address Correlation

Network address correlation is performed using LAN media access control (MAC) or internet protocol (IP) addresses.

To include objects in correlation based on a network address, set a value on one of the following fields.
- aIndMACAddress (1.3.18.0.0.5263)
- iPAddress

Correlation uses 12-character MAC addresses (for example, 10004BF00943). A 14-character MAC address is supported, but the last 2 characters (the link service access point) are removed.

A valid IP address consists of numbers and at least two periods (.) to delimit the numbers.

Free-Form Correlation

Free-form correlation is performed using a free-form string value. Correlation on a free-form string creates a correlated object with a display name that matches the string value.

To include objects in free-form correlation, set the string as the value of the Correlater field. Example valid values include:
- Accounting

- PresidentsOffice
- Building201
- London

You can also enter a multipart string value in the Correlater field. Entering a multipart string enables you to link the correlated object to a hierarchy of correlated aggregate objects as shown in Figure 72:

.

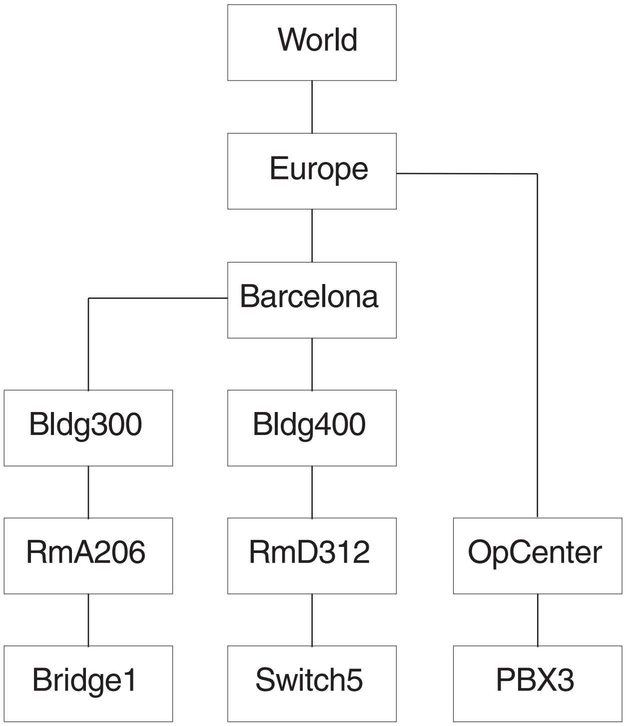


Figure 72. Correlate Objects on Multiple Free-Form Values

To enable correlation to create the objects in Figure 72, set the following values:
- Bridge1 Correlator = 'RmA206 Bldg300 Barcelona Europe'
- Switch5 Correlator = 'RmD312 Bldg400 Barcelona'
- PBX3 Correlator = 'OpCenter Europe World'

This enables you to create or locate a hierarchy of views, based upon organizational or geographic structure, with one command. As with single value free-form correlation, for each string value in a multipart string, a correlated aggregate object is located or created. If parent relationships do not already exist between the different correlated aggregate objects identified in the multipart string, they are created.

Commas or blank spaces can be used to delimit a multi-part string. For example, if you enter a string value of `Jane Doe`, correlation locates or creates two objects – `Jane` and `Doe`.

All of the characters that are supported by the RODM CharVar data type are supported. You can use an underscore character (_) between string values that you want to be treated as one correlated aggregate object (for example, Margaret_Thatcher).

Free-form correlation creates correlated aggregate objects of class GMFHS_Aggregate_Objects_Class. This enables correlation to locate and link to aggregate objects created by BLDVIEWS scripts. BLDVIEWS typically includes objects in views if those objects have a consistent naming scheme (for example, `CPNRTR2` and `CPNHST14`), it builds views from the top down. Multiple free-form correlation does not require objects to have a similarity in object naming; it builds views from the bottom up. Using BLDVIEWS and topology correlation together, you can build custom views that match your enterprise.

Correlated Aggregate Object Classes and Names

Correlated aggregate objects are named using the correlater field value of the first object for which a correlation was found. Valid values include the following values:
- LAN MAC address (for example, 40000A17D006)
- TCP/IP address (for example, 9.37.65.43)
- Free-form correlater (for example, Accounting)

Correlated aggregate objects identified through network address correlation are created on class aggregateSystem. These objects have a multi-part OSI distinguished name that includes a MAC address or TCP/IP address as the last element. For example, `1.3.18.0.0.3519=MultiSys,1.3.18.0.0.6467=40000A17D006`. Correlated aggregate objects identified through free-form correlation are created on class GMFHS_Aggregate_Objects_Class. These objects are named by a free-form correlater value, with no other prefix or suffix (for example, `Accounting`).

For more information about the object names, refer to the aggregateSystem class description in the *IBM Tivoli NetView for z/OS Data Model Reference*.

Object names are defined by the value of the object MyName field. The name used to label these objects on the NetView management console can be either the MyName field value or a user-defined value. See “Correlated Aggregate Object Display Labels” for more information about display labels.

Correlated Object Relationships

Resources with identical Correlater field values are represented by one correlated aggregate object; this includes resources that are managed by different topology agents.

Relationships are created between correlated resources and correlated aggregate objects using links. Links enable more detail, configuration parent, and configuration child navigation between objects and status aggregation.

Correlated Aggregate Object Display Labels

Correlated aggregate objects are displayed using the following symbol:

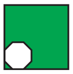


Figure 73. Aggregate Resource Symbol

Correlated aggregate object labels are determined by the first value for which a correlation was found:

Table 34. Correlated Aggregate Object Labels

First Correlation Value	Resource Label
MAC address	LAN workstation aggregate
IP address	IP system aggregate
Correlator field value	Open system aggregate

Correlated Aggregate Object Field Values

The correlation function is triggered when the value of a field on which the FLCMCON method is installed changes. The FLCMCON method triggers the FLCMCOR method. The FLCMCOR method queries the values of the following fields of real objects:

- aIndMACAddress
- segmentNumber
- aUniversallyAdministeredAddress
- adapters
- iPAddress
- netAddress
- sysLocation
- adjacentLinkStationAddress2
- linkName
- ipHostName
- Correlater

The value of these fields is compared to the values of the corresponding fields of the correlated aggregate object. When a value exists on a real object but not on the correlated aggregate object, the value is copied from the real object to both the corresponding field and the DisplayResourceOtherData field of the correlated aggregate object.

Notes:

1. When a value is assigned to a field on the correlated aggregate object, subsequent correlations cannot change the value of the field.
2. If you write an application that uses the value of these fields, query the individual fields rather than parsing the DisplayResourceOtherData field. For more information about these fields, see the *IBM Tivoli NetView for z/OS Data Model Reference*.

Use the NetView management console to display data contained in the DisplayResourcOtherData field. This information is displayed in the NetView management console **Data1** field.

The value in the DisplayResourceOtherData field is not always provided by the correlation function. Information that is provided by the correlation function is identified by a lowercase *a* in the word *address*.

When you set a correlater value in RODMVIEW or the RODMVIEW function of Visual BLDVIEWS, the resultant correlation is only displayed until the next time that RODM is recycled. That can be days or months, depending upon how you run your enterprise. When you set correlater values in a CLIST or BLDVIEWS script, you can rerun that CLIST or BLDVIEWS script, and restore your customized correlations, after RODM is recycled. If your customization includes free-form correlation, there is an easier way to set correlater values. Visual BLDVIEWS (VBV) provides pop-up menus that enable you to select one or more correlated objects, set a value in the Correlater field of those objects and save and run those settings to the host as a BLDVIEWS script. With this method, after RODM is recycled, you can rerun the BLDVIEWS script from the mainframe or the VBV workstation to restore your custom correlations. For more specifics on using Visual BLDVIEWS or BLDVIEWS with topology correlation, see the *IBM Tivoli NetView for z/OS User's Guide: NetView Management Console*.

Using Correlation for Objects You Create

Objects discovered by MultiSystem Manager agents and SNA topology manager logicalLink class (PU) objects are automatically correlated. You can extend correlation to include MultiSystem Manager open data model, GMFHS, and additional SNA topology manager objects. For more information about SNA topology manager, see "Correlating SNA Topology Manager Objects" on page 336.

To include objects that you have created in correlation, perform the following tasks:

- Choose a class to use. You can choose any of the classes enabled for correlation in the FLCSDM8 file. Enabling objects of the open data model requires less setup, and the FLCSOX01 sample file is provided as an example. If your application already creates GMFHS managed resource objects, it is easier to continue using the GMFHS objects.
- Set a value on one or more of the following data fields for each object you want to include in correlation:
  - aIndMACAddress (for example, 1.3.18.0.0.5263)
  - iPAddress
  - Correlater

  The aIndMACAddress and iPAddress fields support correlation based on network addresses and the Correlater field supports free-form correlation.

  You can set field values on the objects using RODMVIEW, CLIST, or BLDVIEWS script. The FLCSOX01 sample file provides an example of a REXX CLIST. This CLIST demonstrates that if your application already creates RODM objects, you can include those objects in correlation by adding just one additional line of code.

Extending Correlation of Objects Created by MultiSystem Manager and SNA Topology Manager

MultiSystem Manager objects and SNA topology manager logicalLink class (PU) objects are automatically correlated. If you have correlatable information about objects that is not discovered by MultiSystem Manager or SNA topology manager agents, you can extend correlation to these objects. To extend the correlation of these objects, perform the following tasks:

- Determine the name of the object
- Set a value on the aIndMACAddress, iPAddress, or Correlater field of the object
- Perform any data model-specific tasks necessary to extend the objects. See "Correlating MultiSystem Manager Objects" on page 336 and "Correlating SNA Topology Manager Objects" on page 336 for more information.

Remember that SNA topology manager and MultiSystem Manager dynamically create, delete, and update objects. If you add field values and then subsequently reacquire topology (for example, by issuing a TOPOSNA or GETTOPO command) or cold start RODM, the values you added can be lost. Because of this, use a CLIST or BLDVIEWS script to reset correlatable field values each time topology is reacquired.

How to Determine Object Names

Object names are defined by the value of the object's MyName field in RODM. Remember that the name of an object that is displayed in a view is usually a simplified version of the object's name in RODM. The name that is displayed in a view usually is not suitable for the object name in RODM. Use RODMVIEW or Visual BLDVIEWS to determine the MyName field values of existing objects.

For a description and syntax of MyName fields, refer to the *IBM Tivoli NetView for z/OS Data Model Reference*.

Correlating MultiSystem Manager Objects

If the FLCMCON method is loaded directly on the field of an object you want to correlate, set a value on the field. To determine which fields have FLCMCON loaded, browse the FLCSDM8 RODM load file. This is all that is required for most MultiSystem Manager objects.

If you want to extend additional network address correlation to objects created by MultiSystem Manager that have the FLCMCON method loaded on the memberOf field, create a link on the memberOf field.

For example, if you want to add MAC address correlation to a Monitor class object that is already correlated on a IP address, create a link on the memberOf field of that object. The link can be to any other object, and the process of creating the link is the same as creating other links in RODM.

Note: Free-form correlation using the Correlater field never requires creation of a link in RODM.

Correlating SNA Topology Manager Objects

SNA topology manager logicalLink class objects are automatically included in correlation because the value of the adjacentLinkStationAddress field can contain the MAC address of the PU. The correlation function determines if this field contains a MAC address. If it does, it treats this field like the aIndMACAddress field.

Because SNA Topology Manager does not discover TCP/IP addresses, SNA PUs are not correlated to resources on which an IP address is discovered unless the MultiSystem Manager IBM Tivoli Network Manager agent also discovers both an IP address and a MAC address on that resource. One example of a resource that has a MAC address and an IP address is a Windows workstation that has an SNA PU and a LAN adapter with IP support. SNA topology manager discovers MAC addresses only on Windows workstations.

To enable IP address correlation for SNA resources, manually set the address on the iPAddress field on an object that is enabled in the FLCSDM8 file. Correlation can then automatically correlate the SNA object to other resources with IP addresses.

Customizing the Correlation Function

All customization of the correlation function is accomplished using the FLCSDM8 RODM load file. After customization, FLCSDM8 must be loaded into RODM. If FLCSDM8 was previously loaded, cold start RODM. If FLCSDM8 was not previously loaded and you have already loaded the other SNA topology manager and MultiSystem Manager load files, load FLCSDM8 without cold starting RODM. You must use the EKGLLOAD sample file to load FLCSDM8. Ensure that you specify the data set and file (FLCSDM8) in the EKGIN3 step.

There are two ways to customize the correlation function:
- Change the display name priority
- Disable correlation for specific classes

Changing the Display Name Priority

You can change the type of display name for a correlated aggregate object, when that object is correlated by network address. When the object is correlated by free-form correlater, the display name is taken from the Correlater field. In that case, the type of display name cannot by changed.

The fields shown in Figure 74 are used to determine the correlated aggregate object display name. To determine which correlated aggregate object field is used to label an object, the correlate function uses a prioritized list of those fields in the FLCSDM8 file. The correlate function queries each field of the aggregate object in the order listed until a non-null value is found; this value is used to label the object. Table 35 lists the default priority used and the agents for which the priorities are used.

Table 35. Correlated Aggregate Object Default Display Name Priority

Priority	Name Type	Discovered By
5	SNA node name	SNATM
6	LAN MAC address	SNATM

You can determine which label is displayed by customizing the order in which the fields are listed.

For example, using the default priority shown in Figure 74, a workstation that contains a MultiSystem Manager IBM Tivoli Network Manager agent is not named using the computer name because the Internet does not define a computer name for managed resources. In this case, the workstation object is labeled using its internet protocol host name.

```
(
 (FIELDID) '1.3.18.0.0.6464'.
           '1.3.18.0.0.3315.2.7.202'       -- computerName
 (FIELDID) '1.3.18.0.0.6464'.
           'ipHostName'                    -- ipHostName
 (FIELDID) '1.3.18.0.0.6464'.
           'iPAddress'                     -- iPAddress
 (FIELDID) '1.3.18.0.0.6464'.
           '1.3.18.0.0.2032'               -- snaNodeName
 (FIELDID) '1.3.18.0.0.6464'.
           '1.3.18.0.0.5263'               -- aIndMACAddress
```

Figure 74. Default Display Name Priority

Now, assume that you have customized the FLCSDM8 file to put TCP/IP address (priority 3) before IP host name (priority 2) as shown in Figure 75 on page 338. In this case, the TCP/IP address is used to label the workstation object because the MultiSystem Manager IBM Tivoli Network Manager agent provides both an IP host name and an IP address, and the IP address name is listed first.

```
(
 (FIELDID) '1.3.18.0.0.6464'.
           '1.3.18.0.0.3315.2.7.202'      --computerName
 (FIELDID) '1.3.18.0.0.6464'.
           'iPAddress'                    -- iPAddress
 (FIELDID) '1.3.18.0.0.6464'.
           'ipHostName'                   -- ipHostName
  (FIELDID) '1.3.18.0.0.6464'.
           '1.3.18.0.0.2032'              -- snaNodeName
 (FIELDID) '1.3.18.0.0.6464'.
           '1.3.18.0.0.5263'              -- aIndMACAddress
 );
```

Figure 75. Customized Display Name Priority

Disabling Correlation for Specific Resources

Correlation is enabled for objects of the classes on which the FLCMCON method is explicitly loaded in the FLCSDM8 file. If you do not want topology correlation to run for a class of managed resource objects, comment out the method load statement that loads the FLCMCON file on the class.

The method load statements are grouped in the FLCSDM8 file by topology agent. To determine which method load statement to comment out:

1. Determine the object display label for a correlated object.
2. Determine the RODM class that the label represents. Use RODMVIEW to determine the class, or refer to the class listings in the *IBM Tivoli NetView for z/OS Data Model Reference* and match the label with the DisplayResourceType values listed.

Note: Using the FLCSDM8 file as shipped, the FLCMCON method is loaded on all of the classes which MultiSystem Manager and SNA topology manager can automatically correlate upon. It also enables correlation for additional classes you might want to extend correlation to. Because the memory and CPU usage for loading a method on an unused class is insignificant, it is not necessary to comment out the method load statements for unused classes.

Chapter 13. Writing RODM Methods

This chapter describes RODM methods. Methods enable you to maintain data in RODM and to automate functions related to the resources represented by objects in RODM. Methods are small executable programs that reside in the RODM address space. They can be run by user applications, by changes to fields in RODM, by other methods, and at RODM initialization. Methods are classified by the way they are run.

The NetView program supplies several general-purpose methods that might meet some or all of your needs. Before you spend time writing your own methods, review the methods that are supplied with the NetView program as described in "Supplied Methods" on page 480 for applicability.

You must install each method, including methods that are supplied with the NetView program, before you can use it. Each method is represented in RODM by an object of the EKG_Method class. These objects are created as part of installing the method. Methods can be dynamically installed, deleted, and refreshed.

Tasks Best Performed with Methods

This section describes which tasks are best performed with methods.

Use a method to do the following:

- Perform multiple actions on more than one object or class in the RODM data cache.

  You can write an object-independent method to process numerous API functions against a set of one or more objects or classes. See "Object-Independent Methods" on page 340 for more information about object-independent methods.
- Load structures and objects at RODM initialization.

  The RODM program supports a special form of the object-independent method called the initialization method. The initialization method can be specified at RODM start up to provide initialization functions. It can load a class hierarchy structure and then create objects of the classes. This function enables the RODM data cache to be established and ready for work following a RODM start up.

  The RODM load function can be used as the initialization method. See "Initialization Method" on page 341 for more information about this method.
- Filter data being changed in the RODM data cache.

  You can write a change method to provide filtering between an application change API function request and the field being changed in the RODM data cache. The change method can alter or reject the change API function request according to policy, security, or validation requirements. See "Change Methods" on page 342 for more information about this method.
- Filter data being queried in the RODM data cache.

  You can write a query method to provide filtering between an application query API function request and the field being queried in the RODM data cache. The query method can alter the data returned from the query API function request according to policy, security, or validation requirements. See "Query Methods" on page 344 for more information about this method.
- Notify applications when data in the RODM data cache changes value.

You can write a notify method to notify applications that are subscribed to an object or class when field values belonging to the object or class are changed. See “Notify Methods” on page 346 for more information about this method.

- Perform multiple actions on more than one field within an object or class.

  You can write a named method to process numerous API functions against a single object or class. See “Named Methods” on page 349 for more information about this method.

Types of Methods

A method is logic in the form of an executable program that is loaded into a RODM address space and is run under certain circumstances. Methods are classified according to the circumstances under which they are run. Several kinds of methods are architected into the RODM product to supply specific kinds of functions. All methods are optional, and the function provided by methods can be used or not, depending on how classes, objects, and methods are defined, organized, and applied in RODM. In broad terms, there are two kinds of methods: *object-independent* methods, and *object-specific* methods.

- Object-independent methods are like callable subroutines that run inside RODM. They can act on many different objects in RODM. Object-independent methods are triggered using the EKG_TriggerOIMethod function, which can be issued by user applications, by other object-independent methods, and asynchronously by object-specific methods.
- Object-specific methods are run only in the context of a particular object. For example, they are run by transactions that refer to a specific object. When an object-specific method is running, it has access only to the data in the fields and subfields of that object. Object-specific methods in RODM can be triggered as side effects of a transaction (the query, change, and notify methods previously described or by explicit reference (named methods that are run upon explicit request).

Methods can refer to data and manipulate data in RODM objects. Through the routines in the method API, methods can query and change the fields and subfields of the RODM objects to which the methods have access. Methods must use the method API to access data in the RODM data cache.

The different methods and their uses are described on the following pages. A pseudocode description of the method interface is included with each explanation. These descriptions describe only the parameters, not the exact interface. The parameters are assumed to be passed to the method by address. The pseudocode examples (in PL/I style) are not intended to imply PL/I parameter passing conventions, such as using descriptors for structures. The method interface is intended to be consistent with the user API style of interface where parameters are pointers directly to the passed data.

Object-Independent Methods

Object-independent methods are like callable subroutines that run inside RODM. They are not associated with any particular RODM object or class. They can act on many different objects in RODM. Object-independent methods are triggered using the EKG_TriggerOIMethod function, which can be issued by user applications, by other object-independent methods, and asynchronously by object-specific methods.

Object-independent methods have these characteristics:

- They can be run from the user API or the method API.

- They can be run by a method for asynchronous execution.
- They can access fields in multiple objects.
- They can issue multiple method API requests to RODM without the target objects being affected by other transactions.

Object-independent method parameters are short-lived parameters. These parameters are defined using the SelfDefining data type and contain application-defined values. These parameters are established dynamically from the EKG_TriggerOIMethod function.

While the standard query and change transactions that a user can submit against RODM are restricted to interactions with one object, an object-independent method can interact in sequence with, or at the same time with, each of several different objects. An object-independent method has access to all the objects in RODM through the method API.

RODM manages the interaction of transactions to ensure that all actions are completed against target entities before allowing access to the entities by other transactions.

Object-independent methods have no long-lived parameters associated with them. One SelfDefining data string, of variable length (up to a maximum of 32767 bytes), is the only parameter passed to an object-independent method when the method is run. RODM does not restrict the contents of that string. You must coordinate the parameter passed when the method is run with the parsing and meaning that the message attaches to the string of bytes that is passed.

Figure 76 shows how an object-independent method is defined in PL/I. Figure 77 shows how an object-independent method is defined in C.

```
ObjIndpMeth: Procedure ( ChStrParm );

 Declare
   ChStrParm     SelfDefiningDataPtr;   /* Pointer to Short-lived, byte string */
   . . . .
   /* code */
   . . . .
 End;
```

Figure 76. Object-Independent Method Procedure Interface for PL/I

```
 VOID ObjIndpMeth(SelfDefiningDataPtr       **in_ChStrParm);
 ....
 /* code */
 ....
```

Figure 77. Object-Independent Method Procedure Interface for C

Initialization Method

The initialization method is a special kind of object-independent method. It is run by RODM at initialization time. When RODM is started with the initialization method, RODM installs, runs, and then frees the method automatically. The main purpose of the initialization method is to set up the initial hierarchy of the RODM data cache. Some functions can be used only by the initialization method. The RODM load function can be used as the RODM initialization method.

Object-Specific Methods

Object-specific methods are as follows:

- Run implicitly as the side effect of a transaction
  - Query method (when querying data)
  - Change method (when changing data)
  - Notify method (after changing data)
- Run explicitly by request through RODM User or Method API
  - Named method (by specifying field name)

Change Methods

A change method is triggered by RODM when a transaction issues the EKG_ChangeField or EKG_ChangeMultipleFields function request to change the value of a field and that field has a change method defined. A change method is not triggered, however, when a transaction issues the EKG_ChangeSubfield function request to change the value in the value subfield of the field. A change method:

- Determines the final value of field to be changed, with the exception of fields of type ObjectLink and ObjectLinkList. Change methods defined on these fields do not change the value of the field. Instead, they determine whether a link or unlink action can proceed.
- Is inherited unless locally overridden.
- Runs in context of a class or object being changed.

The change method parameters are as follows:

field_id
FieldID of the field being changed.

long_lived_parms
A SelfDefining string containing application-defined parameters. These parameters are provided to the change method when it is installed.

short_lived_parms
A SelfDefining string containing application-defined parameters. These parameters are provided to the method dynamically during the API function request that triggers the change method.

data_type
RODM data type of the field being changed.

CharDataLen
The integer length of the new_data if data_type is CharVar or GraphicVar. This length does not include the null terminator for these data types.

New_data
New data for the field from the API call.

A change method can be associated with a field of an object as a subfield of that field. A change method is run every time a transaction is run (a user API or method API transaction) that changes the contents of the field. A change transaction whose target is a simple field triggers whatever change method has been assigned to the change subfield of the target field. Change methods can be triggered by these transactions through either the user API or method API.

A change method is also triggered when a transaction issues the EKG_LinkTrigger function request or the EKG_UnlinkTrigger function request to link two fields in two objects and those fields have change methods defined. These change methods cannot change the value of the fields. The change methods must set a return code to indicate whether the link or unlink can proceed. If the change methods do not exist, or if they do not explicitly set the return code, RODM assumes the return code is zero and the link or unlink proceeds. Change methods on fields other than ObjectLink and ObjectLinkList are run only when the field on which they are defined is directly changed. A change method is not run when the same field on the parent class is changed and the changed value is inherited. A change method is not run by changes in a child object or class. A change method is not run by changes to subfields. The triggering of change methods can be avoided by the use of transactions that manipulate the value subfield of a field.

If a field has a change method defined on it, that change method is responsible for making any changes to the value of that field; RODM will not change the value of that field. The change method must use the EKG_ChangeSubfield function to update the value subfield of the field. If the change method uses the EKG_ChangeField or EKG_ChangeMultipleFields functions to update the value subfield, the change method recursively runs itself. RODM detects and blocks the recursive method execution but does not change the value subfield.

If a change method needs to interact with a resource outside of RODM, it sends any request to the resource asynchronously and set the appropriate flags to indicate that the request has been sent. The change method does not wait for a reply from the real resource before it continues processing.

A change method is associated with a specific field of a specific object. Only a change to that specific field of that object triggers the change method to be run. Change methods for a field of an object can automatically exist on the object by inheritance at the time the object is created. A change method on a field of an object is not triggered by the creation or deletion of that object.

A change subfield has data type MethodSpec. The MethodSpec data type identifies the method that is run. It optionally contains long-lived parameters that are passed to the method when it is run. The long-lived parameters can be used to adapt a general purpose method to a particular situation.

The long-lived parameters can be a list of field identifiers. They are defined when the method is assigned to the change subfield. The list of field identifiers is static. However, the values in the fields are dynamic; they can be changed at any time.

A method can read the contents of fields through the method API. So with a list of field identifiers specifying which fields contain its parameters, a change method can find its own execution-time parameters and take the intended actions. Most methods are written as general-purpose methods by IBM, and several parameters might be required to adapt the general-purpose method to the specific function to be performed to manage a change to a field. This design has the advantage of making parameters to methods visible through the user API for debugging purposes.

Another parameter (besides the long-lived parameters) is passed to a change method when the method is run. The function blocks in the user API and method API for changing fields all include a short-lived parameter, which is SelfDefining data with a maximum length of 254 bytes. When a function block is filled in, a requestor can use these 254 bytes for any data that needs to be passed at invocation time to any methods triggered by the transaction.

To change the value subfield of the field, the change method obtains the data supplied through the API. That information is passed as the fourth and fifth parameters.

Figure 78 shows example change method parameters for PL/I. Figure 79 shows example change method parameters for C.

```
ChngMeth: Procedure ( Field_ID, LLParms, SLParms, DataType, CharDataLen, DataPtr )
 Dcl Field_ID     FieldID;             /* target field of transaction */
 Dcl LLParms     SelfDefiningDataPtr;     /* Pointer to Long-lived field parameters */
 Dcl SLParms     SelfDefiningDataPtr;     /* Pointer to Short-lived Parameter       */
 Dcl DataType    Smallint;                /* Data type of field                     */
 Dcl CharDataLen Integer;                 /* Valid for data type CharVar and GraphicVar */
 Dcl DataPtr     pointer;                 /* Pointer to new data from API call      */
   . . . .
   /* code */
   . . . .
 End;
```

Figure 78. Change Method Procedure Interface for PL/I

```
 VOID ChngMeth(FieldID                     *in_FieldID,
              SelfDefiningDataPtr          **in_LLParms,
              SelfDefiningDataPtr          **in_SLParms,
              Smallint                     *in_DataType,
              Integer                      *in_CharDataLen,
              Pointer                      **in_DataPtr);
 ....
 /* code */
 ....
```

Figure 79. Change Method Procedure Interface for C

Note: For data types of CharVar and GraphicVar, the input data strings are null terminated: CharVar strings by X'00', GraphicVar strings by X'0000'.

The return code and reason code for the entire transaction can be controlled from a change method through calls in the method API available to the method.

Through the method API, a change method has access to:
- Data in fields and subfields of the object upon which it is acting
- A copy of the function block that triggered this method
- Organization of the object including data types of fields

Some of the things a change method can do are the following:
- Stop a transaction upon an error condition and set the return and reason codes using the EKG_SetReturnCode function.
- Change fields and subfields of the target object using the EKG_ChangeSubfield function.
- Add a notification using the EKG_AddNotifySubscription function.
- Take actions on other objects using the EKG_MessageTriggeredAction function.
- Write to the RODM log using the EKG_OutputToLog function.

Query Methods

A query method is run by RODM when a transaction queries the value of a field; but not run when the value subfield is explicitly queried. The query method:
- Can determine final returned data value of the field being queried

- Is inherited unless locally overridden
- Runs in context of a class or object being queried

The query method parameters are:

field_id
FieldID of the field being queried.

long_lived_parms
A SelfDefining string containing application-defined parameters. These parameters are provided to the query method when it is installed.

short_lived_parms
A SelfDefining string containing application-defined parameters. These parameters are provided to the method dynamically during the actual API function request that triggers the query method.

Query methods can be associated with fields of objects. If a query method is defined for a field, the method is run each time the field is queried using the EKG_QueryField function through the user API or method API. If a query method is defined, it is responsible for returning a value for the field to the function that queried the field. The query method can return the current value of the field, or the method can return some other value. For example, a query method can issue a command to some real resource to get the current status of that real resource.

The query can use the EKG_ResponseBlock function to write its response to the caller-provided response block. If the query method does not use the EKG_ResponseBlock function, RODM returns the data in the queried field to the query function. A query method can generate the actual value that is returned. It can check time stamps to verify that the value of a field is current. If you do not want to trigger a query method, use the EKG_QuerySubfield function to query the value subfield of the field rather than querying the field itself.

If a query method submits a command to a real resource to obtain information, it returns immediately to the caller with a reason code indicating that a request for new data has been submitted. No method enters a WAIT state.

A query method is associated with a specific field of a specific object. Only a query of that field of that object triggers the query method to be run.

A query subfield has data type MethodSpec. A query subfield can preserve the name of a query method to be run and a list of field identifiers specifying (long-lived) field parameters to be used by the query method in customizing its behavior to the particular object, field, and environment where the query method is executing. The query method can read the contents of the field parameters using routines available through the method API.

A short-lived parameter is also extracted from the function block submitted by the requesting application and passed to a query method at the time of invocation. Figure 80 on page 346 shows an example of query method parameters for PL/I. Figure 81 on page 346 shows an example of query method parameters for C.

```
QueryMeth: Procedure ( Field_ID, LLParms, SLParms );
 Dcl Field_ID     FieldID;             /* target field of transaction */
 Dcl LLParms     SelfDefiningDataPtr;  /* Pointer to Long-lived field parameters */
 Dcl SLParms     SelfDefiningDataPtr;  /* Pointer to Short-lived Parameter       */
   . . . .
   /* code */
   . . . .
 End;
```

Figure 80. Query Method Procedure Interface for PL/I

```
 VOID QueryMeth(FieldID                     *in_FieldID,
               SelfDefiningDataPtr         **in_LLParms,
               SelfDefiningDataPtr         **in_SLParms);
 ....
 /* code */
 ....
```

Figure 81. Query Method Procedure Interface for C

Notify Methods

Notification methods are run by RODM after certain functions are made. To determine which functions run notification methods, see the description for the function in Chapter 14, “Application Programming Reference,” on page 367.

A notification method:

- Generates notifications to subscribed users
- Is inherited only from class to object
- Runs in context of a class or object being changed
- Can propagate knowledge of field changes to:
  - Other objects
  - Subscribed users

The notification method parameters are as follows:

field_id
FieldID of the field that was changed.

long_lived_parms
SelfDefining string containing application-defined parameters. These parameters are provided to the notification method when it is installed.

short_lived_parms
SelfDefining string containing application-defined parameters. These parameters are provided to the method dynamically during the actual API function request that triggers the notification method.

change_status
Specifies whether or not the changed field value is equal to the old field value.

user_appl_id
UserID of the user that is to receive the notification.

notif_queue_id
Name of the notification queue that is to receive the notification.

user_word
User-supplied information.

A list of notification methods is associated with each field of a class or object that has a notify subfield present. The list is called the subscription list for the field. Every time a field is changed, the associated subscription list of notification methods is processed, and each method in the list is run. The intent of these methods is to propagate knowledge of changes both to other objects and to applications outside RODM that need to be informed about changes. Notification methods can include logic to selectively notify, such as to notify only when a threshold is surpassed.

When a change transaction is specified against a field, all notification methods defined on that field are triggered. These notification methods are triggered regardless of whether or not a change method is defined on the field and whether or not the value of the field actually changes. Each notification method is passed a Change_status parameter by RODM, which informs the method whether or not the value of the field was changed by the change transaction.

To avoid triggering notification methods, use functions that do not trigger methods. These functions do not trigger notification methods:
- EKG_LinkNoTrigger
- EKG_UnlinkNoTrigger
- EKG_ChangeSubfield
- EKG_SwapSubfield

The subscription list on the child is not processed, and the notification methods are not run. Notification methods are active only when values in fields are locally present. This practice is similar to the practice of avoiding triggering change methods where the value in the associated field is inherited, and a change is made to the parent field.

Some notification methods can delete themselves after their first execution. For example, an application submits a RODM transaction that causes a command to be submitted to the target system where the command is attempting to vary a device offline. Completion of the request takes time.

The transaction cannot wait for the response, and the application needs to be informed when the command is complete. The code, which might be a change method implementing the original transaction, places a notification method in the subscription (notification) queue for the field. When the device is varied offline, the notification method pulls itself out of the subscription queue and notifies the original application that the requested vary command has been successfully run.

When a method calls the EKG_AddNotifySubscription function, that method must acquire the required information, identified by the data type SubscriptSpec, to actually perform the function. This information is obtained through long-lived-parameters and short-lived-parameters.

Notification methods are placed in the subscription list of a field upon an explicit request made by an application using the EKG_AddNotifySubscription function in the user API and method API. Notification methods can be deleted from a subscription list using the EKG_DeleteNotifySubscription function.

The subscription list for a field is always processed in the order that the notification methods were placed in the subscription queue. The methods are processed, one at a time, starting with the first method placed in queue.

There is another issue of how inheritance interacts with notification methods. Notification subscriptions are not inherited from a parent class to a child class. However, they are effectively inherited from a class to an object, where the class is the primary parent of the object. Notification subscriptions can be associated with any class or object. When it is associated with a class and that class field changes, the notification list on that class field is run. When a change is made to an object field, the notification subscriptions assigned to the field in that object are run. Any notification subscriptions assigned to the same field in the primary parent are also run, enabling you to use a single notification subscription at the class level for all objects in the class. Methods assigned to an object parent class can use the "WhereAmI" method API to determine the circumstances under which their execution has been triggered.

The NetView program supplies four sample notification methods in source format. Study these methods to learn more about writing your own notification method. The sample methods are the following members of the CNMSAMP data set:
- EKGNEQL
- EKGNLST
- EKGNOTF
- EKGNTHD

These methods are described in "RODM Notification Methods" on page 481.

Figure 82 shows an example of notification parameters for PL/I. Figure 83 shows an example of notification parameters for C.

```
NotifMeth: Procedure ( FieldID, LLParms, SLParms, Change_status,
                       User_Appl_ID, Notif_queue_ID, User_word );

 Declare
   FieldID          Field-identifier,     /* Field-identifier of named field  *
   LLParms          SelfDefiningDataPtr,  /* Pointer to Long-lived field parameters */
   SLParms          SelfDefiningDataPtr,  /* Pointer to Short-lived Parameter       */
   Change_status    Smallint,             /* 0 specifies new data was equal to data*/
                                          /* 1 specifies new data was not equalold data*/
   User_Appl_ID     ApplicationID,        /* unique User identifier      */
   Notif_queue_ID   SubscribeID,          /* Notification queue reference */
   User_word        Anonymous(8);         /* remote user spec            */
   . . . .
   /* code */
   . . . .
 End;
```

Figure 82. Notification Method Procedure Interface for PL/I

```
VOID NotiMeth(FieldID                     *in_FieldID,
              SelfDefiningDataPtr         **in_LLParms,
              SelfDefiningDataPtr         **in_SLParms,
              Smallint                    *in_Change_status,
              ApplicationID               **in_User_Appl_ID,
              SubscribeID                 **in_Notif_queue_ID,
              Anonymous                   **in_User_word);
....
/* code */
....
```

Figure 83. Notification Method Procedure Interface for C

Named Methods

A named method is indicated by a field defined as MethodSpec, containing:
- Method object ID
- Long-lived method parameters

A named method:
- Allows for multiple coordinated actions against an object
- Named method field can also have query, change, notify, prev_val, and time stamp subfields

The named method parameters are:

field_id
FieldID of the field being run.

long_lived_parms
SelfDefining string containing application-defined parameters. These parameters are provided to the named method when it is installed.

short_lived_parms
SelfDefining string containing application-defined parameters. These parameters are provided to the method dynamically during the actual API function request that triggers the named method.

The method is considered *named* because it can be referenced (queried, changed and triggered) using the field name. The field name represents a field in an object with the data type of MethodSpec. A field of this type contains a method name and a list of long-lived field parameters that are available to the method when the method is run. Explicit actions available in the user API and method API are used to trigger named methods.

Named methods enable you to change more than one field of a class or object. RODM locks all of the fields of the target object when a named method is run. No other method or user application can access those fields until the named method completes. This enables you to coordinate the updates to several fields on a target class or object.

Because many named methods can all be associated with all objects of a class, named methods are typically inherited from the class. Many standard transactions against objects can be implemented by either methods that are supplied with the NetView program or user-written methods.

A field of data type MethodSpec, a named method field, can have its own query, change, notify, and other standard subfields. The data in the value subfield of such a field includes the method name and a list of field parameters. The specified field parameters can be the targets of actions taken by the named method, or they can contain arguments to the execution of the named method. As with query and change methods, the long-lived list of field parameters is determined when the named method field is assigned a value. The contents of any fields referenced through the long-lived parameters can be set at any time.

Besides the field parameters, another parameter can be passed at execution time to a named method by the application that triggers the method. This is called a short-lived parameter. Unlike long-lived field parameters, it is not preserved in any way after the named method has run. All short-lived parameters on named methods must be of data type SelfDefining of maximum length 254. Such short-lived parameters are a variable length string of bytes that can be structured in any way that the requesting application and the named method are written to recognize.

The NetView program supplies a sample named method in source format. Study this method to learn more about writing your own named method. The sample method is the member EKGMIMV of the CNMSAMP data set. This method is described in “RODM Named Methods” on page 485.

Figure 84 shows an example of named method parameters for PL/I. Figure 85 shows an example of named method parameters for C.

```
NamedMeth: Procedure ( Field_ID, LLParms, SLParms );
 Dcl Field_ID     FieldID;          /* Field-identifier of named field  */
 Dcl LLParms     SelfDefiningDataPtr;  /* Pointer to Long-lived field parameters */
 Dcl SLParms     SelfDefiningDataPtr;  /* Pointer to Short-lived Parameter       */
   . . . .
   /* code */
   . . . .
 End;
```

Figure 84. Named Method Procedure Interface for PL/I

```
VOID NamedMeth(FieldID                     *in_FieldID,
               SelfDefiningDataPtr         **in_LLParms,
               SelfDefiningDataPtr         **in_SLParms);
....
/* code */
....
```

Figure 85. Named Method Procedure Interface for C

A named method has access to the same data, and has the same abilities as query and change methods. However, the explicit invocation of named methods is at the discretion of applications using RODM, and named methods are free form in the function that they provide if the function can be implemented with the available data and services.

Inheritance in Object-Specific Methods

Query, change, notify, and named methods are all object-specific methods. Of these methods, only named methods are values in fields of RODM objects. Query, change, and notify methods are all stored in subfields of objects. On an object, the named method fields and subfields on fields are inherited from the subfields of the public classes of that object.

In the same way, the values in named method fields and the values in query and change subfields can be inherited through primary inheritance, using the standard principles for supporting inheritance in RODM. Notify methods are inherited from the primary parent to its object children. They are not inherited throughout the class inheritance tree. However, the object fields can additionally have local values that do not override the class-level notification subscriptions. (So standard inheritance of values does not apply to notification subfields.)

Named methods, query methods, change methods and notification methods can also all exist on classes. Change methods on classes (as on objects) can be used to validate changes before they are made, or they can be used to validate a user's authority to make those changes. Query methods can validate a requestor's authority to see the requested data, or they can validate data before it is returned.

Likewise, named methods on classes can be used in ways similar to the ways such methods are used on objects. Complex changes to a class can be run by a named method, or general-purpose functions, applicable to many individual classes, can be implemented with named methods. Finally, notification methods are also valuable on classes.

Change and notification methods on children that are inheriting values from parents are not triggered when the inherited values are changed on parents. Therefore, notification methods are required on parents (which can be classes) so that user applications can be notified when parameters and values change on parents.

The main purpose of the primary hierarchy of classes is to make it easy to specify the organization of and default values in RODM objects. The most common values that are inherited at the object level from the primary hierarchy include:

- Methods and parameters to control the management of RODM data to reflect real-world objects
- Policy parameters that indicate standard limits and thresholds
- Long-lived characteristics, such as capacity, of RODM objects where those characteristics are needed to manage real-world objects

These methods and values appear in fields on classes so they can be stated once and then inherited by many objects through the primary hierarchy.

When a value that is a method is inherited by a child, if that method is triggered and run for a child, execution takes place in the context of the child. While the method resided on the parent, only its name and its long-lived parameters are picked up through the inheritance process. When such a method runs and asks for the contents of a field, it gets the contents of that field on the child entity.

A query, change, or named method installed on a class can fill two roles. The method can be the default change method inherited by children and applied in the context of those children (including children that are objects instead of classes), and it can be triggered in the standard way (query, change of field, direct invocation) in the context of the parent.

Be aware that object-specific method you write can sometimes run on an object and at other times can run on a class. The same kinds of capabilities are available for both objects and classes, using the same method API calls. Many object-specific methods look at the WhatIAm field on the current entity to discover the context in which the method is executing, and different actions might be appropriate in different contexts.

Query, change, named, and notification methods on fields of classes are triggered as part of transactions against those classes just as those kinds of methods are triggered on objects. Also, query, change, and named methods exist on fields of classes to support inheritance of those methods by objects, but inheritance of values in notification subfields is not supported in RODM.

If a notification list exists through inheritance, it begins as a null value. A null value in the notification list field is functionally equivalent to no list at all. Entries can be added to a notification list by using the EKG_AddNotifySubscription function.

In summary, named methods and query, change, and notify subfields all function in the standard way both on private and on public fields of classes. There is no inheritance involving private fields, but query, change, and notification methods are run when the corresponding field is queried or changed. When a field is on a class (as with fields on objects), a change transaction for the field triggers change and notification methods, but a change transaction for the value subfield of a field does not trigger change and notification methods. This function is the same as that supported for objects.

Null Method

RODM provides a special method named NullMeth. You can use the NullMeth object ID in place of any object specific method. NullMeth returns control to its caller without doing any processing. The value NullMeth can be inherited in a field or subfield from a parent class. If the value of a field of type MethodSpec is queried for a null method, the ObjectID for NullMeth is returned in the response block.

Using the NullMeth method name, a query or change subfield that is inherited can be set to do nothing. The effect is the same as if the local subfield does not exist. This is useful where the standard function for a field or subfield is to take some action, but there are a few exceptions where that function is locally overridden to do nothing.

Similarly, an empty notification list acts like no list exists. If the corresponding field changes, no notification methods are triggered, and no one is notified of the event.

Deciding Which Method Type to Use

Before you use a method, you must decide which type of method you need to use. What type of method you use depends on the task you want the method to perform.

When to Use an Object-Independent Method

You use an object-independent method if you want to efficiently manipulate more than one entity in the RODM data cache. An object-independent method can change or query any field in any class or object in the RODM data cache.

When to Use an Object-Specific Method

Object-specific methods are methods that have entities specifically associated with them. You use an object-specific method if you want to manipulate only one entity in the RODM data cache. The specific entity that is manipulated is determined at run time and can be different each time that the method is triggered. To run an action against another object or class, an object-specific method can use the EKG_MessageTriggeredAction function. An object-specific method can also trigger the notification method to inform a user application about an event.

There are four types of object-specific methods:
- Query method
- Change method
- Notify method
- Named method

Each of these methods is designed to perform a specific task and can perform that task only on the entity to which it is associated; it cannot access fields in any other entity. Additionally, object-specific methods can call only the API functions that are designed to be callable from these methods. See “Other Services Available to Object-Specific Methods” on page 364 for a list of API functions that are available to object-specific methods.

Query Method

This object-specific method is triggered when a field that has a non-null query subfield is queried in response to an EKG_QueryField API function. The query method ensures that the data returned to the caller of the EKG_QueryField API function is correct and current.

Use this method to refresh data in an entity field that might be outdated or to enforce policy procedures, validation, or security on the data in the field.

Change Method

This object-specific method is triggered when a field that has a non-null change subfield is changed in response to an EKG_ChangeField function, an EKG_ChangeMultipleFields, an EKG_LinkTrigger function, or an EKG_UnlinkTrigger function. The change method ensures that the functions change, link, or unlink the fields correctly by enforcing data security, data validity, and even policy requirements.

Use this method to enforce policy procedures, validation, or security on the data in an entity field.

Notify Method

This object-specific method is triggered when the value in a field that has a non-null notify subfield is changed. The notify method notifies the applications that are subscribed to the field that the value of the field has changed.

Use this method to notify an application program of a change in the field value of an entity field when that information is essential to the operation of the application.

Named Method

This object-specific method is triggered explicitly by a call to the EKG_TriggerNamedMethod API function. A named method has the capability of performing multiple API functions on all fields within a particular entity. RODM implicitly locks the entity while the method is running. No other method or application can query or change any of the fields of the target entity until the Named method returns control to RODM.

This method is used to perform multiple API functions on a single entity where it is critical that no other method or application can query or change the entity's fields.

Using the Method API

To write methods for RODM, access to RODM data and services is required. The method API provides a set of entry points to RODM that can be called by methods.

A variety of services are available to methods. Some services are available only to object-independent methods, and some are available only to object-specific methods.

Method API calls to RODM pass the following parameters:

- Transaction information block
- Function block
- Response block

The function block can point to additional parameters, such as an entity access information block and a field access information block, which identify the target of the function. The response block is required only for some functions.

The transaction_info_block, function_block, and response_block have the same format as the blocks used by the user API. Table 36 lists where you can find more information about these blocks.

Table 36. Additional Information About Blocks

If You Want More Information on	See Page
Transaction_info_block	307
Response_block	314
Function_block	308

The CALL statement from the PL/I or C language program transfers control to the code segment EKGMAPI. The method must be link-edited with the EKGMAPI module during the link-edit step. Figure 86 shows an example PL/I CALL statement.

```
Declare EKGMAPI Entry( structure, structure, structure );

Call EKGMAPI(  transaction_info_block,
                function_block,
                response_block          /* Null pointer => omitted */
                             );
```

Figure 86. Method API Interface Declaration and Invocation Example

Register Conventions

The method code must follow this register convention:

Register 1
: Points to the first of three consecutive memory locations (a parameter list) that contains addresses of the transaction_info_block, function_block, and response_block.

Register 12
: Is reserved for RODM runtime environment. This register must be preserved by the method. For code written in PL/I and C, this register requirement is consistent with the generated code.

Register 13
: Contains the address for the 72-byte save area of the calling program.

Register 14
: Contains the return address for the calling program.

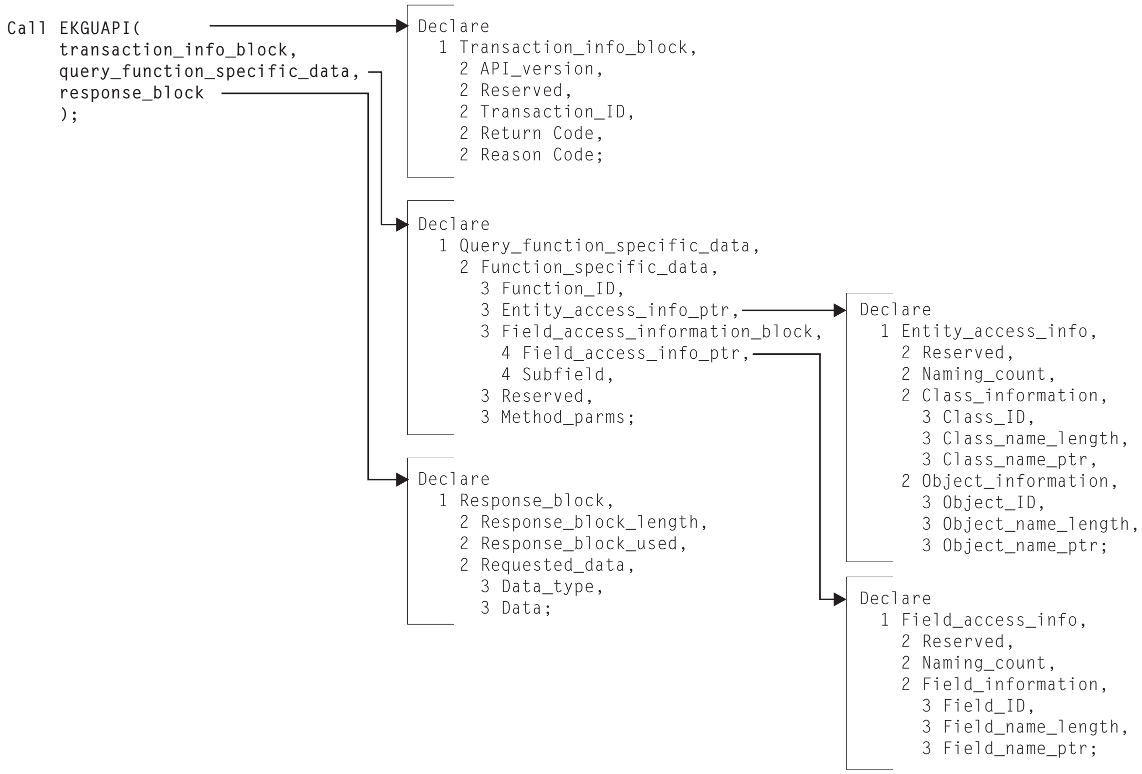
Register 15
: Contains the entry address for the EKGMAPI module.

Usage Notes

The details of all RODM functions are specified in function blocks. The method builds a function block and passes it to RODM to request a desired transaction. The method API functions are described in Chapter 14, "Application Programming Reference," on page 367.

The entity_access_information data, pointed to by the function block, is interpreted the same way for method API calls from object-independent methods as it is from user API calls. However, class and object information is ignored if the call is made from an object-specific method.

The object-specific change, query, notification, and named methods can only access fields within the object or class from which the method API call is performed.



```
Call EKGUAPI(
     transaction_info_block,
     query_function_specific_data,
     response_block
     );
```

```
Declare
  1 Transaction_info_block,
    2 API_version,
    2 Reserved,
    2 Transaction_ID,
    2 Return Code,
    2 Reason Code;
```

```
Declare
  1 Query_function_specific_data,
    2 Function_specific_data,
      3 Function_ID,
      3 Entity_access_info_ptr,
      3 Field_access_information_block,
        4 Field_access_info_ptr,
        4 Subfield,
      3 Reserved,
      3 Method_parms;
```

```
Declare
  1 Response_block,
    2 Response_block_length,
    2 Response_block_used,
    2 Requested_data,
      3 Data_type,
      3 Data;
```

```
Declare
  1 Entity_access_info,
    2 Reserved,
    2 Naming_count,
    2 Class_information,
      3 Class_ID,
      3 Class_name_length,
      3 Class_name_ptr,
    2 Object_information,
      3 Object_ID,
      3 Object_name_length,
      3 Object_name_ptr;
```

```
Declare
  1 Field_access_info,
    2 Reserved,
    2 Naming_count,
    2 Field_information,
      3 Field_ID,
      3 Field_name_length,
      3 Field_name_ptr;
```

Figure 87. Method API Query Field Control Block Sample

API Query Function Control Block Example

Method Parameters

Many transactions have optional parameters that are either being passed to or installed with methods. There are two kinds of method parameters:
- Long-lived parameters
- Short-lived parameters

Long-Lived Parameters

The long-lived parameters are statically defined parameters. Long-lived parameters are:
- Valid only for object-specific methods
- A variable length, SelfDefining string of data
- Restricted to 254 bytes

- Internal meaning is user-defined and user-interpreted

Long-lived parameters are saved in RODM with a method at the time the method is assigned to a subfield, such as when a notification method is installed by the EKG_AddNotifySubscription function or when a named, query, or change method is assigned to a field or subfield.

These long-lived parameters are not immediately used, but are saved until the corresponding method is run (by the appropriate triggering mechanism), and they are made available to that method when the method runs. In this way, general purpose methods can be written and the parameters that provide the desired function specified when the method is assigned to a field or subfield.

Long-lived parameters have the form of a variable length, SelfDefining data string where the length is a maximum of 254 bytes. The content of the 254 bytes of data is not specified by RODM; it is determined by specification of that particular method's interface. The contents of the actual SelfDefining data string cannot be changed after it is specified during method assignment to a field. However, if that long-lived parameter contains a reference to a field within an object, the value of that field can be changed at any time.

Short-Lived Parameters

Short-lived parameters are dynamically defined parameters. Short-lived parameters have the following characteristics:

- Internal meaning is user-defined and user-interpreted for both object-specific and object-independent methods when the method is run using an API request.
- They are a variable length, SelfDefining string of data.
- They are restricted to 254 bytes for object-specific methods.
- They are restricted to 32767 bytes for object-independent methods.

Short-lived parameters are not prestored. They are supplied through the specific transaction request API and are made immediately available to methods being triggered by the transaction. These parameters always have the form of a variable length SelfDefining data string.

Short-lived parameters passed to object-independent methods through the User API can be up to 32767 bytes, but short-lived parameters passed to object-specific methods are restricted to 254 bytes. The meaning of these strings is not defined or limited by RODM. RODM sees only a string of bytes. The requesting user application and the methods being triggered must be written to agree on the contents of this string of bytes.

Installing and Freeing Methods

Before an object-specific method can be assigned to a field or subfield of an object, and before an object-independent method can be run, the method must be installed in RODM. To install a method, create an object of the EKG_Method class.

To install a named method, follow these steps:

1. Determine where you want to install the method.

   For named methods, you must use a field of type MethodSpec on either a class or an object.
2. Create an object of the EKG_Method class.

   Creating this object returns to you the object ID of the newly created object.

3. Use the EKG_ChangeField, the EKG_ChangeSubfield, or EKG_ChangeMultipleFields functions to set the value of the MethodSpec field to the object ID and any long-lived parameters required by your method.

You can also install methods using the RODM load function. When you create an object in the EKG_Method class, RODM loads the method into its address space. Attempting to assign a method name to a field or subfield before the method has been installed results in an error return code from the change transaction.

If an installed method needs to be changed, the EKG_Refresh field in the EKG_Method class enables you to load a new copy of the method into RODM. Trigger the named method specified in the EKG_Refresh field of the method object you want to reload to load the new copy of the method from the library.

When a method is no longer needed, a user can free the storage taken up by the method and can purge the method's name and address from internal RODM tables by executing a delete object transaction against the method object. A method can only be freed if it is not assigned as a value to any field or subfield in RODM. After method has been freed, it cannot be assigned to a field or subfield, and it cannot be run as an object-independent method until it is re-installed.

While other methods need to be installed before use, the null method, NullMeth, is always installed and cannot be freed. An attempt to install or free NullMeth results in an error return code from RODM. Therefore, the method name NullMeth is reserved in RODM, and cannot be used for a user-written method. Other methods that are supplied with the NetView program must be installed before use just like user-written methods.

Synchronous and Asynchronous Execution of Functions

If a method triggers a function or another method, the triggered function or method runs synchronously with the triggering method. The triggering method stops running and does not resume processing until the triggered function or method finishes and returns. The method API provides the EKG_MessageTriggeredAction function, which provides a method with the capability to trigger a function or another method to run asynchronously with it. The triggering method continues to run while the triggered function or method starts, processes, and finishes.

Although the EKG_MessageTriggeredAction function is intended to allow an object-specific method to access entities in the RODM data cache other than the one it is associated with, it can also be called by an object-independent method. Also, the EKG_MessageTriggeredAction function enables the following functions to run asynchronously with the triggering method:
- Change or swap the contents of a field or subfield
- Link or unlink two objects
- Revert inheritance of a field
- Create and delete objects

Method Anchor Service

RODM provides a callable method anchor service that will return a pointer to an 8-byte work area. This area is cleared to zeros prior to invoking the method, and the contents of the area is preserved when the method causes other methods to be triggered.

It is intended that this area be used for communication between the component modules of large, complex methods. Note that it cannot be used to communicate between methods, because it is cleared by RODM each time a method is run.

Run the EKGMANC service routine using the following code for PL/I:

```
DCL WORK_AREA  CHAR(8) BASED(WORK_AREA_PTR);
DCL WORK_AREA_PTR POINTER;
CALL EKGMANC(WORK_AREA_PTR);
```

For C use the following code:

```
char *work_area_ptr;
EKGMANC(&work_area_ptr);
```

There is no return or reason code from the EKGMANC call. The address of the work area is always returned.

Coding Your RODM Method

The following sections describe some of the details of writing your own methods. These sections include information about compiler options, link-editing, and restrictions. Be sure to review both the general restrictions and the restrictions for the programming language you are using.

Installation Written Methods

Installation written methods can be written in PL/I or in C. These methods can use the multicultural support of the PL/I language. DBCS character strings can be manipulated as graphic constants.

Installation supplied methods can reference RODM data stored in either SBCS or DBCS formats.

After your method has been coded, you can run the method using test data and debugging aids to find any syntax or logic errors. Refer to the *IBM Tivoli NetView for z/OS Programming: PL/I and C* for additional information. Install your method by link-editing it to the appropriate user library pointed to by the STEPLIB DD statement in your start up JCL for RODM.

NetView Methods

The NetView program includes a basic set of RODM methods. You can write your own methods in either PL/I or C. You can supplement or replace methods that are supplied with the NetView program with your methods. All methods that are supplied with the NetView program reside in the CNMLINK target library for the NetView program.

Note: Enterprise PL/I for z/OS cannot be used to compile method code, including your method code and method code that is supplied with the NetView program.

Currently, the following methods are supplied with RODM:

EKGNOTF
Notify for any change

EKGNLST
Notify if changed value is equal to one value in a list of values

EKGNEQL
Notify if changed value is equal to a specific value

EKGNTHD
Notify if changed value is within a specified threshold

EKGCTIM
Change method to trigger an Object-independent method to complete an action asynchronously

EKGMIMV
Named method to increment a value

EKGSPPI
Object-independent method used by the RODM automation platform

All notification methods return, in the notification block, the current value, previous value, and time stamp (if these subfields are defined) from the field that is causing the notification message.

The methods that are supplied with the NetView program for RODM are described in the following sections on a functional basis. All parameters that are passed to methods are specified as SelfDefining data strings.

Programming Language Specific Preprocessor Statements

When compiling your program or linking your source code, add the following preprocessor statements.

Compiling IBM C Methods

If you are compiling your methods using the IBM C language, follow these guidelines:

- Code the following pragma statement:

  ```
  #pragma linkage(csect,PLI)
  ```

  where `csect` is the name of the external entry-point csect.
- If any RODM control blocks are referenced in the modules, include file EKG3CINC.H in your source file. This file includes all of the RODM function and response blocks, and the function prototype statements for the RODM entry points EKGMANC, EKGUAPI, EKGMAPI, and EKGWAIT.
- If no RODM control blocks are referenced in the modules but the modules do call EKGMANC, EKGUAPI, EKGMAPI, or EKGWAIT, include file EKG3CEEP.H in your source file.
- Do not specify the RENT option when compiling.

The following is an example of IBM C source for coding a method:

```
#pragma linkage(thisproc,PLI)

#include "EKG3CINC.H"
  /* or */
#include "EKG3CEEP.H"
```

```
void thismethod(void arg)
{
   /* code */
}
```

Compiling IBM PL/I Methods

If you are compiling your methods using the IBM PL/I language, follow these guidelines:

- If any RODM control blocks are referenced in the modules, include file EKG1IINC in your source file. This file includes all of the RODM function and response blocks, and the function prototype statements for the RODM entry points EKGMANC, EKGUAPI, EKGMAPI, and EKGWAIT.
- If no RODM control blocks are referenced in the modules but the modules do call EKGMANC or EKGMAPI, include file EKG1IEEP in your source file.
- Specify the REENTRANT option when compiling.
- Specify the MACRO preprocessor compiler option if you include RODM macros in your method.

The following is an example of IBM PL/I source for coding a method:

```
*PROCESS MACRO;
 thismethod: proc;

%include ekglib(EKG1IINC);
    or
%include ekglib(EKG1IEEP);

/* code */

end thismethod;
```

Linking Methods that Call EKGMAPI Directly

Specify the following link-edit control statements when linking a method that calls EKGMAPI directly:

```
<method object code>

INCLUDE SYSLIB(EKGMAPI)
ENTRY method_name
NAME method_name(R)
```

Specify these link-edit options:
- AMODE=31
- RMODE=ANY or RMODE=24
- RENT

Restrictions on Methods

All RODM methods must run in PSW key 8, which is the default. Do not change the PSW key in any method.

PL/I Language Restrictions

Installation defined methods written in PL/I require a PL/I compiler that is supported by RODM. These PL/I programs are expected to clean up after execution is complete for a particular invocation; all dynamically allocated storage is freed. In addition, PL/I programs that run in the RODM address space must observe certain the following restrictions:

- Use of PLITEST

The PLITEST facility is not available to programs running in the RODM address space.

- Use of FETCH and RELEASE

  PL/I procedures cannot be fetched or released by other PL/I procedures. The user API supports adding and deleting methods. These services can be used in place of FETCH and RELEASE.
- Use of DATE built-in function

  The PL/I DATE built-in function cannot be called by a program running in the RODM address space.
- Use of TIME built-in function

  The PL/I TIME built-in function cannot be called by a program running in the RODM address space.
- Use of file I/O

  PL/I file I/O cannot be used by programs running in the RODM address space. No RODM method attempts to access SYSPRINT. However, the RODM output to log function can be used for file I/O.
- Interlanguage communication

  Interlanguage calls to COBOL and FORTRAN routines cannot be used. Only interlanguage calls to C and assembler are permitted.
- Delays

  The execution of a method cannot be suspended. Methods complete as soon as possible.
- Wait

  The execution of a method cannot be suspended.
- Use of PL/I DISPLAY statement

  The PL/I DISPLAY statement writes its output to the RODM type-1 log record. Because of performance and logging impacts, the PL/I DISPLAY statement is not usually used. Instead, use the EKG_OutputToLog API function.
- Use of PL/I multitasking

  The PL/I multitasking facilities cannot used. Task management is handled by RODM facilities and not the PL/I facilities. The task, event, and priority options of the CALL statement cannot be used, and do not use the COMPLETION, STATUS, and PRIORITY built-in functions.
- Use of MAIN option

  User methods cannot be coded with the PL/I MAIN option of the PROCEDURE statement.
- Linkage field

  All methods must be reentrant. In addition to writing reentrant code, the REENTRANT option of the PROCEDURE statement must be used.
- Cannot use controlled storage variables, or anything using pseudo-register vectors, such as file I/O and fetch/release
- Programs must not request CHECKPOINT, SORT, or PLIDUMP
- PL/I options for CHECK and FLOW must not be used
- Use of On-Units and Signal
  - PL/I programs cannot perform attention handling; that On-unit will not get control
  - PL/I programs must not signal ERROR or FINISH
  - PL/I programs must not contain On-error or On-finish statements

C Language Restrictions

Methods must be compiled using the NORENT option. Methods must not be prelinked using the C prelink facility.

The following C functions cannot be used in RODM methods:
- Atexit()
- Exit()
- Main()
- All file and stream input/output statements and library functions

Do not specify the static storage class specifier for any data in a method.

The RODM output to log function can be used for file input/output.

Restrictions in General

An object-specific method can query and manipulate only the object or class with which the method is associated.

The following are restrictions on methods:

- Named methods

  Named methods can be run to run synchronously with the caller directly from the user API, by an object-independent method through the method API, or by a named method through the method API. Also, named methods can be triggered to run asynchronous to the caller through the message interface provided in the method API.

  Named methods cannot be triggered for asynchronous execution through the user API.
- Object-independent methods

  Object-independent methods can be run to run synchronously with the caller from the user API or the method API. Also, they can be triggered from any method, through the message interface provided in the method API, to run asynchronous to that method.

  Object-independent methods cannot be triggered for asynchronous execution through the user API.
- Change methods

  Change methods cannot be used on system-defined fields. See “System-Defined Fields” on page 211 for a complete list of these fields.

  Change methods used on LINK fields, that is the fields of data type ObjectLink or ObjectLinkList, are triggered by EKG_LinkTrigger and EKG_UnlinkTrigger functions. These change methods have the following restrictions:
  - They cannot change fields.
  - They cannot perform a link or unlink function.
  - They must set a return code if the return code is non-zero.
    - A zero return code allows the link or unlink to continue.
    - A non-zero return code prohibits the link or unlink.
    - If the change methods exist, the return codes from the change methods defined to both objects must be zero in order for the link or unlink to continue.
- Notification methods

  A particular combination of a User_appl_ID, notification method, SubscribeID, and long-lived parameters uniquely specify a notification method and can be assigned only one time to a particular notification subfield.

- All methods
  - All methods must be written as reentrant.
  - Methods cannot query a notification queue or suspend their own execution.
  - When RODM is operating on a z/OS system, methods must adhere to operating system constraints placed on applications running in cross-memory mode; for example, the methods must not use any service that requires the execution of an IBM z/Architecture® SVC instruction.
  - If a method uses recovery routines such as ESTAE, ESTAX, SPIE, or STAE, the recovery routines must be set up to percolate so that RODM regains control after any abend.
  - Use of the method API to synchronously run another method must not cause recursive execution of any previously run method.
  - The response block overflow buffer is not available to methods. If the response block supplied by a method is too small for the data returned by the function, the data that does not fit in the supplied response block is discarded.

RODM Method Services

Some RODM functions can be used by all types of methods; others can be used only by certain types of methods. The following sections lists the types of methods and the RODM functions that each can use.

Services Available to both Object-Specific and Object-Independent Methods

When you design your program, you can implement the following functions, available for use in both object-independent and object-specific methods.

- Querying RODM Data
  - EKG_QueryField (See “EKG_QueryField — Query a Field” on page 411)
  - EKG_QueryMultipleSubfields (See “EKG_QueryMultipleSubfields — Query Multiple Value Subfields” on page 418)
  - EKG_QuerySubfield (See “EKG_QuerySubfield — Query a Subfield” on page 426)
  - EKG_QueryEntityStructure (See “EKG_QueryEntityStructure — Query Structure of an Entity” on page 409)
  - EKG_QueryFieldStructure (See “EKG_QueryFieldStructure — Query Structure of a Field” on page 415)
  - EKG_QueryFieldID (See “EKG_QueryFieldID — Query Field Identifier” on page 412)
  - EKG_QueryFieldName (See “EKG_QueryFieldName — Query a Field Name” on page 413)
- Actions against RODM Data
  - EKG_ChangeField (See “EKG_ChangeField — Change a Field” on page 376)
  - EKG_ChangeMultipleFields (See “EKG_ChangeMultipleFields — Change Multiple Fields” on page 377)
  - EKG_ChangeSubfield (See “EKG_ChangeSubfield — Change a Subfield” on page 378)
  - EKG_RevertToInherited (See “EKG_RevertToInherited — Revert to Inherited Value” on page 429)
  - EKG_AddNotifySubscription (See “EKG_AddNotifySubscription — Add Notification Subscription” on page 373)
  - EKG_DeleteNotifySubscription (See “EKG_DeleteNotifySubscription — Delete Notification Subscription” on page 393)

  - EKG_TriggerNamedMethod (See "EKG_TriggerNamedMethod — Trigger a Named Method" on page 438)
- Additional Method Support
  - EKG_SendNotification
  - EKG_MessageTriggeredAction
  - EKG_SetReturnCode
  - EKG_OutputToLog
  - EKG_ResponseBlock (can be used in named and query object-specific methods and object-independent methods)
  - EKG_QueryFunctionBlockContents

This list of query and action functions is a subset of the transactions available to RODM users through the user API.

Both the user API and method API use the same function blocks to specify the function requested for queries and actions with the queries generating responses that are returned in response blocks. Also, a named method can generate data that is returned in a response block.

See Chapter 11, "Writing Applications that Use RODM," on page 301 for the formats for all these function blocks and response blocks. As in the user API, the user of the method API is responsible for allocating and freeing the storage in which function and response blocks reside. The method API function blocks for the additional method support functions are described in this section.

Other Services Available to Object-Independent Methods

The following additional services are available to object-independent methods through the method API and the user API.

- EKG_LinkNoTrigger, EKG_LinkTrigger (See "EKG_LinkNoTrigger, EKG_LinkTrigger — Link Two Objects" on page 402)
- EKG_UnlinkNoTrigger, EKG_UnlinkTrigger (See "EKG_UnlinkNoTrigger, EKG_UnlinkTrigger — Unlink Two Objects" on page 441)
- EKG_CreateObject (See "EKG_CreateObject — Create an Object" on page 388)
- EKG_DeleteObject (See "EKG_DeleteObject — Delete an Object" on page 395)
- EKG_TriggerOIMethod (See "EKG_TriggerOIMethod — Trigger an Object-Independent Method" on page 440)

Other Services Available to Object-Specific Methods

The following additional services are available *only* to object-specific methods:
- EKG_WhereAmI
- EKG_QueryObjectName

Services Available to the Initialization Method

The initialization method is the only method that can use the following functions. The method can run these functions at RODM initialization time to create the RODM data structure and load the data into the RODM data cache.

- Administrative functions
  - EKG_CreateClass (See "EKG_CreateClass — Create a Class" on page 386)
  - EKG_CreateField (See "EKG_CreateField — Create a Field" on page 387)
  - EKG_CreateSubfield (See "EKG_CreateSubfield — Create a Subfield" on page 390)
- Control functions

  - EKG_Checkpoint (See "EKG_Checkpoint — Checkpoint RODM to DASD" on page 380)

The access to the above mentioned functions is similar to the access available through the user API. These functions are run by calls to RODM using the method API. Use of these functions requires the standard function block definitions.

The method API functions and interfaces available to the initialization method also include all those enabled in object-independent methods, with the following exceptions. Do not use these exceptions within the initialization method.

- EKG_SendNotification
  - This function fails because no Notification_queues can be registered at the time the initialization method is running.
- EKG_ResponseBlock
  - No response block is passed to the initialization method, so the data is lost.
- EKG_QueryFunctionBlockContents
  - No function block is used to initiate the initialization method execution, so no data is available.
- EKG_CreateObject to create an EKG_NotificationQueue object
  - Notification queues are named by concatenating a User_appl_ID to the queue name. This function always fails for the initialization method because no User_appl_ID is available during initialization.

If the initialization method uses the message interface to start an asynchronous task, RODM initialization continues without waiting for the completion of that asynchronous task.

RODM Method Library

To access the method API services, RODM provides a library that contains entry points for method API services. This library is called the RODM Method Library and is given the default name CNMLINK.

This library is especially intended for use with C and PL/I programs. To access a service such as EKGMAPI, declare EKGMAPI as an external entry in your program. To resolve the external name, use the CNMLINK library.

Member EKGMIMV of data set CNMSAMP in the sample library contains an example showing how EKGMAPI can be called from a named method to increment the value of a specified field by the value of a field.

Chapter 14. Application Programming Reference

The details of all transactions against RODM data are specified in function blocks. A user builds a function block and passes it to RODM to request a desired transaction. All function blocks contain a Function_ID which specifies the function being requested from RODM.

Summarizing RODM Functions

This chapter describes each of the RODM functions. The major categories of functions follow:
- Access functions
- Control functions
- Administrative functions
- Action functions
- Query functions
- RODM user API services
- RODM method API services

See Chapter 11, "Writing Applications that Use RODM," on page 301 for an explanation of how function blocks are used in application programs. See Chapter 13, "Writing RODM Methods," on page 339 for an explanation of how function blocks are used in methods.

Access Functions

Access functions enable a user application program to connect to and disconnect from RODM.

EKG_Connect or EKG_ConnectLong: Connect to RODM
The connect function is called to connect the user to RODM.

EKG_Disconnect: Disconnect from RODM
The disconnect function is called to end the connection between the user and RODM.

Control Functions

Control functions allow a user application program that has the appropriate access level to checkpoint RODM data to DASD or to stop RODM, with or without checkpointing data.

EKG_Checkpoint: Checkpoint RODM
Checkpoint RODM data to DASD.

EKG_Stop: Stop RODM
Stop the RODM subsystem.

Administrative Functions

Use the RODM administrative functions, with the appropriate function blocks passed as parameters, to delete or create classes, fields, and subfields. Because response blocks are not needed in administrative calls, set the response block pointer to null.

When a RODM class is initially created, it contains the system-defined fields and the public fields of its primary parent. The values of these fields are inherited from their primary parent. Classes are differentiated from their parent by the existence of additional fields or by setting different values in the fields that do exist. Most frequently, a child class needs to have more fields than exist on the parent. These additional fields must be explicitly added to the class. RODM has no set limit of the number of fields a class can contain.

You can add a field to a class. You can add a subfield only to a field that is already in place. You cannot add a field directly to an object.

EKG_CreateClass: Create a Class
Create a new class in the RODM data cache.

EKG_CreateField: Create a Field
Add a new field to a class.

EKG_CreateSubfield: Create a Subfield
Add a new subfield to a field in a class.

EKG_DeleteClass: Delete a Class
Remove a class from the RODM data cache.

EKG_DeleteField: Delete a Field
Delete a field from a class.

EKG_DeleteSubfield: Delete a Subfield
Delete a subfield from a field in a class.

Action Functions

Action functions change values, create and delete objects and links between objects, add and delete notification subscriptions, and trigger named and object-independent methods. Action functions can be submitted in list form using the EKG_ExecuteFunctionList function to enable multiple actions with a single interface call.

EKG_AddNotifySubscription: Add a Notification Subscription
Subscribe to a field.

EKG_AddObjDelSubs: Add an Object Deletion Subscription
Subscribe to an object for notification of deletion.

EKG_ChangeField: Change a Field
Change the value of a field.

EKG_ChangeMultipleFields: Change Multiple Fields
Change the value of multiple fields of an object.

EKG_ChangeSubfield: Change a Subfield
Change the value of a subfield.

EKG_CreateObject: Create an Object
Create an object in the RODM data cache.

EKG_DeleteNotifySubscription: Delete a Notification Subscription
Delete a subscription to a field.

EKG_DeleteObject: Delete an Object
Delete an object in the RODM data cache.

EKG_DelObjDelSubs: Delete an Object Deletion Subscription
Delete a subscription to an object.

EKG_LinkNoTrigger: Link Two Objects
Link two objects; do not run notify methods.

EKG_LinkTrigger: Link Two Objects
Link two objects; run notify methods.

EKG_RevertToInherited: Revert to Inherited Value
Remove the local copy of the data value from a field and replace it with the inherited value.

EKG_SwapField: Swap a Field
Compare and swap field data with new data.

EKG_SwapSubfield: Swap a Subfield
Compare and swap subfield data with new data.

EKG_TriggerNamedMethod: Trigger a Named Method
Run a named method.

EKG_TriggerOIMethod: Trigger an Object-Independent Method
Run an object independent method.

EKG_UnlinkNoTrigger: Unlink Two Objects
Unlink two objects; do not run notify methods.

EKG_UnlinkTrigger: Unlink Two Objects
Unlink two objects; run notify methods.

Query Functions

Query functions enable a user application program to query the values contained in fields, subfields, notification queues, and access blocks. Query functions can be submitted in list form using the EKG_ExecuteFunctionList function to enable multiple actions with a single interface call.

The contents of the field or information to be queried is returned in the response block.

If a field of an object or class is being queried and there is a query method associated with the field, that query method is run before the contents of the field is retrieved. That method has the opportunity to change the contents of the field before the data in the field is read and returned to the caller. A query method can explicitly set the returned value of the query operation by using the EKG_ResponseBlock function. If a query method uses the EKG_ResponseBlock function, RODM does not place any data into the response block.

EKG_Locate: Locate Objects Using Public Indexed Field
Provide a list of all objects in RODM that match a specified search criteria.

EKG_QueryEntityStructure: Query Structure of an Entity
Provide a list of all fields within a class or object, specifying each field's name, data type, and inheritance state.

EKG_QueryField: Query Field
Obtain the value of a field.

EKG_QueryFieldID: Query Field Identifier
Convert a field name to its field identifier.

EKG_QueryFieldName: Query Field Name
Convert a field identifier to its field name.

EKG_QueryFieldStructure: Query Structure of a Field
Provide organization of a field (that is, data type, local copy indicator, and subfield map).

EKG_QueryMultipleSubfields: Query Multiple Value Subfields
Obtain the value of multiple subfields for an object.

EKG_QueryNotifyQueue: Query Notification Queue
Obtain next queue element, if available.

EKG_QueryResponseBlockOverflow : Query Response Block Overflow
Obtains any overflow response block data.

EKG_QuerySubfield: Query Subfield
Obtain the value of a subfield.

RODM User API Services

EKG_ExecuteFunctionList: Execute a List of Functions
Enable user application programs to pass a list of RODM functions in a single function call.

RODM Method API Services

EKG_LockObjectList: Lock List of Objects
This API was used to enable object-independent methods to explicitly lock objects. It is no longer necessary, but is maintained for compatibility.

EKG_MessageTriggeredAction: Trigger an Action by a Message
Provide object-specific methods with the ability to trigger an asynchronous API function for another object or class.

EKG_QueryFunctionBlockContents: Query Function Block Contents
Provide methods with the contents of the function block of the function request that triggered the method.

EKG_QueryObjectName: Query Object Name
Allow an object-specific method to convert an ObjectID to the corresponding object name.

EKG_OutputToLog: Output to Log
Provide the ability to output information to the RODM log.

EKG_ResponseBlock: Output to Response Block
Appends method-defined information to the caller's response block, except for Query methods, which overwrite the response block.

EKG_SendNotification: Send a Notification
Provide the facility for notification methods to send notification information blocks to notification queues when a field is changed.

EKG_SetReturnCode: Set Return and Reason Codes
Enable a method to set the return code and reason code for the method caller.

EKG_UnlockAll: Unlock all Held Entities
This method was used to free all locks held. It is no longer necessary, but is maintained for compatibility.

EKG_WhereAmI: Where Am I
Enable an object-specific method to determine the class, object, and field for which it was triggered.

Function Reference

This section describes each of the functions available from the RODM user application programming interface and the RODM method application programming interface. The format of this section is described in "Function Reference Format." The functions are listed in alphabetical order by function name.

Function Reference Format

This section describes the format of the RODM function descriptions contained in this chapter. The functions are listed in alphabetical order by function name. Following each function name is a function description. Each function description contains the following reference sections:
- Purpose
- Function block format
- Examples
- Summary
- Usage

These reference sections are described in the following sections.

Purpose

The purpose section of each function description explains what the function does.

Function block format

The function block format describes the function block that you need to pass to the function. If the function returns a response block, the response block is also described in this section.

The function block format table contains five columns:

Offset The offset in decimal bytes to the beginning of the parameter.

Length
The length in decimal bytes of the parameter. If the length of a parameter is variable, the length column contains a dash (—) character.

Type The RODM abstract data type of the parameter. A few parameters do not use the defined RODM abstract data types. The PL/I or data types are listed for parameters which do not use RODM abstract data types.

Use The use is either In for data input to the function, or Out for data output by the function. For reserved fields and fields not used by a particular function, the use column contains a dash (—).

Parameter Name
The name of the parameter. Each parameter is described in "Function Parameter Descriptions" on page 445. This is the actual name used in the example function block or response block supplied with RODM.

Examples

The examples section lists the names of the code examples provided by RODM for each function. Provided in both PL/I and C, these examples are on the samples tape that was shipped with the NetView product. Include the example function block and response block in your user application or method for each function you plan to use. Use the parameter names that are provided to access the function. This will limit the impact to your program of any service that might be applied to RODM.

The example function blocks and example response blocks for PL/I contain the preprocessor macro substitution variable *EKG_Boundary*. This variable is converted to `UNALIGNED BASED(*)`, which is required for PL/I programs.

The usage coding examples are pieces of actual code that illustrate how to set up and call each function. Use the usage coding examples to learn about calling the function. Note, however, that these examples might not be suitable for inclusion in your programs.

The names in the examples table are the member names of each example. The default data set name for function block samples and response block samples is NETVIEW.V6R1M0.SCNMMAC1. The default data set name for usage coding examples is NETVIEW.V6R1M0.CNMSAMP. For example, the complete name of the function block example in PL/I for the EKG_Connect function is NETVIEW.V6R1M0.SCNMMAC1(EKG11101). The complete name of the PL/I usage coding example for this function is NETVIEW.V6R1M0.CNMSAMP(EKG51101).

Summary

The summary table lists the following topics for each function:

Function ID
The function identifier used by RODM to determine which function has been requested.

Type The type of function, such as access or query.

User API
Specifies whether this function can be used by user applications.

Object-specific method
Specifies whether this function can be used by object-specific methods.

Object-independent method
Specifies whether this function can be used by object-independent methods.

Initialization method
Specifies whether this function can be used by initialization methods.

Methods triggered
Specifies whether this function triggers query, change, or notification methods and which methods are triggered.

Triggered by EKG_MessageTriggeredAction
Specifies whether this function can be run asynchronously by the EKG_MessageTriggeredAction function.

Authorization
Specifies the minimum authorization level that a user application must be assigned in order to use this function.

User applications must be authorized to use specific RODM functions. Each function specifies the required authorization level. Applications can use all functions with a required authorization level equal to or less than the authorization level of the application. Each application's authorization level is specified when the application User_appl_ID is created in the security system profile. See the *IBM Tivoli NetView for z/OS Security Reference* for information about defining authorization levels.

Usage Notes

This topic provides additional function information and limitations.

The parameters used by each function are described in “Function Parameter Descriptions” on page 445. This section describes in general what each parameter does. Function-specific differences in parameters, such as maximum data length, are listed in the usage section for the specific function.

The return codes and associated reason codes issued by RODM functions are listed in “RODM Return and Reason Codes” on page 452. This section also includes cross reference tables that list all of the reason codes that each function uses and all of the functions that use a particular reason code. You can use this information to design the error handling routines for your user applications and methods.

The final section in this chapter describes the methods that are supplied with the NetView program. These include notification and change methods you can use with RODM. “Supplied Methods” on page 480 describes each method and the parameters you pass to it.

EKG_AddNotifySubscription — Add Notification Subscription

Purpose

This function adds a notification method to a field on an object or a class. RODM places the notification method in a subscription list associated with the field. If the specified notification queue does not exist, RODM creates the notification queue using the specified User_appl_ID.

Function Block Format

Table 37. Function Block for the EKG_AddNotifySubscription Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Pointer	In	Entity_access_info_ptr
008	4	Pointer	In	Field_access_info_ptr
012	8	ApplicationID	In	User_appl_ID
020	8	SubscribeID	In	Notification_queue
028	8	Anonymous(8)	In	User_word
036	8	ObjectID	In	Notify_method
044	4	SelfDefiningDataPtr	In	Long_lived_parm

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 38. Example Names for the EKG_AddNotifySubscription Function

Example	Name
PL/I function block	EKG11412
PL/I response block	None
PL/I usage coding	EKG51412
C function block	EKG31412
C response block	None
C usage coding	EKG61412

Summary

Table 39. Summary of the EKG_AddNotifySubscription Function

Function ID	1412
Type	Action
User API	Yes
Object-specific method	Yes
Object-independent method	Yes
Initialization method	Yes
Methods triggered	Notification method of MyObjectChildren field of the EKG_NotificationQueue class triggered if the notification queue object is created
Triggered by the EKG_MessageTriggeredAction function	No
Authorization	3

Usage

See "RODM Notification Process" on page 318 for more information about notification subscriptions.

A notification subscription, consisting of a User_appl_ID, Notification_queue, method ObjectID, and Long_lived_parm is added to a field one time. If a second request specifying the same information is sent, the request is rejected.

The class, object, and field access information from the function block specify where the subscription is to be installed. If the value subfield of the designated field is changed by the EKG_ChangeField or EKG_ChangeMultipleFields functions, the requested notification method is run.

When a notification method is run, it is provided the value of the Long_lived_parm field from the function block. The method cannot modify the Long_lived_parm.

Users can assign notification subscriptions to both an object and its parent class where both are run when a change is made to the object field. When these notifications are added, RODM does not validate that duplicate subscriptions have not been added between the class and object. Duplicate subscriptions are rejected only at the individual class or object level.

EKG_AddObjDelSubs — Add Object Deletion Subscription

Purpose

This function adds a deletion-subscription to an object; RODM sends you a notification block if the object is deleted.

Function Block Format

Table 40. Function Block for the EKG_AddObjDelSubs Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID

Table 40. Function Block for the EKG_AddObjDelSubs Function (continued)

Offset	Length	Type	Use	Parameter Name
004	4	Pointer	In	Entity_access_info_ptr
008	8	ApplicationID	In	User_appl_ID
016	8	SubscribeID	In	Notification_queue
024	8	Anonymous(8)	In	User_word
032	4	SelfDefiningDataPtr	In	Long_lived_parm

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 41. Example Names for the EKG_AddObjDelSubs Function

Example	Name
PL/I function block	EKG11417
PL/I response block	None
PL/I usage coding	EKG51417
C function block	EKG31417
C response block	None
C usage coding	EKG61417

Summary

Table 42. Summary of the EKG_AddObjDelSubs Function

Function ID	1417
Type	Action
User API	Yes
Object-specific method	Yes
Object-independent method	Yes
Initialization method	No
Methods triggered	No
Triggered by the EKG_MessageTriggeredAction function	No
Authorization	3

Usage

A deletion-notification subscription, consisting of a User_appl_ID, Notification_queue, and Long_lived_parm, is added to an object one time. If a second request specifying the same information is sent, the request is rejected.

The object access information from the function block specifies where the subscription is to be installed. If the designated object is deleted by the EKG_DeleteObject function, a notification block is sent to the user application. The content of the notification block is the output from the EKG_QueryNotifyQueue function. For more information, see “EKG_QueryNotifyQueue — Query Notification Queue” on page 421.

EKG_ChangeField — Change a Field

Purpose

This function changes the value of a field of either an object or a class. This function triggers any change or notification methods that are defined on the field.

Function Block Format

Table 43. Function Block for the EKG_ChangeField Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Pointer	In	Entity_access_info_ptr
008	4	Pointer	In	Field_access_info_ptr
012	2	Smallint	In	Subfield
014	2	Smallint	In	Data_type
016	4	Integer	In	New_char_data_length
020	4	Pointer	In	New_data_ptr
024	4	SelfDefiningDataPtr	In	Method_parms

Note that the Subfield parameter at offset 012 is not currently used.

See "Function Parameter Descriptions" on page 445 for more information about the parameters listed. See "Abstract Data Type Reference" on page 223 for more information about the abstract data types listed.

Examples

Table 44. Example Names for the EKG_ChangeField Function

Example	Name
PL/I function block	EKG11401
PL/I response block	None
PL/I usage coding	EKG51401
C function block	EKG31401
C response block	None
C usage coding	EKG61401

Summary

Table 45. Summary of the EKG_ChangeField Function

Function ID	1401
Type	Action
User API	Yes
Object-specific method	Yes
Object-independent method	Yes
Initialization method	Yes
Methods triggered	Change and notification methods triggered
Triggered by the EKG_MessageTriggeredAction function	Yes

Table 45. Summary of the EKG_ChangeField Function (continued)

Authorization	3

Usage

The new value pointed to by New_data_ptr must be of the same data type as the target field being changed. The new value must be formatted correctly for that data type. The Data_type field must specify the same data type as the target field.

You cannot use this function to change fields that have a data type of ObjectID, ObjectIDList, ObjectLink, ObjectLinkList, ClassID, ClassIDList, or ClassLinkList. These fields are set either by RODM, or by the LINK and UNLINK transactions.

You cannot use this function to change the RODM system-defined fields that have read-only access, such as MyName and MyID.

Multiple field values can be changed using the EKG_ChangeMultipleFields function.

EKG_ChangeMultipleFields — Change Multiple Fields

Purpose

This function enables you to change the value of multiple fields of an object. This function triggers change and notification methods that are defined on the field.

Function Block Format

Table 46. Function Block for the EKG_ChangeMultipleFields Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Pointer	In	Entity_access_info_ptr
008	4	Integer	In	Number_of_fields
: First element, array of structure				
012	4	Pointer	In	Field_access_info_ptr
016	2	Anonymous(2)	—	Reserved
018	2	Smallint	In	Data_type
020	4	Integer	In	New_char_data_length
024	4	Pointer	In	New_data_ptr
028	4	SelfDefiningDataPtr	In	Method_parms
032	4	Integer	In	Return_code
036	4	Integer	In	Reason_code

See "Function Parameter Descriptions" on page 445 for more information about the parameters listed. See "Abstract Data Type Reference" on page 223 for more information about the abstract data types listed.

Examples

Table 47. Example Names for the EKG_ChangeMultipleFields Function

Example	Name
PL/I function block	EKG11419

Table 47. Example Names for the EKG_ChangeMultipleFields Function (continued)

Example	Name
PL/I response block	None
PL/I usage coding	EKG51419
C function block	EKG31419
C response block	None
C usage coding	EKG61419

Summary

Table 48. Summary of the EKG_ChangeMultipleFields Function

Function ID	1419
Type	Action
User API	Yes
Object-specific method	Yes
Object-independent method	Yes
Initialization method	Yes
Methods triggered	Change and notification methods triggered
Triggered by the EKG_MessageTriggeredAction function	Yes
Authorization	3

Usage

The new value pointed to by New_data_ptr must be of the same data type as the target field being changed. The new value must be formatted correctly for that data type. The Data_type field must specify the same data type as the target field.

You cannot use this function to change fields that have a data type of ObjectID, ObjectIDList, ObjectLink, ObjectLinkList, ClassID, ClassIDList, or ClassLinkList. These fields are set either by RODM or by the LINK and UNLINK transactions.

You cannot use this function to change the RODM system-defined fields that have read-only access, such as MyName and MyID.

EKG_ChangeSubfield — Change a Subfield

Purpose

This function enables you to change the value of a subfield without triggering change and notification methods.

Function Block Format

Table 49. Function Block for the EKG_ChangeSubfield Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Pointer	In	Entity_access_info_ptr
008	4	Pointer	In	Field_access_info_ptr
012	2	Smallint	In	Subfield

Table 49. Function Block for the EKG_ChangeSubfield Function (continued)

Offset	Length	Type	Use	Parameter Name
014	2	Smallint	In	Data_type
016	4	Integer	In	New_char_data_length
020	4	Pointer	In	New_data_ptr
024	4	—	—	Not used

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 50. Example Names for the EKG_ChangeSubfield Function

Example	Name
PL/I function block	EKG11403
PL/I response block	None
PL/I usage coding	EKG51403
C function block	EKG31403
C response block	None
C usage coding	EKG61403

Summary

Table 51. Summary of the EKG_ChangeSubfield Function

Function ID	1403
Type	Action
User API	Yes
Object-specific method	Yes
Object-independent method	Yes
Initialization method	Yes
Methods triggered	No
Triggered by the EKG_MessageTriggeredAction function	Yes
Authorization	3

Usage

If the value subfield is to be changed, the data type of the new data must be identical with that of the field. For other subfields, the data type of the subfield is determined by the subfield type, and RODM checks that the data_type field in the function block is compatible with the specified subfield.

The change of a value subfield does not cause the prev_val and timestamp subfields to be updated, nor does it run a change or notification method.

EKG_Checkpoint — Checkpoint RODM to DASD

Purpose

This function causes RODM to write a copy of its in-storage data to a checkpoint data set. Use this checkpoint data set to recover RODM data after a system failure.

Function Block Format

Table 52. Function Block for the EKG_Checkpoint Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID

See "Function Parameter Descriptions" on page 445 for more information about the parameters listed. See "Abstract Data Type Reference" on page 223 for more information about the abstract data types listed.

Examples

Table 53. Example Names for the EKG_Checkpoint Function

Example	Name
PL/I function block	EKG11201
PL/I response block	None
PL/I usage coding	EKG51201
C function block	EKG31201
C response block	None
C usage coding	EKG61201

Summary

Table 54. Summary of the EKG_Checkpoint Function

Function ID	1201
Type	Control
User API	Yes
Object-specific method	No
Object-independent method	No
Initialization method	Yes
Methods triggered	Notification
Triggered by the EKG_MessageTriggeredAction function	No
Authorization	4

Usage

The EKG_Checkpoint function writes RODM data to predefined and preallocated VSAM linear data sets, which are called RODM checkpoint data sets.

The checkpoint function is controlled using the CHECKPOINT_FUNCTION statement in member EKGCUST. Use this statement to either disable the checkpoint function or control how the checkpoint function reacts when a checkpoint failure occurs. See the *IBM Tivoli NetView for z/OS Administration Reference* for more information.

The data that the EKG_Checkpoint function writes to the checkpoint data sets includes the following:

- The RODM master window—a RODM data area that resides in the RODM address space and contains RODM system information. The RODM master window data is written to the master window checkpoint file.
- RODM translation window—a RODM data area that resides in the RODM address space and contains the address information that enables correct data mapping and addressing in the RODM data cache. RODM translation window data is written to the translation window checkpoint file.
- RODM data windows—RODM data areas that reside in data spaces and contain the actual data in the data cache. RODM data-window data is written to data window checkpoint files.

The checkpoint process includes the following steps:

1. Begin checkpoint—RODM sends a message to the console, notifying the operator that RODM is quiescing.
2. Quiescing—during the checkpoint quiesce period, RODM allows method API requests, but rejects new user API requests. At the end of the quiesce period, if no user API, method API, or asynchronous transactions are still running, RODM proceeds to the next step in the checkpoint process, first stage checkpoint. Otherwise, RODM issues a Write-To-Operator with Reply (WTOR) message requesting directions from the operator. The operator must then select one of three options:

Option	**Meaning**
1	Perform the quiesce again. Choose this option if a checkpoint is really desired, but give RODM another quiesce period to successfully quiesce.
2	Unconditionally, start first stage checkpoint. Choose this option if a checkpoint is immediately necessary or after having tried option one.
3	Stop the checkpoint request. Choose this option if option one has been attempted or if critical RODM tasks must not be stopped.

3. First stage checkpoint—after the quiescence time period ends and all transactions have finished processing or the operator has requested an unconditional checkpoint, RODM writes the master window and the translation windows to their respective checkpoint files.
4. Second stage checkpoint—after the first stage checkpoint ends, RODM sends a message to the console notifying the operator that transactions can now resume. RODM then begins writing the data windows, one at a time, to the data window checkpoint files. User applications can make transaction requests during this checkpoint stage. However, a transaction will fail if the specific data window that it needs access to is being written to a data window checkpoint file or has not yet been written to a data window checkpoint file.
5. End of checkpoint—after all data windows have been written to data window checkpoint files, RODM sends a message to the console notifying the operator that the checkpoint process has completed, and two EKG_System object fields are updated, depending on whether or not the checkpoint process was successful.

The EKG_LastCheckpointID field of the EKG_System object is updated by RODM to reflect the transaction ID of the of the last checkpoint transaction if the checkpoint process is successful. Otherwise, the EKG_LastCheckpointID field remains unchanged.

The EKG_LastCheckpointResult field of the EKG_System object is updated with the current transaction ID for a checkpoint process issued from a MODIFY command, or the transaction ID of the user API requesting the checkpoint process. The EKG_LastCheckpointResult field also reflects the result of the checkpoint process by use of return and reason codes. Application programs that are subscribed to this field receive notification that the checkpoint process has completed.

With the exception of the checkpoint process, all transactions issued across the RODM user API are synchronous in that the user does not regain execution control until the transaction has completed. With the checkpoint process, the application regains control when the checkpoint request has been recorded. The checkpoint operation is actually processed asynchronously with other processing in the application. This same asynchronous processing for the checkpoint process also applies to an operator-requested checkpoint process, through the MODIFY command.

Coding Checkpoint Control: RODM updates the EKG_LastCheckpointResult field in the EKG_System class each time RODM completes a checkpoint operation. The EKG_LastCheckpointResult field contains the transaction ID of the transaction requesting the checkpoint operation and the return and reason codes indicating the result of the checkpoint operation. Applications can subscribe to this field to be notified of the completion of each checkpoint operation.

Subscribe to the EKG_LastCheckpointResult field to be notified of the result of the checkpoint. The user can then query the field and determine the result of the checkpoint operation. If the checkpoint operation is not successful, the user can then determine why the checkpoint process failed.

A user application can keep a record or journal of its transactions with RODM. If RODM fails between checkpoint operations, the application can then determine which transactions have been checkpointed by RODM and which transactions have to be resent. All transactions in that journal numerically the same or lower than the EKG_LastCheckPointID field are reflected in the checkpoint data sets of the successfully completed checkpoint operations and can be erased from the journal. All transactions numerically higher than the EKG_LastCheckPointID field have to be reset to restore RODM to its status before the failure.

From the beginning of a checkpoint operation until stage 1 is completed, RODM rejects any additional transaction requests and provides a return code and reason code identifying that condition if keyword TRANSPARENT_CHECKPOINT=NO is specified in the customization file.

User applications can subscribe to the EKG_LastCheckpointID field, the EKG_LastCheckpointResult field, or to both fields, using the EKG_AddNotifySubscription function. See “EKG_AddNotifySubscription — Add Notification Subscription” on page 373. You can use the EKGNOTF notification method that is supplied with the NetView program for this subscription. See “RODM Notification Methods” on page 481 for a description of EKGNOTF.

EKG_Connect — Connect to RODM

Purpose

The connect function enables an application program to use RODM. This is the first function that the application can issue to RODM.

Function Block Format

Table 55. Function Block for the EKG_Connect Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	8	Char(8)	In	User_password
012	4	Pointer	In	Stop_ECB
016	8	TransID	Out	Last_checkpoint_ID
024	4	Anonymous(4)	—	Reserved

See "Function Parameter Descriptions" on page 445 for more information about the parameters listed. See "Abstract Data Type Reference" on page 223 for more information about the abstract data types listed.

Examples

Table 56. Example Names for the EKG_Connect Function

Example	Name
PL/I function block	EKG11101
PL/I response block	None
PL/I usage coding	EKG51101
C function block	EKG31101
C response block	None
C usage coding	EKG61101

Summary

Table 57. Summary of the EKG_Connect Function

Function ID	1101
Type	Access
User API	Yes
Object-specific method	No
Object-independent method	No
Initialization method	No
Methods triggered	Notification
Triggered by the EKG_MessageTriggeredAction function	No
Authorization	1

Usage

The User_appl_ID is used to determine the users access authority and to associate registered event control blocks (ECBs) with the appropriate user.

If the system on which RODM is installed is protected by a system authorization facility, the user can connect to RODM using a blank user ID. RODM obtains the user ID from the system authorization facility and uses it to determine the user's access authority in RODM. If the system is not protected by a system authorization facility, the user cannot connect to RODM using a blank user ID.

When a user application issues an EKG_Connect function request, RODM creates a user object from the EKG_User system-defined class.

An access block, as described in “Access Block” on page 305, must be passed. The user's sign_on_token parameter in the access block is set by RODM. This parameter must not be changed by the user application for subsequent calls to RODM.

A user can disconnect from RODM without purging the subscription notification queue. Before the notification queues that are owned by this user application ID can again be posted, all ECB addresses associated with all notification queues for this user and with subscription notifications must be reset for the new address space.

All tasks in the address space from which the EKG_Connect function was issued can access RODM either by connecting to RODM with unique, RODM authorized user IDs, or by using the sign_on_token. The sign_on_token is not valid when the connecting TCB ends or the EKG_Disconnect function is performed.

EKG_ConnectLong — Connect to RODM

Purpose

The connectlong function enables an application program to use RODM. This is the first function the application can issue to RODM.

Function Block Format

Table 58. Function Block for the EKG_ConnectLong Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	100	Char(100)	In	User_password_phrase
104	4	Pointer	In	Stop_ECB
108	8	TransID	Out	Last_checkpoint_ID
116	4	Anonymous(4)	—	Reserved

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 59. Example Names for the EKG_ConnectLong Function

Example	Name
PL/I function block	EKG11100
PL/I response block	None
PL/I usage coding	EKG51100

Table 59. Example Names for the EKG_ConnectLong Function (continued)

Example	Name
C function block	EKG31100
C response block	None
C usage coding	EKG61100

Summary

Table 60. Summary of the EKG_ConnectLong Function

Function ID	1100
Type	Access
User API	Yes
Object-specific method	No
Object-independent method	No
Initialization method	No
Methods triggered	Notification
Triggered by the EKG_MessageTriggeredAction function	No
Authorization	1

Usage

The User_appl_ID is used to determine the users access authority and to associate registered ECBs with the appropriate user.

If the system on which RODM is installed is protected by a system authorization facility, the user can connect to RODM using a blank user ID. RODM obtains the user ID from the system authorization facility and uses it to determine the user's access authority in RODM. If the system is not protected by a system authorization facility, the user cannot connect to RODM using a blank user ID.

When a user application issues an EKG_ConnectLong function request, RODM creates a user object from the EKG_User system-defined class.

An access block, as described in “Access Block” on page 305, must be passed. The user's sign_on_token parameter in the access block is set by RODM. This parameter must not be changed by the user application for subsequent calls to RODM.

A user can disconnect from RODM without purging the subscription notification queue. Before notification queues owned by this user application ID can again be posted, all ECB addresses associated with all notification queues for this user and with subscription notifications must be reset for the new address space.

All tasks in the address space from which the EKG_ConnectLong function was issued can access RODM either by connecting to RODM with unique, RODM authorized user IDs, or by using the sign_on_token. The sign_on_token is not valid when the connecting TCB ends or the EKG_Disconnect function is performed.

EKG_CreateClass — Create a Class

Purpose

This function creates a new class as the child of a specified parent class in the RODM data cache. RODM adds the new class ID entry to the MyClassChildren linked-list field of the parent of the new class.

Function Block Format

Table 61. Function Block for the EKG_CreateClass Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Pointer	In	Class_access_info_ptr
008	4	Pointer	In	Parent_access_info_ptr
012	4	SelfDefiningDataPtr	In	Method_parms

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 62. Example Names for the EKG_CreateClass Function

Example	Name
PL/I function block	EKG11302
PL/I response block	None
PL/I usage coding	EKG51302
C function block	EKG31302
C response block	None
C usage coding	EKG61302

Summary

Table 63. Summary of the EKG_CreateClass Function

Function ID	1302
Type	Administrative
User API	Yes
Object-specific method	No
Object-independent method	No
Initialization method	Yes
Methods triggered	Notification methods on MyClassChildren and WhatIAm fields of parent class triggered
Triggered by the EKG_MessageTriggeredAction function	No
Authorization	5

Usage

Specify the class name and RODM returns the associated ID.

Classes are created only with system-defined fields and those fields that are inherited through the primary hierarchy. All additional fields must be added explicitly by calls to RODM.

Creating a class changes the value of the WhatIAm field of the parent of the class if the parent did not have any class children.

EKG_CreateField — Create a Field

Purpose

This function creates a new field on a class in the RODM data cache.

Function Block Format

Table 64. Function Block for the EKG_CreateField Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Pointer	In	Class_access_info_ptr
008	4	Pointer	In	Field_access_info_ptr
012	2	Smallint	In	Field_type_flag
014	2	Smallint	In	Data_type
016	4	Bit(32)	In	Subfield_map

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 65. Example Names for the EKG_CreateField Function

Example	Name
PL/I function block	EKG11304
PL/I response block	None
PL/I usage coding	EKG51304
C function block	EKG31304
C response block	None
C usage coding	EKG61304

Summary

Table 66. Summary of the EKG_CreateField Function

Function ID	1304
Type	Administrative
User API	Yes
Object-specific method	No
Object-independent method	No
Initialization method	Yes
Methods triggered	No

Table 66. Summary of the EKG_CreateField Function (continued)

Triggered by the EKG_MessageTriggeredAction function	No
Authorization	5

Usage

The initial value for a field is the null value of the field's data type.

When a field is created, RODM applies the following rules:

- If the field being added to a class is public and has the same name and fields (that is, data type and subfield definitions) as a public field already defined in a subclass, the field is defined in the specified class and the subclass defined field is treated as a local value for that field (this affects what value is inherited below the subclass). If the data type of the field in the subclass is different from the new data type, the new definition is rejected.
- If the new field being added is a private field, no check is made for subclass definitions.
- If a new field definition is for a public field and there is an existing private definition in a subclass of the specified class, the new field definition is rejected.

If the field already exists and has exactly the same data type and subfield definitions as was requested, a warning return code is generated and a reason code describing that condition is returned. The original field is left as previously defined.

If a subfield that is not valid is specified, RODM does not create that subfield. However, RODM does create the field and all valid requested subfields. RODM issues the warning return code 4 with reason code 100.

EKG_CreateObject — Create an Object

Purpose

This function creates a new object in the RODM data cache. RODM adds the new object ID entry to the MyObjectChildren linked-list field of the parent of the new object.

Function Block Format

Table 67. Function Block for the EKG_CreateObject Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Pointer	In	Entity_access_info_ptr
008	4	SelfDefiningDataPtr	In	Method_parms

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 68. Example Names for the EKG_CreateObject Function

Example	Name
PL/I function block	EKG11409
PL/I response block	None
PL/I usage coding	EKG51409
C function block	EKG31409
C response block	None
C usage coding	EKG61409

Summary

Table 69. Summary of the EKG_CreateObject Function

Function ID	1409
Type	Action
User API	Yes
Object-specific method	No
Object-independent method	Yes
Initialization method	Yes [1]
Methods triggered	Notification methods on MyClassChildren and WhatIAm fields of parent class triggered
Triggered by the EKG_MessageTriggeredAction function	Yes
Authorization	5 (create method object) 3 (create other object)

: [1] Initialization methods cannot create objects of the EKG_NotificationQueue class.

Usage

The Entity_access_info_ptr must point to an entity access block that specifies the class which is the parent of the object being created. The Object_name_ptr of the entity access block is optional. If the Object_name_ptr is specified, it must point to a field of type ObjectName that contains the name of the requested new object. Otherwise, RODM assigns the new object a name.

If you are creating an object of the EKG_Method class or the EKG_NotificationQueue class, the object name is required. Object names for these classes are limited to 8 characters.

The object name is not returned to the caller through this interface, but can be accessed by querying the MyName field of the object. RODM assigns names in the form EKG*ddddddd* where *ddddddd* is a decimal number from 0000000 to 9999999. If you specify the object name, do not specify an object name that begins with EKG.

The Object_ID field in the entity access block is set by RODM when the object is successfully created. The Method_Parms short_lived_parameters are passed to the notification method on the MyObjectChildren field of the class and the notification method, if one exists, is triggered.

When a new object is created, it contains all of the public locally- defined and inherited fields that appear on the class of the new object. The values in these fields are initially the default values inherited from the class except for the system-defined fields, which are set by RODM, and fields of type ObjectLink, which are empty fields.

All subfields, wherever they exist, begin existence on a new object with inherited values except for the notify subfield. A Notify subfield starts out with the null value.

If the parent class does not have any object children when this object is created, RODM updates the WhatIAm field of the class to indicate that the class now has object children.

EKG_CreateSubfield — Create a Subfield

Purpose

This function creates one or more subfields for an existing field in an existing class in the RODM data cache.

Function Block Format

Table 70. Function Block for the EKG_CreateSubfield Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Pointer	In	Class_access_info_ptr
008	4	Pointer	In	Field_access_info_ptr
012	4	Bit(32)	In	Subfield_map

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 71. Example Names for the EKG_CreateSubfield Function

Example	Name
PL/I function block	EKG11306
PL/I response block	None
PL/I usage coding	EKG51306
C function block	EKG31306
C response block	None
C usage coding	EKG61306

Summary

Table 72. Summary of the EKG_CreateSubfield Function

Function ID	1306
Type	Administrative
User API	Yes

Table 72. Summary of the EKG_CreateSubfield Function (continued)

Object-specific method	No
Object-independent method	No
Initialization method	Yes
Methods triggered	No
Triggered by the EKG_MessageTriggeredAction function	No
Authorization	5

Usage

Subfields can be created only on an existing field of a class. Subfields must be created in the class in which the field was created.

If a specified subfield already exists and other specified subfields do not exist, the subfields that do not exist are created and a warning return code is generated.

EKG_DeleteClass — Delete a Class

Purpose

This function deletes an existing class from the RODM data cache. RODM removes the value in the MyID field of the deleted class from the MyClassChildren linked-list field of the parent of the deleted class.

Function Block Format

Table 73. Function Block for the EKG_DeleteClass Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Pointer	In	Class_access_info_ptr
008	4	SelfDefiningDataPtr	In	Method_parms

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 74. Example Names for the EKG_DeleteClass Function

Example	Name
PL/I function block	EKG11303
PL/I response block	None
PL/I usage coding	EKG51303
C function block	EKG31303
C response block	None
C usage coding	EKG61303

Summary

Table 75. Summary of the EKG_DeleteClass Function

Function ID	1303
Type	Administrative
User API	Yes
Object-specific method	No
Object-independent method	No
Initialization method	No
Methods triggered	Notification methods on MyClassChildren and WhatIAm fields of parent class triggered
Triggered by the EKG_MessageTriggeredAction function	No
Authorization	5

Usage

You cannot delete a RODM system-defined class or a class that has children.

Deleting a class will change the value of the WhatIAm field of the parent of the class if the parent class no longer has any class children.

EKG_DeleteField — Delete a Field

Purpose

This function deletes a field from a class in the RODM data cache. The field is also deleted from any classes and objects that inherit the field from this class.

Function Block Format

Table 76. Function Block for the EKG_DeleteField Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Pointer	In	Class_access_info_ptr
008	4	Pointer	In	Field_access_info_ptr

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 77. Example Names for the EKG_DeleteField Function

Example	Name
PL/I function block	EKG11305
PL/I response block	None
PL/I usage coding	EKG51305
C function block	EKG31305
C response block	None
C usage coding	EKG61305

Summary

Table 78. Summary of the EKG_DeleteField Function

Function ID	1305
Type	Administrative
User API	Yes
Object-specific method	No
Object-independent method	No
Initialization method	No
Methods triggered	No
Triggered by the EKG_MessageTriggeredAction function	No
Authorization	5

Usage

Fields can be deleted only from classes; they cannot be deleted from objects.

Deletion of a public field on a class removes the existence of that field from all descendant classes.

Before a public field can be deleted from a class, you must delete all objects created from that class and from descendent classes of that class.

Local values assigned to a field are discarded when that field is deleted.

Private fields can be deleted at any time.

EKG_DeleteNotifySubscription — Delete Notification Subscription

Purpose

This function deletes one or more notification subscriptions from a field.

Function Block Format

Table 79. Function Block for the EKG_DeleteNotifySubscription Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Pointer	In	Entity_access_info_ptr
008	4	Pointer	In	Field_access_info_ptr
012	24	RecipientSpec	In	Subscription_info
036	8	ObjectID	In	Notify_method
044	4	SelfDefiningDataPtr	In	Long_lived_parm

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 80. Example Names for the EKG_DeleteNotifySubscription Function

Example	Name
PL/I function block	EKG11413
PL/I response block	None
PL/I usage coding	EKG51413
C function block	EKG31413
C response block	None
C usage coding	EKG61413

Summary

Table 81. Summary of the EKG_DeleteNotifySubscription Function

Function ID	1413
Type	Action
User API	Yes
Object-specific method	Yes
Object-independent method	Yes
Initialization method	Yes
Methods triggered	Notification methods triggered
Triggered by the EKG_MessageTriggeredAction function	No
Authorization	3

Usage

Deleting a notification subscription does not delete the notification blocks that are queued on the notification queue when the delete function is issued. The notification queue object is not deleted.

The notification subscription that is to be deleted is uniquely identified by four fields: the User_appl_ID field, the Notification_queue field, the Notify_method field, and the Long_lived_parm field. Using these four fields, the EKG_DeleteNotifySubscription function deletes one or more notification subscriptions based on the first of the following rules that applies:

1. If the Notification_queue field is set to an asterisk followed by seven blanks ("*   "), and the Notify_method and Long_lived_parm fields are set to null values, all subscriptions associated with the specified User_appl_ID field are deleted.
2. If the Notification_queue field is set to an asterisk followed by seven blanks ("*   "), all subscriptions associated with the specified User_appl_ID, Notify_method, and Long_lived_parm fields are deleted.
3. If the Notify_method field is set to the null value, RODM deletes the notification subscriptions that meet the other criteria without considering the value in the Notify_method field.
4. If the Long_lived_parm field is set to the null value, RODM deletes the notification subscriptions that meet the other criteria without considering the value in the Long_lived_parm field.

Specifying User_appl_ID as a null value does not have the same effect as specifying null values for the other parameters. A Null User_appl_ID value is interpreted the same here as for the EKG_AddNotifySubscription function; it requires RODM to supply a default value. The default is determined exactly as for the EKG_AddNotifySubscription function (see "EKG_AddNotifySubscription — Add Notification Subscription" on page 373).

To specify a null Long_lived_parm, declare a pointer to the Long_lived_parm data type with a value of zero.

EKG_DeleteObject — Delete an Object

Purpose

This function deletes an existing object from a specified class. RODM deletes the object ID of the deleted object from the MyObjectChildren field of the parent class of the deleted object.

Function Block Format

Table 82. Function Block for the EKG_DeleteObject Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Pointer	In	Entity_access_info_ptr
008	4	SelfDefiningDataPtr	In	Method_parms

See "Function Parameter Descriptions" on page 445 for more information about the parameters listed. See "Abstract Data Type Reference" on page 223 for more information about the abstract data types listed.

Examples

Table 83. Example Names for the EKG_DeleteObject Function

Example	Name
PL/I function block	EKG11410
PL/I response block	None
PL/I usage coding	EKG51410
C function block	EKG31410
C response block	None
C usage coding	EKG61410

Summary

Table 84. Summary of the EKG_DeleteObject Function

Function ID	1410
Type	Action
User API	Yes
Object-specific method	No
Object-independent method	Yes
Initialization method	Yes

Table 84. Summary of the EKG_DeleteObject Function (continued)

Methods triggered	Notification methods on MyClassChildren and WhatIAm fields of object class triggered
Triggered by the EKG_MessageTriggeredAction function	Yes
Authorization	5 (delete method object) 3 (delete other object)

Usage

The Method_parms data is passed to any notification methods assigned to the MyObjectChildren and WhatIAm fields on the object class.

All ObjectLink type links from all fields of the target object to other objects must be deleted before this object is deleted. RODM returns an error if ObjectLink type links still exist.

If the parent class of this object does not have any children after this object is deleted, RODM updates the WhatIAm field of the class to indicate that it is now a class with no children.

EKG_DeleteSubfield — Delete a Subfield

Purpose

This function deletes one or more subfields from the specified field of a class in the RODM data cache. The subfields must be deleted from the field in the class where the field was created. RODM also deletes the subfields from any class or object that inherits the specified field.

Function Block Format

Table 85. Function Block for the EKG_DeleteSubfield Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Pointer	In	Class_access_info_ptr
008	4	Pointer	In	Field_access_info_ptr
012	4	Bit(32)	In	Subfield_map

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 86. Example Names for the EKG_DeleteSubfield Function

Example	Name
PL/I function block	EKG11307
PL/I response block	None
PL/I usage coding	EKG51307
C function block	EKG31307
C response block	None

Table 86. Example Names for the EKG_DeleteSubfield Function (continued)

Example	Name
C usage coding	EKG61307

Summary

Table 87. Summary of the EKG_DeleteSubfield Function

Function ID	1307
Type	Administrative
User API	Yes
Object-specific method	No
Object-independent method	No
Initialization method	No
Methods triggered	No
Triggered by the EKG_MessageTriggeredAction function	No
Authorization	5

Usage

You can delete a subfield only from the class on which it was created. If a subfield is defined on a parent class, you must delete it from that parent class, not from any child classes that inherit the subfield.

You cannot delete the value subfield. The value of Subfield_map bit 1 must always be 0 (zero) for this function.

If you instruct RODM to delete a subfield that does not exist, RODM returns a warning; it does, however, delete any other subfields that you instructed it to delete, if they exist.

Before a subfield of a public field can be deleted from a class, you must delete all objects created from that class and from descendent classes of that class.

EKG_DelObjDelSubs — Delete Object Deletion Subscription

Purpose

This function deletes a deletion-subscription for an object.

Function Block Format

Table 88. Function Block for the EKG_DelObjDelSubs Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Pointer	In	Entity_access_info_ptr
008	8	ApplicationID	In	User_appl_ID
016	8	SubscribeID	In	Notification_queue
024	8	Anonymous(8)	In	User_word
032	4	SelfDefiningDataPtr	In	Long_lived_parm

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 89. Example Names for the EKG_DelObjDelSubs Function

Example	Name
PL/I function block	EKG11418
PL/I response block	None
PL/I usage coding	EKG51418
C function block	EKG31418
C response block	None
C usage coding	EKG61418

Summary

Table 90. Summary of the EKG_DelObjDelSubs Function

Function ID	1418
Type	Action
User API	Yes
Object-specific method	Yes
Object-independent method	Yes
Initialization method	No
Methods triggered	No
Triggered by the EKG_MessageTriggeredAction function	No
Authorization	3

Usage

Deleting a deletion-subscription does not delete the notification blocks that are queued on the notification queue when the delete function is issued. The notification queue object is not deleted.

The subscription that is to be deleted is uniquely identified by three fields: the User_appl_ID field, the Notification_queue field, and the Long_lived_parm field. Using these three fields, the EKG_DelObjDelSubs function deletes one or more deletion-subscriptions based on the first of the following rules that applies:

1. If the Notification_queue field is set to an asterisk followed by seven blanks ("*   "), and the Long_lived_parm field is set to null values, all subscriptions associated with the specified User_appl_ID field are deleted.
2. If the Notification_queue field is set to an asterisk followed by seven blanks ("*   "), all subscriptions associated with the specified User_appl_ID and Long_lived_parm fields are deleted.
3. If the Long_lived_parm field is set to the null value, RODM deletes the notification subscriptions that meet the other criteria without considering the value in the Long_lived_parm field.

Specifying User_appl_ID as a null value does not have the same effect as specifying null values for the other parameters. A null User_appl_ID value is interpreted the same here as for the EKG_AddObjDelSubs function; it requires RODM to supply a default value. The default is determined exactly as for the EKG_AddObjDelSubs function (see "Function Parameter Descriptions" on page 445).

To specify a null Long_lived_parm, declare a pointer to the Long_lived_parm data type with a value of zero.

EKG_Disconnect — Disconnect from RODM

Purpose

This function disconnects the user application from RODM.

Function Block Format

Table 91. Function Block for the EKG_Disconnect Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID

See "Function Parameter Descriptions" on page 445 for more information about the parameters listed. See "Abstract Data Type Reference" on page 223 for more information about the abstract data types listed.

Examples

Table 92. Example Names for the EKG_Disconnect Function

Example	Name
PL/I function block	EKG11102
PL/I response block	None
PL/I usage coding	EKG51102
C function block	EKG31102
C response block	None
C usage coding	EKG61102

Summary

Table 93. Summary for the EKG_Disconnect Function

Function ID	1102
Type	Access
User API	Yes
Object-specific method	No
Object-independent method	No
Initialization method	No
Methods triggered	Notification
Triggered by the EKG_MessageTriggeredAction function	No
Authorization	1

Usage

After you disconnect from RODM, RODM does not accept any other function requests with your disconnected access block until you issue the EKG_Connect or EKG_ConnectLong function request.

Processing of notification queues and subscriptions when you disconnect from RODM is controlled by setting the EKG_StopMode field of your user object. If you do not intend to reconnect later, set EKG_StopMode in your user object to 1 to cause all notification subscriptions to be deleted. See the EKG_StopMode field in "EKG_User Class" on page 201.

When you disconnect, all notification queues on behalf of your user application ID that are in active status (EKG_Status in the corresponding objects in class EKG_NotificationQueue is set to 1) continue to accumulate notification blocks. If you reconnect at a later time, you must reestablish notification ECBs (field EKG_ECBAddress) within all of your existing notification queue objects before any notifications can be received.

When you disconnect from RODM, your user object is deleted if all subscriptions are deleted (or none were established) and notification queues are purged.

EKG_ExecuteFunctionList — Execute a List of Functions

Purpose

This function runs a list of RODM functions with a single interface call. RODM manages the function list to ensure that the target entities are not affected by other transactions during the call.

Function Block Format

Table 94. Function Block for the EKG_ExecuteFunctionList Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Integer	In	Number_of_Functions
: First element, array of structure				
008	0	Structure	—	Function_info_array
008	4	Pointer	In	Function_block_ptr
012	4	Pointer	Out	Response_block_reference
016	4	Integer	Out	Response_block_used
020	4	Integer	Out	Return_code
024	4	Integer	Out	Reason_code
: Second element, array of structure (if used)				
028	0	Structure	—	Function_info_array
028	4	Pointer	In	Function_block_ptr
032	4	Pointer	Out	Response_block_reference
036	4	Integer	Out	Response_block_used
040	4	Integer	Out	Return_code
044	4	Integer	Out	Reason_code

Note: Function block contains Number_of_functions array elements

Table 95. Response Block for the EKG_FunctionList Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Response_block_length
004	4	Integer	Out	Response_block_used
008	—	—	Out	Response_data

Note: A response block is not required if no function returns data. Response_block_used is the total for all functions. The function block contains the amounts used by individual functions.

See "Function Parameter Descriptions" on page 445 for more information about the parameters listed. See "Abstract Data Type Reference" on page 223 for more information about the abstract data types listed.

Examples

Table 96. Example Names for the EKG_ExecuteFunctionList Function

Example	Name
PL/I function block	EKG11600
PL/I response block	None
PL/I usage coding	EKG51600
C function block	EKG31600
C response block	None
C usage coding	EKG61600

Summary

Table 97. Summary of the EKG_ExecuteFunctionList Function

Function ID	1600
Type	User API Service
User API	Yes
Object-specific method	No
Object-independent method	No
Initialization method	No
Methods triggered	Yes
Triggered by EKG_MessageTriggeredAction function	No
Authorization	2 (list of queries only) 3 (list includes actions)

Authorization levels: EKG_ExecuteFunctionList can perform only action functions and query functions. These action and query functions cannot have authorization levels greater than 3.

Usage

The return code and reason code returned in the transaction information block for the EKG_ExecuteFunctionList function are the highest return code for any individual function, and its corresponding reason code.

RODM manages the function list to ensure that the target entities are not affected by other transactions during the call.

If the response block overflow situation is encountered, all output length values (response_block_used parameters) are set by RODM, but pointer values (for example, response_block_reference parameters) for transaction results that are contained wholly in the overflow buffer are set to null. When you retrieve the overflow block, it is your responsibility to parse that data using the length information returned on the original call.

If the list contains functions not authorized to you, those functions are skipped (no action will be attempted) and an error return code and reason code are set for those functions.

EKG_LinkNoTrigger, EKG_LinkTrigger — Link Two Objects

Purpose

These functions are used to establish a link between two fields on two objects. The EKG_LinkTrigger function triggers change methods and notification methods; the EKG_LinkNoTrigger function does not.

Function Block Format

Table 98. Function Block for EKG_LinkNoTrigger Function and the EKG_LinkTrigger Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Pointer	In	Entity_access_info_ptr_1
008	4	Pointer	In	Field_access_info_ptr_1
012	4	Pointer	In	Entity_access_info_ptr_2
016	4	Pointer	In	Field_access_info_ptr_2
020	4	SelfDefiningDataPtr	In	Method_parms[1]

:

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 99. Example Names for the EKG_LinkNoTrigger Function and the EKG_LinkTrigger Function

Example	Name
PL/I function block (EKG_LinkTrigger)	EKG11405
PL/I function block (EKG_LinkNoTrigger)	EKG11406
PL/I response block	None
PL/I usage coding (EKG_LinkTrigger)	EKG51405
PL/I usage coding (EKG_LinkNoTrigger)	EKG51406
C function block (EKG_LinkTrigger)	EKG31405
C function block (EKG_LinkNoTrigger)	EKG31406
C response block	None

Table 99. Example Names for the EKG_LinkNoTrigger Function and the EKG_LinkTrigger Function (continued)

Example	Name
C usage coding (EKG_LinkTrigger)	EKG61405
C usage coding (EKG_LinkNoTrigger)	EKG61406

Summary

Table 100. Summary of the EKG_LinkNoTrigger Function and the EKG_LinkTrigger Function

Function ID EKG_LinkNoTrigger EKG_LinkTrigger	 1406 1405
Type	Action
User API	Yes
Object-specific method	No
Object-independent method	Yes
Initialization method	Yes
Methods triggered EKG_LinkTrigger EKG_LinkNoTrigger	Change methods and notification methods No
Triggered by the EKG_MessageTriggeredAction function	Yes
Authorization	3

Usage

Links can be performed only on fields within objects. Fields of classes cannot be linked. The fields being linked must be on different objects.

Each of the two fields to be linked must be type ObjectLink or ObjectLinkList. Use an ObjectLink field if you need only one link. Use an ObjectLinkList field if you need more than one link for a field.

No assumption can be made regarding the order of links within a field of type ObjectLinkList.

If a link is performed on a field of type ObjectLink that was previously linked to another field, the link function will fail.

If a link is performed on a field of type ObjectLinkList that was previously linked to another field, the link function will succeed. If the field that it is linked to is also of type ObjectLinkList, the link is added and previous links are retained.

Do not use EKG_LinkNoTrigger with GMFHS resources.

When the EKG_LinkTrigger function is issued, the link operation is performed before the notification methods are triggered. If there are change methods defined on one or both of the fields to be linked, the link proceeds after the change methods, but only if one of the following is true:

- Both change methods explicitly set a zero return code with EKG_SetReturnCode.

- Neither change method sets a return code. In this case, RODM assumes a zero return code and the link proceeds.

If the link does not proceed, the notification methods are not triggered. If the objects are successfully linked, the notification methods are triggered in the following order:

1. Notification methods for the field specified by Field_access_info_ptr_1
2. Notification methods for the field specified by Field_access_info_ptr_2
3. Notification methods for the parent class of the first field
4. Notification methods for the parent class of the second field

EKG_Locate—Locate Objects Using Public Indexed Field

Purpose

This function returns the list of object IDs of all objects in RODM that match the search criteria. The search criteria is specified as the value of a character field that has been defined as public_indexed. See "Indexed Fields" on page 220 for a description of using public indexed fields and the EKG_Locate function.

Function Block Format

Table 101. Function Block for the EKG_Locate Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Anonymous(4)	—	Reserved, must be X'00000000'
008	4	Pointer	In	Field_access_info_ptr
012	2	Smallint	In	Data_type, must be 4 or 32
014	2	Anonymous(2)	—	Reserved, must be X'0000'
016	4	Integer	In	Indexed_data_length
020	4	Pointer	In	Indexed_data_ptr

Table 102. Response Block for the EKG_Locate Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Response_block_length
004	4	Integer	Out	Response_block_used
008	—	ObjectIDList	Out	Requested_data

See "Function Parameter Descriptions" on page 445 for more information about the parameters listed. See "Abstract Data Type Reference" on page 223 for more information about the abstract data types listed.

Examples

Table 103. Example Names for the EKG_Locate Function

Example	Name
PL/I function block	EKG11509
PL/I response block	EKG21509
PL/I usage coding	EKG51509
C function block	EKG31509

Table 103. Example Names for the EKG_Locate Function (continued)

Example	Name
C response block	EKG41509
C usage coding	EKG61509

Summary

Table 104. Summary of the EKG_Locate Function

Function ID	1509
Type	Query
User API	Yes
Object-specific method	Yes
Object-independent method	Yes
Initialization method	Yes
Methods triggered	No
Triggered by EKG_MessageTriggeredAction function	No
Authorization	2

Usage

The EKG_Locate function acts on all objects in RODM with the specified field, regardless of the class the objects are in.

The EKG_Locate function works with fields of data types CharVar and IndexList that are created as public_indexed only. If you use the EKG_Locate function on a field named DisplayResourceName, RODM will return the Object IDs of all objects matching the search criteria regardless of case of the field or search criteria. For DBCS values, you can get unexpected matches.

EKG_LockObjectList — Lock List of Objects

Purpose

This function was previously used to obtain explicit locks for a list of objects. RODM now controls locking automatically, and this function is no longer necessary. This function remains available for compatibility with existing applications. No changes to existing applications that use this function are required.

Function Block Format

Table 105. Function Block for the EKG_LockObjectList Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Integer	In	Object_list_length
: First element, array of structure				
008	0	Structure	—	Object_array
008	8	ObjectID	In	Object_ID
016	4	Integer	Out	Reason_code[1]

Table 105. Function Block for the EKG_LockObjectList Function (continued)

Offset	Length	Type	Use	Parameter Name
: Second element, array of structure (if used)				
020	0	Structure	—	Object_array
020	8	ObjectID	In	Object_ID
028	4	Integer	Out	Reason_code[1]

Note: Function block contains Object_list_length array elements

See "Function Parameter Descriptions" on page 445 for more information about the parameters listed. See "Abstract Data Type Reference" on page 223 for more information about the abstract data types listed.

Examples

Table 106. Example Names for the EKG_LockObjectList Function

Example	Name
PL/I function block	EKG12002
PL/I response block	None
PL/I usage coding	EKG52002
C function block	EKG32002
C response block	None
C usage coding	EKG62002

Summary

Table 107. Summary of the EKG_LockObjectList Function

Function ID	2002
Type	Method API Service
User API	No
Object-specific method	No
Object-independent method	Yes
Initialization method	Yes
Methods triggered	No
Triggered by EKG_MessageTriggeredAction function	No
Authorization	None

Usage

For compatibility with existing applications, the value 0 is always returned in the Reason_code field.

EKG_MessageTriggeredAction — Trigger an Action by a Message

Purpose

This function runs a RODM function asynchronously. It enables an object-specific method to act on other objects in the data cache.

Function Block Format

Table 108. Function Block for the EKG_MessageTriggeredAction Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Pointer	In	Function_block_ptr

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 109. Example Names for the EKG_MessageTriggeredAction Function

Example	Name
PL/I function block	EKG12009
PL/I response block	None
PL/I usage coding	EKG52009
C function block	EKG32009
C response block	None
C usage coding	EKG62009

Summary

Table 110. Summary of the EKG_MessageTriggeredAction Function

Function ID	2009
Type	Method API Service
User API	No
Object-specific method	Yes
Object-independent method	Yes
Initialization method	Yes
Methods triggered	No
Triggered by EKG_MessageTriggeredAction function	No
Authorization	None

Usage

Not all functions can be run by the EKG_MessageTriggeredAction function. The entry “Triggered by EKG_MessageTriggeredAction function” in the Summary table for each function tells you whether that function can be run by this function.

The method that uses the EKG_MessageTriggeredAction function receives a return code and reason code that specifies whether the function request was accepted by RODM. However, the method cannot determine when the action takes place. To detect problems with methods triggered and functions run by the EKG_MessageTriggeredAction function, subscribe to the EKG_LastAsyncError field of the EKG_System and EKG_User classes. See “Asynchronous Error Notification” on page 325 for more information.

Functions run by the EKG_MessageTriggeredAction function cannot return a response block to the calling method.

This function is intended for use in object-specific methods; it enables the object-specific method to act on an object other than the object with which the method is associated. However, object-independent methods can also use this function.

EKG_OutputToLog — Output to Log

Purpose

This function writes a log record to the current RODM log data set. This enables methods to record error or diagnostic information.

Function Block Format

Table 111. Function Block for the EKG_OutputToLog Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Pointer	In	Log_message
008	2	Smallint	In	Message_CCSID

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 112. Example Names for the EKG_OutputToLog Function

Example	Name
PL/I function block	EKG12008
PL/I response block	None
PL/I usage coding	EKG52008
C function block	EKG32008
C response block	None
C usage coding	EKG62008

Summary

Table 113. Summary of the EKG_OutputToLog Function

Function ID	2008
Type	Method API Service
User API	No
Object-specific method	Yes
Object-independent method	Yes
Initialization method	Yes
Methods triggered	No
Triggered by EKG_MessageTriggeredAction function	No

Table 113. Summary of the EKG_OutputToLog Function (continued)

Authorization	None

Usage

RODM maintains a log (a VSAM entry sequence data set) where methods can write character strings (type 1 log records). This is the same log where RODM writes error records for error condition in RODM.

RODM places the method name, a time stamp, a unique transaction identifier, and the log record type at the beginning of the record in the RODM log.

EKG_QueryEntityStructure — Query Structure of an Entity

Purpose

This function queries the structure of an object or class and returns a list of its fields. The field list includes the field name, field ID, data type, and inheritance status.

Function Block Format

Table 114. Function Block for the EKG_QueryEntityStructure Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Pointer	In	Entity_access_info_ptr
008	4	—	—	Not used
012	2	—	—	Not used
014	2	Anonymous(2)	—	Reserved
016	4	—	—	Not used

Table 115. Response Block for the EKG_QueryEntityStructure Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Response_block_length
004	4	Integer	Out	Response_block_used
008	2	Smallint	Out	Field_info_element_size
010	2	Smallint	Out	Field_info_count
: First element, array of structure				
012	0	Structure	—	Field_info_array
012	4	FieldID	Out	Field_ID
016	2	Bit(16) bit 0 bit 1 bit 2	— Out Out Out	Bit_map • Private_public_flag • Local_inherited_flag • Indexed_flag
018	2	Smallint	Out	Data_type
020	67	ShortName	Out	Field_name
087	1	—	—	Reserved
: Second element, array of structure (if used)				
088	0	Structure	—	Field_info_array

Table 115. Response Block for the EKG_QueryEntityStructure Function (continued)

Offset	Length	Type	Use	Parameter Name
088	4	FieldID	Out	Field_ID
092	2	Bit(16) bit 0 bit 1 bit 2	— Out Out Out	Bit_map • Private_public_flag • Local_inherited_flag • Indexed_flag
094	2	Smallint	Out	Data_type
096	67	ShortName	Out	Field_name
161	1	—	—	Reserved

Note: Function block contains Field_info_count array elements

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 116. Example Names for the EKG_QueryEntityStructure Function

Example	Name
PL/I function block	EKG11503
PL/I response block	EKG21503
PL/I usage coding	EKG51503
C function block	EKG31503
C response block	EKG41503
C usage coding	EKG61503

Summary

Table 117. Summary of the EKG_QueryEntityStructure Function

Function ID	1503
Type	Query
User API	Yes
Object-specific method	Yes
Object-independent method	Yes
Initialization method	Yes
Methods triggered	No
Triggered by EKG_MessageTriggeredAction function	No
Authorization	2

Usage

The response data contains an array that consists of one array element for each field in the object or class. There are Field_info_count elements in the response block; each element is of size Field_info_element_size.

EKG_QueryField — Query a Field

Purpose

This function queries the value of a field on an object or a class.

Function Block Format

Table 118. Function Block for the EKG_QueryField Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Pointer	In	Entity_access_info_ptr
008	4	Pointer	In	Field_access_info_ptr
012	2	—	—	Not used
014	2	Anonymous(2)	—	Reserved
016	4	SelfDefiningDataPtr	In	Method_parms

Table 119. Response Block for the EKG_QueryField Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Response_block_length
004	4	Integer	Out	Response_block_used
008	2	Smallint	Out	Data_type
010	—	Anonymous	Out	Data

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 120. Example Names for the EKG_QueryField Function

Example	Name
PL/I function block	EKG11501
PL/I response block	EKG21501
PL/I usage coding	EKG51501
C function block	EKG31501
C response block	EKG41501
C usage coding	EKG61501

Summary

Table 121. Summary of the EKG_QueryField Function

Function ID	1501
Type	Query
User API	Yes
Object-specific method	Yes
Object-independent method	Yes
Initialization method	Yes

Table 121. Summary of the EKG_QueryField Function (continued)

Methods triggered	Query method for the target field triggered
Triggered by EKG_MessageTriggeredAction function	No
Authorization	2

Usage

If there is a query method on the field, the Method_parm field is passed to that query method when the method is run. If there is no query method on the field, the Method_parm field is ignored.

If the value subfield is queried and the data type returned is CharVar, the data string is immediately followed by a null terminating byte of X'00'. If the value subfield is queried and the data type returned is GraphicVar, the data string is immediately followed by a null terminating double-byte of X'0000'.

For a successful query, RODM returns a reason code that specifies whether the returned value is a local value or an inherited value.

Multiple field values can be queried using the EKG_QueryMultipleSubfields function.

EKG_QueryFieldID — Query Field Identifier

Purpose

This function returns a field ID from a specified field name.

Function Block Format

Table 122. Function Block for the EKG_QueryFieldID Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	—	—	Not used
008	4	Pointer	In	Field_access_info_ptr
012	2	—	—	Not used
014	2	Anonymous(2)	—	Reserved
016	4	—	—	Not used

Table 123. Response Block for the EKG_QueryFieldID Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Response_block_length
004	4	Integer	Out	Response_block_used
008	4	FieldID	Out	Field_ID

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 124. Example Names for the EKG_QueryFieldID Function

Example	Name
PL/I function block	EKG11505
PL/I response block	EKG21505
PL/I usage coding	EKG51505
C function block	EKG31505
C response block	EKG41505
C usage coding	EKG61505

Summary

Table 125. Summary of the EKG_QueryFieldID Function

Function ID	1505
Type	Query
User API	Yes
Object-specific method	Yes
Object-independent method	Yes
Initialization method	Yes
Methods triggered	No
Triggered by EKG_MessageTriggeredAction function	No
Authorization	2

Usage

The Field_ID in the Field_access_info_ptr is ignored for this function.

This function obtains a field ID from the specified field name. If the field name is not defined for any class, RODM issues return code 4 with reason code 56.

Because all identical field names defined across all classes in the RODM data cache share the same field ID, the class information is not necessary for this function to distinguish identical field names in different classes.

Note: To obtain the object ID associated with an object name, query the MyID field of the object under a specified class; to obtain the class ID associated with a class name, query the MyID field of the class.

EKG_QueryFieldName — Query a Field Name

Purpose

This function returns a field name from a specified field ID.

Function Block Format

Table 126. Function Block for the EKG_QueryFieldName Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID

Table 126. Function Block for the EKG_QueryFieldName Function (continued)

Offset	Length	Type	Use	Parameter Name
004	4	Pointer	In	Entity_access_info_ptr
008	4	Pointer	In	Field_access_info_ptr
012	2	—	—	Not used
014	2	Anonymous(2)	—	Reserved
016	4	—	—	Not used

Table 127. Response Block for the EKG_QueryFieldName Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Response_block_length
004	4	Integer	Out	Response_block_used
008	67	ShortName	Out	Field_name

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 128. Example Names for the EKG_QueryFieldName Function

Example	Name
PL/I function block	EKG11506
PL/I response block	EKG21506
PL/I usage coding	EKG51506
C function block	EKG31506
C response block	EKG41506
C usage coding	EKG61506

Summary

Table 129. Summary of the EKG_QueryFieldName Function

Function ID	1506
Type	Query
User API	Yes
Object-specific method	Yes
Object-independent method	Yes
Initialization method	Yes
Methods triggered	No
Triggered by EKG_MessageTriggeredAction function	No
Authorization	2

Usage

This function obtains a field name from the specified field ID in an object or class. If the field ID is not defined for the object or class, a warning message with a reason code is returned.

While all identical field names defined across all classes in the RODM data cache share the same field ID, not all identical field IDs share the same field name. However, all field IDs within a given object or class are unique within that object or class. Therefore, the object or class information is necessary to uniquely identify the field name from the specified field ID.

To obtain the object name associated with an object ID, query the MyName field of the object; to obtain the class name associated with a class ID, query the MyName field of the class.

You must set the Field_ID parameter in the field access information block for this function. The Field_name parameter in the field access information block is ignored for this function.

EKG_QueryFieldStructure — Query Structure of a Field

Purpose

This function queries the definition of a field and returns the data type, inheritance state, and subfield map of the specified field.

Function Block Format

Table 130. Function Block for the EKG_QueryFieldStructure Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Pointer	In	Entity_access_info_ptr
008	4	Pointer	In	Field_access_info_ptr
012	2	—	—	Not used
014	2	Anonymous(2)	—	Reserved
016	4	—	—	Not used

Table 131. Response Block for the EKG_QueryFieldStructure Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Response_block_length
004	4	Integer	Out	Response_block_used
008	2	Smallint	Out	Data_type
010	2	Smallint	Out	Inheritance_state
012	4	Bit(32)	Out	Subfield_map
016	4	Bit(32)	Out	Local_copy_map

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 132. Example Names for the EKG_QueryFieldStructure Function

Example	Name
PL/I function block	EKG11504
PL/I response block	EKG21504
PL/I usage coding	EKG51504
C function block	EKG31504
C response block	EKG41504
C usage coding	EKG61504

Summary

Table 133. Summary of the EKG_QueryFieldStructure Function

Function ID	1504
Type	Query
User API	Yes
Object-specific method	Yes
Object-independent method	Yes
Initialization method	Yes
Methods triggered	No
Triggered by EKG_MessageTriggeredAction function	No
Authorization	2

Usage

The value of the notify subfield is never inherited. If a notify subfield exists, it always contains a locally defined value. This value is initially null.

The values of subfields with data types ClassLinkList, ObjectLink, and ObjectLinkList are never inherited. If these subfields exist, they always contain locally defined values. These values are initially null.

The value subfield is always locally created. Its value can be inherited or locally defined. The value is initially inherited.

The values of the prev_val and timestamp subfields are never inherited. If these subfields exist, they always contain locally defined values. These values are initially null.

EKG_QueryFunctionBlockContents — Query Function Block Contents

Purpose

This method API function obtains a copy of the function block of the user API or method API function request that triggered this method. This function enables a triggered method to get information about the function that triggered it.

Function Block Format

Table 134. Function Block for the EKG_QueryFunctionBlockContents Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID

Table 135. Response Block for the EKG_QueryFunctionBlockContents Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Response_block_length
004	4	Integer	Out	Response_block_used
008	4	Integer	Out	Function_block_origin
012	—	Anonymous	Out	Function_block_copy

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 136. Example Names for the EKG_QueryFunctionBlockContents Function

Example	Name
PL/I function block	EKG12001
PL/I response block	EKG22001
PL/I usage coding	EKG52001
C function block	EKG32001
C response block	EKG42001
C usage coding	EKG62001

Summary

Table 137. Summary of the EKG_QueryFunctionBlockContents Function

Function ID	2001
Type	Method API Service
User API	No
Object-specific method	Yes
Object-independent method	Yes
Initialization method	No
Methods triggered	No
Triggered by EKG_MessageTriggeredAction function	No
Authorization	None

Usage

If this function is called by a change, query, or notify method, this function returns the function block contents of the function that caused the method to be triggered. For example, if an EKG_ChangeField function triggers a notify method, the

EKG_QueryFunctionBlockContents function issued by the notify method returns the function block of the EKG_ChangeField function.

If this function is called by an object-independent method, this function returns the function block contents of the EKG_TriggerOIMethod function.

If this function is called by a named method, this function returns the function block contents of the EKG_TriggerNamedMethod function.

The function block data returned by this function is put in Function_block_copy. The pointers in the function block point to the corresponding information blocks in the same Function_block_copy. The method using the EKG_QueryFunctionBlockContents function can use these pointers to get all the information contained in Function_block_copy.

Because all pointers in the returned function block are adjusted to point to the data in the response block, the method cannot use these pointers to change RODM data or the original function block.

The data referenced by the pointers in the returned function block is placed in the response block immediately following the copy of the function block.

If the size of the response block is not sufficient to contain all of the returned function block data, the Response_block_used field is set to the actual size required and the data in the response block is truncated.

If the new data value cannot be placed in the response block of a returned function block containing change API function data, the other function block data is provided but the New_data_ptr is set to null.

If either the new data value or the old data value cannot be placed in the response block of a returned function block containing swap API function data, the other function block data is provided and RODM does the following:

- If the value specified by the New_data_ptr pointer cannot be placed in the response block, RODM sets the New_data_ptr and the Old_data_ptr to null.
- Otherwise, the new data value is placed in the response block:
  - If the value specified by the Old_data_ptr pointer cannot be placed in the response block, RODM sets the Old_data_ptr to null.

A response block size deficiency is not considered to be a response block overflow condition. RODM returns the truncated data and the required data length but the method must reinitiate the request with a larger response block if it is to obtain the omitted data.

EKG_QueryMultipleSubfields — Query Multiple Value Subfields

Purpose

This function queries multiple value subfields for an object with a single call to the user API or the method API. This function queries object subfields, not class subfields. It does not trigger any associated query methods.

Function Block Format

Table 138. Function Block for the EKG_QueryMultipleSubfields Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Pointer	In	Entity_access_info_ptr
008	4	Integer	In	Number_of_subfields
Note: First element, array of structure				
012	0	Structure	—	Field_info_array
012	4	Pointer	In	Field_access_info_ptr
016	4	Anonymous(4)	—	Reserved
020	4	Pointer	Out	Response_block_reference
024	4	Integer	Out	Response_block_used
028	4	Integer	Out	Return_code
032	4	Integer	Out	Reason_code
Note: Second element, array of structure (if used)				
036	0	Structure	—	Field_info_array
036	4	Pointer	In	Field_access_info_ptr
040	4	Anonymous(4)	—	Reserved
044	4	Pointer	Out	Response_block_reference
048	4	Integer	Out	Response_block_used
052	4	Integer	Out	Return_code
056	4	Integer	Out	Reason_code

Note: Function block contains Number_of_subfields array elements

Table 139. Response Block for the EKG_QueryMultipleSubfields Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Response_block_length
004	4	Integer	Out	Response_block_used
008	0	Anonymous(1)	Out	Requested_info_array
Note: First and subsequent elements, array of requested information				
008	2	Smallint		Data_type
010	—	Anonymous		Data_value

Array notes:

- Response block contains Number_of_subfields array elements if all subfield queries are successful. Unsuccessful queries are not included in the array.
- The Response_block_used field in the function block defines the length of the corresponding element in the response block.
- The return code and reason code are for each individual subfield queried.

See "Function Parameter Descriptions" on page 445 for more information about the parameters listed. See "Abstract Data Type Reference" on page 223 for more information about the abstract data types listed.

Examples

Table 140. Example Names for the EKG_QueryMultipleSubfields Function

Example	Name
PL/I function block	EKG11508
PL/I response block	EKG21508
PL/I usage coding	EKG51508
C function block	EKG31508
C response block	EKG41508
C usage coding	EKG61508

Summary

Table 141. Summary of the EKG_QueryMultipleSubfields Function

Function ID	1508
Type	Method API Service
User API	Yes
Object-specific method	Yes
Object-independent method	Yes
Initialization method	Yes
Methods triggered	No
Triggered by EKG_MessageTriggeredAction function	No
Authorization	2

Usage

The EKG_QueryMultipleSubfields function does not trigger any query methods.

The value specified in the number_of_subfields field cannot exceed 100,000.

It is your responsibility to provide the Entity_access_info_block, the Response_block, the number of queried fields, and a list of field IDs or field names (which are specified in the Field_access_info_blocks—one block per field requested).

The return code and reason code returned in the transaction information block for the EKG_QueryMultipleSubfields function is the highest return code for any individual query and the first corresponding reason code.

If the response block overflow situation is encountered, all output length values (response_block_used parameters) are set by RODM, but pointer values (for example, response_block_reference parameters) for transaction results that are contained completely in the overflow buffer are set to null. When you retrieve the overflow block with EKG_QueryResponseBlockOverflow, it is your responsibility to parse that data using the length information returned on the original call. The overflow processing is only available to the user API; the method API for this function discards any overflow data.

If the subfield queried returns data of type CharVar, the data string is immediately followed by a null terminating byte of X'00'. If the subfield queried returns data of type GraphicVar, the data string is immediately followed by a null terminating double-byte of X'0000'.

After a successful query, RODM returns a reason code that specifies whether the returned value is a local value or an inherited value.

EKG_QueryNotifyQueue — Query Notification Queue

Purpose

This function returns the next notification block from the specified notification queue.

Function Block Format

Table 142. Function Block for the EKG_QueryNotifyQueue Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	8	SubscribeID	In	Notification_queue

Table 143. Response Block for the EKG_QueryNotifyQueue Function (Notification Block)

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Response_block_length
004	4	Integer	Out	Response_block_used
008	2	Smallint	Out	Notification_queue_count
010	2	Smallint	Out	Response_block_type
012	4	ClassID	Out	Class_ID
016	8	ObjectID	Out	Object_ID
024	4	FieldID	Out	Field_ID
028	2	Smallint	Out	Subfield
030	8	ApplicationID	Out	User_appl_ID
038	8	SubscribeID	Out	Notification_queue
046	8	MethodName	Out	Method_name
054	8	Anonymous(8)	Out	User_word
062	—	SelfDefining	Out	User_area

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 144. Example Names for the EKG_QueryNotifyQueue Function

Example	Name
PL/I function block	EKG11507
PL/I response block	EKG21507
PL/I usage coding	EKG51507

Table 144. Example Names for the EKG_QueryNotifyQueue Function (continued)

Example	Name
C function block	EKG31507
C response block	EKG41507
C usage coding	EKG61507

Summary

Table 145. Summary of the EKG_QueryNotifyQueue Function

Function ID	1507
Type	Query
User API	Yes
Object-specific method	No
Object-independent method	No
Initialization method	No
Methods triggered	No
Triggered by EKG_MessageTriggeredAction function	No
Authorization	2

Usage

If the queried notification queue is not empty, the first (oldest) notification block on the notification queue is returned in the response block, and that notification block is deleted from the notification queue. The Notification_queue_count field in the response block specifies the number of notification blocks in the notification queue prior to this function call. A Notification_queue_count value greater than zero indicates that a notification block was placed in the response block.

The Class_ID, Object_ID, Field_ID, and Subfield fields of the response block specify the object or class, field, and subfield where the method that generated the notification is located.

- If the Class_ID and Object_ID are both null, an object-independent method triggered the notification. In that case, the Field_ID and Subfield are set to zero.
- If the Object_ID is null, but the Class_ID is not null, the field is in the class.
- If the Object_ID field is not null, the Class_ID field specifies the object class, and the field is in the object.
- If the executing method that called the notification function was a query, change, or notify method, the Subfield field is set to the identifier of that type of subfield. In this case, the Field_ID field specifies the field that was possibly changed, thus causing this notification to be generated.
- If the Subfield field specifies the notify subfield, the field was changed and a notification method was triggered.
- If the executing method was a named method, the Subfield field is set to 1 for the value subfield.
- If the executing method was an object-independent method, the Subfield field is set to zero.

The User_appl_ID that is returned identifies the user that caused the notification to be triggered.

The Notification queue field contains the same notification queue name that was specified in the original subscription.

The User_word field might contain the same user information that was specified in the original subscription, but the notification method actually determines the value returned in this field.

The Method_name field specifies the name of the notifying method.

The User_area string contains a maximum of 32767 bytes of data supplied by the notifying method.

EKG_QueryObjectName — Query Object Name

Purpose

This function returns the object name of an object when you supply the object ID. This function can be used by object-specific methods only. The object-specific method can use this function to get the object name of any object, not just the object with which the method is associated.

Function Block Format

Table 146. Function Block for the EKG_QueryObjectName Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	8	ObjectID	In	Object_ID

Table 147. Response Block for the EKG_QueryObjectName Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Response_block_length
004	4	Integer	Out	Response_block_used
008	67	ShortName	Out	Class_name
075	1	—	—	Reserved
076	—	ObjectName	Out	Object_name

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 148. Example Names for the EKG_QueryObjectName Function

Example	Name
PL/I function block	EKG12011
PL/I response block	EKG22011
PL/I usage coding	EKG52011
C function block	EKG32011
C response block	EKG42011
C usage coding	EKG62011

Summary

Table 149. Summary of the EKG_QueryObjectName Function

Function ID	2011
Type	Method API Service
User API	No
Object-specific method	Yes
Object-independent method	No
Initialization method	No
Methods triggered	No
Triggered by EKG_MessageTriggeredAction function	No
Authorization	None

Usage

Object-specific methods have access to the ObjectIDs of other objects through link fields. This function enables the object-specific method to associate the object name with an object ID. The EKG_MessageTriggeredAction function enables the object-specific method to then take some action on another object.

This function does not trigger the query method on the MyName field if one is present.

EKG_QueryResponseBlockOverflow — Query for Response Block Overflow

Purpose

This function queries the response block overflow buffer. The overflow buffer contains excess output from a user application function that previously caused a response block overflow.

Function Block Format

Table 150. Function Block for the EKG_QueryResponseBlockOverflow Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Anonymous(4)	—	Reserved
008	8	TransID	In	Correlation_ID

Table 151. Response Block for the EKG_QueryResponseBlockOverflow Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Response_block_length
004	4	Integer	Out	Response_block_used
008	—	Anonymous	Out	Data

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 152. Example Names for the EKG_QueryResponseBlockOverflow Function

Example	Name
PL/I function block	EKG11510
PL/I response block	EKG21510
PL/I usage coding	EKG51510
C function block	EKG31510
C response block	EKG41510
C usage coding	EKG61510

Summary

Table 153. Summary of the EKG_QueryResponseBlockOverflow Function

Function ID	1510
Type	Query
User API	Yes
Object-specific method	No
Object-independent method	No
Initialization method	No
Methods triggered	No
Triggered by EKG_MessageTriggeredAction function	No
Authorization	2

Usage

The Data field in the response block contains the continuation of the data in the response block that was returned by the original function. For data types that have length fields or headers, the length field or header is usually stored in the original response block.

RODM provides an overflow buffer for functions called from user application programs only. For query methods that return a value to a user API query request, all data output to the response block by the method is returned to the caller. If the amount of data exceeds the size of the user-supplied response block, RODM places the excess data in the response block overflow buffer.

For all other methods and for query methods that are triggered by a method API query request, all data output to the response block by the method might not be returned to the caller. If the amount of data exceeds the size of the method-supplied response block, RODM truncates the data to the size of the response block and discards the excess.

If RODM places data in the overflow buffer, you must use the EKG_QueryResponseBlockOverflow function to retrieve the contents of the buffer before RODM accepts any other function requests using the specified access block.

You can make only one call for the overflow buffer to retrieve the overflow data. If the Response_block_length specified is less than the amount of data in the buffer, RODM fills the response block based on the specified size and discards any remaining data.

Response block overflow buffers maintained by RODM are identified by Transaction_IDs. Specify the Transaction_ID value returned in the transaction information block of the function that caused the overflow as the Correlation_ID parameter for this function request.

If you want to discard the data in the overflow buffer without using it, set Response_block_length to 0 when you call the EKG_QueryResponseBlockOverflow function.

See "Response Block" on page 314 for additional information about response block overflow.

EKG_QuerySubfield — Query a Subfield

Purpose

This function queries the value of a subfield of a field on an object or a class.

Function Block Format

Table 154. Function Block for EKG_QuerySubfield Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Pointer	In	Entity_access_info_ptr
008	4	Pointer	In	Field_access_info_ptr
012	2	Smallint	In	Subfield
014	2	Anonymous(2)	—	Reserved
016	4	—	—	Not used

Table 155. Response Block for the EKG_QuerySubfield Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Response_block_length
004	4	Integer	Out	Response_block_used
008	2	Smallint	Out	Data_type
010	—	Anonymous	Out	Data

See "Function Parameter Descriptions" on page 445 for more information about the parameters listed. See "Abstract Data Type Reference" on page 223 for more information about the abstract data types listed.

Examples

Table 156. Example Names for the EKG_QuerySubfield Function

Example	Name
PL/I function block	EKG11502
PL/I response block	EKG21502

Table 156. Example Names for the EKG_QuerySubfield Function (continued)

Example	Name
PL/I usage coding	EKG51502
C function block	EKG31502
C response block	EKG41502
C usage coding	EKG61502

Summary

Table 157. Summary of the EKG_QuerySubfield Function

Function ID	1502
Type	Query
User API	Yes
Object-specific method	Yes
Object-independent method	Yes
Initialization method	Yes
Methods triggered	No
Triggered by EKG_MessageTriggeredAction function	No
Authorization	2

Usage

Querying a RODM managed subfield, prev_val or timestamp, for example, differs from querying other subfields. RODM-managed subfield values always correspond to their respective value subfield. If an object has a local value for the value subfield and a managed subfield exists, that managed subfield has either of the following two values:

- If the prev_val or timestamp existed at the time the field value was set, the prev_val or timestamp subfields have a local value reflecting appropriate values.
- If these subfields were created subsequent to the last setting of the local field value, these subfields contain a Null value.

When a RODM-managed subfield is queried:

- If the field has a local value and the managed subfield exists, its local value is returned.
- If the field has no local value, a value for the managed subfield is determined from the inherited field.

If the subfield queried returns data of type CharVar, the data string is immediately followed by a null terminating byte of X'00'. If the subfield queried returns data of type GraphicVar, the data string is immediately followed by a null terminating double-byte of X'0000'.

Notification subfield values are never inherited. The EKG_QuerySubfield function, when triggered against a notification subfield, returns a value only if the subfield is locally defined. Subfields with data types ClassLinkList, ObjectLink, and ObjectLinkList are never inherited. The EKG_QuerySubfield function, when triggered against a value, prev_val, or timestamp subfield, returns a value only if the subfield is locally defined. Otherwise the query returns the null value.

After a successful query, RODM returns a reason code that specifies whether the returned value is a local value or an inherited value.

EKG_ResponseBlock — Output to Response Block

Purpose

This function writes data to the current response block. The data is of the SelfDefining type.

Function Block Format

Table 158. Function Block for the EKG_ResponseBlock Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	SelfDefiningDataPtr	In	Data_to_be_returned

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 159. Example Names for the EKG_ResponseBlock Function

Example	Name
PL/I function block	EKG12004
PL/I response block	None
PL/I usage coding	EKG52004
C function block	EKG32004
C response block	None
C usage coding	EKG62004

Summary

Table 160. Summary of the EKG_ResponseBlock Function

Function ID	2004
Type	Method API Service
User API	No
Object-specific method	Query and named only
Object-independent method	Yes
Initialization method	No
Methods triggered	No
Triggered by EKG_MessageTriggeredAction function	No
Authorization	None

Usage

Each time an object-independent or named method runs this function, a new SelfDefining data string is appended to the current response block. Each time a query method runs this function, a new SelfDefining data string overwrites the current response block.

If the size of the data pointed to by Data_to_be_returned is larger than the size of the current response block, RODM truncates the data to the size of the response block and issues a warning return code. This function does not write to the response block overflow buffer.

The EKG_ResponseBlock function writes data to the current response block. For this function, the current response block is the response block of the method that issued this function. Because methods can call other methods, this might not be the same as the function block of the method that was first run.

When this function is used by a query method, the following actions are taken by RODM:

- RODM uses the length field from the self-defining string to determine response block storage requirements and removes that field from the data. This means that the application sees the exact same format of data in the response block regardless of whether the data was provided directly by RODM or by a method through the use of this function.
- The value returned to the user through this self-defining string cannot be a null string (that is, the length of the self-defining string must be greater than 2). If the self-defining string is not formatted properly, RODM does not modify the response block.

EKG_RevertToInherited — Revert to Inherited Value

Purpose

This function deletes the locally defined value of a field or subfield of an object or class. This causes the field or subfield to inherit the value defined on its parent class.

Function Block Format

Table 161. Function Block for the EKG_RevertToInherited Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Pointer	In	Entity_access_info_ptr
008	4	Pointer	In	Field_access_info_ptr
012	2	Smallint	In	Subfield

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 162. Example Names for the EKG_RevertToInherited Function

Example	Name
PL/I function block	EKG11411

Table 162. Example Names for the EKG_RevertToInherited Function (continued)

Example	Name
PL/I response block	None
PL/I usage coding	EKG51411
C function block	EKG31411
C response block	None
C usage coding	EKG61411

Summary

Table 163. Summary of the EKG_RevertToInherited Function

Function ID	1411
Type	Action
User API	Yes
Object-specific method	Yes
Object-independent method	Yes
Initialization method	Yes
Methods triggered	No
Triggered by the EKG_MessageTriggeredAction function	Yes
Authorization	3

Usage

Fields and subfields which are locally created on a class are not inherited from a parent class. Because these fields and subfields are not inherited, there is no inherited value for them to revert to. RODM issues a warning return code if the target of this function is locally created.

You cannot use the EKG_RevertToInherited function with any of the following fields or subfields:

- System-defined fields
- Fields of data type ObjectLink or ObjectLinkList
- Notify subfield
- Prev_val subfield
- Timestamp subfield
- System fields defined as read-only under the following system classes:
  - EKG_System class
  - EKG_User class
  - EKG_Method class
  - EKG_NotificationQueue class

If the prev_val or timestamp subfields are defined and the value subfield is the target of the EKG_RevertToInherited function, the prev_val and timestamp subfields also revert to inherited values. See "RODM Subfields" on page 213 for more information about inheritance of the prev_val and timestamp subfields.

Specify the Subfield parameter as 0 to cause all subfields of the field except the notify subfield to revert to their inherited values. You cannot specify the Subfield parameter as 4 (notify), 5 (prev_val), or 6 (timestamp).

When reverting to inherited values, the subfields of the same field can inherit values from different levels of parent classes. For example, the value of the value subfield can be inherited from the immediate parent class, and the value of the query subfield can be inherited from the parent class of the parent class.

EKG_SendNotification — Send a Notification

Purpose

This function sends a notification block to a specified notification queue when the value of a field within an object or class changes.

Function Block Format

Table 164. Function Block for the EKG_SendNotification Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	8	ApplicationID	In	User_appl_ID
012	8	SubscribeID	In	Notification_queue
020	8	Anonymous(8)	In	User_word
028	4	SelfDefiningDataPtr	In	Method_output_message

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 165. Example Names for the EKG_SendNotification Function

Example	Name
PL/I function block	EKG12005
PL/I response block	None
PL/I usage coding	EKG52005
C function block	EKG32005
C response block	None
C usage coding	EKG62005

Summary

Table 166. Summary of the EKG_SendNotification Function

Function ID	2005
Type	Method API Service
User API	No
Object-specific method	Yes
Object-independent method	Yes
Initialization method	No
Methods triggered	No
Triggered by EKG_MessageTriggeredAction function	No

Table 166. Summary of the EKG_SendNotification Function (continued)

Authorization	None

Usage

This function creates a notification block and places the notification block in the specified Notification_queue for the specified User_appl_ID. If the specified Notification_queue is empty, RODM posts the user's ECB associated with this queue.

For more information about notification, see “EKG_AddNotifySubscription — Add Notification Subscription” on page 373, “EKG_DeleteNotifySubscription — Delete Notification Subscription” on page 393, and “EKG_QueryNotifyQueue — Query Notification Queue” on page 421. If the posting of the user's ECB for the notification queue fails, RODM purges all notification queues and subscriptions based on the value of the EKG_StopMode field in the EKG_User_Class object. See “EKG_User Class” on page 201 for the possible values of EKG_StopMode.

EKG_SetReturnCode — Set Return and Reason Codes

Purpose

This function sets the return code and reason code that a method returns to the caller of the method.

Function Block Format

Table 167. Function Block for the EKG_SetReturnCode Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Integer	In	Value_for_return_code
008	4	Integer	In	Value_for_reason_code

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 168. Example Names for the EKG_SetReturnCode Function

Example	Name
PL/I function block	EKG12006
PL/I response block	None
PL/I usage coding	EKG52006
C function block	EKG32006
C response block	None
C usage coding	EKG62006

Summary

Table 169. Summary of the EKG_SetReturnCode Function

Function ID	2006

Table 169. Summary of the EKG_SetReturnCode Function (continued)

Type	Method API Service
User API	No
Object-specific method	Yes
Object-independent method	Yes
Initialization method	Yes
Methods triggered	No
Triggered by EKG_MessageTriggeredAction function	No
Authorization	None

Usage

The EKG_SetReturnCode function changes the return code of the caller to the value of the Value_for_return_code parameter if he value of Value_for_return_code is greater than the previous value of the return code. This function sets the value of the reason code of the caller to the value of the Value_for_reason_code parameter if the return code was changed.

The value of Value_for_return_code can be 0, 4, 8, or 12. The value of Value_for_reason_code can be from 0 to 65535. If you write methods that issue reason codes, use reason codes in the range 49152-65535.

Use the following guidelines for any return codes issued by methods that you write:

Return Code	**Meaning**
0	If reason code is also 0, the operation was successful and there are no complications. If the reason code is not 0, the operation was successful, but there are messages which might be logged.
4	A problem was encountered, retry the request or function later. The reason code might supply more information.
8	The request or function failed because of a logic error. Do not retry the request or function. The reason code might supply more information.
12	The request or function failed because RODM is not available. Do not retry the request or function. The reason code might supply more information.

If the method that calls EKG_SetReturnCode is triggered from within a transaction that is initiated by a function that is contained in the list of an EKG_ExecuteFunctionList user API call, the return code and the reason code are propagated to the individual return code and reason code fields for that function in the list. In addition, if this return code is the highest return code of all functions in the list, this return code and reason code become the EKG_ExecuteFunctionList user API transaction return code and reason code set in the transaction information block.

When the EKG_SetReturnCode function is called and the specified return code is greater than or equal to EKG_MLogLevel in the EKG_User class object, RODM writes a type-3 log record for object-specific methods and a type-4 log record for object-independent methods. If this function is requested by a method running asynchronously, RODM compares the return code to the MLOG_LEVEL customization parameter and then writes the log record as described above. When a log record is written from a method that is running asynchronously, RODM sets the EKG_LastAsyncError field to the return code and triggers notification methods for all applications that are subscribed to this field.

For more information about how RODM determines return and reason codes, see “Error Conditions in Transactions” on page 317.

Method writers must be aware of the implications of issuing return and reason codes from methods. See “Error Conditions in Transactions” on page 317 for information about how an application might interpret reason and return codes that are returned by methods.

EKG_Stop — Stop RODM

Purpose

This function stops the RODM program that you are connected to. You can optionally specify that RODM perform a checkpoint operation before stopping.

Function Block Format

Table 170. Function Block for the EKG_Stop Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	2	Smallint	In	Stop_type

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 171. Example Names for the EKG_Stop Function

Example	Name
PL/I function block	EKG11202
PL/I response block	None
PL/I usage coding	EKG51202
C function block	EKG31202
C response block	None
C usage coding	EKG61202

Summary

Table 172. Summary of the EKG_Stop Function

Function ID	1202
Type	Control
User API	Yes
Object-specific method	No
Object-independent method	No
Initialization method	No

Table 172. Summary of the EKG_Stop Function (continued)

Methods triggered	Notification methods installed on the EKG_LastCheckpointID field are triggered only if the checkpoint is successful. Notification methods installed on the EKG_LastCheckpointResult field are triggered whenever a checkpoint is requested. Notification methods cannot be installed on any other fields.
Triggered by the EKG_MessageTriggeredAction function	No
Authorization	6

Usage

After RODM is stopped by the use of this function, it can be restarted only with an operator command.

EKG_SwapField — Swap a Field

Purpose

This function compares the value of the target field with a specified test value. If they are equal, this function changes the value of the target field to the specified new value.

Function Block Format

Table 173. Function Block for the EKG_SwapField Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Pointer	In	Entity_access_info_ptr
008	4	Pointer	In	Field_access_info_ptr
012	2	—	—	Not used
014	2	Smallint	In	Data_type
016	4	Integer	In	New_char_data_length
020	4	Pointer	In	New_data_ptr
024	4	Integer	In	Old_char_data_length
028	4	Pointer	In	Old_data_ptr
032	4	SelfDefiningDataPtr	In	Method_parms

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 174. Example Names for the EKG_SwapField Function

Example	Name
PL/I function block	EKG11402
PL/I response block	None
PL/I usage coding	EKG51402

Table 174. Example Names for the EKG_SwapField Function (continued)

Example	Name
C function block	EKG31402
C response block	None
C usage coding	EKG61402

Summary

Table 175. Summary of the EKG_SwapField Function

Function ID	1402
Type	Action
User API	Yes
Object-specific method	No
Object-independent method	No
Initialization method	No
Methods triggered	Notification and Change methods triggered
Triggered by the EKG_MessageTriggeredAction function	Yes
Authorization	3

Usage

RODM compares the value of the field that is the target of this function with the test value pointed to by Old_data_ptr. If the values are equal, RODM changes the value of the target field to the new value pointed to by New_data_ptr. If the values are not equal, RODM does not change the value of the field and issues return code 8 with reason code 39.

The data type of the new data must be the same as the data type of the target field. The EKG_SwapField function cannot be used for fields with a data type of ObjectID, ObjectIDList, ObjectLink, ObjectLinkList, ClassID, ClassIDList, or ClassLinkList.

If New_data_ptr is null, RODM sets the field to the null value for its data type.

If a change method is defined for the target field, RODM triggers the change method if the value pointed to by Old_data_ptr is equal to the value of the target field. If RODM triggers a change method, RODM passes the value of New_data_ptr to the change method instead of changing the value of the field.

If notification methods are defined for the target field, RODM triggers the notification methods when the target field is successfully changed by this function or by the change method for the target field. If the target field is on an object, RODM also triggers the notification methods defined for the same field in the object's parent class.

The EKG_SwapField function issues return code 0 if it successfully updates the value of the target field. The reason code indicates the details of the change:

Reason code	Explanation
0	A local value existed and was changed.

26 The existing value is the same as the new value.

142 An inherited value existed and was replaced by a local value.

If both 0 (zero) and 26 or both 26 and 142 can be issued, RODM always issues 26.

EKG_SwapSubfield — Swap a Subfield

Purpose

This function compares the value of the target subfield with a specified test value. If they are equal, this function changes the value of the target subfield to the specified new value.

Function Block Format

Table 176. Function Block for the EKG_SwapSubfield Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Pointer	In	Entity_access_info_ptr
008	4	Pointer	In	Field_access_info_ptr
012	2	Smallint	In	Subfield
014	2	Smallint	In	Data_type
016	4	Integer	In	New_char_data_length
020	4	Pointer	In	New_data_ptr
024	4	Integer	In	Old_char_data_length
028	4	Pointer	In	Old_data_ptr
032	4	—	—	Not used

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 177. Example Names for the EKG_SwapSubfield Function

Example	Name
PL/I function block	EKG11404
PL/I response block	None
PL/I usage coding	EKG51404
C function block	EKG31404
C response block	None
C usage coding	EKG61404

Summary

Table 178. Summary of the EKG_SwapSubfield Function

Function ID	1404
Type	Action
User API	Yes

Table 178. Summary of the EKG_SwapSubfield Function (continued)

Object-specific method	No
Object-independent method	No
Initialization method	No
Methods triggered	No
Triggered by the EKG_MessageTriggeredAction function	Yes
Authorization	3

Usage

RODM compares the value of the subfield that is the target of this function with the test value pointed to by Old_data_ptr. If the values are equal, RODM changes the value of the target subfield to the new value pointed to by New_data_ptr. If the values are not equal, RODM does not change the value of the subfield and issues return code 8 with reason code 39.

The data type of the new data must be the same as the data type of the existing subfield. The EKG_SwapSubfield function cannot be used for subfields with a data type of ObjectID, ObjectIDList, ObjectLink, ObjectLinkList, ClassID, ClassIDList, or ClassLinkList.

If New_data_ptr is null, RODM sets the subfield to the null value for its data type.

RODM does not trigger any methods or update the prev_val and timestamp subfields when the value of a subfield is changed by this function.

The EKG_SwapSubfield function issues return code 0 (zero) if it successfully updates the value of the target subfield. The reason code indicates the details of the change:

Reason code	Explanation
0	A local value existed and was changed.
26	The existing value is the same as the new value.
142	An inherited value existed and was replaced by a local value.

If both 0 (zero) and 26 or both 26 and 142 can be issued, RODM always issues 26.

EKG_TriggerNamedMethod — Trigger a Named Method

Purpose

This function triggers a named method within a specified object or class.

Function Block Format

Table 179. Function Block for the EKG_TriggerNamedMethod Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Pointer	In	Entity_access_info_ptr
008	4	Pointer	In	Field_access_info_ptr
012	4	SelfDefiningDataPtr	In	Method_parms

Table 180. Response Block for the EKG_TriggerNamedMethod Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Response_block_length
004	4	Integer	Out	Response_block_used
008	—	Anonymous	Out	Concat_of_strings

See "Function Parameter Descriptions" on page 445 for more information about the parameters listed. See "Abstract Data Type Reference" on page 223 for more information about the abstract data types listed.

Examples

Table 181. Example Names for the EKG_TriggerNamedMethod Function

Example	Name
PL/I function block	EKG11415
PL/I response block	EKG21415
PL/I usage coding	EKG51415
C function block	EKG31415
C response block	EKG41415
C usage coding	EKG61415

Summary

Table 182. Summary of the EKG_TriggerNamedMethod Function

Function ID	1415
Type	Action
User API	Yes
Object-specific method	Yes
Object-independent method	Yes
Initialization method	Yes
Methods triggered	No
Triggered by the EKG_MessageTriggeredAction function	Yes
Authorization	5 (trigger EKG_Refresh named method) 3 (trigger other named method)

Usage

The Field_access_info_ptr must point to a field of type MethodSpec. The method_parameter_list of this MethodSpec field becomes the long-lived parameters of the named method. The SelfDefining string pointed to by the Method_Parms parameter becomes the short-lived parameters sent to the named method. This SelfDefining string has a maximum length of 254 bytes.

A named method can act only on fields in the object or class in which the named method is defined.

If a named method causes an overflow in the response block, the named method itself will receive a return code and reason code for the overflow. However, the method might not pass this return code and reason code back to the program that triggered the method. Always compare the Response_block_length parameter with the Response_block_used parameter returned in the response block if a named method is triggered. If the value of the Response_block_used parameter is larger than the value of the Response_block_length parameter, an overflow occurred.

EKG_TriggerOIMethod — Trigger an Object-Independent Method

Purpose

This function triggers an object-independent method.

Function Block Format

Table 183. Function Block for the EKG_TriggerOIMethod Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	8	MethodName	In	Method_name
012	4	SelfDefiningDataPtr	In	Method_parms

Table 184. Response Block for the EKG_TriggerOIMethod Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Response_block_length
004	4	Integer	Out	Response_block_used
008	—	Anonymous	Out	Concat_of_strings

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 185. Example Names for the EKG_TriggerOIMethod Function

Example	Name
PL/I function block	EKG11416
PL/I response block	EKG21416
PL/I usage coding	EKG51416
C function block	EKG31416
C response block	EKG41416
C usage coding	EKG61416

Summary

Table 186. Summary of the EKG_TriggerOIMethod Function

Function ID	1416
Type	Action
User API	Yes

Table 186. Summary of the EKG_TriggerOIMethod Function (continued)

Object-specific method	No
Object-independent method	Yes
Initialization method	Yes
Methods triggered	No
Triggered by the EKG_MessageTriggeredAction function	Yes
Authorization	3

Usage

The field pointed to by Method_parms has a maximum length of 32767 bytes.

An object-independent method must be installed by creating a method object under the EKG_Method class before it can be triggered by this function.

If an object-independent method causes an overflow in the response block, the object-independent method itself will receive a return code and reason code for the overflow. However, the method might not pass this return code and reason code back to the program that triggered the method. Always compare the Response_block_length parameter with the Response_block_used parameter returned in the response block if an object-independent method is triggered. If the value of the Response_block_used parameter is larger than the value of the Response_block_length parameter, an overflow occurred.

EKG_UnlinkNoTrigger, EKG_UnlinkTrigger — Unlink Two Objects

Purpose

These functions delete a link between two objects. The EKG_UnlinkTrigger function triggers change methods and notification methods; the EKG_UnlinkNoTrigger function does not.

Function Block Format

Table 187. Function Block for the EKG_UnlinkNoTrigger Function and the EKG_UnlinkTrigger Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID
004	4	Pointer	In	Entity_access_info_ptr_1
008	4	Pointer	In	Field_access_info_ptr_1
012	4	Pointer	In	Entity_access_info_ptr_2
016	4	Pointer	In	Field_access_info_ptr_2
000	4	Pointer	In	Method_parms[1]

:

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 188. Example Names for the EKG_UnlinkNoTrigger Function and the EKG_Unlink Trigger Function

Example	Name
PL/I function block (EKG_UnlinkTrigger)	EKG11407
PL/I function block (EKG_UnlinkNoTrigger)	EKG11408
PL/I response block	None
PL/I usage coding (EKG_UnlinkTrigger)	EKG51407
PL/I usage coding (EKG_UnlinkTrigger)	EKG51407
PL/I usage coding (EKG_UnlinkNoTrigger)	EKG51408
C function block (EKG_UnlinkTrigger)	EKG31407
C function block (EKG_UnlinkNoTrigger)	EKG31408
C response block	None
C usage coding (EKG_UnlinkTrigger)	EKG61407
C usage coding (EKG_UnlinkNoTrigger)	EKG61408

Summary

Table 189. Summary of the EKG_UnlinkNoTrigger Function and the EKG_UnlinkTrigger Function

Function ID EKG_UnlinkNoTrigger EKG_UnlinkTrigger	1408 1407
Type	Action
User API	Yes
Object-specific method	No
Object-independent method	Yes
Initialization method	Yes
Methods triggered EKG_UnlinkTrigger EKG_UnlinkNoTrigger	Change methods and notification methods No
Triggered by the EKG_MessageTriggeredAction function	Yes
Authorization	3

Usage

No assumption can be made regarding the order of links within a field of type ObjectLinkList.

The fields being unlinked must be of type ObjectLink or ObjectLinkList. The fields must have been linked using the EKG_LinkNoTrigger function or the EKG_LinkTrigger function. An ObjectLink field has only one link. An ObjectLinkList field can have more than one link for a field.

Do not use EKG_UnlinkNoTrigger with GMFHS resources.

When the EKG_UnlinkTrigger function is issued, the unlink operation is performed before the notification methods are triggered. If there are change methods defined on one or both of the fields to be unlinked, the unlink proceeds after the change methods, but only if one of the following is true:

- Both change methods explicitly set a zero return code with EKG_SetReturnCode.
- Neither change method sets a return code. In this case, RODM assumes a zero return code and the unlink proceeds.

If the unlink operation does not proceed, the notification methods are not triggered. If the fields are successfully unlinked, the notification methods are triggered in the following order:

1. Notification methods for the field specified by Field_access_info_ptr_1
2. Notification methods for the field specified by Field_access_info_ptr_2
3. Notification methods for the parent class of the first field
4. Notification methods for the parent class of the second field

EKG_UnlockAll — Unlock All Held Entities

Purpose

This function was previously used to free all locks held by a level-1 object-independent method. RODM now controls locking automatically, and this function is no longer necessary. This function remains available for compatibility with existing applications. No changes to existing applications that use this function are required.

Function Block Format

Table 190. Function Block for the EKG_UnlockAll Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 191. Example Names for the EKG_UnlockAll Function

Example	Name
PL/I function block	EKG12003
PL/I response block	None
PL/I usage coding	EKG52003
C function block	EKG32003
C response block	None
C usage coding	EKG62003

Summary

Table 192. Summary of the EKG_UnlockAll Function

Function ID	2003
Type	Method API Service
User API	No
Object-specific method	No
Object-independent method	Yes

Table 192. Summary of the EKG_UnlockAll Function (continued)

Initialization method	Yes
Methods triggered	No
Triggered by EKG_MessageTriggeredAction function	No
Authorization	None

EKG_WhereAmI — Where Am I

Purpose

This function returns the class, object, field, and subfield to which the method name is assigned and the context in which the object-specific method is being run.

Function Block Format

Table 193. Function Block for the EKG_WhereAmI Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Function_ID

Table 194. Response Block for the EKG_WhereAmI Function

Offset	Length	Type	Use	Parameter Name
000	4	Integer	In	Response_block_length
004	4	Integer	Out	Response_block_used
008	4	ClassID	Out	Class_ID
012	8	ObjectID	Out	Object_ID
020	4	FieldID	Out	Field_ID
024	2	Smallint	Out	Subfield
026	2	Anonymous(2)	—	Reserved
028	8	ObjectID	Out	Requesting_method_ID

See “Function Parameter Descriptions” on page 445 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.

Examples

Table 195. Example Names for the EKG_WhereAmI Function

Example	Name
PL/I function block	EKG12007
PL/I response block	EKG22007
PL/I usage coding	EKG52007
C function block	EKG32007
C response block	EKG42007
C usage coding	EKG62007

Summary

Table 196. Summary of the EKG_WhereAmI Function

Function ID	2007
Type	Method API Service
User API	No
Object-specific method	Yes
Object-independent method	No
Initialization method	No
Methods triggered	No
Triggered by EKG_MessageTriggeredAction function	No
Authorization	None

Usage

The Subfield parameter indicates the type of method. The Subfield parameter is set to 1 for named methods.

The Object_ID parameter is set to null if the method is defined on a class.

Function Parameter Descriptions

`Bit_map`
A bit map of flags describing a field. Bit_map is made up of the Private_public_flag and Local_inherited_flag.

`Change_status`
The Change_status parameter is used to inform a method whether or not the value of a field has changed.

`Class_access_info_ptr`
The Class_access_info_ptr is a pointer to an entity access information block where only the class information is used by this function call. The object information in that access block must be set to null values if the naming_count information is set to zero.

`Class_ID`
The class identifier.

`Class_name`
The name of the class this function acts on.

`Concat_of_strings`
A response data string of the Anonymous type. The string is a concatenation of zero or more SelfDefining data strings.

`Correlation_ID`
The unique ID of a transaction assigned by RODM.

`Data`
The data returned by the RODM function. This data is of type Data_type. For the Data parameter of an overflow block, the data type is specified in the original response block for the function that caused the overflow.

`Data_to_be_returned`
The Data_to_be_returned parameter must be set by the caller to point at whatever is to be concatenated into the data area of the response block.

Data_type
The RODM abstract data type of the specified parameter.

Entity_access_info_ptr
Pointer to the entity access information block that specifies the entity this function acts on.

Entity_access_info_ptr_1
The pointer to the entity access information block that specifies the first entity this function acts on.

Entity_access_info_ptr_2
The pointer to the entity access information block that specifies the second entity this function acts on.

Field_access_info_ptr
The pointer to the field access information block that specifies the field of the object this function acts on.

Field_access_info_ptr_1
The pointer to the field access information block that specifies the field of the first object this function acts on.

Field_access_info_ptr_2
The pointer to the field access information block that specifies the field of the second object this function acts on.

Field_ID
The field identifier.

Field_info_array
For EKG_QueryEntityStructure, an array of parameters describing the fields that make up an object or a class. For EKG_QueryMultipleSubfields, an array of fields whose value subfields will be queried.

Field_info_count
The number of fields in Field_info_array.

Field_info_element_size
The size of each element of Field_info_array.

Field_name
The name of the field. Variable length field with maximum length 67 bytes.

Field_type_flag
A Field_type_flag specifies whether the new field is to be public, private, or public-indexed. Valid values are:

Value	Meaning
1	Public
2	Private
3	Public-indexed

Function_block_copy
A copy of the queried function block. The Function_block_copy parameter contains a copy of the function block for the function that triggered the executing method.

Function_block_origin
The Function_block_origin parameter specifies whether the originating function was called by a user application or by a method. Valid values are:

Value	Meaning
1	User application

2	Method

`Function_block_ptr`
The pointer to the function block for a function to be run. See the description of the specific function for the format of the function block.

`Function_ID`
The function ID that identifies this function to RODM.

`Function_info_array`
The array of functions to be run.

`Indexed_data_length`
Length of the indexed data that RODM is attempting to locate.

`Indexed_data_ptr`
Pointer to the indexed data that RODM is attempting to locate. Indexed data is of type CharVar or IndexList. Indexed_data_ptr must point to the first byte of the character data of a CharVar data value or an individual IndexList data item. The length of the character string must be specified in Indexed_data_length.

`Inheritance_state`
The value of this field is always 1.

`Last_checkpoint_ID`
The transaction ID of the last checkpoint request. The Last_checkpoint_ID is set to zero when RODM is cold-started.

`Local_copy_map`
The Local_copy_map is a bit map defined as follows (bits are numbered 1–32 from left to right). RODM sets a Local_copy_map bit to 1 in an output block to indicate that the corresponding subfield contains locally-defined data.

Bit	Subfield
1	Value
2	Query
3	Change
4	Notify
5	Prev_val
6	Timestamp
7–32	Reserved

`Local_inherited_flag`
A flag that specifies whether a field is locally defined or is inherited from a parent class. Valid values are:

Value	Meaning
0	Locally defined
1	Inherited

`Log_message`
The Log_message parameter points to the character string to be written to the RODM log. This is an AnonymousVar string of a maximum 32709 bytes.

`Long_lived_parm`
This is the pointer to long-lived-parameters passed to the notification methods. The parameters identified by this pointer have a maximum length of 254 bytes.

`Message_CCSID`
The Message_CCSID value identifies the code page and character set definition used for the string pointed to by Log_message. This value can be used by applications which process the RODM log data set.

Method_name
The name of the method that this function triggers or the name of the method that put this notification block on the notification queue.

Method_output_message
A pointer to the data that is placed on the notification queue by the calling method and is passed to the user application. The maximum length of the message is 32767 bytes.

Method_parms
The pointer to the short-lived parameters passed to a method. The short-lived parameters are passed to the notification method associated with the object the function acts on. For the EKG_SwapField function, the short-lived parameters are also passed to the change method. For the EKG_QueryField function, the short-lived parameters are passed to the query method instead of the notification method. For the EKG_TriggerNamedMethod and EKG_TriggerOIMethod functions, the short-lived parameters are passed to the method being triggered.

New_char_data_length
The length of the new data for data types CharVar and GraphicVar. This parameter is ignored for other data types. The data pointed to must be the first byte of the character data and the length must be specified in the New_Char_data_length parameter.

New_data_ptr
The pointer to the new data that is to replace the value of the target field.

Notification_queue
The Notification_queue specified by the function. See “RODM Notification Process” on page 318.

Notification_queue_count
The number of notification blocks on the notification_queue before this function acts on the queue.

Notify_method
The object ID of the notification method that is associated with this notification subscription.

Number_of_fields
A value that specifies the number of fields to be changed.

Number_of_functions
A value that specifies the number of functions to be run. You specify one element of Function_information_array for each function.

Number_of_subfields
A value that specifies the number of value subfields to be queried. You specify one element of Field_info_array for each query.

Object_array
The array of objects this function acts on.

Object_ID
The object identifier of the object this function acts on, or one element of Object_array of objects this function acts on.

Object_list_length
The number of objects in the array.

Object_name
The name of the object this function acts on.

`Old_char_data_length`
The length of the old data if the data type of the old data is CharVar or GraphicVar. This parameter is ignored for other data types.

`Old_data_ptr`
The pointer to the old data.

`Private_public_flag`
The Private_public_flag specifies whether a field is private (not inherited by its children) or public (inherited by its children). Valid values are:

Value	Meaning
0	Public
1	Private

`Parent_access_info_ptr`
The Parent_access_info_ptr is the pointer to an entity access information block where only the class information is used by this function call. The object information in that access block must be set to null values if the Naming_count information is set to zero.

`Reason_code`
The reason code from RODM.

`Requesting_method_ID`
The method Object_ID of the current method object.

`Response_block_length`
The length in bytes of the response block supplied by the method or application using this function. This value must include 8 bytes for the Response_block_length and Response_block_used parameters.

`Response_block_reference`
The pointer set by RODM to the address within the response of the first byte of returned data for this function. This parameter is set to zero when no data is returned. One common response block is shared by all operations originating from a single user API call. These interactions include any that are specified in an EKG_ExecuteFunctionList or EKG_QueryMultipleSubfields function call.

`Response_block_type`
The response_type_block specifies whether a notification block was generated by a notification method or by an object-deletion subscription. Valid values are:

Value	Meaning
1	Generated by a notification method
2	Generated when an object was deleted and an object-deletion subscription existed for that object

`Response_block_used`
The length in bytes of the data returned by RODM. If the response block supplied by the method or application is too small to hold the data that is to be returned, the value of Response_block_used is set to the size that the response block was in order to hold the data. This value is larger than the value of Response_block_length and includes 8 bytes for the Response_block_length and Response_block_used parameters. This parameter is set to zero when no data is returned.

If a transaction provides response block data and does not cause a response block overflow, the Response_block_used parameter is less than or equal to the Response_block_length parameter. If the transaction does cause a response block overflow, the Response_block_used parameter is greater than the Response_block_length parameter.

`Response_data`

The area in an EKG_ExecuteFunctionList response block that contains the data returned by query functions. Use Response_block_reference pointers (see above) in the function block to retrieve the data for individual functions. The format is the same as that following the 8-byte header in the normal response block for the function.

`Return_code`

The Return_code and Reason_code values indicate status of this particular function request. The highest numeric value is duplicated in the Transaction_info_block parameter for of the EKGUAPI call. If there is a tie for the worst error, the first among the worst is reported.

`Stop_ECB`

The parameter used to notify users that the current version of RODM is stopping in response to either an operator request or an API request. If a user application calls EKGWAIT, this ECB must always be included in the list.

`Stop_type`

Specify Stop_type of 1 to stop RODM after it has quiesced and performed a checkpoint operation. Specify Stop_type of 2 to stop RODM after it has quiesced without performing a final checkpoint operation.

`Subfield`

Identifies the specific subfield for this function. Valid values for all functions except EKG_WhereAmI are:

Value	Subfield
0	All subfields except Notify (valid only for EKG_RevertToInherited function)
1	Value
2	Query
3	Change
4	Notify
5	Prev_val
6	Timestamp

Valid values for the EKG_WhereAmI function are:

Value	Subfield
1	Value (method must be named method)
2	Query
3	Change
4	Notify (notification method)

`Subfield_map`

The Subfield_map is a bit map defined as follows (bits are numbered 0–31 from left to right). Setting a bit to 1 specifies that the function acts on that subfield. RODM sets a Subfield_map bit to 1 in an output block to indicate that the corresponding subfield exists.

Bit	Subfield
0	Value
1	Query
2	Change
3	Notify
4	Prev_val
5	Timestamp
6–31	Reserved (must be set to zero)

Subscription_info
Specifies the notification subscription this function acts on. Subscription_info is defined as data type RecipientSpec and contains the User_appl_ID and Notification_queue for the specified subscription.

User_appl_ID
If the User_appl_ID parameter in the function block is set to the null value (blank) for this field, RODM will default to the User_appl_ID value defined in the Access_block that starts this transaction. For a subscription notification, the User_appl_ID parameter specifies the application which is being notified. If a method initiated through the message interface specifies a null User_appl_ID, the name supplied by RODM is that which was specified in the Access_block which originally issued the message transaction.

For an APF (authorized program facility) authorized program, the User_password does not need to be specified. The User_appl_ID without the User_password identifies the user to RODM and determines the user's authority level. For application programs that are not APF authorized, the User_password is required. The User_appl_ID and the User_password are combined to identify the user to RODM, and to determine the user's authority level using the EKG_Connect or EKG_ConnectLong function.

User_password
For application programs that are not APF authorized, both the User_appl_ID and the User_password are required to be specified in the RODM access block to validate the user authority level and to connect to RODM. The validated User_appl_ID is used by RODM to determine the specific level of access authority granted to the user. This parameter is a maximum of 8 bytes with shorter values left justified in the parameter and padded on the right with blanks.

In performing the validation of the User_appl_ID and User_password for programs that are not APF authorized, RODM uses the RACROUTE interfaces on z/OS systems. The user ID, password and access authorization level are assumed to have been registered to the security manager supporting those interfaces.

If a User_appl_ID is specified, the User_password value must be valid for programs that are not APF authorized. If the User_appl_ID parameter in the Access_block is all blanks, both for programs that are APF authorized and for programs that are not APF authorized, the User_password field is ignored. A system authorization facility (SAF) product such as Resource Access Control Facility (RACF), attempts to associate an authorized user ID with this function call. If that user ID is not located, the connection request is rejected. If a verified user ID is found, it is put into the User_appl_ID parameter of the Access_block.

User_area
A data area containing the data supplied by the method that put the notification block on the notification queue.

User_word
The User_word parameter is intended to be the information passed to the notification method through the invocation parameters. The parameter is set by the caller in the function block used by the EKG_AddNotifySubscription function, saved with the subscription request in RODM, made available to a notification method as a passed parameter, and is assumed to be passed to the notification function unmodified when notification takes place. The notification method determines the final value for User_word.

`Value_for_reason_code`
The reason code passed to the caller of the method.

`Value_for_return_code`
The return code passed to the caller of the method.

RODM Return and Reason Codes

For each function call you make to RODM, the RODM program issues a return code and reason code. The reason code gives you more specific information about the possible cause of a problem.

The following four sections describe the possible reason codes for each of the four return codes. The tables provide explanations and suggested corrective actions. "List of Reason Codes for Each Function" on page 470 and "List of Functions for Each Reason Code" on page 472 provide a cross-reference so that you can determine the codes that are issued for any particular function call you use. "List of Reason Codes from Supplied Methods" on page 479 lists the reason codes returned by the methods that are supplied with the NetView program.

Reason codes can fall into one of three ranges based on which program or method issued the reason code:

Range	**Issued By**
0–32767	RODM application programming interfaces
32768–49151	Methods that are supplied with the NetView program
49152–65535	Customer-written methods

If you write methods that issue reason codes, use reason codes in the range 49152–65535.

Reason codes in the range **32781–32996** are issued by the EKGCTIM, EKGMIMV, EKGNEQL, EKGNLST, EKGNOTF, EKGNTHD, and FLBTRNMM methods that are supplied with the NetView program. These reason codes are issued when the method receives an error or warning from a RODM transaction. Subtract 32780 from the reason code issued by the method to get the original value issued by RODM for the transaction. You can then look up the original value in the following tables. The methods issue the return code for the transaction without change.

Reason codes in the range **32810–32904** are issued by the EKGSPPI method when it receives an error from the program-to-program interface module CNMCNETV. The reason code issued is 32809 plus the return code from CNMCNETV. Subtract 32809 from the reason code issued by the EKGSPPI method. The result is the return code from CNMCNETV. See the *IBM Tivoli NetView for z/OS Application Programmer's Guide* for the meaning of this return code.

Writers of methods must be aware of the implications of issuing return and reason codes from methods. See "Error Conditions in Transactions" on page 317 for information about how an application might interpret reason and return codes that are returned by methods.

The *IBM Tivoli NetView for z/OS Troubleshooting Guide* contains additional information about troubleshooting RODM problems, especially abend problems.

Reason Codes for Return Code 0

Table 197 describes the reason codes that are returned with return code 0.

Table 197. Reason Codes for Return Code 0

Reason Code	Description	Corrective Action
0	The system successfully performs the requested function.	None
26	The new data value is the same as the old data value. If a local copy did not previously exist for the field, one is created.	None
48	Not used.	None
142	The system performs the request successfully and a local copy is created.	None
143	The system performs the request successfully and the returned value is an inherited value.	None
167	Not used.	None
180	The user object will not be deleted when the user disconnects from RODM. The possible cause is that links from the user object to the queue object are not removed because the StopMode specifies to keep the queue objects.	None
185	The Disconnect is successful. The user object is not deleted from RODM because links to Notification Queue objects still exist.	Try to connect and disconnect again.
32769	Compared data values do not match.	Specify the value subfield for the data to be compared.

Reason Codes for Return Code 4

Table 198 describes the reason codes that are returned with return code 4.

Table 198. Reason Codes for Return Code 4

Reason Code	Description	Corrective Action
1	The system rejects the request because RODM is doing one of the following: • Quiescing—waiting for all current transactions to complete following a checkpoint request. • Writing the master window and the translation windows to the checkpoint data sets. RODM rejects all new user API requests and returns this reason code.	Retry the request after the checkpoint process is completed.
2	The system rejects the request because RODM is starting.	Retry the request after RODM is initialized completely.
3	The system rejects the request because RODM is stopping.	Restart the specified RODM or connect to another existing RODM by updating the RODM_name field in the RODM access block. Retry the request.

Table 198. Reason Codes for Return Code 4 (continued)

Reason Code	Description	Corrective Action
5	The system rejects the request because RODM was stopped with a checkpoint request. The specified Sign_on_token is no longer valid.	If this reason code was a result of the EKG_Connect or EKG_ConnectLong function, retry the request after restarting the specified RODM. If this reason code was not result of a EKG_Connect or EKG_ConnectLong function, connect to another RODM by correcting the RODM_name field in the access block to get a new Sign_on_token. Retry the request with the new Sign_on_token.
6	The system rejects the request because RODM was stopped without a checkpoint request. The specified Sign_on_token is no longer valid.	If this reason code was a result of the EKG_Connect or EKG_ConnectLong function, retry the request after restarting the specified RODM. If this reason code was not result of a EKG_Connect or EKG_ConnectLong function, connect to another RODM by correcting the RODM_name field in the access block to get a new Sign_on_token. Retry the request with the new Sign_on_token.
24	The system cannot trigger one or more methods in the notification list. The original transaction itself completed successfully. Possible causes are that the notification method is recursive, or there are errors in executing the method.	Make sure that all methods in the notification list are valid.
27	The response block is not large enough. An overflow block is created. An overflow block is not created for query functions issued by a method.	Retrieve the data from the overflow block using the query response block overflow function.
28	RODM log files are not available. Both the primary and secondary log files can not be opened or written successfully, or the LOGT command was issued. The transaction failed.	Contact the system administrator.
29	The log record size is larger than the default maximum of 32761 bytes. The record is truncated to 32761 bytes.	Check the size of the Method_parms in the function block or check the size of the log message specified for the Output to Log (2008) function.
30	The Stop_ECB in the function block is null. This user will not be notified when the specified RODM is stopping.	None
34	The specified queue object is created but the link with the user object cannot be created. The required storage might not be available.	None
38	The operator stopped the checkpoint request by direct response to a WTOR issued by RODM. This reason code is contained in the EKG_LastCheckpointResult field of the EKG_System object and is not returned through the method API or user API.	Contact operator.
40	The system does not change the field value because the field already contains the primary inheritance value.	None
41	The system rejects the request because the field is locally created.	None

Table 198. Reason Codes for Return Code 4 (continued)

Reason Code	Description	Corrective Action
42	The specified method is a null module because it has been deleted by an unsuccessful module refresh. The transaction failed.	Refresh the method and retry the request. If not successful, delete the method and reinstall it.
44	There is no message in the specified notification queue for the user.	None
46	The overflow block is cleaned without retrieving because the response block provided by the user is null.	None
47	Some of the overflow data is discarded because the response block provided by the user is not large enough.	None
48	Not used.	None
49	Not used.	None
50	The posting fails because the user has not requested a WAIT on the specified ECB address, or because the specified ECB address is not valid. The queue objects or the subscriptions will be deleted according to the StopMode of the user object.	None
52	The system rejects the request because the specified class does not exist or the parent class of the specified object ID does not exist. RODM sets the return code to 4 for query functions and to 8 for other functions.	Correct the class or parent class. Retry the request.
54	The system rejects the request because the specified object does not exist. RODM sets the return code to 4 for query functions and to 8 for other functions.	Correct the object data. Retry the request.
56	The system rejects the request because the specified field does not exist. RODM sets the return code to 4 for query functions and to 8 for other functions.	Correct the field data. Retry the request.
57	The system rejects the request because the specified primary parent of the object is not a class with object children. RODM sets the return code to 4 for query functions and to 8 for other functions.	Correct the primary parent class data. If the primary parent class data is correct, verify the class ID portion of the object ID. Retry the request.
62	The system rejects the request because the specified subfield does not exist. RODM sets the return code to 4 for query functions and to 8 for other functions.	Correct the subfield data. Retry the request.
72	The target fields have already been linked. The system has taken no action.	Correct the entity and field information. Retry the request.
75	The target fields are not linked. The system has taken no action.	Correct the field information. Retry the request.
81	The system rejects the request because the specified method is not installed. RODM sets the return code to 4 for requests to delete a method object and to 8 for other functions.	Install the specified method. Retry the request.
92	The system rejects the request because the field to be created already exists under the specified class.	Correct the field data. Retry the request.

Table 198. Reason Codes for Return Code 4 (continued)

Reason Code	Description	Corrective Action
97	A field with the specified field name already exists on a child class. The new field is created on the parent class and the existing field on the child class is marked as containing locally defined data.	None
100	One or more requested subfields are not valid. Any valid subfields are created. RODM sets the return code to 4 for create field functions and to 8 for create subfield functions.	Correct the subfield map. Retry the request.
104	One or more specified subfields already exist.	Correct the subfield map. Retry the request.
110	The system rejects the request because the specified object name is used by another object under the specified parent class.	Correct the object name. Retry the request.
112	The system rejects the request because the specified field already has a notification subscription with the same parameters.	Correct the request data. Retry the request.
133	The system cannot update the value of the timestamp subfield. There might not be enough storage.	Issue another transaction for the same resource and check the return and reason codes from that transaction. Return code 12, reason code 211 means there is not enough storage. If the problem is caused by not enough storage, free storage and retry the request.
146	One or more specified subfields do not exist in the specified field.	Correct the subfield information. Retry the request.
158	The notification cannot be placed in the notification queue because the queue has reached its maximum limit.	Query the notification queue content or enlarge the value of EKG_Maximum_Q_Entries.
173	The system performs the request successfully and one notification queue is created by RODM.	Change the EKG_ECBAddress of this notification queue object to a valid value.
174	The notification information block has been put into the notification queue. The system cannot post the specified user because the ECB address is null.	None
175	Part of the user message is truncated because it is longer than 32767 bytes.	None
181	The notification cannot be attached to the specified queue because the queue is not active.	Change the EKG_Status value of the specified queue object.
182	The notification has been put in the notification queue. The system cannot post the specified user.	None
183	The information from the notification block has been put in the response block. The system cannot release the storage used by the notification block.	None
191	The system rejects the request because the specified method object is the NullMeth object.	Correct the method object information. Retry the request.
204	The original data in the response block is overwritten.	None
205	Not used.	None
206	Not used.	None

Table 198. Reason Codes for Return Code 4 (continued)

Reason Code	Description	Corrective Action
208	The response block overflow data is discarded because the user has specified to not save overflow data.	If the response block overflow data is needed, change the value of the EKG_RBOverflowAction field to save. Retry the request.
209	The user request to wait on a list of ECBs cannot be completed because an ECB address of 0 is found.	Correct the ECB address.
221	Not used.	None
604	A correlated aggregate object was not created because the agent provided an incorrect correlation value (network address).	Modify the agent (distributed manager) to provide a valid network address.
605	A correlated aggregate object was not created because a correlated aggregate object already exists.	None
32770	Part of the method output message from the notification method that is supplied with the NetView program is discarded because the length exceeds 32767 bytes. The request completed successfully.	Correct the method output message.
45081	A method encountered an error but was able to complete its function. Either an incorrect field value was provided, for which RODM used a default value, or the method detected a notification method failure after it successfully changed the value of a field in RODM.	The condition that caused this error must be corrected to avoid future failures. The method logs information on the error in messages written as type-1 RODM log entries. If the error is caused by a notification method failure, the message includes the reason code set by the notification method. If the error was caused by an incorrect field value, the RODM log specifies the field, the incorrect value, and the default value used in its place. Correct the incorrect value.

Reason Codes for Return Code 8

Table 199 describes the reason codes that are returned with return code 8.

Table 199. Reason Codes for Return Code 8

Reason Code	Description	Corrective Action
8	The system rejects the request because the API version is not valid.	Correct the API version information in the transaction information block. Retry the request.
9	The system rejects the request because the caller is not authorized to use the requested function.	Make sure that the User_appl_ID is correct or contact the system administrator to change the authority level.
10	The system rejects the request because the function ID is not valid.	Correct the function ID. Retry the request.
11	The requested function is not complete because the system does not have enough storage to copy the short-lived parameters into RODM.	Remove unused entities and fields or contact the system administrator. Retry the request.
13	The system rejects the request because the specified RODM is not found.	Start the RODM with the specified name or correct the RODM_name field in the RODM access block. Retry the request.

Table 199. Reason Codes for Return Code 8 (continued)

Reason Code	Description	Corrective Action
14	The system rejects the request because an incorrect Sign_on_token is detected. The user application does not connect to the specified RODM using the EKG_Connect or EKG_ConnectLong function, or the Sign_on_token was changed. If two or more applications connect to RODM using the same user ID and one application disconnects from RODM, the Sign_on_token for the remaining applications is canceled by RODM. This reason code is issued when a remaining application sends a function request to RODM.	Make sure the user application does not modify the Sign_on_token. Connect to the specified RODM using the EKG_Connect or EKG_ConnectLong function to get a valid Sign_on_token. Retry the request with the new Sign_on_token.
15	The system rejects the request because the number of concurrently executing API function calls has reached the limit specified in the customization file.	Retry the request later or increase the CONCURRENT_USERS value in the RODM customization file. Warm start RODM.
16	The system rejects the request because no RODM currently exists in the system.	Start the RODM with the specified name. Retry the request.
17	The system rejects the request because the RODM service module in CSA is not found.	Contact the system administrator.
18	The system rejects the request because the specified function is not allowed for this method.	Correct the function ID in the function block. Retry the request.
21	The system cannot perform the requested list of functions because the number of list requests provided by the user is zero or negative.	Correct the Number_of_functions field. Retry the request.
22	The system rejects the request because the notification queue name is null.	Correct the notification queue name. Retry the request.
23	The system rejects the request because the data (types CharVar, GraphicVar, MethodSpec, SelfDefining, or BERVar) passed to RODM is not valid.	Correct the data. Retry the request.
33	The system rejects the request because no storage is available for storing the log record information.	Delete unused entities. Retry the request.
35	Checkpoint master window error. The VSAM data set identified by the EKGMAST DD statement in the RODM start up JCL is not available or not usable. This reason code is contained in the EKG_LastCheckpointResult field of the EKG_System object and is not returned through the method API or user API.	Contact the system administrator.
36	Checkpoint translation window error. The VSAM data set identified by the EKGTRAN DD statement in the RODM start up JCL is not available or not usable. This reason code is contained in the EKG_LastCheckpointResult field of the EKG_System object and is not returned through the method API or user API.	Contact the system administrator.

Table 199. Reason Codes for Return Code 8 (continued)

Reason Code	Description	Corrective Action
37	Checkpoint data window error. One or more of the VSAM data sets identified by the DD statements in the RODM start up JCL whose names have a prefix of EKGD are not available or not usable. This reason code is contained in the EKG_LastCheckpointResult field of the EKG_System object and is not returned through the method API or user API.	Contact the system administrator.
39	The system rejects the request because the data pointed to by Old_data_ptr is not equal to the target field.	Correct Old_data_ptr. Retry the request.
52	The system rejects the request because the specified class does not exist or the parent class of the specified object ID does not exist. RODM sets the return code to 4 for query functions and to 8 for other functions.	Correct the class or parent class. Retry the request.
54	The system rejects the request because the specified object does not exist. RODM sets the return code to 4 for query functions and to 8 for other functions.	Correct the object data. Retry the request.
56	The system rejects the request because the specified field does not exist. RODM sets the return code to 4 for query functions and to 8 for other functions.	Correct the field data. Retry the request.
57	The system rejects the request because the specified primary parent of the object is not a class with object children. RODM sets the return code to 4 for query functions and to 8 for other functions.	Correct the primary parent class data. If the primary parent class data is correct, verify the class ID portion of the object ID. Retry the request.
60	The system rejects the request because the field type is public and there are still objects existing under the class or descendent classes.	Delete the objects under the class before deleting the public field or its subfields.
61	The system rejects the request because the subfield number is not valid.	Correct the subfield number. Retry the request.
62	The system rejects the request because the specified subfield does not exist. RODM sets the return code to 4 for query functions and to 8 for other functions.	Correct the subfield data. Retry the request.
65	The system rejects the request because this function does not apply to fields with data type ObjectLink or ObjectLinkList.	Correct the function ID or field identifier. Retry the request.
66	The system rejects the request because the data type of the new data is not the same as the data type of the specified field.	Correct the data type or field. Retry the request.
67	The system rejects the request because this function does not apply to a system-defined field.	Correct the function ID or the field. Retry the request.
70	The system rejects the request because this function does not apply to a notify subfield.	Correct the subfield or function ID. Retry the request.
71	The system rejects the request because this function does not apply to a prev_val or timestamp subfield.	Correct the subfield or function ID. Retry the request.

Table 199. Reason Codes for Return Code 8 (continued)

Reason Code	Description	Corrective Action
73	The system rejects the request because the two target objects are identical.	Correct the entity information. Retry the request.
74	The system rejects the request because the field data type is not allowed for a link or unlink function.	Correct the field information. Retry the request.
76	The system rejects the request because the notify subfield does not exist.	Create a notify subfield for the specified field.
77	The system rejects the request because this function does not apply to some of the system-defined fields.	Correct the fields. Retry the request.
79	The system rejects the request because the specified function block pointer in the list is null.	Correct the function block pointer. Retry the request.
80	The system rejects the request because this module recursively calls itself.	Update the related methods to remove the recursive call. Retry the request.
81	The system rejects the request because the specified method, or a method called by the specified method, is not installed. RODM sets the return code to 4 for requests to delete a method object and to 8 for other functions.	Install the specified method. If the specified method is installed correctly, ensure that all methods called by the specified method are installed correctly. Retry the request.
83	The system rejects the request because the response block length is less than eight bytes.	Correct the response block length. Retry the request.
84	The user has already connected to RODM.	None
85	The system rejects the request because the specified Stop_type is not valid.	Correct the Stop_type. You can specify the value of Stop_type as 1 or 2. Retry the request.
86	The system rejects the request because the specified class name is not valid or is a RODM reserved class name.	Correct the class name. Retry the request.
87	The system rejects the request because the specified class name has been used by another class.	Correct the class name. Retry the request.
89	The system rejects the request because the universal class or a system-created class cannot be deleted.	Correct the class information. Retry the request.
90	The system rejects the request because some entities exist under the specified class.	Delete all entities under the specified class. Retry the request.
91	The system rejects the request because the specified field name is not valid or is a reserved RODM field name.	Correct the field name. Retry the request.
93	The system rejects the request because the field to be created already exists in the subclass and has a different data type or different subfields.	Correct the field data. Retry the request.
94	The system rejects the request because the field to be created already exists in a child class with a different field type.	Correct the field data. Retry the request.
95	The system rejects the request because the field type flag is not valid.	Correct the field type flag. You can specify the value of the field type flag as 1, 2, or 3. Retry the request.

Table 199. Reason Codes for Return Code 8 (continued)

Reason Code	Description	Corrective Action
96	The system rejects the request because the data type is not valid or is a reserved data type.	Correct the data type. You cannot create fields using reserved data types. Retry the request.
98	The system rejects the request because a user application is not allowed to delete system-defined fields.	Correct the field information. Retry the request.
100	One or more requested subfields are not valid. Any valid subfields are created. RODM sets the return code to 4 for create field functions and to 8 for create subfield functions.	Correct the subfield map. Retry the request.
103	The system rejects the request because the field or subfield does not exist under the specified class.	Correct the class information to specify the class where the field or subfield exists. Retry the request.
106	The system rejects the request because the value subfield cannot be deleted.	Correct the subfield name. Retry the request.
107	The system rejects the request because the method object name is not valid.	Correct the method name. Retry the request.
108	The system rejects the request to delete the method because the method is in use.	Check the value of the EKG_UsageCount field of the method object. If the value is greater than 0, the method is being used; retry the request later.
109	The system rejects the request because the user-provided object name is not valid or is a RODM reserved object name.	If the request was a non_connect request, correct the object name. If the request was a connect request, correct the User_appl_ID so that it conforms to the rules for RODM object names. Retry the request.
111	The system rejects the request because the specified object is linked to other objects.	Unlink all other objects from the specified object. Retry the request.
113	The system rejects the request because the specified subscription does not exist.	Correct the request data. Retry the request.
115	The system rejects the request because the data type for this field is not valid for this function.	Correct the field data type. Retry the request.
117	A function in the list is rejected because the function ID in the function information array is not valid. Functions with valid function IDs are processed.	Correct the function ID in the function information array.
120	The system rejects the request because an overflow block with the specified correlation ID does not exist.	Correct the correlation ID. Retry the request.

Table 199. Reason Codes for Return Code 8 (continued)

Reason Code	Description	Corrective Action
127	The system rejects the request because the user ID is not authorized to RODM.	If you **are** running RODM with security active, ensure that the task that is trying to connect to RODM is defined to your security product, and has read access to the appropriate RODM resources defined in the RODMMGR class. For example, the user must have access to at least the RODM1 resource in the RODMMGR class to connect to RODM. (The `RODM` portion of RODM1 is determined by the SEC_RNAME keyword in EKCUST.) If the task that is trying to connect to RODM is a started procedure, ensure that you have defined the task to the STARTED class in the SAF product. In RACF, this can also be accomplished by defining the task in the started procedure table, ICHRIN03; however, using the STARTED class is preferred. If you **are not** running RODM with security active, it is possible that you are trying to connect to RODM with a blank user ID. This is not allowed. You must specify a user ID on the connect request when security is not active. If you run the RODM loader when security is not active, you will also get this reason code because the loader first tries to connect with a blank user ID. It will then automatically attempt to connect with a non-blank user ID. In this case, the reason code can be ignored. **Note:** Running with a blank user ID is allowed when RODM is running with security active because the user ID can be extracted from the SAF product. To run with security active you must: • Have an SAF product installed • Have a security class active for RODM (RODMMGR or user defined) • Identify the security class with the SEC_CLASS keyword in EKGCUST.
128	The system rejects the request for one of the following reasons: • The password or password phrase has expired. • The password or password phrase is not authorized. • The user ID was revoked in the SAF product.	If the password or password phrase is expired or not authorized, correct the problem and retry the request. If the password or password phrase is not the problem, have the security administrator check the status of the user ID in the SAF product.
130	The system rejects the request because a connection was requested in cross-memory mode.	Issue the connection request in non-cross-memory mode.
131	The system rejects the request because the overflow block has not been cleaned.	Issue a query response block overflow request to retrieve the overflow data. Retry the request.

Table 199. Reason Codes for Return Code 8 (continued)

Reason Code	Description	Corrective Action
134	The system cannot update the value of the prev_val subfield. There might not be enough storage.	Issue another transaction for the same resource and check the return and reason codes from that transaction. Return code 12, reason code 211 means there is not enough storage. If the problem is caused by not enough storage, free some storage and retry the request.
135	The system rejects the request because the length of the long-lived parameters is not valid.	Correct the parameter lengths. Retry the request.
136	The system rejects the request because the length of the Method_parms is not valid.	Correct the parameter length. The maximum length is 254 bytes. Retry the request.
138	The system rejects the request because the Old_data_ptr is null.	Correct the Old_data_ptr. Retry the request.
139	The system rejects the request because the field ID is not specified and the field name pointer or field name length is not valid.	Specify either a valid field ID or a valid field name pointer and field name length. Retry the request.
140	The system rejects the request because the class ID is not specified and the class name is not valid.	Specify a valid class ID or a valid class name. Retry the request.
141	The system rejects the request because the specified field of data type ObjectLink is already linked to another field.	Correct the field information. Retry the request.
144	The system rejects the request because a system-created field or subfield cannot be deleted by a user application.	Correct the field or subfield information. Retry the request.
145	The system rejects the request because the specified field or subfield is read only.	Correct the field or subfield information. Retry the request.
147	The system rejects the request because the length of the new data is not valid.	Correct the data length. Retry the request.
148	The system rejects the request because the create subfield function is not valid for a system-defined field.	Correct the field information. Retry the request.
150	The system rejects the request because the object ID is not specified and the object name information is not valid.	Specify a valid object ID or a valid object name. Retry the request.
159	The system rejects the request because the object directory or the field name table has reached its maximum size limit.	Select another object or field. Retry the request.
160	The system rejects the request because the field name is not specified.	Specify the field name. Retry the request.
163	The system rejects the request because the pointer to the entity access information block is not valid.	Correct the entity access information block pointer. Retry the request.
164	The system rejects the request because the pointer to the field access information block is not valid.	Correct the field access information block pointer. Retry the request.
165	The system rejects the request because the Naming_count of the entity access information block is not valid.	Correct the Naming_count value. Valid Naming_count values are 0, 1, and 2. Retry the request.
166	The system rejects the request because the Naming_count of the field access information block is not valid.	Correct the Naming_count value. Valid Naming_count values are 0 and 1. Retry the request.

Table 199. Reason Codes for Return Code 8 (continued)

Reason Code	Description	Corrective Action
169	The system rejects the request because the object ID is not specified.	Specify the object ID. Retry the request.
170	The system rejects the request because a user application cannot create or delete a system object.	Correct the parent class information. Retry the request.
176	The system rejects the request because the new data is not valid.	Correct the new data. Valid values for EKG_StopMode are 1, 2, and 3; for EKG_Status and EKG_MTraceFlag are 0 and 1; for EKG_RBOverflowAction and EKG_ExternalLogState are 1 and 2. Valid values for EKG_LogLevel and EKG_MLogLevel are 0—999. Retry the request.
186	The system rejects the request because the user application cannot create classes under the system classes.	Correct the parent class information. Retry the request.
187	The system rejects the request because the specified subfield map is null.	Correct the subfield map. Retry the request.
192	The system rejects the request because the specified function ID is not valid asynchronous execution.	Correct the function ID. Retry the request.
193	The return or reason code set by the method is not valid.	Correct the return or reason code in the method. Valid return codes are 0, 4, 8, and 12. Valid reason codes are from 0 to 65535.
195	The system rejects the request because the system field cannot be changed at the class level.	Correct the data. Retry the request.
201	The system rejects the request because the data to be return is a null string.	Correct the data. Retry the request.
202	The system rejects the request because the checkpoint function is disabled.	Make sure that all checkpoint data sets are defined when RODM is started.
203	The system rejects the request because there is no response block.	Specify a response block. Retry the request.
207	The EKG_Connect or EKG_ConnectLong function cannot be completed. Possible causes are that RACF is active but the class that is specified in the customization file is not active in RACF or the class is not defined in RACF.	Contact the system administrator.
210	The user request to wait on a list of ECBs cannot be completed because the number of ECBs is zero.	Correct the number of ECBs in the list.
214	The system rejects the request because the Naming_count of the Entity Access Information Block is not valid. Because the function needs valid object access information, the Naming_count of the Entity Access Information Block must be 0 or 2.	Correct the Naming_count. Valid values are 0 and 2. Retry the request.
215	The system rejects the request because the user is not allowed to update EKG_MTraceFlag of NullMeth.	Correct the method object information.
220	The system rejects the link or unlink request because one or both of the change methods defined to the fields to be linked or unlinked returned a non-zero return code.	Examine the change method to see what criteria it uses to allow links or unlinks and make sure you meet those criteria.

Table 199. Reason Codes for Return Code 8 (continued)

Reason Code	Description	Corrective Action
223	The system rejects the query multiple subfields request because the Number_of_subfields field of the function block was specified as zero, less than zero, or greater than 100,000. The system rejects the change multiple fields request because the Number_of_subfields field of the function block was specified as zero, less than zero, or greater than 256.	Specify a correct value for the Number_of_subfields field. Specify a correct value for the Number_of_Fields field.
224	The system rejects the request because the input data type is not allowed for an indexed field.	Correct the input data type or the Field_type_flag. Retry the request.
225	The system rejects the request because the field has not been created with the corresponding Field_type_flag.	Correct the field name, or the field ID and field type information. Retry the request.
226	You tried to connect a program that is not APF authorized with a blank password or password phrase.	Either specify the correct password or password phrase in the User_password field of the function block, or make the program APF authorized.
227	The system rejects the request because a reserved field in the function block is not zero.	Ensure that all of the reserved fields in the function block are set to zero. Retry the request.
228	The system rejects the request because the indexed data length field for the locate function is not valid.	Ensure that the indexed data length field is between 0 and 32767 for data type CharVar, and between 0 and 254 for data type IndexList. Retry the request.
229	The system rejects the request because the index data value pointer for the locate function is not valid.	Correct the indexed data value pointer. Retry the request.
230	The system rejects the request because the length of the IndexList field does not equal the sum of each element including each element's 2-byte length field.	Ensure that the length is correct. Retry the request.
231	The system rejects the request because the IndexList field contains a duplicate value.	Ensure that each value is unique within the field. Retry the request.
232	The system rejects the request because a length found in a value of the IndexList field is not valid.	Ensure that the length of each value is between 0 and 254 bytes. Retry the request.
32768	The data specified in the Long_lived_parm is not valid. The error might be in the request code, option code or enable change_status parameter. The error might also be that the data type of the tested value is not valid. The request failed.	Correct the Long_lived_parm.
32771	The system rejects the request because the data type of the value subfield queried is not valid.	Verify the correct data type for the method being used. See "Supplied Methods" on page 480 for a description of the methods that are supplied with the NetView program. Correct the parameter list passed to the method.
32772	The system rejects the notification method that is supplied with the NetView program because the data type of the value in the specified field is not valid. The request failed.	Correct the field data type of the specified field. Valid field data types are Smallint, Integer, Floating, TimeStamp or CharVar.
32790	The short-lived parameter passed to the method is a null pointer.	Correct the pointer.

Table 199. Reason Codes for Return Code 8 (continued)

Reason Code	Description	Corrective Action
32791	One or more data items in the short-lived parameter is not of data type CharVar.	Correct the short-lived parameter.
32792	One or more data items in the short-lived parameter is too long.	Correct the short-lived parameter.
32793	Incorrect number of data items in the short-lived parameter.	Correct the short-lived parameter.
32794	The RCVRID_CHARVAR value in the short-lived parameter passed to the EKGSPPI method is blank or null.	Specify a valid value for RCVRID_CHARVAR.
32795	The ASSIST_CHARVAR value in the short-lived parameter passed to the EKGSPPI method is not valid.	Specify a valid value for ASSIST_CHARVAR.
32796	The TASKINFO_CHARVAR value in the short-lived parameter passed to the EKGSPPI method is not valid.	Specify a valid value for TASKINFO_CHARVAR.
32797	The TASKNAME_CHARVAR value in the short-lived parameter passed to the EKGSPPI method is blank or null.	Specify a valid value for TASKNAME_CHARVAR.
32798	The CMD_CHARVAR value in the short-lived parameter passed to the EKGSPPI method is blank or null.	Specify a valid value for CMD_CHARVAR.
45057	The DUIFCUAP method parameters specify deleting the AggregationParent to AggregationChild link between two objects. However, the specified objects do not have this link.	If the objects were never linked, or if the objects were previously unlinked by the DUIFCUAP method, no action is needed. If the objects were unlinked without using the DUIFCUAP method, run the DUIFFAWS method. If the objects were unlinked using the DUIFCUAP method, but the method did not complete successfully, run the DUIFFAWS method.
45058	The DUIFCUAP method parameters specify creating the AggregationParent to AggregationChild link between two objects. However, the specified objects already have this link.	If the objects were previously linked by the DUIFCUAP method, no action is needed. If the objects were linked without using the DUIFCUAP method, run the DUIFFAWS method. If the objects were linked using the DUIFCUAP method, but the method did not complete successfully, run the DUIFFAWS method.
45061	Not used.	None
45066	The DUIFCUAP does not create the requested link. Creating the requested link creates a loop in the aggregation hierarchy, or a loop already exists in the aggregation hierarchy above the objects for which the link was requested. Information about the loop path is written to the RODM log.	Correct the parameters passed to DUIFCUAP to specify valid objects to be linked, or remove the loop from the aggregation hierarchy. Run DUIFFAWS to make sure that aggregate objects are properly initialized. Run DUIFCUAP again to create the desired link.
45070	The short-lived input parameters provided to a method are not valid. The parameters might have been supplied by the INVOKED_WITH RODM load function primitive statement. The parameters are written to the RODM log.	Check the RODM log and verify that the parameters sent to the method have the correct format and value for the method.

Table 199. Reason Codes for Return Code 8 (continued)

Reason Code	Description	Corrective Action
45071	An object specified in the input parameters to the DUIFCUAP method or the DUIFCLRT method does not exist in RODM. Information about the missing object is written to the RODM log.	Create the missing object or correct the input parameters for the method and retry the request.
45077	An error occurred for a method that was triggered for this transaction. Diagnostic information is written to the RODM log.	Check the RODM log for information on the specific error that occurred.
45078	An error occurred while processing a transaction. The RODM data cache might contain inconsistent field values.	Check the RODM log for information on the specific error that occurred. Correct the specific error. Repeat the transaction or run the DUIFFAWS method.
45079	An error occurred while processing a transaction. Some part of the change required for the transaction was completed, but not all of it.	Check the RODM log for information on the specific error that occurred. Correct the specific error. Repeat the transaction.
45080	The value or data type of the data specified by the New_data_ptr parameter for an EKG_ChangeField function request is not valid.	Check the RODM log for information on the specific field where the error occurred. Correct the error. Repeat the transaction.
45082	An error occurred while processing a transaction. The value of the DisplayStatus field of one or more aggregate objects might be incorrect.	Check the RODM log for information on the specific error that occurred. Correct the specific error. Repeat the transaction or run the DUIFFRAS method.
45083	An object passed to the method in the self-defining method parameters is not in the expected class.	Verify that the method parameters are valid. For GMFHS method DUIFCLRT, the first object specified in the method parameters must be a real, aggregate, or shadow object, and the second object specified must be a display resource type object. For GMFHS method DUIFCUAP, the first object specified in the method parameters must be a real or aggregate object and the second object specified must be an aggregate object.
45092	An attempt to connect the GMFHS application to RODM failed. Another GMFHS application is already connected to RODM.	Make sure that the name of the RODM application as specified in the GMFHS initialization member (DUIGINIT) is correct. Only one GMFHS application can connect to RODM at a time.
45093	The version of GMFHS methods installed in RODM is incompatible with the version of the GMFHS application that attempted a connection with RODM.	Make sure that the name of the RODM application as specified in the GMFHS initialization member (DUIGINIT) is correct. The version of the GMFHS application must be the same as the version of the GMFHS methods installed in RODM.

Reason Codes for Return Code 12

Table 200 describes the reason codes that are returned with return code 12.

Table 200. Reason Codes for Return Code 12

Reason Code	Description	Corrective Action
7	Not used.	None

Table 200. Reason Codes for Return Code 12 (continued)

Reason Code	Description	Corrective Action
19	All or some of the response block overflow data is discarded because the overflow block does not have enough storage.	Issue the query response block overflow function to clean up the overflow block. Retry the request using a larger response block. The Response_block_used field in the response block contains the amount of storage needed for the response data.
20	The requested function might not complete because an abend occurred during the transaction.	Verify that the control blocks passed to RODM for the transaction are valid. See the *IBM Tivoli NetView for z/OS Troubleshooting Guide* for information on diagnosing abend problems.
25	The system rejects the request because the transaction tries to update data in a data window currently being written by RODM in a checkpoint process.	Retry the request later.
63	The system rejects the request to create a method object because the system cannot load the module of the specified method into the RODM address space.	Verify that the method exists in the method library.
68	Not used.	None
82	The module of the specified method has been deleted by an unsuccessful module refresh.	Refresh the module of the method again.
118	Not used.	None
121	The system rejects the request because there is not enough storage. Storage has run out in one of the following places: • In the VSAM translation checkpoint data sets • In the translation window	The most likely reason for this return and reason code is that the VSAM data set is too small. If this is the case, message EKG1116I is also written to the console. If you receive this message, increase the size of the RODM translation checkpoint data set. The checkpoint data set size is specified by DDname EKGTRAN in the RODM startup JCL.
122	The system rejects the request because there is not enough storage. Storage has run out in one of the following places: • In the VSAM checkpoint data sets • In the RODM dataspace	The most likely reason for this return and reason code is that the VSAM checkpoint data sets are too small. If this is the case, you will also receive message EKG1117I on the system console. If you receive this message, increase the size of the RODM data window checkpoint data set or add another data window checkpoint data set. The checkpoint data sets are specified by DDname EKGDnnn in the RODM startup JCL.
123	Not used.	None
124	The system rejects the request because there is no ID available for the class.	Delete unused entities. Retry the request.
126	The system rejects the request because there is no ID available for the field.	Delete unused fields. Retry the request.
156	The system rejects the request to create a queue object because there is no storage for the notification queue block.	Delete unused entities. Retry the request later.
157	The system rejects the request because there is no storage for the notification information block.	Retry the request later.
177	The system rejects the request because no system-generated object name is available.	Specify the object name. Retry the request.

Table 200. Reason Codes for Return Code 12 (continued)

Reason Code	Description	Corrective Action
179	The system rejects the request because the system cannot create the user object. The possible cause is that not enough storage is available.	Retry the request later.
188	Not used.	None
194	The system cannot complete the request because the method has an execution error.	Check the RODM log record for further information.
198	The system rejects the request because the system cannot change the fields of the user object. There might not be enough storage available.	Free some storage and retry the request.
199	An operator canceled the user transaction.	Check with the operator.
200	The system cancels the user transaction because of RODM is quiescing.	Retry the request or method later.
211	The system cannot process the error because no storage is available. The storage held is not released. The system cannot be used until it is restarted.	Contact the system administrator to restart RODM.
212	The system cannot complete the transaction because an unrecoverable error occurred. RODM will write a type-2 log record to the RODM log.	Check the content of the log record for information about the transaction that caused the abend. See the *IBM Tivoli NetView for z/OS Troubleshooting Guide* for information on diagnosing abend problems.
213	The requested function did not complete because an abend occurred when RODM accessed the interface blocks of the application or method.	Check the interface blocks for errors that can cause address exceptions. See the *IBM Tivoli NetView for z/OS Troubleshooting Guide* for information on diagnosing abend problems.
216	Not used.	None
240	The RODM transaction did not complete normally. An ABEND might have occurred.	Check the RODM log for information on the specific error that occurred. After correcting the error, repeat the transaction.
600	An EKG_QueryMultipleSubfields request issued by the correlation function failed for one real object.	Ignore this error if the correlation function performed correctly. If the correlation function did not perform correctly, contact IBM Software Support.
601	An EKG_QueryMultipleSubfields request issued by the correlation function failed for one aggregate object.	Ignore this error if the correlation function performed correctly. If the correlation function did not perform correctly, contact IBM Software Support.
602	An EKG_ChangeMultipleFields request issued by the correlation function failed for one aggregate object.	Ignore this error if the correlation function performed correctly. If the correlation function did not perform correctly, contact IBM Software Support.
603	An EKG_Locate request issued by the correlation function failed for one real object.	Ignore this error if the correlation function performed correctly. If the correlation function did not perform correctly, contact IBM Software Support.
603	An EKG_Locate request issued by the correlation function failed for one real object.	Ignore this error if the correlation function performed correctly. If the correlation function did not perform correctly, contact IBM Software Support.

Table 200. Reason Codes for Return Code 12 (continued)

Reason Code	Description	Corrective Action
604	An aggregateSystem object was not created by the correlate function because the correlatable value was less than 2 characters in length.	Ignore this error if the correlation function performed correctly. If the correlation function did not perform correctly, contact IBM Software Support.
605	An EKG_CreateObject request issued by the correlate function failed for one aggregate object.	Ignore this error if the correlation function performed correctly. If the correlation function did not perform correctly, contact IBM Software Support.
606	An EKG_TriggerOIMethod request issued by the correlate function failed to link to a DisplayResourceType for one aggregate object.	Ignore this error if the correlation function performed correctly. If the correlation function did not perform correctly, contact IBM Software Support.
45085	Not used.	None
45086	An error occurred when the objects in a view changed.	Check the RODM log for information on the specific error that occurred. After correcting the error, repeat the transaction.

List of Reason Codes for Each Function

Table 201 lists the function IDs of the RODM API functions and the reason codes returned by each function.

Table 201. Reason Codes for API Functions

Function ID	Reason Codes
Common reason codes for user API	0 1 2 3 5 6 8 9 10 13 14 15 16 17 20 23 25 131 199 200 211 212 213 240
Common reason codes for method API	0 10 18 20 192
1101	30 84 109 127 128 130 179 198 207
1102	180 198
1201	35 36 37 38 202
1202	85 202
1302	24 52 86 87 121 122 124 136 140 163 165 186 32768 32769 32770 32772
1303	24 52 89 90 136 140 163 165 32768 32769 32770 32772
1304	52 91 92 93 94 95 96 97 100 121 122 126 139 140 159 160 163 164 165 166
1305	52 60 98 103 139 140 144 163 164 165 166
1306	52 100 103 104 122 139 140 148 163 164 165 166 187
1307	52 60 98 103 106 139 140 144 146 163 164 165 166 187
1401	24 26 42 52 54 56 57 65 66 67 80 81 122 133 134 135 136 139 140 142 145 147 150 163 164 165 166 176 194 195 215 230 231 232 32768 32769 32770 32771 32772

Table 201. Reason Codes for API Functions (continued)

Function ID	Reason Codes
1402	24 26 39 52 54 56 57 65 66 67 80 81 122 133 134 135 136 138 139 140 142 145 147 150 163 164 165 166 176 194 195 215 230 231 232 32768 32769 32770 32772
1403	26 52 54 56 57 61 62 65 66 67 70 71 81 122 135 139 140 142 145 147 150 163 164 165 166 176 195 215 230 231 232
1404	26 39 52 54 56 57 61 62 65 66 67 70 71 81 122 135 138 139 140 142 145 147 150 163 164 165 166 176 195 215 230 231 232
1405	24 52 54 56 57 72 73 74 122 133 136 139 140 141 145 150 163 164 166 214 220 32768 32769 32770 32772
1406	52 54 56 57 72 73 74 122 133 139 140 141 145 150 163 164 166 214
1407	24 52 54 56 57 73 74 75 133 136 139 140 145 150 163 164 166 214 220 32768 32769 32770 32772
1408	52 54 56 57 73 74 75 133 139 140 145 150 163 164 166 214
1409	22 24 34 52 63 107 109 110 121 122 136 140 150 156 159 163 165 170 177 214 604 605 32768 32769 32770 32772
1410	22 24 52 54 57 81 107 108 111 113 136 140 150 163 170 191 214 32768 32769 32770 32772
1411	40 41 52 54 56 57 61 62 65 67 70 71 122 139 140 145 150 163 164 165 166
1412	22 52 54 56 57 76 77 81 112 122 135 139 140 150 156 163 164 165 166 173
1413	22 52 54 56 57 76 77 113 135 139 140 150 163 164 165 166
1415	42 52 54 56 57 80 81 82 115 136 139 140 150 163 164 165 166 191 194 214 32768 32771
1416	11 42 80 81 191 194
1417	22 52 54 57 77 109 112 122 135 140 150 156 163 173 214
1418	22 52 54 57 77 109 113 135 140 150 163 214
1419	24 26 42 52 54 56 57 65 66 67 80 81 122 133 134 135 136 139 140 142 145 147 150 163 164 165 166 176 194 215 223 227 230 231 232 602 32768 32769 32770 32771 32772
1501	19 27 42 52 54 56 57 80 83 136 139 140 143 150 163 164 165 166 194 208
1502	19 27 52 54 56 57 61 62 83 136 139 140 143 150 163 164 165 166 208
1503	19 27 52 54 57 83 139 140 150 163 164 165 166 208
1504	19 27 52 54 56 57 83 139 140 150 163 164 165 166 208

Table 201. Reason Codes for API Functions (continued)

Function ID	Reason Codes
1505	19 27 56 139 164 166 208
1506	19 27 52 54 56 57 83 139 140 150 163 164 165 166 208
1507	19 22 27 44 54 57 83 183 208
1508	27 52 56 83 139 140 150 163 164 165 166 223 600 601
1509	139 164 166 224 225 227 228 229 603
1510	46 47 120
1600	21 79 83 117
2001	27 83
2002	21 52 54 118
2004	27 83 201 203 204
2005	22 50 157 158 174 175 181 182
2006	28 29 33 193 45086
2008	28 29 33
2011	19 27 54 169 208
EKGWAIT	209 210

List of Functions for Each Reason Code

Table 202 lists the RODM reason codes and the function IDs of the RODM API functions that return each reason code. See Table 203 on page 478 to resolve a function ID to its function name.

Table 202. Function IDs for Each Reason Code

Reason Code	Function ID
0	Common reason code for user API and method API.
1	Common reason code for user API.
2	Common reason code for user API.
3	Common reason code for user API.
5	Common reason code for user API.
6	Common reason code for user API.
8	Common reason code for user API.
9	Common reason code for user API.
10	Common reason code for user API and method API.
11	1416
13	Common reason code for user API.
14	Common reason code for user API.
15	Common reason code for user API.
16	Common reason code for user API.
17	Common reason code for user API.

Table 202. Function IDs for Each Reason Code (continued)

Reason Code	Function ID
18	Common reason code for method API.
19	1501 1502 1503 1504 1505 1506 1507 2011
20	Common reason code for user API and method API.
21	1600 2002
22	1409 1410 1412 1413 1417 1418 1507 2005
23	Common reason code for user API.
24	1302 1303 1401 1402 1405 1407 1409 1410 1419
25	Common reason code for user API.
26	1401 1402 1403 1404 1419
27	1501 1502 1503 1504 1505 1506 1507 1508 2001 2004 2011
28	2006 2008
29	2006 2008
30	1101
33	2006 2008
34	1409
35	1201
36	1201
37	1202
38	1201
39	1402 1404
40	1411
41	1411
42	1401 1402 1415 1416 1419 1501
44	1507
46	1510
47	1510
50	2005
52	1302 1303 1304 1305 1306 1307 1401 1402 1403 1404 1405 1406 1407 1408 1409 1410 1411 1412 1413 1415 1417 1418 1419 1501 1502 1503 1504 1506 1508 2002
54	1401 1402 1403 1404 1405 1406 1407 1408 1410 1411 1412 1413 1415 1417 1418 1419 1501 1502 1503 1504 1506 1507 2002 2011
56	1401 1402 1403 1404 1405 1406 1407 1408 1411 1412 1413 1415 1419 1501 1502 1504 1505 1506 1508
57	1401 1402 1403 1404 1405 1406 1407 1408 1410 1411 1412 1413 1415 1417 1418 1419 1501 1502 1503 1504 1506 1507
60	1305 1307

Table 202. Function IDs for Each Reason Code (continued)

Reason Code	Function ID
61	1403 1404 1411 1502
62	1403 1404 1411 1502
63	1409
65	1401 1402 1403 1404 1411 1419
66	1401 1402 1403 1404 1419
67	1401 1402 1403 1404 1411 1419
70	1403 1404 1411
71	1403 1404 1411
72	1405 1406
73	1405 1406 1407 1408
74	1405 1406 1407 1408
75	1407 1408
76	1412 1413
77	1412 1413 1417 1418
79	1600
80	1401 1402 1415 1416 1419 1501
81	1401 1402 1403 1404 1410 1412 1415 1416 1419
82	1415
83	1501 1502 1503 1504 1506 1507 1508 1600 2001 2004
84	1101
85	1202
86	1302
87	1302
89	1303
90	1303
91	1304
92	1304
93	1304
94	1304
95	1304
96	1304
97	1304
98	1305 1307
100	1304 1306
103	1305 1306 1307
104	1306
106	1307
107	1409 1410
108	1410

Table 202. Function IDs for Each Reason Code (continued)

Reason Code	Function ID
109	1101 1409 1417 1418
110	1409
111	1410
112	1412 1417
113	1410 1413 1418
115	1415
117	1600
118	2002
120	1510
121	1302 1304 1409
122	1302 1304 1306 1401 1402 1403 1404 1405 1406 1409 1411 1412 1417 1419
124	1302
126	1304
127	1101
128	1101
130	1101
131	Common reason code for user API.
133	1401 1402 1405 1406 1407 1408 1419
134	1401 1402 1419
135	1401 1402 1403 1404 1412 1413 1417 1418 1419
136	1302 1303 1401 1402 1405 1407 1409 1410 1415 1419 1501 1502
138	1402 1404
139	1304 1305 1306 1307 1401 1402 1403 1404 1405 1406 1407 1408 1411 1412 1413 1415 1419 1501 1502 1503 1504 1505 1506 1508 1509
140	1302 1303 1304 1305 1306 1307 1401 1402 1403 1404 1405 1406 1407 1408 1409 1410 1411 1412 1413 1415 1417 1418 1419 1501 1502 1503 1504 1506 1508
141	1405 1406
142	1401 1402 1403 1404 1419
143	1501 1502
144	1305 1307
145	1401 1402 1403 1404 1405 1406 1407 1408 1411 1419
146	1307
147	1401 1402 1403 1404 1419
148	1306
150	1401 1402 1403 1404 1405 1406 1407 1408 1409 1410 1411 1412 1413 1415 1417 1418 1419 1501 1502 1503 1504 1506 1508

Table 202. Function IDs for Each Reason Code (continued)

Reason Code	Function ID
156	1409 1412 1417
157	2005
158	2005
159	1304 1409
160	1304
163	1302 1303 1304 1305 1306 1307 1401 1402 1403 1404 1405 1406 1407 1408 1409 1410 1411 1412 1413 1415 1417 1418 1419 1501 1502 1503 1504 1506 1508
164	1304 1305 1306 1307 1401 1402 1403 1404 1405 1406 1407 1408 1411 1412 1413 1415 1419 1501 1502 1503 1504 1505 1506 1508 1509
165	1302 1303 1304 1305 1306 1307 1401 1402 1403 1404 1409 1411 1412 1413 1415 1419 1501 1502 1503 1504 1506 1508
166	1304 1305 1306 1307 1401 1402 1403 1404 1405 1406 1407 1408 1411 1412 1413 1415 1419 1501 1502 1503 1504 1505 1506 1508 1509
169	2011
170	1409 1410
173	1412 1417
174	2005
175	2005
176	1401 1402 1403 1404 1419
177	1409
179	1101
180	1102
181	2005
182	2005
183	1507
186	1302
187	1306 1307
191	1410 1415 1416
192	Common reason code for method API.
193	2006
194	1401 1402 1415 1416 1419 1501
195	1401 1402 1403 1404
198	1101 1102
199	Common reason code for user API.
200	Common reason code for user API.
201	2004
202	1201 1202
203	2004

Table 202. Function IDs for Each Reason Code (continued)

Reason Code	Function ID
204	2004
207	1101
208	1501 1502 1503 1504 1505 1506 1507 2011
209	EKGWAIT
210	EKGWAIT
211	Common reason code for user API.
212	Common reason code for user API.
213	Common reason code for user API.
214	1405 1406 1407 1408 1409 1410 1415 1417 1418
215	1401 1402 1403 1404 1419
220	1405 1407
223	1419 1508
224	1509
225	1509
226	1101
227	1419 1509
228	1509
229	1509
230	1401 1402 1403 1404 1419
231	1401 1402 1403 1404 1419
232	1401 1402 1403 1404 1419
240	Common reason code for user API.
600	1508
601	1508
602	1419
603	1509
604	1409
605	1409
32768	1302 1303 1401 1402 1405 1407 1409 1410 1415 1419
32769	1302 1303 1401 1402 1405 1407 1409 1410 1419
32770	1302 1303 1401 1402 1405 1407 1409 1410 1419
32771	1401 1415 1419
32772	1302 1303 1401 1402 1405 1407 1409 1410 1419
45086	2006

List of Function Names by Function ID

Table 203 lists the RODM API function names by their function ID.

Table 203. Function Names by Function ID

Function ID	Function Name
1100	EKG_ConnectLong
1101	EKG_Connect
1102	EKG_Disconnect
1201	EKG_Checkpoint
1202	EKG_Stop
1302	EKG_CreateClass
1303	EKG_DeleteClass
1304	EKG_CreateField
1305	EKG_DeleteField
1306	EKG_CreateSubfield
1307	EKG_DeleteSubfield
1401	EKG_ChangeField
1402	EKG_SwapField
1403	EKG_ChangeSubfield
1404	EKG_SwapSubfield
1405	EKG_LinkTrigger
1406	EKG_LinkNoTrigger
1407	EKG_UnLinkTrigger
1408	EKG_UnLinkNoTrigger
1409	EKG_CreateObject
1410	EKG_DeleteObject
1411	EKG_RevertToInherited
1412	EKG_AddNotifySubscription
1413	EKG_DeleteNotifySubscription
1415	EKG_TriggerNamedMethod
1416	EKG_TriggerOIMethod
1417	EKG_AddObjDelSubs
1418	EKG_DelObjDelSubs
1419	EKG_ChangeMultipleFields
1501	EKG_QueryField
1502	EKG_QuerySubfield
1503	EKG_QueryEntityStructure
1504	EKG_QueryFieldStructure
1505	EKG_QueryFieldID
1506	EKG_QueryFieldName
1507	EKG_QueryNotifyQueue
1508	EKG_QueryMultipleSubfields
1509	EKG_Locate

Table 203. Function Names by Function ID (continued)

Function ID	Function Name
1510	EKG_QueryResponseBlockOverflow
1600	EKG_ExecuteFunctionList
2001	EKG_QueryFunctionBlockContents
2002	EKG_LockObjectList
2003	EKG_UnlockAll
2004	EKG_ResponseBlock
2005	EKG_SendNotification
2006	EKG_SetReturnCode
2007	EKG_WhereAmI
2008	EKG_OutputToLog
2009	EKG_MessageTriggeredAction
2011	EKG_QueryObjectName

List of Reason Codes from Supplied Methods

Table 204 lists the methods that are supplied with the NetView program and the reason codes that are returned by each method.

Table 204. Reason Codes for Supplied Methods

Method	Reason Codes
DUIFCATC	45070 45077 45078 45081 45088
DUIFCCAN	45077 45081 45088
DUIFCLRT	45070 45071 45077 45078 45081 45083 45088
DUIFCUAP	45065 45070 45071 45077 45081 45088
DUIFCUUS	45070 45077 45078 45081 45088
DUIFECDS	45070 45077 45078 45079 45081 45088
DUIFFAWS	45088
DUIFFIRS	45070 45077 45078 45081 45088
DUIFFRAS	45077 45081 45088
DUIFFSUS	45070 45077 45078 45081 45088
DUIFRFDS	45077 45081 45088
DUIFVCFT	45070 45077 45081 45088
EKGCTIM	32768 32771 32780
EKGMIMV	32768 32771 32780
EKGNEQL	32768 32769 32770 32772 32780 32954
EKGNLST	32768 32769 32770 32772 32780 32954
EKGNOTF	32768 32770 32780 32954
EKGNTHD	32768 32769 32770 32772 32780 32954
EKGSPPI	32780 32790 32791 32792 32793 32794 32795 32796 32797 32798 32809+

Maximizing RODM Performance

This section describes how to maximize system performance while running RODM. The structure and size of the data model, the design of methods, and the design of user applications all affect performance.

Data Model Structure and Size

Execution time for some functions increases as the number of classes between the object and the universal class (root) increases. Keep the number of vertical classes to a minimum. For best performance, do not exceed 100 vertical classes.

Method Design

Use functions that do not trigger methods whenever possible in methods you write. This limits the scope of actions resulting from a single transaction and reduces system utilization.

User Application Design

If you do not need to trigger the query method for a field, and your data model contains many vertical classes, you can improve performance by using the query subfield function instead of the query field function.

The RODM notification process uses resources for each notification subscription. Delete any unneeded notification subscriptions.

Customization Parameters and System Fields

For the best performance, set the RODM logging levels so that logging is kept to a minimum. The suggested value for the LOG_LEVEL and MLOG_LEVEL customization parameters, and the corresponding EKG_LogLevel and EKG_MLogLevel fields in the EKG_User class is 8.

Note: Values smaller than 8 can cause GMFHS to report method errors.

Indexed Fields

The EKG_Locate function makes it easier for an application to retrieve a list of Object IDs. Rather than scanning the entire data model using the query field functions, use the EKG_Locate function which scans just the tables that contain the Object IDs.

For better performance, the indexed field must be created before populating the data model. Improved performance can also be gained by ensuring that objects have indexed field values where the first 254 bytes are unique.

Supplied Methods

This section provides a brief introduction to the methods that are supplied with the NetView program. These methods are provided with RODM to supply specific kinds of functions. You can replace a supplied method and add locally developed ones. Supplied methods use the method API. These methods are described in this section on a functional basis. All parameters passed to methods are specified as SelfDefining data strings.

RODM Notification Methods

In addition to notifying the required subscriber that the data has changed, all RODM notification methods return the values of the value, prev_val, and timestamp subfields. This data is returned to the subscriber in the User_area of the notification queue block. See "EKG_QueryNotifyQueue — Query Notification Queue" on page 421 for a description of this block. If the User_area cannot contain all the data, a null data string is returned. The order of the data returned is:

1. The value in the value subfield
2. The value in the prev_val subfield
3. The value in the timestamp subfield

The data type of the returned data is SelfDefining. Each output value is preceded by a tag code (corresponding to the numbers 1, 2, and 3 above) to identify which subfield the data came from. If a particular subfield is not defined, that tag code is not included in the SelfDefining data string. Table 205 is an example of the data that is returned in the data string.

Table 205. Example User_data Returned with EKGNOTF Notification Method

Offset	Length	Value	Description
000	2	34	Total length of SelfDefining string
002	2	21	Smallint data type code
004	2	01	Value field indicator
006	2	10	Value data type flag (Integer)
008	4	*value*	Value data (Integer)
012	2	21	Smallint data type code
014	2	02	Prev_val field indicator
016	2	10	Prev_val data type flag (Integer)
018	4	*prev_val*	Prev_val data (Integer)
022	2	21	Smallint data type code
024	2	03	Timestamp field indicator
026	2	27	Timestamp data type flag (TimeStamp)
028	8	*timestamp*	Timestamp data (TimeStamp)

The notification methods that are supplied with the NetView program notify subscribers only when the data value of the field changes such that the new value is different from the old value. In addition, each method must be passed a parameter specifying how the notification must be performed, as follows:

Always
A notification is sent to the subscriber specified to the method through its invocation parameters each time the method is run.

Once A single notification is generated and the method then deletes itself from the field's notification list.

If a notification method is installed on the field of an object, then when a change is made to the object field, the notification subscriptions assigned to that field are run. After the notifications of the object are processed, any notification subscriptions assigned to the same field in the primary parent are run.

Methods that perform comparison operations to determine if a notification generated can be assigned only to fields of the following data types:

- Smallint
- Integer
- Float
- TimeStamp
- CharVar

EKGNOTF: General Notification

Function

Notify its subscriber of any change to the associated field value.

Long-lived-parameters

A 2-byte integer code designating the execution option of Always or Once.

Table 206. EKGNOTF Long-lived-parameter Description

Offset	Length	Value	Description
000	2	8	Total length of SelfDefining string
002	2	21	Smallint data type code
004	2	1 or 2	Two byte integer (1=always, 2=once)
006	2	21	Smallint data type code
008	2	1 or 2	Two byte integer (1=notify only if new value is different from previous value, 2=notify always)

Short-lived-parameters

None required.

EKGNEQL: Notify If Equal

Function

Notify its subscriber when any change to the associated field value causes that field to be equal to the long-lived-parameter. The function must be sensitive to all supported RODM data types in order to determine how to make the appropriate comparison.

Long-lived-parameters

A 2-byte integer code designating the execution option of always or once followed by the value being tested against the subscribed field. The long-lived-parameter specifies a Field_ID within the current object where the test value is specified.

Table 207. EKGNEQL Long-lived-parameter Description

Offset	Length	Value	Description
000	2	14	Total length of SelfDefining string
002	2	21	Smallint data type code
004	2	1 or 2	Two byte integer (1=always, 2=once)
006	2	21	Smallint data type code
008	2	1 or 2	Two byte integer (1=notify only if new value is different from previous value, 2=notify always)
010	2	26	Smallint data type code (FieldID)
012	4	Field_ID	Field_ID of test value

Short-lived-parameters

None required.

EKGNLST: Notify if Equal to List

Function

Notify its subscriber when any change to the associated field value causes that field to equal one of the values in the long-lived-parameter. The function must be sensitive to all supported RODM data types in order to determine how to make the appropriate comparison.

Long-lived-parameters

A 2-byte integer code designating the execution option of always or once followed by the number of values in the list and the list of values being tested against the subscribed field. The long-lived-parameter specifies a Field_ID within the current object where the comparison list count is specified and a list of Field_IDs where the test values are specified.

Table 208. EKGNLST Long-lived-parameter Description

Offset	Length	Value	Description
000	2	14+(N*6)	Total length of SelfDefining string
002	2	21	Smallint data type code
004	2	1 or 2	Two byte integer (1=always, 2=once)
006	2	21	Smallint data type code
008	2	1 or 2	Two byte integer (1=notify only if new value is different from previous value, 2=notify always)
010	2	10	Smallint data type code (Integer)
012	4	N (range 0..n)	Number of following Field_IDs
016	2	26	Smallint data type code (FieldID)
018	4	Field_ID	Field_ID of first test value **Note:** Element of array
010+(N*6)	2	26	Smallint data type code (FieldID)
012+(N*6)	4	Field_ID	Field_ID of Nth test value

Short-lived-parameters

None required.

EKGNTHD: Notify If Outside Threshold

Function

Notify its subscriber when any change to the associated field value causes that field to fall outside the threshold specified in the long-lived-parameter. This method provides three options.

- The user specifies an upper bound. Subscribers are notified if the value of the associated field is greater than the parameter.
- The user specifies a lower bound. Subscribers are notified if the value of the associated field is less than the parameter.
- The user specifies a pair of parameter values. Subscribers are notified if value of the associated field is greater than the larger parameter or less than the smaller parameter.

Long-lived-parameters

A 2-byte integer code designating the execution option of always or once, followed by the particular function being performed and the threshold values. The long-lived-parameter specifies a Field_ID within the current object where the function code is specified and Field_IDs as required to specify the threshold values.

Table 209. EKGNTHD Long-lived-parameter Description

Offset	Length	Value	Description
000	2	20 or 26	Total length of SelfDefining string
002	2	21	Smallint data type code
004	2	1 or 2	Two byte integer (1=always, 2=once)
006	2	21	Smallint data type code
008	2	1 or 2	Two byte integer (1=notify only if new value is different from previous value, 2=notify always)
010	2	10	Integer data type code
012	4	1, 2, or 3	Option code (1=upper bound, 2=lower bound, 3=range)
016	2	26	Smallint data type code (FieldID)
018	4	Field_ID	For 1, upper bound; for 2 or 3, lower bound
: Next parameters for option code 3 only			
022	2	26	Smallint data type code (FieldID)
024	4	Field_ID	For 3, upper bound

Short-lived-parameters
None required.

RODM Change Methods

EKGCTIM: Trigger Object-Independent Method

Function
Using the message facility, trigger an object-independent method to perform some designated function asynchronous to the execution of the invoking method. If, for example, the object-independent method is intended to communicate with a real status sender, a SWAP function block can be passed, in order to communicate old and new value information from the field associated with this method. This can let the object-independent method tell the real status sender to change a real device status from old to new state.

Long-lived-parameters
List of Field_IDs where data is provided to build the required function block to be passed to the object-independent method. Each consecutive 4 bytes of this parameter string is interpreted as a FieldID of a field within the current object. The specified fields are queried, and the information is placed in the function block of the EKG_TriggerOIMethod function.

Table 210. EKGCTIM Long-lived-parameter Description

Offset	Length	Value	Description
000	2	12	Total length of SelfDefining string
002	2	26	Smallint data type code (FieldID)
004	4	Field_ID	Field containing method name
008	2	26	Smallint data type code (FieldID)
010	4	Field_ID	Field containing Short-lived-parameter list as a SelfDefining string

Short-lived-parameters
None required.

RODM Named Methods

EKGMIMV: Increment Value

Function
Increment the value of a specified field, defined within the current object, by a specified value.

Long-lived-parameters
Two Field_IDs are required. The first four bytes of the string specifies the Field_ID of field to be incremented. The second four bytes specifies the Field_ID of the field containing the increment value. These fields must be integer data type and the increment value can be negative causing the designated field value to be decremented.

Table 211. EKGMIMV Long-lived-parameter Description

Offset	Length	Value	Description
000	2	12	Total length of SelfDefining string
002	2	26	Smallint data type code (FieldID)
004	4	Field_ID	Field to be incremented
008	2	26	Smallint data type code (FieldID)
010	4	Field_ID	Field containing increment value

Short-lived-parameters
None required.

EKGCTIM: Trigger Object-Independent Method

This is the same function as the change method described for this function. This method, when installed in RODM, can be used in either manner.

RODM Object-Independent Methods

EKGSPPI: Send a command to NetView

The EKGSPPI method is one of the services in the RODM automation platform. See Chapter 8, “Using the RODM Automation Platform,” on page 189 for more information about automation tasks using NetView. An extensive RODM automation scenario using the EKGSPPI method and the automation platform is contained in the chapter entitled the *IBM Tivoli NetView for z/OS Automation Guide*.

Function: This object-independent method sends commands to the DSIQTSK task in NetView. DSIQTSK then dispatches the commands to an autotask, which issues the commands. NetView supplies two example methods that call the EKGSPPI, one change method named EKGCPPI and one object-independent method named EKGOPPI. You can use these example methods as models for your own methods that trigger EKGSPPI.

The best way to trigger the EKGSPPI method is using the EKG_MessageTriggeredAction function. This enables EKGSPPI to run asynchronously. The EKG_MessageTriggeredAction function specifies the EKG_TriggerOIMethod function, which contains the parameters passed to EKGSPPI.

Long-lived parameters: None required.

Short-lived parameters: EKGSPPI accepts a short-lived parameter with the SelfDefining data type. The short-lived parameter contains seven data items. Each data item is data type CharVar or data type AnonymousVar. All seven data items must appear in the order shown, but some can have a null value. The EKGSPPI method deletes leading blank characters from the value specified for each data item.

The names used for the data items are the variable names used in the sample methods EKGCPPI and EKGOPPI. The seven data items in the short-lived parameter are:

`RCVRID_CHARVAR`
This data item specifies the name of the command receiver to which EKGSPPI sends commands. This is the name supplied on the ID field of the CMDRCVR defined in the DSIQTSKI initialization file for the DSIQTSK task. The EKGSPPI method converts this name to uppercase. This name has a maximum of 8 characters.

`ASSIST_CHARVAR`
This data item specifies whether the command is to be sent to a NetView operator before it is run. The command is issued in the form of a message (DWO670I). If SAVECMD is specified in the automation table trap for DWO670I, the command can be saved for the operator that the SAVECMD is routed to. The operator can use the ASSISCMD to display the command on the panel. The operator can issue, modify, or cancel the command from the NetView assist panel. Valid values are:

Value	**Meaning**
ASSIST	Send the command to an operator
NOASSIST	Issue the command without sending it to an operator
null or blanks	Issue the command without sending it to an operator

This value has a maximum of 8 characters.

`TASKINFO_CHARVAR`
This data item specifies whether the command is run by a specific NetView autotask. Valid values are:

Value	**Meaning**
ANY	DSIQTSK routes the command to the next autotask (after the most recently used autotask) defined to DSIQTSK. Autotasks are used in the order in which they are defined in the DSIQTASKI member of DSIPARM.
ONLY	DSIQTSK routes the command to the autotask specified in the short-lived parameter data item TASKNAME_CHARVAR. If the specified autotask is not available, the command is not issued.
ONLYANY	DSIQTSK routes the command to the autotask specified in the short-lived parameter data item TASKNAME_CHARVAR. If the specified autotask is not available, DSIQTSK routes the command to the next autotask (after the most recently used autotask) defined to

DSIQTSK. Autotasks are used in the order in which they are defined in the DSIQTASKI member of DSIPARM.

null or blanks
DSIQTSK routes the command to the next autotask (after the most recently used autotask) defined to DSIQTSK. Autotasks are used in the order in which they are defined in the DSIQTASKI member of DSIPARM.

This value has a maximum of 8 characters.

`TASKNAME_CHARVAR`
This data item specifies the name of the autotask that DSIQTSK routes the command to. This is the name specified by the TASK statement of DSIQTSKI, the initial member of DSIQTSK task. If TASKINFO_CHARVAR is ONLY or ONLYANY, TASKNAME_CHARVAR is required. The EKGSPPI method converts this name to uppercase. This value has a maximum of 8 characters.

`SENDER_CHARVAR`
This data item identifies the sender of the command for commands which specify ASSIST_CHARVAR as ASSIST. This name is included in the message sent to the operator. The EKGSPPI method converts this name to uppercase. This value has a maximum of 8 characters.

`CMD_CHARVAR`
This data item specifies the command to be issued. A COMMAND_CHARVAR value is required. This value has a maximum of 240 characters.

`CMD_DESC_CHARVAR`
This data item specifies a description of the command to be issued. You can specify blanks or null for this value. This value has a maximum of 780 characters. This description is displayed on the assist panel if ASSIST is specified for the ASSIST_CHARVAR data item in short-lived parameters.

Output: The command is sent to the DSIQTSK task in NetView.

You can run the EKGSPPI method using the RODM load function. Figure 88 on page 488 shows an example of invoking EKGSPPI using a RODM load function primitive statement.

Note: The RODM load function is not an APF (authorized program facility) authorized program. If the NetView program-to-program interface command receiver managed by DSIQTSK requires APF authorization, the job fails and a return code of 8 with a reason code of 32832 is issued by the EKGSPPI method.

```
OP EKGSPPI INVOKED_WITH       -- Trigger the EKGSPPI method --
   (SELFDEFINING)
   ( (CHARVAR) 'CNM01'        -- Command receiver name --
     (CHARVAR) 'NOASSIST'     -- Issue without operator intervention --
     (CHARVAR) 'ONLYANY'      -- Use named autotask if available --
     (CHARVAR) 'AUTO1'        -- Autotask name --
     (CHARVAR) 'LOAD FUN'     -- Name of sender of command --
     (CHARVAR) 'some reasonable command goes here'
                              -- Command to be sent --
     (CHARVAR) 'This command is sent using the RODM load function.'
               ' It is an example of triggering the EKGSPPI method '
               ' using a RODM load function primitive statement.'
                              -- Command description --             );
```

Figure 88. Example RODM Load Function Primitive Statement to Invoke EKGSPPI

GMFHS Methods

The methods described in this section are supplied for use with GMFHS. You can also use these methods with automation code that you write.

Use only these GMFHS methods for the described purposes. For example, do not use a named method as an object-independent method.

In addition to the GMFHS methods described in this section, GMFHS uses other methods which cannot be used by your programs. Do not use the methods in this list with programs you write:

- DUIFCAAP
- DUIFCADT
- DUIFCAPC
- DUIFCASB
- DUIFCATC
- DUIFCCAP
- DUIFCDTC
- DUIFCDUC
- DUIFCGRA
- DUIFCGRT
- DUIFCGR2
- DUIFCGR3
- DUIFCLSR
- DUIFCLS2
- DUIFCLS3
- DUIFCMUU
- DUIFCRDC
- DUIFCRTP
- DUIFCRTU
- DUIFCRUC
- DUIFCSRT
- DUIFCURA
- DUIFCUTC
- DUIFEGSN
- DUIFITKN
- DUIFRAIP
- DUIFRRTC
- DUIFVCVT
- DUIFVDRT
- DUIFVEFC

- DUIFVEVF
- DUIFVEXV
- DUIFVFPV
- DUIFVGET
- DUIFVIEW
- DUIFVLST
- DUIFVLTT
- DUIFVMDR
- DUIFVNGI
- DUIFVNGN
- DUIFVNOI
- DUIFVNOT
- DUIFVPFR
- DUIFVSUB
- DUIFVTKN
- DUIFVUNS
- DUIFVUPD
- DUIFVVLC

DUIFCCAN: Clear All Notes

This object-independent method can be run by any application to clear the note field on all UserStatus flags for all real and aggregate objects in RODM.

Function: Use the DUIFCCAN method to clear all note fields without going through the topology console for each real and aggregate object. An operator ID of "DUIFCCAN" will be set to indicate that the note was cleared by this method, instead of an operator.

Input: This method does not require input parameters and can be triggered with the following RODM load function primitive statement:

```
OP DUIFCCAN INVOKED WITH;
```

Output: If this method encounters errors, it sets a return and reason code and writes a type 1 record to the RODM log. Table 204 on page 479 lists the reason codes that can be returned by this method.

DUIFCLRT: Link Resource Type Method

This method is an object-independent method that is run to link or unlink:

- The DisplayResourceType field of a real, aggregate, or shadow object to the Resources field of an object of the Display_Resource_Type_Class.
- The DisplayResourceType field of a View_Information_Reference_Object to the Resources field of an object of the Display_Resource_Type_Class.

Function: Use the DUIFCLRT method to ensure that the DisplayStatus field value of the affected aggregate resources is recalculated when the DisplayResourceType field of a real or aggregate resource is changed. These changes might occur:

- If the DisplayResourceType value of a GMFHS_Managed_Real_Objects_Class object is changed, the DefaultAggregationPriorityCopy value of that object might need to be changed. If this change affects the effective aggregation priority of that real resource, the aggregate resources affected by that change must be updated and their DisplayStatus values recalculated. To make this change, the DUIFCLRT method triggers the DUIFCAPC method.
- If the DisplayResourceType link is changed in an object of the GMFHS_Aggregate_Objects_Class, GMFHS recalculates the DisplayStatus field value for that aggregate.

The DUIFCLRT method cannot be triggered by other methods, including the EKGLISLM and EKGLIILM initialization methods. Do not trigger the DUIFCUAP method using another method.

Figure 89 is an example of triggering the DUIFCLRT method using a RODM load function primitive statement.

```
OP DUIFCLRT INVOKED_WITH (SELFDEFINING)
 (
  (SMALLINT) 1
  (CHARVAR) '
  (CHARVAR) 'Display_Resource_Type_Class.DUIXC_RTN_NN_DOMAIN_AGG'
  (OBJECTID) 'View_Information_Reference_Class'.
   '1.3.18.0.0.2150_Reference'
 );
```

Figure 89. RODM Load Function Primitive Statement Invoking DUIFCLRT

Input: Specify the input parameters to the DUIFCLRT method using ***three*** of the four items in a SELFDEFINING data type. The items are summarized in Table 212, followed by a complete description of each item.

Table 212. Input Values for DUIFCLRT Operation

Item	Description	Data Type	Required/Optional
1	Link or unlink	CHARVAR or SMALLINT	Required
2	Resource object	CHARVAR or OBJECTID	Optional [1]
3	Display resource type	CHARVAR or OBJECTID	Required
4	View information reference object	CHARVAR or OBJECTID	Optional [1]

Note: [1] Either the Resource Object or the View Information Resource Object must be specified; however, both cannot be specified.

1 The first item specifies the operation, and can be the CHARVAR data type or the SMALLINT data type. Valid values are:

Table 213. Input Values for DUIFCLRT Operation

Operation	CHARVAR	SMALLINT
Link resources	LINK	1
Unlink resources	UNLINK	2

2 The second item specifies the real, aggregate, or shadow object being linked or unlinked, and can be the CHARVAR data type or the OBJECTID data type. This item is optional, however, if it is not specified, the fourth item must be specified. If you are not specifying this item, the null character must be specified. For example, use the following code:

```
(CHARVAR) '
```

For a CHARVAR item, specify the class name and the object name separated with a period. For an OBJECTID item, specify the class name within single quotation marks and the object name within single quotation marks, separated by a period. For example, use the following code:

```
(CHARVAR) 'Display_Resource_Type_Class.DUIXC_RTN_NN_DOMAIN_AGG'
(OBJECTID) 'Display_Resource_Type_Class'.'DUIXC_RTN_NN_DOMAIN_AGG'
```

If the class name or object name used in a CHARVAR data item contains a period, enclose the name in two single quotation marks. For example, if the class name was `Class.name`, use the following code:

```
(CHARVAR)'Class.name'.Object'
```

If the class name or object name used in a CHARVAR or OBJECTID data item contains a single quotation mark (') character, use two single quotation marks to specify the single quotation mark. For example, if the name of an object was `Greg'sObject`, use the following code:

```
(CHARVAR)'Class.Greg'sObject'
```

3 The third item specifies the Display_Resource_Type_Class object being linked or unlinked. This item is required. The format for the third item is the same as the format for the second item.

4 The fourth item specifies the View_Information_Reference _Object being linked or unlinked. This item is optional; however, if it is not specified, the second item must be specified. If you are not specifying this item, the null character must be specified. For example, use the following code:

```
(CHARVAR) '
```

The format for the fourth item is the same as the format for the second item.

Output: The link or unlink is performed.

If this method encounters errors, it sets a return and reason code and writes a type1 record to the RODM log. Table 204 on page 479 lists the reason codes that can be returned by this method.

DUIFCUAP: Update Aggregation Path Method

This is an object-independent method which is to be run whenever two resource objects are to be linked or unlinked using the AggregationChild field in an object of the GMFHS_Aggregate_Objects_Class and the AggregationParent field in a different GMFHS_Aggregate_Objects_Class object or GMFHS_Managed_Real_Objects_Class object.

Function: Use this method to ensure that the "Value." (count) fields and the DisplayStatus field value in the aggregate resource and its aggregation ancestors above the link or unlink are updated to reflect the addition (for a link) or deletion (for an unlink) of real resource aggregation descendants.

Use of this method also prevents the introduction of loops into the aggregation hierarchy. An aggregation hierarchy loop occurs when the AggregationParent field of an aggregate object contains a link to the AggregationChild field of the same object or to an object that has an AggregationParent field that is linked either directly or through other aggregate objects to the AggregationChild field of the first aggregate object.

While GMFHS is operating, use only the DUIFCUAP method to add aggregate resources to or delete aggregate resources from aggregation hierarchies. Note that this requirement is not enforced by RODM.

GMFHS only uses the DUIFCUAP method indirectly, using the RODM load function because GMFHS does not otherwise change the aggregation hierarchy.

The DUIFCUAP method cannot be triggered by other methods, including the EKGLISLM and EKGLIILM initialization methods. Do not trigger DUIFCUAP using another method. Figure 90 on page 492 is an example of triggering the

DUIFCUAP method using a RODM load function primitive statement.

```
OP DUIFCUAP INVOKED_WITH (SELFDEFINING)
   ((CHARVAR)'LINK'
   (CHARVAR)'GMFHS_Aggregate_Objects_Class.ETHERNET'
   (CHARVAR)'GMFHS_Aggregate_Objects_Class.WESTCTR');
```

Figure 90. RODM load function primitive statement invoking DUIFCUAP

Input: Specify the input parameters to the DUIFCUAP method using three items in a SELFDEFINING data type.

- The first item specifies the operation, and can be the CHARVAR data type or the SMALLINT data type. Valid values are:

Table 214. Input Values for DUIFCUAP Operation

Operation	CHARVAR	SMALLINT
Link resources	LINK	1
Unlink resources	UNLINK	2

- The second item specifies the real or aggregate object being linked or unlinked that is lower in the aggregation hierarchy. This data item can be the CHARVAR data type or the OBJECTID data type. For a CHARVAR item, specify the class name and the object name separated with a period. For an OBJECTID item, specify the class name within single quotation marks and the object name within single quotation marks, separated by a period. For example:

  ```
  (CHARVAR)'GMFHS_Aggregate_Objects_Class.ETHERNET'
  (OBJECTID)'GMFHS_Aggregate_Objects_Class'.'ETHERNET'
  ```

  If the class name or object name used in a CHARVAR data item contains a period, enclose the name in two single quotation marks. For example, if the class name was `Class.name`, code:

  ```
  (CHARVAR)'Class.name'.Object'
  ```

  If the class name or object name used in a CHARVAR or OBJECTID data item contains a single quotation mark (') character, use two single quotation marks to specify the single quotation mark. For example, if the name of an object was `Greg'sObject`, code:

  ```
  (CHARVAR)'Class.Greg'sObject'
  ```

- The third item specifies the aggregate object being linked or unlinked that is higher in the aggregation hierarchy. The format for the third item is the same as the format for the second item.

Output: The link or unlink is performed.

If this method encounters errors, it sets a return and reason code and writes a type 1 record to the RODM log. Table 204 on page 479 lists the reason codes that can be returned by this method.

DUIFCUUS: Update User Status Method

This is a named method installed on the UpdateUserStatus field of all objects under the GMFHS_Displayable_Objects_Parent class during the initial RODM structure load for GMFHS. The GMFHS_Monitorable_Objects_Class inherits this method.

Function: Use this method for any application that must change the UserStatus field value of any descendent class of GMFHS_Displayable_Objects_Parent_Class, including the GMFHS_Managed_Real_Objects_Class, the GMFHS_Aggregate_Objects_Class, and GMFHS_Shadow_Objects_Class.

Input: The following input is required for DUIFCUUS_Update_User_Status method:

- A 4-byte mask specifying which bits of UserStatus to change
- A 4-byte UserStatus containing the new values
- An 8-byte character field containing the operator ID, method name, or product that is changing the UserStatus field
- A 20 byte block of reserved fields

See the *IBM Tivoli NetView for z/OS Data Model Reference* for a description of the UserStatus field, including bit values.

The following examples illustrate how to set the UserStatus bits. The bits have been split into lines to help show the different values.

Required bits:

- First 16 bytes contain the mask, UserStatus and operator ID.
- Next 20 bytes are reserved.

The following example RODM load function primitive statement indicates that OPER1 set the mark bit for the WESTCTR object.

```
OP 'GMFHS_Aggregate_Objects_Class'.'WESTCTR'.'UpdateUserStatus'
      INVOKED_WITH (SELFDEFINING)
      ((ANONYMOUSVAR)X'8000000080000000D6D7C5D9F1404040'
                      '0000000000000000000000000000000000000000');
```

The following example RODM load function primitive statement indicates that OPER1 cleared the mark bit for the WESTCTR object.

```
OP 'GMFHS_Aggregate_Objects_Class'.'WESTCTR'.'UpdateUserStatus'
      INVOKED_WITH (SELFDEFINING)
      ((ANONYMOUSVAR)X'8000000000000000D6D7C5D9F1404040'
                      '0000000000000000000000000000000000000000');
```

Notes:

1. The minimum number of bytes that can be sent as input to DUIFCUUS is 36. Set the mask, UserStatus, and operator ID as desired and set the remaining 20 bytes to zero.
2. When specifying an operator ID:
   - The operator ID must be 8 bytes
   - The operator ID can be all blanks

The DUIFCUUS method restricts the bits that can be changed based on the class of the object being changed.

- The marked bit (0x80000000) can be changed for any object.
- The suspended (0x20000000) and automatically clear suspended (0x60000000) bits can be changed only for objects of classes that are children of the GMFHS_Real_Objects_Class.

  Note: A shortcut to suspending real objects is possible by setting the suspended bit of an aggregate. The aggregate itself is not suspended; instead the Child Suspended bit (0x00800000) is set for the aggregate and all real objects who are children of the aggregate inherit the suspended bit. The automatic resume bit can be set in addition to the suspended bit, and it will also be inherited by the real object children.

- The child suspended bit (0x00800000) can be cleared for an aggregate. The suspended and automatic resume bits of all real object children of the aggregate will also be cleared.
- The aggregate threshold inconsistency bit (0x08000000) can be changed only for objects of class GMFHS_Aggregate_Objects_Class.
- The automation in progress bit (0x04000000) can be changed for any object.
- The not monitored bit can be changed only for objects of children that are children of the GMFHS_Real_Objects_Class.

Output: If this method is triggered using the EKG_TriggerNamedMethod function, supply a response block for the output. The response block must be at least 22 bytes. The Concat_of_strings field in the response block is a SelfDefining string with the following format:

Table 215. Output from DUIFCUUS Method

Offset	Length	Value	Description
000	2	12	Total length of SelfDefining string
002	2	30	Data type AnonymousVar
004	2	8	Length of AnonymousVar data
006	8		Value of timestamp subfield of UserStatus field after update

If this method encounters errors, it sets a return and reason code and writes a type 1 record to the RODM log. Table 204 on page 479 lists the reason codes that can be returned by this method.

DUIFECDS: Change Display Status Method

This method is a named method that is installed on the ChangeDisplayStatus field of all objects that are defined on the GMFHS_Managed_Real_Objects_Class.

Function: This method changes the DisplayStatus field of an object of the GMFHS_Managed_Real_Objects_Class and reports to the caller the effect of the change. The DisplayStatus field is changed only if one of the following conditions is satisfied:

- The unconditional change input parameter is non-zero
- The time input parameter is greater-than or equal-to the SourceStatusUpdateTime field of the object to be changed

The following example RODM load function primitive statement sets the DisplayStatus of object TRMD401 to 129 (satisfactory) only if the value of the SourceStatusUpdateTime field is less-than or equal-to 930402143000Z0000.

```
OP 'GMFHS_Managed_Real_Objects_Class'.'TRMD401'.'ChangeDisplayStatus'
  INVOKED_WITH (SELFDEFINING)
  ((ANONYMOUSVAR)X'000000810011F9F3F0F4F0F2F1F4F3F0F0F0E9F0F0F0F00000');
```

Input: The input is standard for a named method. The following short_lived_parm input is required for DUIFECDS_Change_Display_Status method:

- Display_status (Integer) New DisplayStatus

- Source_status_time (CharVar(17)) New SourceStatusUpdateTime in UTC (Coordinated Universal Time) format. The time stamp provided to DUIFECDS must be normalized to UTC, that is, the sign and offset portions of the time stamp must be `Z0000`.
- Unconditional_change (Smallint). If 0, this method changes the DisplayStatus of the target object only if the SourceStatusUpdateTime field of the target object is less than the Source_status_time input parameter. If not 0, this method changes the DisplayStatus of the target object without checking the Source_status_time input parameter.

Output: If this method is triggered using the EKG_TriggerNamedMethod function, supply a response block for the output. The response block must be at least 22 bytes. The Concat_of_strings field in the response block is a SelfDefining string with the following format:

Table 216. Output from DUIFECDS Method

Offset	Length	Value	Description
000	2	12	Total length of SelfDefining string
002	2	30	Data type AnonymousVar
004	2	16	Length of AnonymousVar data
006	4		Integer new value of DisplayStatus field
010	4		Integer previous value of DisplayStatus field
014	8		Value of timestamp subfield of DisplayStatus field after update

If this method does not change the DisplayStatus field of the target object because the unconditional change parameter is 0 and the time parameter is less than the SourceStatusUpdateTime field, the method sets the output parameters as follows:
- New DisplayStatus is set to the current value of DisplayStatus.
- Previous DisplayStatus is set to the current value of DisplayStatus.
- Timestamp is set to 0.

If this method encounters errors, it sets a return and reason code and writes a type 1 record to the RODM log. Table 204 on page 479 lists the reason codes that can be returned by this method.

DUIFFAWS: Aggregation Warm Start Method

This is an object-independent method that is run to initialize the fields related to status aggregation in the real and aggregate objects in the RODM data cache. GMFHS runs this method:
- During initialization of the configuration definition at startup
- When GMFHS recovers a lost connection to RODM
- When a CONFIG NETWORK command is processed

To disable the DUIFFAWS method, code the AGGRST=NO parameter in the GMFHS startup procedure or code LCON-AGGRST-REQUIRED=NO in the GMFHS DUIGINIT file.

Function: This method reinitializes:

- The DefaultAggregationPriorityValue field of each real object that is linked to a Display_Resource_Type_Class object
- The following fields of each aggregate object that is linked to a Display_Resource_Type_Class object:

- NOXCPTCount
- PriorityXCPTCount
- SuspendedCount
- StatusGroupCounts
- TotalRealResourceCount
- UnknownCount
- XCPTCount

After reinitializing these fields, this method recalculates the status for each aggregate object.

You can trigger the DUIFFAWS method using the RODM load function if a failure or application error causes one or more of the aggregate object fields in the previous list to be incorrect.

The following RODM load function statement triggers the DUIFFAWS method:

```
OP DUIFFAWS INVOKED_WITH;
```

Input: There are no input parameters for this method.

Output: If this method encounters errors, it sets a return and reason code and writes a type 1 record to the RODM log. Table 204 on page 479 lists the reason codes that can be returned by this method.

DUIFFIRS: Set Initial Resource Status Method

This method is triggered by GMFHS after the initialization of the configuration definition. It is triggered for each Non_SNA_Domain_Class object for which resource status solicitation will not be done and which is linked to an NMG_Class object which has an AgentStatusEffect field that indicates that the ability to receive alerts for the resources in the domain is not dependent on the AgentStatus of the NMG.

This method is also triggered when a gateway communication session is established for a non-SNA domain for which resource status solicitation will be done if the value of the InitialResourceStatus field of the domain is not 132 (unknown).

Function: This method is triggered by GMFHS during initialization of the configuration. It is triggered for each non-SNA domain for which resource status solicitation will not be done if the non-SNA domain is associated with an NMG that specifies AgentStatusEffect as 0.

This method is also triggered when status solicitation starts for resources within a non-SNA domain if the value of InitialResourceStatus field of the non-SNA domain is not equal to 132 (unknown).

Input: The inputs required for DUIFFIRS_Set_Initial_Resource_Status method are:

- RODM ObjectID of a Non_SNA_Domain_Class object.
- Time in UTC time stamp format to be associated with the change.
- Unconditional change indicator. If the 2-byte field is not equal to 0, this method sets all resources in the non-SNA domain to the value of the InitialResourceStatus field for the domain. If the unconditional change indicator is equal to 0, this method sets resources in the non-SNA domain to the value of the InitialResourceStatus field only if the resource specifies DisplayStatus equal to 132 (unknown).

The following hex string is an example of the input parameter to the DUIFFIRS method. This example specifies a target object in the SNA_Domain_Class which has a RODM object identifier value of X'00010010F9DC34AA'. The time is specified as 1430Z on 2 May, 1993. The unconditional change indicator is set to 1, so all resources in the domain will be updated. The input parameter is:

```
X'00010010F9DC34AAF9F3F0F5F0F2F1F4F3F0F0F0E9F0F0F0F00001'
```

Output: If this method encounters errors, it sets a return and reason code and writes a type 1 record to the RODM log. Table 204 on page 479 lists the reason codes that can be returned by this method.

DUIFFRAS: Recalculate Aggregate Status Method

This object-independent method can be triggered to recalculate the DisplayStatus of all aggregate objects.

Function: This method recalculates the status of every aggregate object based on each aggregate's status counter.

Input: This method requires no input parameters. This method can be triggered with the following RODM load function primitive statement:

```
OP DUIFFRAS INVOKED_WITH;
```

Output: If this method encounters errors, it sets a return and reason code and writes a type 1 record to the RODM log. Table 204 on page 479 lists the reason codes that can be returned by this method.

DUIFFSUS: Set Unknown Status Method

This object-independent method is triggered to set the DisplayStatus field value of all the real objects linked to the Resources field of a specified Non_SNA_Domain_Class object to 132 (unknown). GMFHS triggers this method:

- After the configuration definition is initialized for each non-SNA for which the DUIFFIRS method is not triggered
- When the AgentStatus field of an NMG_Class object that is linked to the ReportsToAgent field of the Non_SNA_Domain_Class object changes from 1 (satisfactory) or 3 (intermediate) to 0 (unknown) or 2 (unsatisfactory) and the AgentStatusEffect field value indicates that the ability to receive alerts for the resources in the domain is affected by the AgentStatus of the NMG
- When GMFHS receives an alert that indicates the transaction program or element manager associated with the domain is down

Function: This method sets value of the DisplayStatus field of all real resource objects linked to the Resources field of the specified Non_SNA_Domain_Class object to 132 (unknown). It sets the value of the SourceStatusUpdateTime field of each of these objects to the specified value.

Input: The inputs required for DUIFFSUS_Set_Unknown_Status method are:

- DomainObjectID representing Domain's RODM object identifier
- StatusUpdateTime representing New value for SourceStatusUpdateTime field in UTC format

The following hex string is an example of the input parameter to the DUIFFSUS method. This example specifies a target object in the SNA_Domain_Class which has a RODM object identifier value of X'00010010F9DC34AA'. The time is specified as 1430Z on 2 May, 1993. The input parameter is:

```
X'00010010F9DC34AAF9F3F0F5F0F2F1F4F3F0F0F0E9F0F0F0F0'
```

Output: If this method encounters errors, it sets a return and reason code and writes a type 1 record to the RODM log. Table 204 on page 479 lists the reason codes that can be returned by this method.

DUIFRFDS: Refresh DisplayStatus Change Method DUIFCRDC

This object-independent method can be run by any application to change the DisplayStatus field to the current DisplayStatus value for every real and aggregate resource defined in RODM.

Function: This method is useful when the DisplayStatus mapping table, DUIFSMT, has been changed. Instead of waiting for a status change from the network to trigger an exception view update, method DUIFRFDS can be run to cause the status change, which recalculates the exception state of the objects. The appropriate exception views are then updated.

Input: This method requires no input parameters and can be triggered with the following RODM load function primitive statement:

```
OP DUIFRFDS INVOKED_WITH;
```

See sample CNMSJH13 for an example of triggering the method.

Output: If this method encounters errors, it sets a return and reason code and writes a type 1 record to the RODM log. Table 204 on page 479 lists the reason codes that can be returned by this method.

DUIFVCFT: Change Exception State

This object-independent method can be run by a user method to change the exception state of an object.

Function: The user method that runs method DUIFVCFT is specified by the USRXMETH keyword in DisplayStatus mapping table DUIFSMT. Sample user methods DUIFCUXM and DUIFCUX2 run method DUIFVCFT to set either value XCPT or NOXCPT in the ResourceTraits field the same way a real DisplayStatus change is processed. DUIFVCFT will then trigger a method to determine if the change in exception state will cause the object to be added to or deleted from any open exception views.

Input: Table 217 lists the input parameters for method DUIFVCFT:

Table 217. Input Values for DUIFVCFT

Parameter	Data Type	Length of Field
Total_Length	SMALLINT	2
Data_Type	SMALLINT	2
Data_Length	SMALLINT	2
Resource_Object_ID	OBJECTID	8
Requested_exception_status	INTEGER	4

Output: The ResourceTraits field of the resource is updated to reflect the requested exception state.

If this method encounters errors, it sets a return and reason code and writes a type 1 record to the RODM log. Table 204 on page 479 lists the reason codes that can be returned by this method.

Notes:

1. Resource_Object_ID is the object id of the resource whose changed DisplayStatus triggered the user method.
2. Set Requested_exception_status to 0 if you do *not* want the resource to have an exception state. DUIFVCFT will set value NOXCPT in the ResourceTraits field for this resource.
3. Set Requested_exception_status to 1 if you *do* want the resource to have an exception state. DUIFVCFT will set value XCPT in the ResourceTraits field for this resource.
4. See "Creating a DisplayStatus Method for Exception Views" on page 111 for more information.

DUIFVINS: Install View Granularity Method (DUIFVNOT)

This object-independent method installs method DUIFVNOT on a class or field.

Function: DUIFVINS must be run for each new class or connectivity field that is added to the data model.

DUIFVNOT is inherited by all objects of a class. For a list of all the fields on which GMFHS installs DUIFVNOT, see sample FLBTRDME.

Input: Table 218 lists the input parameters for method DUIFVINS:

Table 218. Input Values for DUIFVINS

Parameter	Data Type	Length of Field
enable_change_status	SMALLINT	2
rule	INTEGER	4
notification_method	OBJECTID	8
class	CLASSID	4
field	FIELDID	4

enable_change_status

This parameter is used to prevent view change notifications (VCNs) when a field is set to its previous value.

The values for this parameter are:

0 Used if either the prev_val subfield does not exist on the field, or if a VCN must be issued even if the field is changed to its previous value.

1 Used if the prev_val subfield exists on the field, and if a VCN must not be issued when the field is changed to its previous value.

rule The criteria used to determine if a field change results in a VCN being issued. It is implicit in each of these rules, with the exception of ANY_FIELD_OBJECT_CHANGE, that the objectID or classID and fieldID involved in the change are used to construct at least one view that is currently open.

The values for this parameter are:

1 OBJECT_CHANGE: Send a view update if the field changes at the object level.

2 VALUE_INCREASE: Send a view update if the field changes at the object level and the value of the field increases.

3	VALUE_DECREASE: Send a view update if the field changes at the object level and the value of the field decreases.
4	CONNECTIVITY: This rule applies to the ObjectLink and ObjectLinkList data types. Send a view update if the field changes at the object level and the link or unlink results in a change to the connectivity displayed in the view. For the following view types, only one of the objects needs to be currently in a view to indicate a view change: Configuration Parents Configuration Logical Configuration Physical Configuration Backbone Configuration Child Configuration Child II Configuration Child III For all other view types, both objects must be in a view to indicate a view change.
5	CLASS_CHANGE: Send a view update if the field changes at the class level.
6	OBJECT_OR_CLASS_CHANGE: Send a view update if the field changes at the object or class level.
7	ANY_FIELD_OBJECT_CHANGE: Send a view update if the field changes at the object level whether or not the field was used to construct the view. This is for customers that want to monitor fields that are not involved in view building, including exception views. The other rules do not result in a VCN for exception views. See "Defining Exception View Objects and Criteria" on page 100 for more information.
5000	LU_CHANGE: Send a view update if the field changes on an LU-type object and the monitoringLuCollection field indicates the LU collection is not in transition.

notification_method
The object ID of the notification method DUIFVNOT.

class The class ID on which DUIFVNOT must be installed.

field The field ID on which DUIFVNOT must be installed.

The following is an example of a RODM loader statement to run DUIFVINS:

```
OP DUIFVINS INVOKED_WITH (SELFDEFINING)
 (
  (SMALLINT) 0
  (INTEGER)  1
  (OBJECTID) EKG_Method.DUIFVNOT
  (CLASSID)  GMFHS_Real_Objects_Class
  (FIELDID)  GMFHS_Real_Objects_Class.DisplayResourceType
 );
```

Output: If this method encounters errors, it sets a return and reason code and writes a type 1 record to the RODM log. Table 204 on page 479 lists the reason codes that can be returned by this method.

Part 5. Appendixes

Appendix A. RODM Tools

The NetView program provides the following tools for use with RODM:

- RODMView is an interactive application program for viewing and updating the values of fields in the RODM data cache. See "RODMView."
- The RODM unload function can be used to unload classes, objects, and fields. See "RODM Unload Function" on page 537.
- FLCARODM (RODM Access Facility) provides a fast and efficient REXX interface to RODM. See "FLCARODM" on page 541.
- RODM Collection Manager enables the arbitrary grouping of objects into views or aggregates and dynamically manages the views or aggregates. See "RODM Collection Manager" on page 585.
- The Visual BLDVIEWS (VBV) application simplifies the management of RODM views and information by providing a graphical interface to the BLDVIEWS tool and the RODMView tool. See "Visual BLDVIEWS" on page 586.
- The BLDVIEWS tool is used for defining custom views that match your network layout and your preferred style of monitoring it. See "BLDVIEWS" on page 586.
- DELVIEWS can be used for deleting certain customized views. See "DELVIEWS" on page 655.

Some panels in this appendix show GMFHS information.

RODMView

Use the RODMView application program to view and update the values of fields in the RODM data cache. RODMView runs under an OST task in the NetView program. The following RODMView topics are covered:

- Navigating within RODMView
- RODMView restrictions
- Starting RODMView
- Using the RODMView functions

Navigating within RODMView

You can navigate within RODMView in the following ways:

- Using the main menu
- Using accelerator PF keys
- Using the PF keys displayed at the bottom of a panel

Panel data entry fields are identified by underscored lines, and a command line is at the bottom of each panel.

Navigating Using Menus

RODMView has a main menu panel, which is illustrated in Figure 92 on page 506. To navigate to the option that you want, enter the corresponding selection number, or select the appropriate line with the cursor and press Enter. If you enter an option that is not valid, an error message is displayed.

From any RODMView panel, you can navigate directly to the panel of another RODMView function by pressing an associated accelerator PF key. Accelerator PF keys PF13–PF22 correspond to option numbers 1–10 respectively, as shown in Table 219 on page 504.

Table 219. Accelerator PF Keys and Options

PF Key	Option	Panel
PF13	Option 1	Access and control
PF14	Option 2	Simple query
PF15	Option 3	Compound query
PF16	Option 4	Locate actions
PF17	Option 5	Link/unlink
PF18	Option 6	Change field
PF19	Option 7	Subfield actions
PF20	Option 8	Create actions
PF21	Option 9	Delete actions
PF22	Option 10	Method actions

On many PC-based terminal emulators, PF keys in the range of PF13–PF22 are accessed by holding down the shift key (or other control key) and pressing a PF key in the range 1–10, whose numbers correspond directly to the option numbers.

A list of active PF keys is displayed across the bottom of the RODMView panels. The PF keys that are displayed and the function they perform vary depending on the panel that is displayed. Table 220 lists the PF keys and corresponding functions:

Table 220. PF Key Function

PF Key	Function
PF1	Displays help information.
PF2	Ends the command and exits.
PF3	Returns control to the previous panel.
PF4	Clears query input fields.
PF5	• Repeats the last find request when viewing query output. • Redisplays the last query or locates output when viewing query and locate panels.
PF6	Rolls to the next application in the ring.
PF7	Goes back to the previous panel.
PF8	Goes forward to the next panel.
PF9	Copies the query output to the NetView log.
PF10	When the cursor is on a hexadecimal object ID of query or locate output, copies that object ID to the input line of another panel.
PF11	When the cursor is on a SystemView® class or field name of query or locate output, translates between the SystemView textual name and numeric identifier.
PF12	Recalls commands entered on the RODMView command line.

RODMView Restrictions

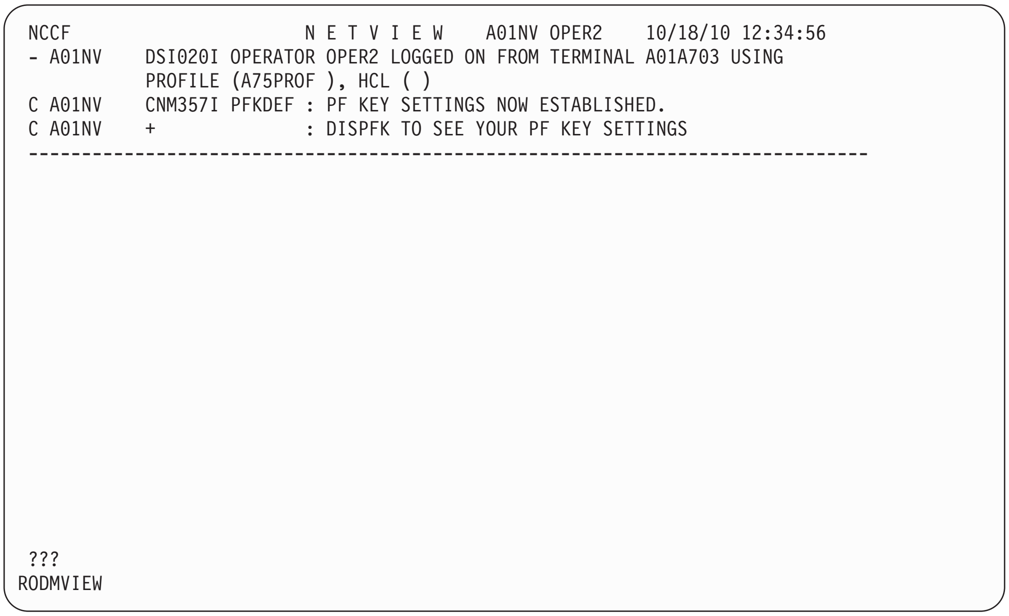
The following is a list of RODMView restrictions:

- The length of a command that RODMView can run is 240 characters. You can shorten the command that RODMView runs by using class, object, or field IDs instead of lengthy names.
- The object name input fields are limited to a maximum of 64 bytes on all RODMView panels even though object names can be a total of 254 bytes in RODM. You can get around the character limit by using the object ID instead of the name or by using pattern-matching characters (wild cards) in the name.
- Only the query function supports wild cards.
- Only one copy of RODMView can be run on a single NetView session at a time. If you attempt to run a second copy of RODMView, the program exits and the previous copy of RODMView regains control.
- You can restrict certain keywords of the EKGVACTM command processor.

  Reference: For a list of keywords that can be protected, See the *IBM Tivoli NetView for z/OS Administration Reference*. You cannot restrict keywords for any of the other RODMView command processors.

Starting RODMView

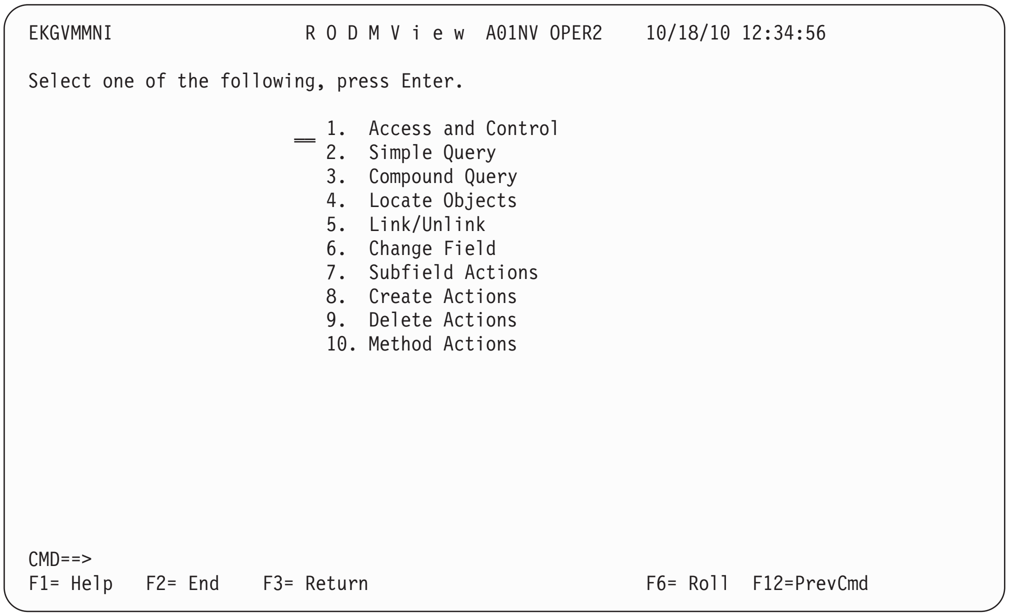
To start RODMView, enter **RODMVIEW** on the NetView NCCF command line as shown in Figure 91.



```
 NCCF                    N E T V I E W    A01NV OPER2    10/18/10 12:34:56
 - A01NV    DSI020I OPERATOR OPER2 LOGGED ON FROM TERMINAL A01A703 USING
            PROFILE (A75PROF ), HCL ( )
 C A01NV    CNM357I PFKDEF : PF KEY SETTINGS NOW ESTABLISHED.
 C A01NV    +              : DISPFK TO SEE YOUR PF KEY SETTINGS
 -------------------------------------------------------------------------------


















 ???
RODMVIEW
```

Figure 91. RODMView NetView Command Line Call

The RODMView main menu is displayed as shown in Figure 92 on page 506.



```
EKGVMMNI                  R O D M V i e w  A01NV OPER2    10/18/10 12:34:56

Select one of the following, press Enter.

                            1.  Access and Control
                         __ 2.  Simple Query
                            3.  Compound Query
                            4.  Locate Objects
                            5.  Link/Unlink
                            6.  Change Field
                            7.  Subfield Actions
                            8.  Create Actions
                            9.  Delete Actions
                            10. Method Actions











CMD==>
F1= Help   F2= End    F3= Return                         F6= Roll  F12=PrevCmd
```

Figure 92. RODMView Main Menu — EKGVMMNI

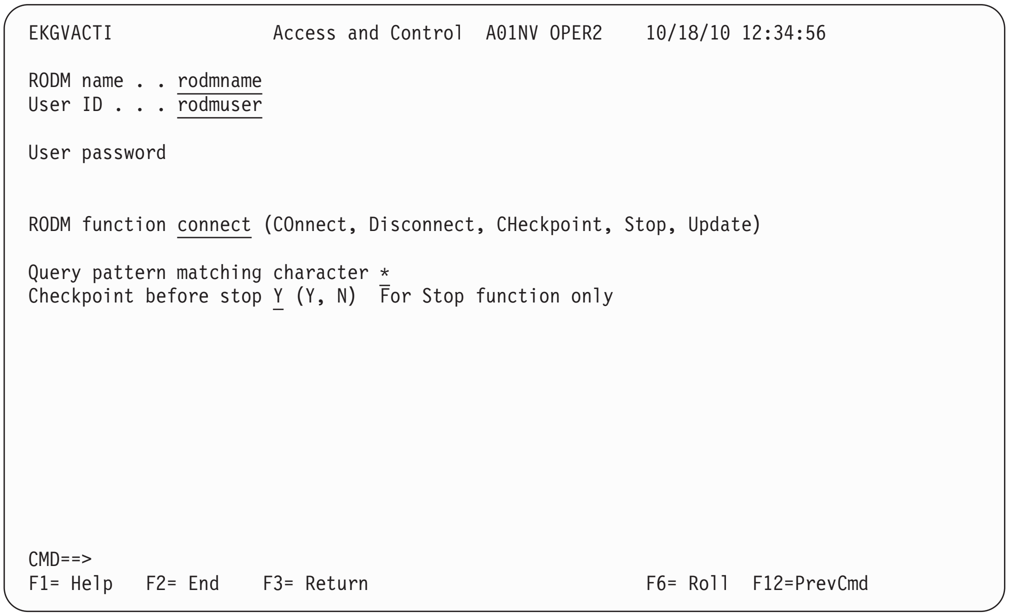
From the RODMView menu you can choose any of the available functions. The three ways to choose an option are as follows:

- Enter the corresponding number at the prompt next to the selections.
- Move the cursor to the line of the selection and press Enter.
- Use the accelerator PF keys.

You must be signed on to RODM before using any of the other functions.

Access and Control Function

Select **1. Access and control,** from the main menu to display the Access and Control panel as shown in Figure 93.



```
EKGVACTI               Access and Control  A01NV OPER2    10/18/10 12:34:56

RODM name . . rodmname
User ID . . . rodmuser

User password


RODM function connect (COnnect, Disconnect, CHeckpoint, Stop, Update)

Query pattern matching character *
Checkpoint before stop Y (Y, N)  For Stop function only











CMD==>
F1= Help   F2= End    F3= Return                         F6= Roll  F12=PrevCmd
```

Figure 93. RODMView Access and Control Panel — EKGVACTI

Enter the RODM name and one of the following functions:

- COnnect
- Disconnect
- CHeckpoint
- Stop
- Update

Notes:

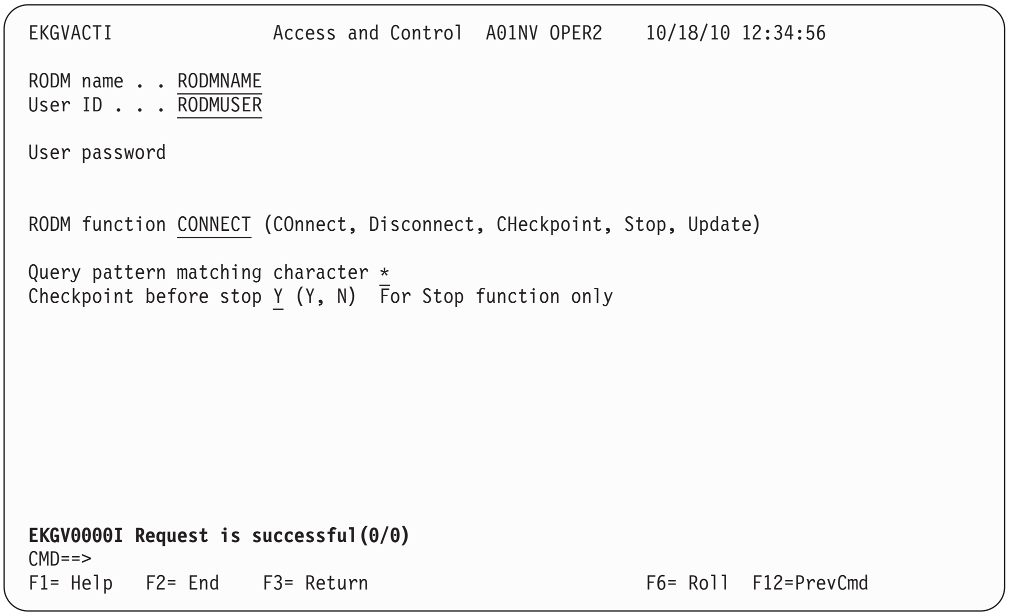
1. The capitalized letters of the functions indicate the minimum letters that you can enter to specify a function. For example, type **CO** to specify the connect function.
2. RODM must be started before you can connect to RODM with RODMView.

If you do not specify the user ID, the NetView operator ID is used as the default. If you do not specify the user password, blanks are used as the user password.

The query pattern-matching character is the character that is used as a wild card when issuing queries. Note that the asterisk (*) is valid as part of an object name, and might not be suitable for use as a wild card. The connect function assigns the value to the wild card. To change it without disconnecting and reconnecting, use the update function. If the character is changed on this panel, it is only be effective if the connect or update request is successful.

If a system authorization facility is enabled on your system, RODM uses it. Your user ID must be authorized to perform the functions you select. The user ID might not be the same one as your NetView operator ID. Check with your security administrator if you are unsure. To avoid access conflicts with other RODM users and applications, it is best for each RODM user to have a unique RODM user ID across your z/OS system.

After you enter the information in the required fields and press Enter, a message is displayed near the bottom of the panel informing you of the outcome of your request.



```
EKGVACTI              Access and Control  A01NV OPER2    10/18/10 12:34:56

RODM name . . RODMNAME
User ID . . . RODMUSER

User password


RODM function CONNECT (COnnect, Disconnect, CHeckpoint, Stop, Update)

Query pattern matching character *
Checkpoint before stop Y (Y, N)  For Stop function only









EKGV0000I Request is successful(0/0)
CMD==>
F1= Help   F2= End    F3= Return                        F6= Roll  F12=PrevCmd
```

Figure 94. RODMView Message for a Successful Connection

The message line in the lower-left corner of Figure 94 on page 507 indicates that the request was successful with return and reason codes of 0 (zero) from RODM. Return and reason codes appear in parentheses next to the message. In this example, both the return and reason codes are 0.

When RODMView receives these return and reason code combinations from RODM, it tries to convert the combination and to display an associated RODMView message. Because the RODM return and reason code combinations are numerous, RODMView only translates the most common combinations. In the case that RODM returns a return/reason pair that RODMView does not translate, the RODM reason code and return code are displayed in the following message:

```
EKGV8037E RODM return code/reason code is (return_code/reason_code)
```

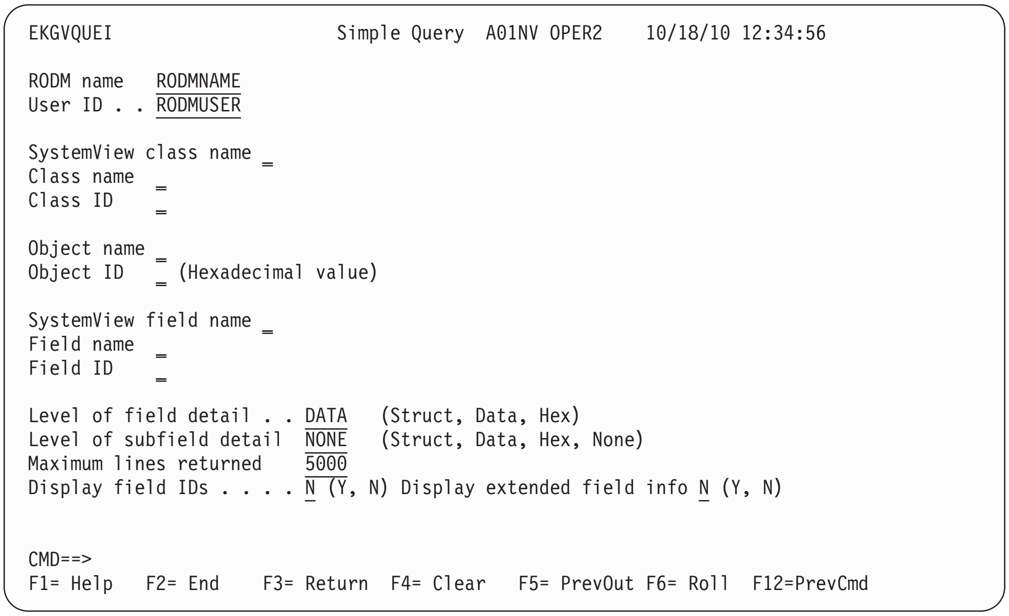
All RODM-specific return and reason codes are the range of 0–49151. See “RODM Return and Reason Codes” on page 452 for more information.

If any of the RODMView command processors encounters a problem that is not due specifically to RODM, the reason code is greater than 67000. These reason codes are converted by RODMView and the corresponding message is displayed.

When you have successfully signed on to RODM, press PF3 to return to the RODMView main menu.

Simple Query Function

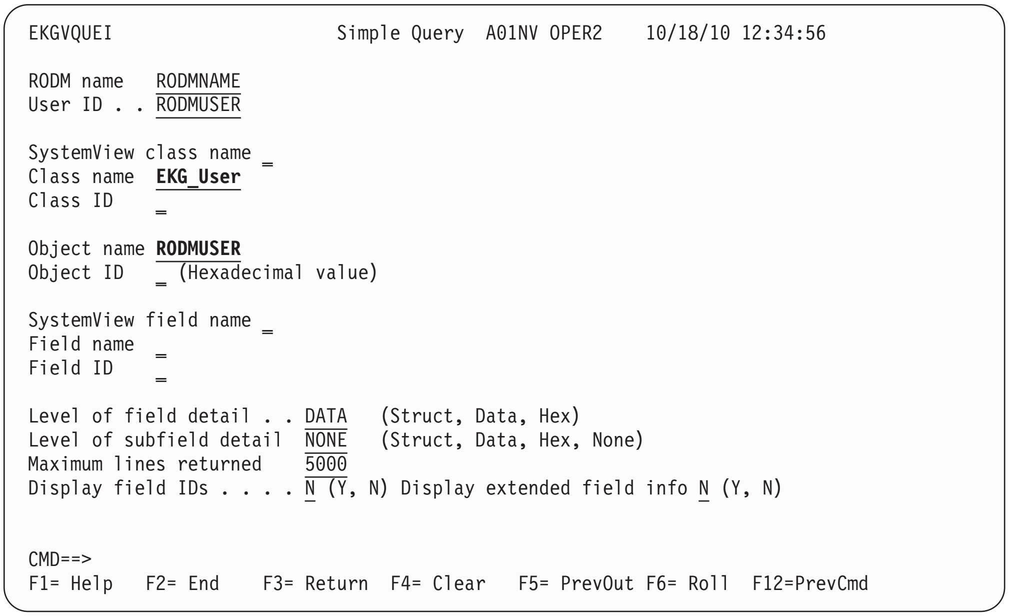
From the RODMView main menu, select **2. Simple Query** to perform different kinds of queries at various levels of detail. The Simple Query panel is displayed as shown in Figure 95.



```
EKGVQUEI                    Simple Query  A01NV OPER2    10/18/10 12:34:56

RODM name   RODMNAME
User ID . . RODMUSER

SystemView class name _
Class name  _
Class ID    _

Object name _
Object ID   _ (Hexadecimal value)

SystemView field name _
Field name  _
Field ID    _

Level of field detail . . DATA   (Struct, Data, Hex)
Level of subfield detail  NONE   (Struct, Data, Hex, None)
Maximum lines returned    5000
Display field IDs . . . . N (Y, N) Display extended field info N (Y, N)


CMD==>
F1= Help   F2= End    F3= Return  F4= Clear   F5= PrevOut F6= Roll  F12=PrevCmd
```

Figure 95. RODMView Query Panel — EKGVQUEI

Type the criteria for which you want RODMView to base the query request and press Enter. For example, if you want to display the object representing your user ID in the EKG_User class, enter the information as shown in Figure 96 on page 509. Note that objects created on the EKG_User class represent users that are currently signed on to RODM.

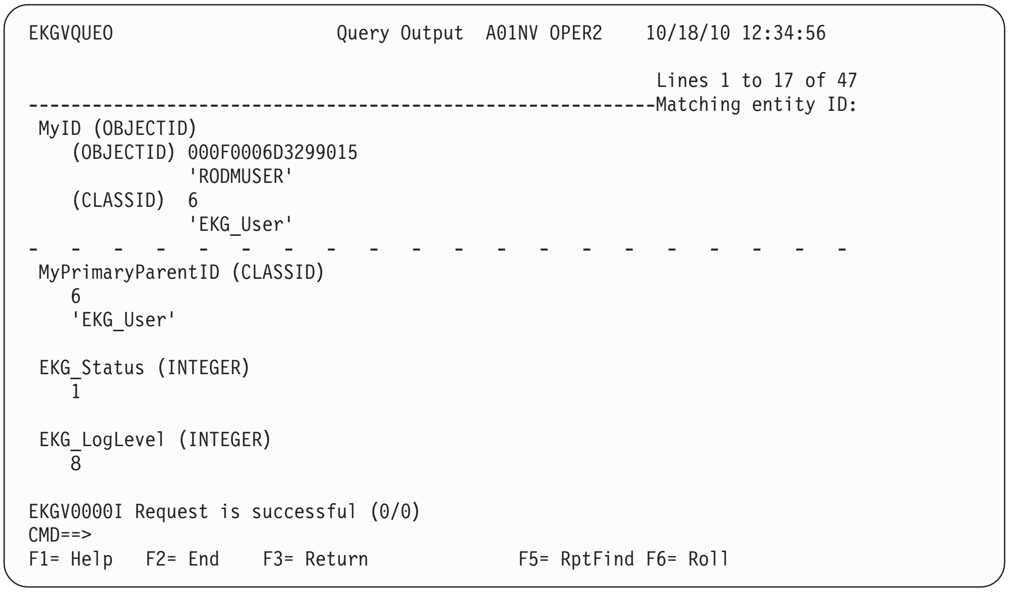


```
EKGVQUEI                  Simple Query  A01NV OPER2    10/18/10 12:34:56

RODM name   RODMNAME
User ID . . RODMUSER

SystemView class name _
Class name  EKG_User
Class ID    _

Object name RODMUSER
Object ID   _ (Hexadecimal value)

SystemView field name _
Field name  _
Field ID    _

Level of field detail . . DATA   (Struct, Data, Hex)
Level of subfield detail  NONE   (Struct, Data, Hex, None)
Maximum lines returned    5000
Display field IDs . . . . N (Y, N) Display extended field info N (Y, N)


CMD==>
F1= Help   F2= End    F3= Return  F4= Clear   F5= PrevOut F6= Roll  F12=PrevCmd
```

Figure 96. RODMView Querying Your User ID

Note that, except for SystemView class and field names, RODM is case-sensitive for class, object, and field names.

If the specified object exists, the output are displayed as shown in Figure 97.



```
EKGVQUEO                  Query Output  A01NV OPER2    10/18/10 12:34:56

                                                           Lines 1 to 17 of 47
-----------------------------------------------------------Matching entity ID:
 MyID (OBJECTID)
    (OBJECTID) 000F0006D3299015
               'RODMUSER'
    (CLASSID)  6
               'EKG_User'
-   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
 MyPrimaryParentID (CLASSID)
    6
    'EKG_User'

 EKG_Status (INTEGER)
    1

 EKG_LogLevel (INTEGER)
    8

EKGV0000I Request is successful (0/0)
CMD==>
F1= Help   F2= End    F3= Return              F5= RptFind F6= Roll
```

Figure 97. RODMView Query Output Panel

The Query Output panel shown in Figure 97 shows (in the upper-right corner) that 47 lines of output are available, the first 17 of which are displayed on the current panel.

The 0 return and reason codes in the message indicate that the request was successful.

For each class entity or field class that RODMView finds that matches the search criteria, the entity identifier is displayed under the header, Matching entity ID:, followed by the fields you have specified. In this example, because the query criteria is very specific, only one entity is found. Leave the Field name and Field ID fields blank to display all of the fields of this object.

You can also query RODM by numeric identifiers rather than by names. The identifier of an entity can be found by querying it by name. The identifiers are displayed in the Matching entity ID section and in the MyID field of that entity for the sake of clarity.

If numeric identifiers are used at the same time as the corresponding name, the numeric identifier takes precedence and the names are ignored. For example, if you query by specifying **EKG_System** for the Class Name and **1** for the Class ID, the class that is queried is the UniversalClass because its identifier is 1. The name EKG_System is disregarded by RODM because a numeric identifier is present.

For each field that exists on the object you query, the field name is displayed, its data type is displayed in parentheses, and its value is displayed (under the field name). In some cases, additional information is automatically obtained about the field.

For example, the RODM-defined data type ClassID is an integer. Because it is helpful to know what class name corresponds to the number, RODMView further queries RODM to match the class name with its ID. See the MyPrimaryParentID field in Figure 97 on page 509.

For those fields that have no value assigned to them, a blank line follows the line containing the field name and field data type.

From the query output panel, you can page backward or forward through the output using PF7 and PF8, or by typing the **UP** and **DOWN** commands on the command line.

The following table is a summary of output control commands available on the command line of the Query Output panel:

Table 221. Query Output Control Commands

Command	Explanation
UP *n*	Scrolls output up one page, or optionally by *n* lines.
DOWN *n*	Scrolls output down one page, or optionally by *n* lines
TOP	Scrolls output to the top.
BOTTOM	Scrolls output to the bottom.
F *find_word*	Search for *find_word* from the current panel to the end of output.
F *find_word* PREV	Search for *find_word* from the current panel to the beginning of output. The keyword PREV can be abbreviated as P.

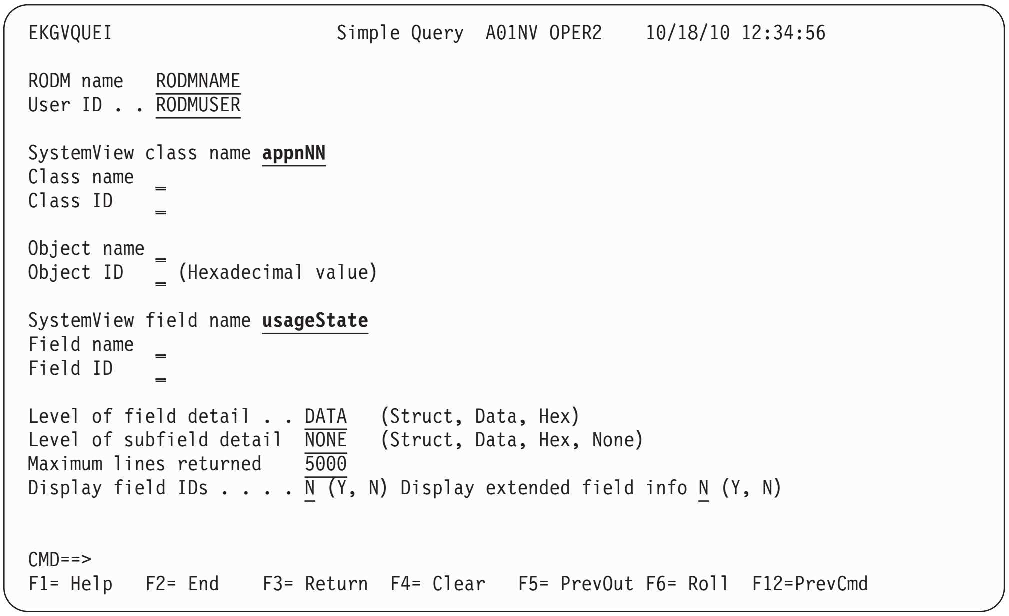
Note: When you are searching for a word using the F command, the *find_word* must be a single string of alphanumeric characters. Spaces are not permitted even if they are enclosed in single quotation marks.

You can search for a single word anywhere in the output, starting from the current panel to the end of the output, by typing the command **F**

find_word on the command line. Similarly, you can search for a word from your current position on the panel to the start of the output by typing the command **F find_word PREV** or **F find_word P** on the command line.

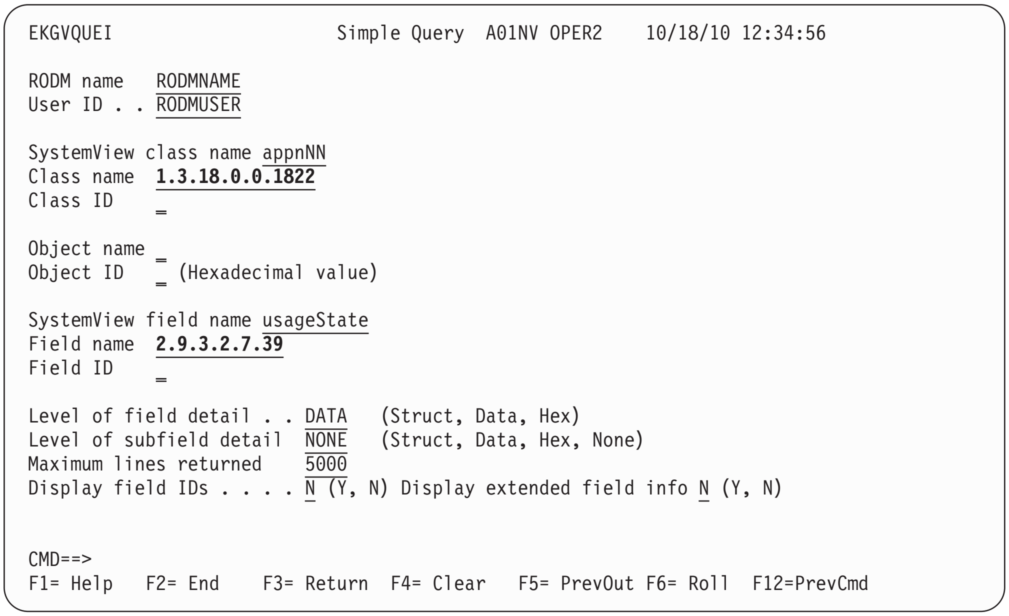
Querying RODM Using SystemView Class and Field Names

Some RODM applications, for example, NetView MultiSystem Manager, use a special naming convention for the SystemView data model. This convention consists of numbers separated by periods to represent the SystemView name. RODMView can translate the SystemView data model textual class name. For example, it can translate the SystemView class name `appnNN` and the SystemView field name `usageState` as shown in Figure 98, to the equivalent RODM class name `1.3.18.0.0.1822` and field name `2.9.3.2.7.39` as shown in Figure 99 on page 512.



```
EKGVQUEI                   Simple Query  A01NV OPER2    10/18/10 12:34:56

RODM name   RODMNAME
User ID . . RODMUSER

SystemView class name appnNN
Class name  _
Class ID    _

Object name _
Object ID   _ (Hexadecimal value)

SystemView field name usageState
Field name  _
Field ID    _

Level of field detail . . DATA   (Struct, Data, Hex)
Level of subfield detail  NONE   (Struct, Data, Hex, None)
Maximum lines returned    5000
Display field IDs . . . . N (Y, N) Display extended field info N (Y, N)


CMD==>
F1= Help   F2= End    F3= Return  F4= Clear   F5= PrevOut F6= Roll  F12=PrevCmd
```

Figure 98. RODMView Simple Query Specifying SystemView Class and Field Names



```
EKGVQUEI                  Simple Query  A01NV OPER2    10/18/10 12:34:56

RODM name   RODMNAME
User ID . . RODMUSER

SystemView class name appnNN
Class name  1.3.18.0.0.1822
Class ID    _

Object name _
Object ID   _ (Hexadecimal value)

SystemView field name usageState
Field name  2.9.3.2.7.39
Field ID    _

Level of field detail . . DATA   (Struct, Data, Hex)
Level of subfield detail  NONE   (Struct, Data, Hex, None)
Maximum lines returned    5000
Display field IDs . . . . N (Y, N) Display extended field info N (Y, N)


CMD==>
F1= Help   F2= End    F3= Return  F4= Clear   F5= PrevOut F6= Roll  F12=PrevCmd
```

Figure 99. RODMView Simple Query-Translated SystemView Textual Class and Field Names

Querying RODM Using Pattern-Matching Characters

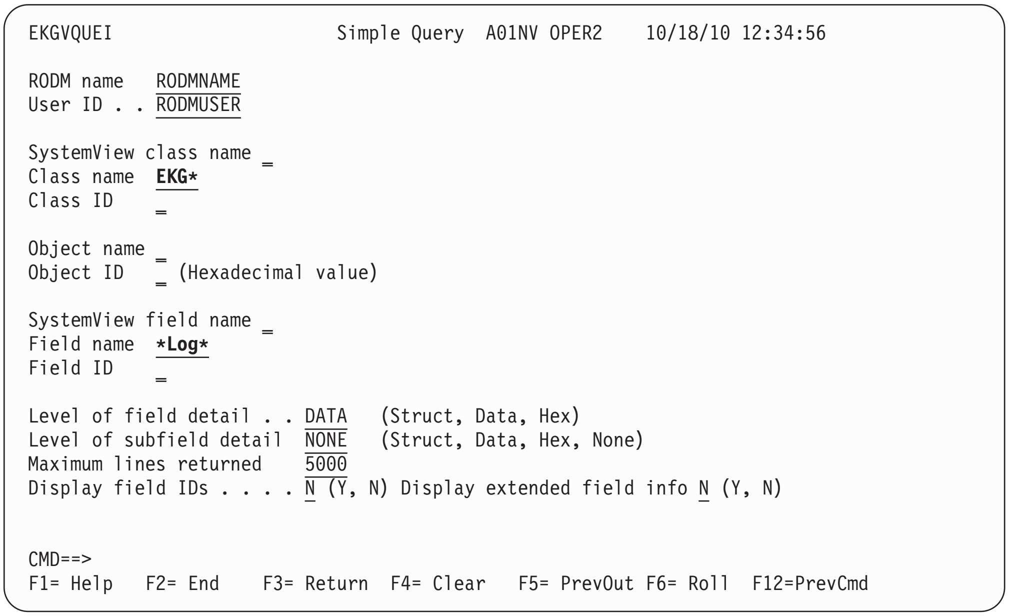
Use pattern-matching characters to specify a search using less specific criteria. For example, if you know the name of an object you want to find but do not know what class it exists under, or if you know a class name contains a certain word, pattern-matching characters (wild cards) can be used.

Pattern-matching characters in RODMView are available for Class name, Object name, and Field name input fields for the query functions only.

The default pattern-matching character in RODMView is the asterisk (*), but it can be changed by the user on the Access and Control panel. Note that an asterisk is a valid character in an object name, and unexpected results can occur when querying for objects that contain asterisks in their names. The following are examples of search strings that use pattern-matching characters:

Test*	Matches on a name starting with Test
***Test**	Matches on a name ending with Test
Test	Matches on a name that contains Test anywhere within it
*	Matches every name

For example, to query all the fields related to logging and defined on classes starting with the letters EKG, specify the query as shown in Figure 100 on page 513.

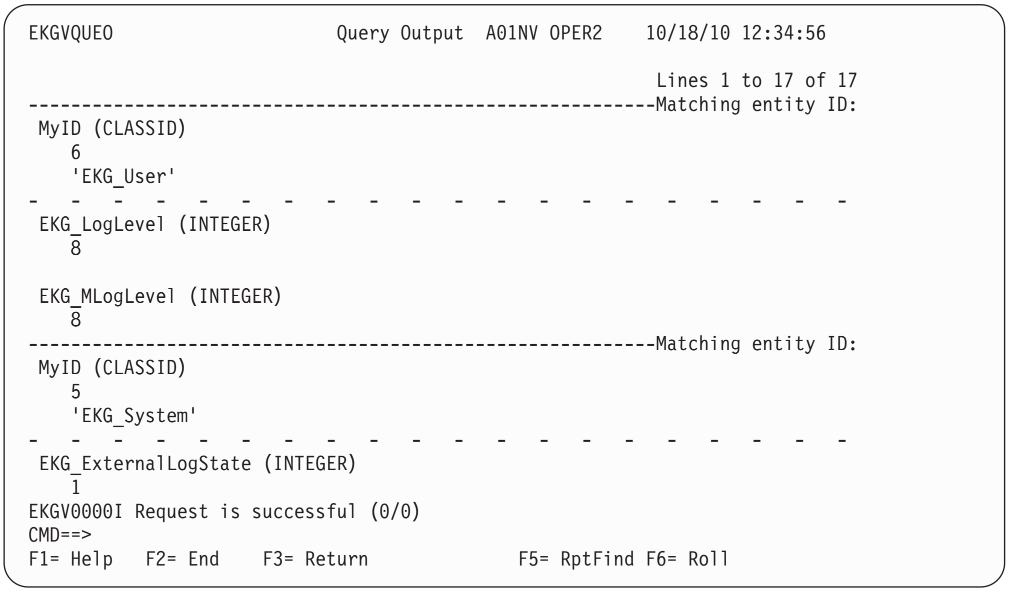


```
EKGVQUEI                     Simple Query  A01NV OPER2    10/18/10 12:34:56

RODM name   RODMNAME
User ID . . RODMUSER

SystemView class name _
Class name  EKG*
Class ID    _

Object name _
Object ID   _ (Hexadecimal value)

SystemView field name _
Field name  *Log*
Field ID    _

Level of field detail . . DATA   (Struct, Data, Hex)
Level of subfield detail  NONE   (Struct, Data, Hex, None)
Maximum lines returned    5000
Display field IDs . . . . N (Y, N) Display extended field info N (Y, N)


CMD==>
F1= Help   F2= End    F3= Return  F4= Clear   F5= PrevOut F6= Roll  F12=PrevCmd
```

Figure 100. RODMView Query for Fields That Contain the Word Log

RODMView searches for all fields that contain Log in their names. Every class defined in RODM is searched.

Figure 101 illustrates the output panel for a typical RODM.



```
EKGVQUEO                     Query Output  A01NV OPER2    10/18/10 12:34:56

                                                           Lines 1 to 17 of 17
-----------------------------------------------------------Matching entity ID:
 MyID (CLASSID)
    6
    'EKG_User'
-   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
 EKG_LogLevel (INTEGER)
    8

 EKG_MLogLevel (INTEGER)
    8
-----------------------------------------------------------Matching entity ID:
 MyID (CLASSID)
    5
    'EKG_System'
-   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
 EKG_ExternalLogState (INTEGER)
    1
EKGV0000I Request is successful (0/0)
CMD==>
F1= Help   F2= End    F3= Return              F5= RptFind F6= Roll
```

Figure 101. RODMView Query Output for Fields Containing 'Log'

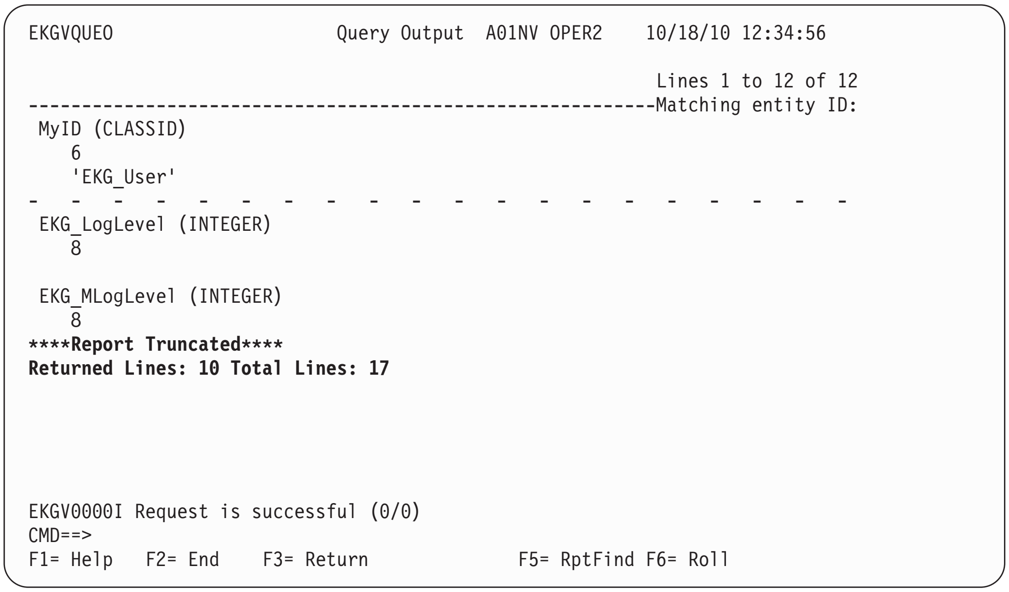
As shown in Figure 101, RODMView found two classes that have Log in their field names: the EKG_User class and the EKG_System class. The EKG_User class has two fields matching the criteria: EKG_LogLevel and EKG_MLogLevel. The EKG_System class has the field EKG_ExternalLogState.

The output from the above example shows information at the class level. To see the same information at the object level, enter a pattern-matching character in the Object Name input field and on the Class name input field and press Enter.

Some queries display a large number of lines, particularly when using pattern-matching characters. The query request does not display more lines than specified in the Maximum lines returned field. If you specify 0, RODMView defaults to 5000. If the response to a query results in more lines being returned than specified by the Maximum lines returned field, you are notified in the last two lines that this has occurred.

Note:

Figure 102 illustrates the results of the previous query request where Maximum lines returned is set to 10 and the lines returned by the query are 17. Notice that the query request completed successfully and the excess lines are not displayed. The last two lines displayed indicate that the query report is truncated. In this example, increase the Maximum lines returned to a value greater than or equal to 17 to prevent the query report being truncated.



```
EKGVQUEO                     Query Output  A01NV OPER2    10/18/10 12:34:56

                                                           Lines 1 to 12 of 12
-----------------------------------------------------------Matching entity ID:
 MyID (CLASSID)
    6
    'EKG_User'
-   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
 EKG_LogLevel (INTEGER)
    8

 EKG_MLogLevel (INTEGER)
    8
****Report Truncated****
Returned Lines: 10 Total Lines: 17





EKGV0000I Request is successful (0/0)
CMD==>
F1= Help   F2= End    F3= Return             F5= RptFind F6= Roll
```

Figure 102. RODMView Excessively Large Query Output

Compound Query Function

From the RODMView main menu, select **3. Compound Query** to perform different kinds of queries at various levels of detail using multiple criteria. The Compound Query panel is displayed as shown in Figure 103 on page 515.

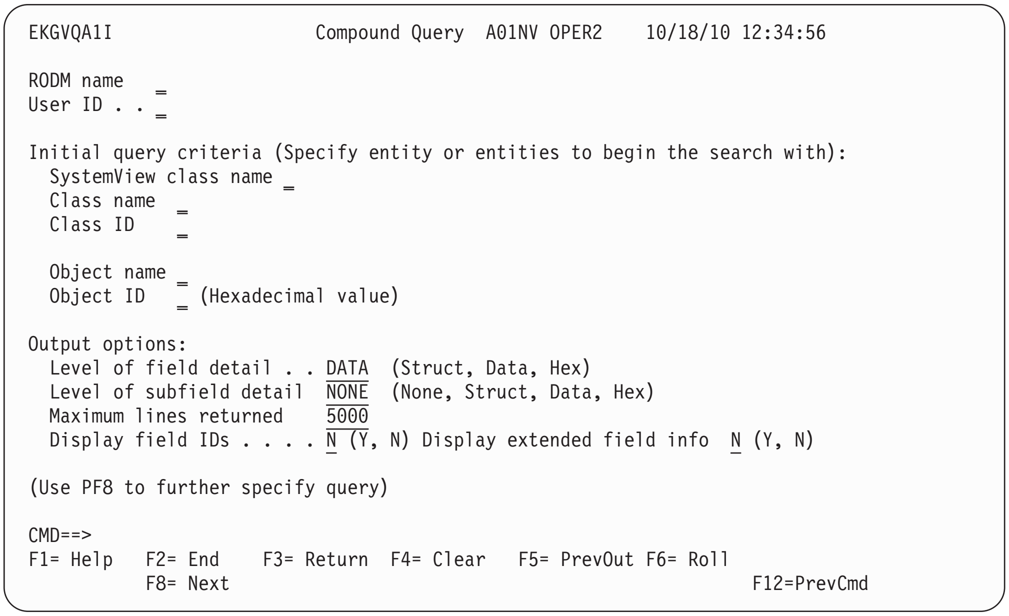
The criteria the simple query function uses to display classes and objects are the class and object names themselves. You can use the compound query function not only to search for classes and objects in the same manner, but also to select only those classes or objects that meet other criteria. For example, it is possible to search for all objects in RODM that have a particular value in a field. It is also possible to search for all objects that are linked to other objects through a field and that have a particular value in a field.

From the RODMView main menu, select **3. Compound Query**. Four panels are used to specify the query:

- Use the Compound Query panel EKGVQA1I (shown in Figure 103 on page 515), to specify where to begin the search by specifying the class and object names.

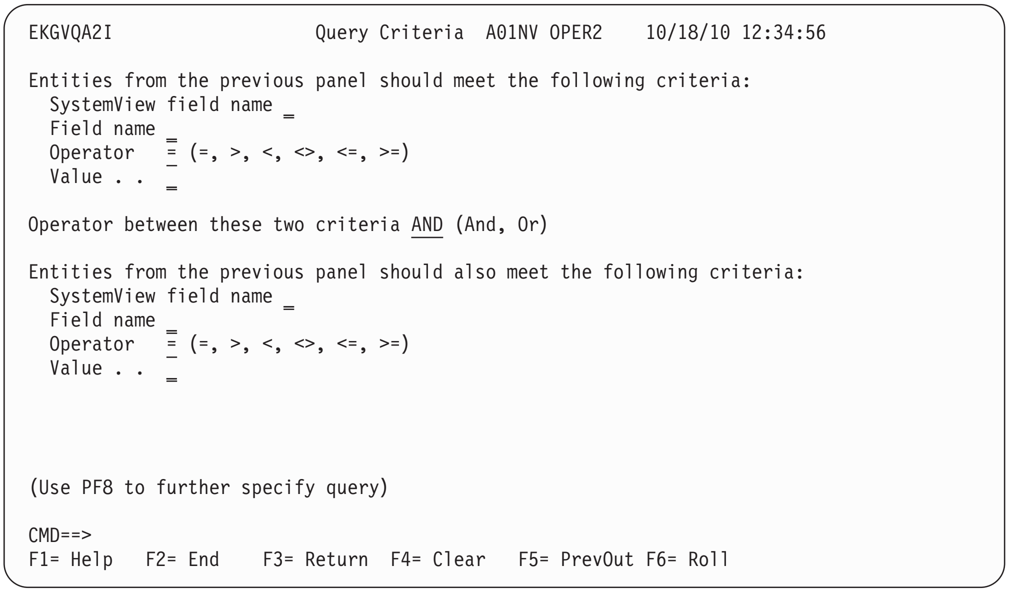
- Use panel EKGVQA2I (shown in Figure 104), to specify criteria that the classes or objects must meet to be displayed.
- Use panel EKGVQA3I (shown in Figure 105 on page 516), to specify a field that is followed to query any linked entities. You can also specify criteria that the entities found on the traversed field must meet to be displayed.
- Use panel EKGVQA4I (shown in Figure 106 on page 516), to specify which fields (or all fields, if left blank) are displayed of the entities that met all the search criteria you entered.

Use PF7 and PF8 to navigate among the four Compound Query panels. To clear all the input fields on all of the panels, press PF4; note that RODMView asks for verification.



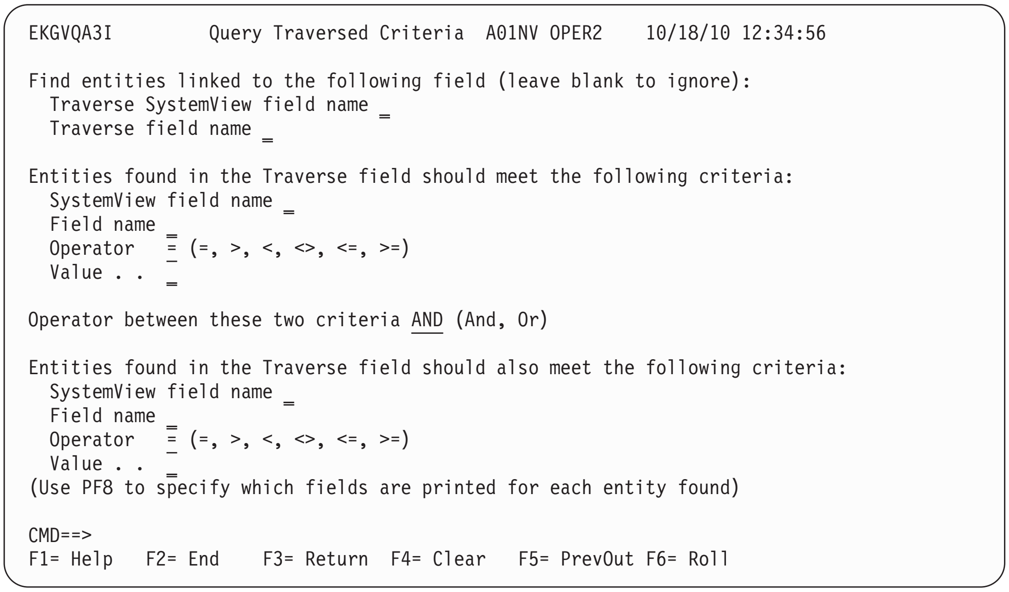
```
EKGVQA1I                 Compound Query  A01NV OPER2    10/18/10 12:34:56

RODM name    _
User ID . .  _

Initial query criteria (Specify entity or entities to begin the search with):
  SystemView class name _
  Class name  _
  Class ID    _

  Object name _
  Object ID   _ (Hexadecimal value)

Output options:
  Level of field detail . . DATA  (Struct, Data, Hex)
  Level of subfield detail  NONE  (None, Struct, Data, Hex)
  Maximum lines returned    5000
  Display field IDs . . . . N (Y, N) Display extended field info  N (Y, N)

(Use PF8 to further specify query)

CMD==>
F1= Help   F2= End    F3= Return  F4= Clear   F5= PrevOut F6= Roll
           F8= Next                                          F12=PrevCmd
```

Figure 103. RODMView Compound Query Panel 1 — EKGVQA1I



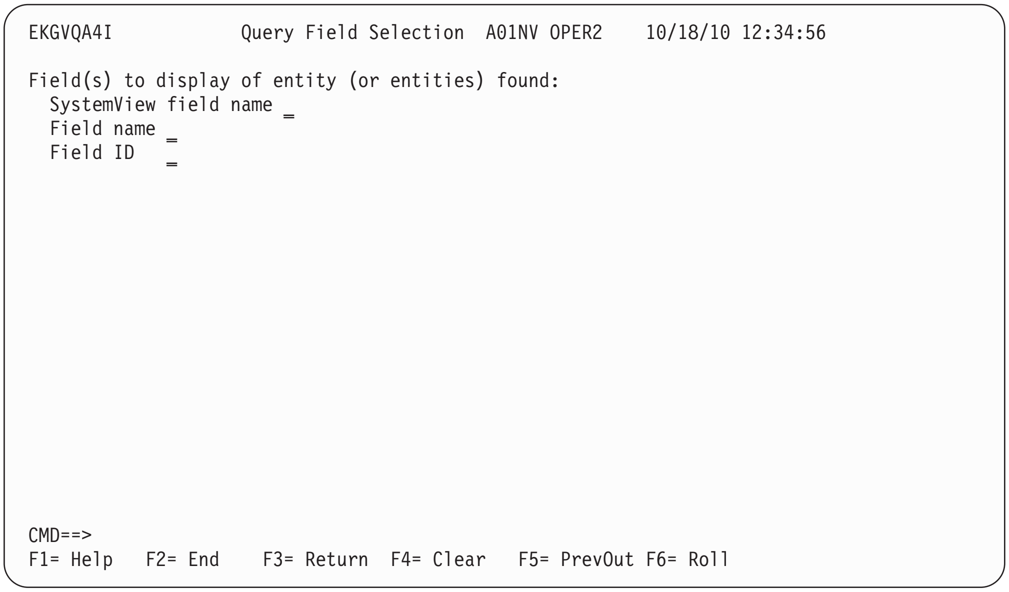
```
EKGVQA2I                 Query Criteria  A01NV OPER2    10/18/10 12:34:56

Entities from the previous panel should meet the following criteria:
  SystemView field name _
  Field name _
  Operator   _ (=, >, <, <>, <=, >=)
  Value . .  _

Operator between these two criteria AND (And, Or)

Entities from the previous panel should also meet the following criteria:
  SystemView field name _
  Field name _
  Operator   _ (=, >, <, <>, <=, >=)
  Value . .  _




(Use PF8 to further specify query)

CMD==>
F1= Help   F2= End    F3= Return  F4= Clear   F5= PrevOut F6= Roll
```

Figure 104. RODMView Query Criteria Panel 2 — EKGVQA2I



```
EKGVQA3I        Query Traversed Criteria  A01NV OPER2    10/18/10 12:34:56

Find entities linked to the following field (leave blank to ignore):
  Traverse SystemView field name _
  Traverse field name _

Entities found in the Traverse field should meet the following criteria:
  SystemView field name _
  Field name _
  Operator   = (=, >, <, <>, <=, >=)
  Value . .  _

Operator between these two criteria AND (And, Or)

Entities found in the Traverse field should also meet the following criteria:
  SystemView field name _
  Field name _
  Operator   = (=, >, <, <>, <=, >=)
  Value . .  _
(Use PF8 to specify which fields are printed for each entity found)

CMD==>
F1= Help   F2= End    F3= Return  F4= Clear   F5= PrevOut F6= Roll
```

Figure 105. RODMView Query Traversed Criteria Panel 3 — EKGVQA3I



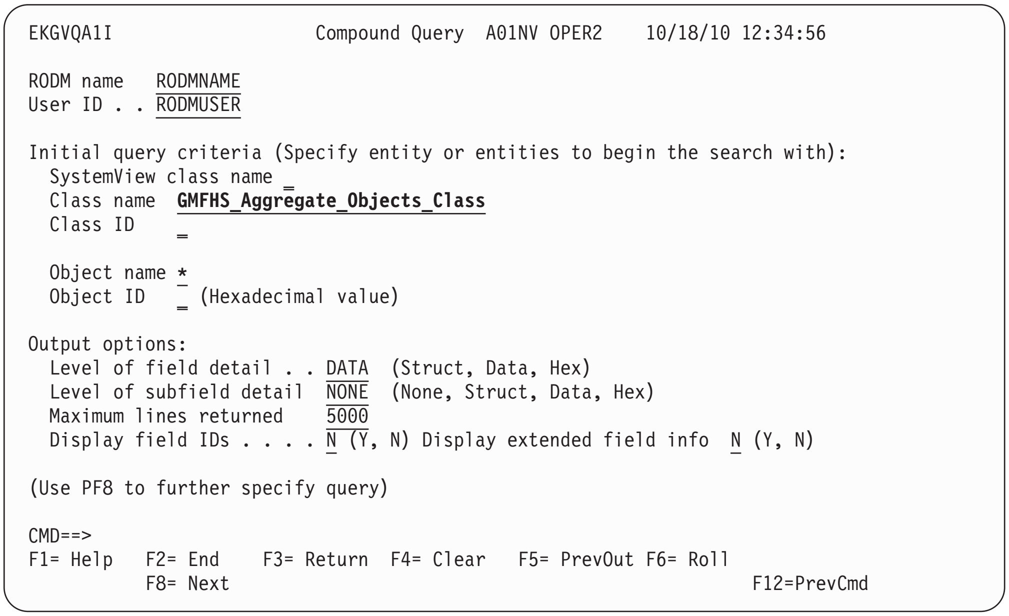
```
EKGVQA4I           Query Field Selection  A01NV OPER2    10/18/10 12:34:56

Field(s) to display of entity (or entities) found:
  SystemView field name _
  Field name _
  Field ID   _


















CMD==>
F1= Help   F2= End    F3= Return  F4= Clear   F5= PrevOut F6= Roll
```

Figure 106. RODMView Query Field Selection Panel 4 — EKGVQA4I

The following sections provide two examples of using the compound query function. Definitions from the GMFHS sample network are used.

Compound Query Example 1

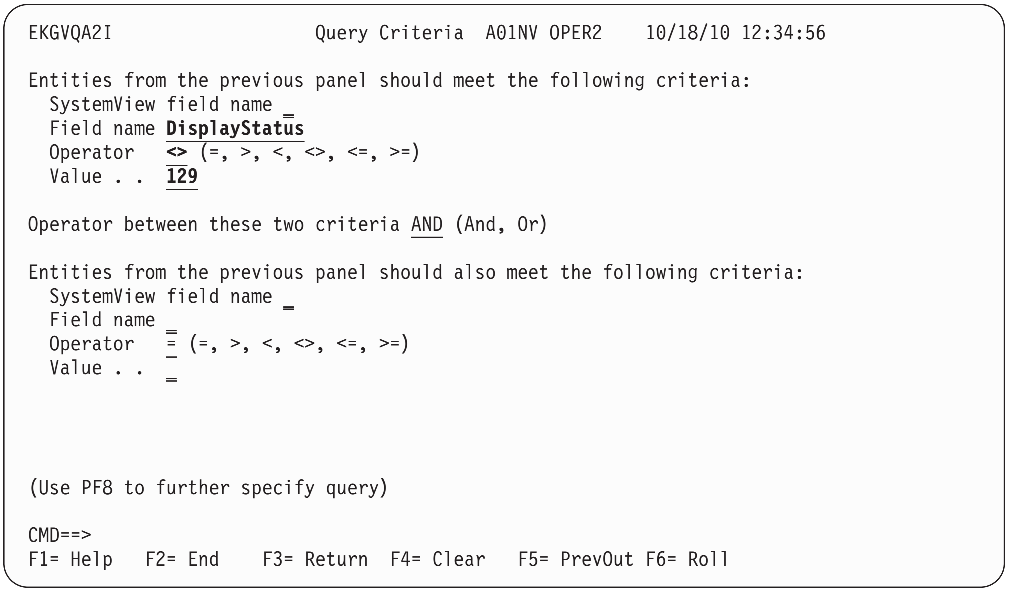
The first example shows how to use the compound query function to find aggregate objects with non-satisfactory status. To do this, type **GMFHS_Aggregate_Objects_Class** for the **Class name**, and the pattern-matching character (*) for the **Object name** on panel EKGVQA1I, as shown in Figure 107 on page 517.



```
EKGVQA1I                   Compound Query  A01NV OPER2    10/18/10 12:34:56

RODM name   RODMNAME
User ID . . RODMUSER

Initial query criteria (Specify entity or entities to begin the search with):
  SystemView class name _
  Class name  GMFHS_Aggregate_Objects_Class
  Class ID    _

  Object name *
  Object ID   _ (Hexadecimal value)

Output options:
  Level of field detail . . DATA  (Struct, Data, Hex)
  Level of subfield detail  NONE  (None, Struct, Data, Hex)
  Maximum lines returned    5000
  Display field IDs . . . . N (Y, N) Display extended field info  N (Y, N)

(Use PF8 to further specify query)

CMD==>
F1= Help   F2= End    F3= Return  F4= Clear   F5= PrevOut F6= Roll
           F8= Next                                         F12=PrevCmd
```

Figure 107. Starting a Compound Query on the GMFHS_Aggregate_Objects_Class

To select those objects that have an unsatisfactory status, press PF8 on the first compound query panel to scroll to the second compound query panel, EKGVQA2I. Specify that the DisplayStatus field is to have a value other than 129, as shown in Figure 108.

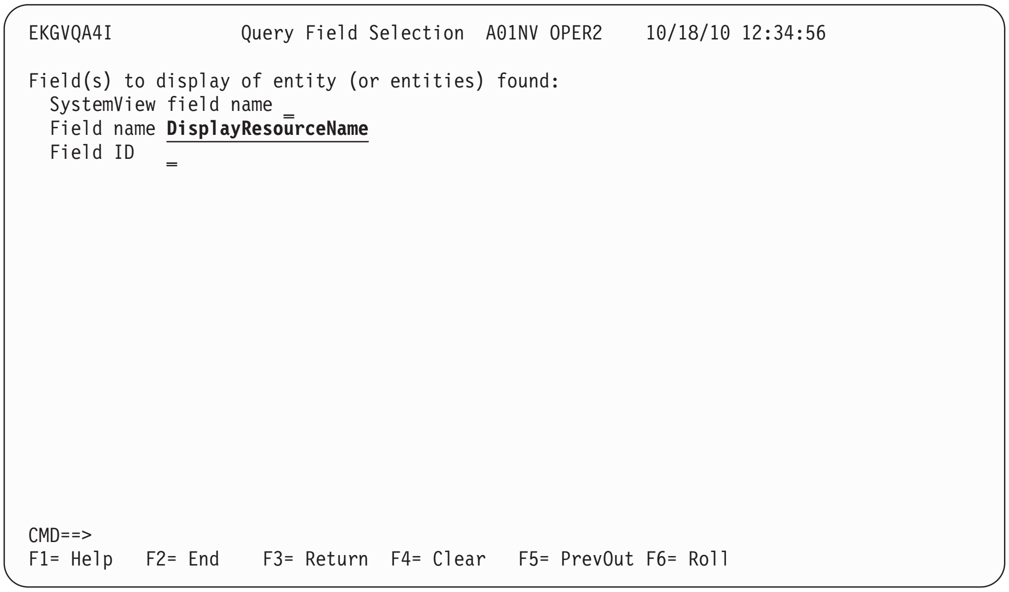


```
EKGVQA2I                   Query Criteria  A01NV OPER2    10/18/10 12:34:56

Entities from the previous panel should meet the following criteria:
  SystemView field name _
  Field name DisplayStatus
  Operator   <> (=, >, <, <>, <=, >=)
  Value . .  129

Operator between these two criteria AND (And, Or)

Entities from the previous panel should also meet the following criteria:
  SystemView field name _
  Field name _
  Operator   = (=, >, <, <>, <=, >=)
  Value . .  _




(Use PF8 to further specify query)

CMD==>
F1= Help   F2= End    F3= Return  F4= Clear   F5= PrevOut F6= Roll
```

Figure 108. Selecting Only Those Entities that Have Nonsatisfactory DisplayStatus

Because no values are specified for any other input fields, RODMView ignores these input fields.

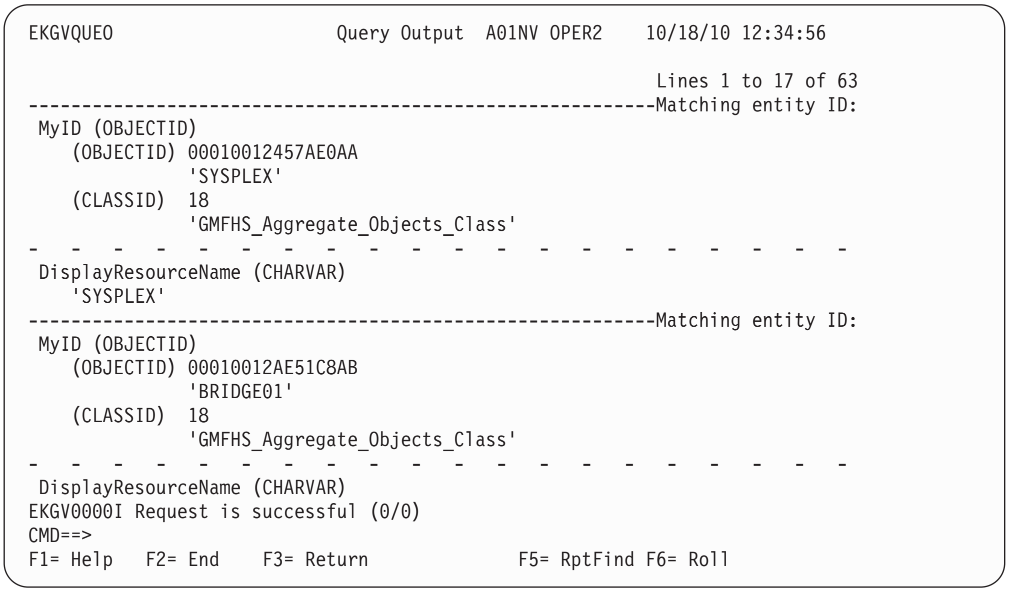
You can restrict the fields that are displayed for the entities found that meet the criteria. For example, to display only the DisplayResourceName of the entities found, press PF8 twice to display the fourth panel EKGVQA4I, and fill in the input fields as shown in Figure 109 on page 518.



```
EKGVQA4I          Query Field Selection  A01NV OPER2    10/18/10 12:34:56

Field(s) to display of entity (or entities) found:
  SystemView field name _
  Field name DisplayResourceName
  Field ID   _

















CMD==>
F1= Help   F2= End    F3= Return  F4= Clear   F5= PrevOut F6= Roll
```

Figure 109. Selecting Only the DisplayResourceName Field to be Displayed

After the compound query specification has been completed, press Enter to run the query. If all of the GMFHS Sample Network aggregate objects were in unsatisfactory status, the output is displayed as shown in Figure 110.



```
EKGVQUEO                   Query Output  A01NV OPER2    10/18/10 12:34:56

                                                          Lines 1 to 17 of 63
-----------------------------------------------------------Matching entity ID:
 MyID (OBJECTID)
    (OBJECTID) 00010012457AE0AA
               'SYSPLEX'
    (CLASSID)  18
               'GMFHS_Aggregate_Objects_Class'
-   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
 DisplayResourceName (CHARVAR)
    'SYSPLEX'
-----------------------------------------------------------Matching entity ID:
 MyID (OBJECTID)
    (OBJECTID) 00010012AE51C8AB
               'BRIDGE01'
    (CLASSID)  18
               'GMFHS_Aggregate_Objects_Class'
-   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
 DisplayResourceName (CHARVAR)
EKGV0000I Request is successful (0/0)
CMD==>
F1= Help   F2= End    F3= Return              F5= RptFind F6= Roll
```

Figure 110. Compound Query Example 1 Output

There are 63 lines of output available, but only 17 lines are visible on the output panel at a time, as shown in Figure 110. Use PF8 to scroll through the output to display all of the entities that met the criteria.

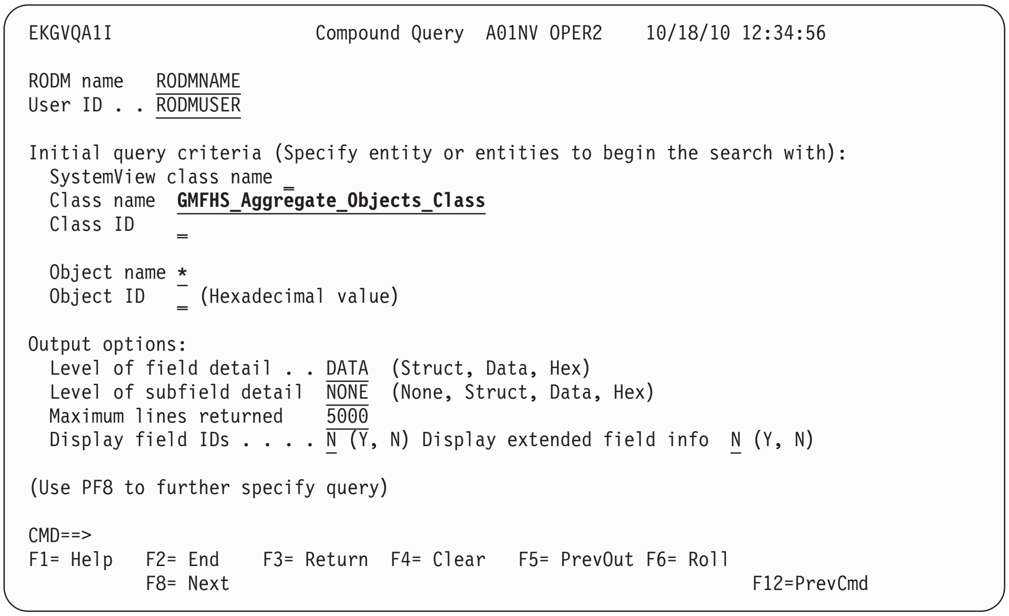
Compound Query Example 2

The second example shows how to use the compound query function to find all of the physically connected (through the ComposedOfPhysical link) objects of aggregates that have a non-satisfactory status, while the aggregate objects have a satisfactory status. This compound query example uses the following criteria:

- Which objects to start with (all aggregates that have satisfactory status)

- Which field to traverse (the ComposedOfPhysical link)
- The criteria to apply to the objects on the other side of the link (a non-satisfactory status).

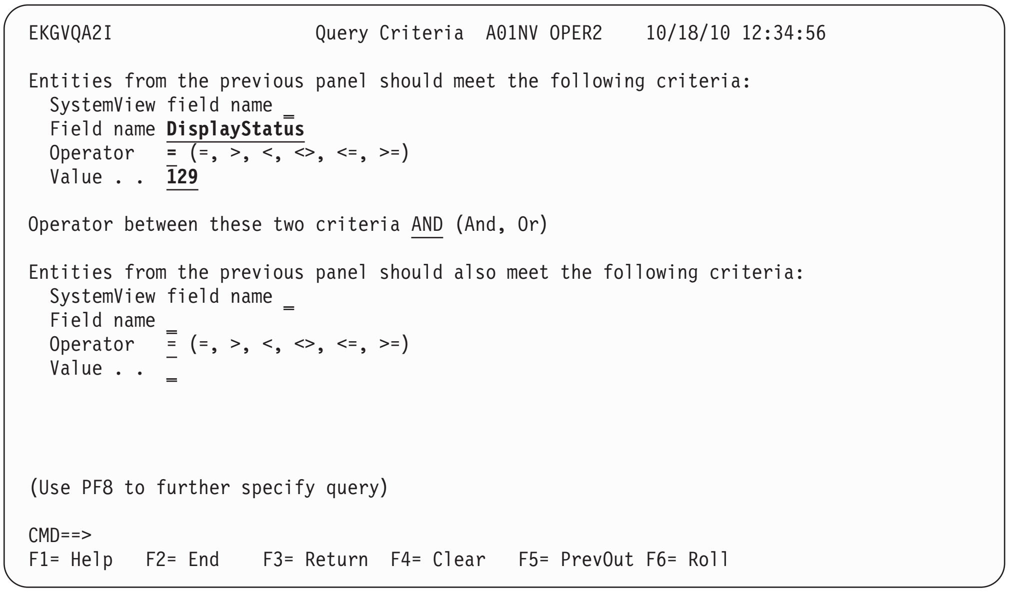
To do this, specify `GMFHS_Aggregate_Objects_Class` for the **Class name** and the pattern matching character (*) for the **Object name** on panel EKGVQA1I as shown in Figure 111.



```
EKGVQA1I                 Compound Query  A01NV OPER2    10/18/10 12:34:56

RODM name   RODMNAME
User ID . . RODMUSER

Initial query criteria (Specify entity or entities to begin the search with):
  SystemView class name _
  Class name  GMFHS_Aggregate_Objects_Class
  Class ID
              _

  Object name *
  Object ID   _ (Hexadecimal value)

Output options:
  Level of field detail . . DATA  (Struct, Data, Hex)
  Level of subfield detail  NONE  (None, Struct, Data, Hex)
  Maximum lines returned    5000
  Display field IDs . . . . N (Y, N) Display extended field info  N (Y, N)

(Use PF8 to further specify query)

CMD==>
F1= Help   F2= End    F3= Return  F4= Clear   F5= PrevOut F6= Roll
           F8= Next                                         F12=PrevCmd
```

Figure 111. Starting a Compound Query on the GMFHS_Aggregate_Objects_Class

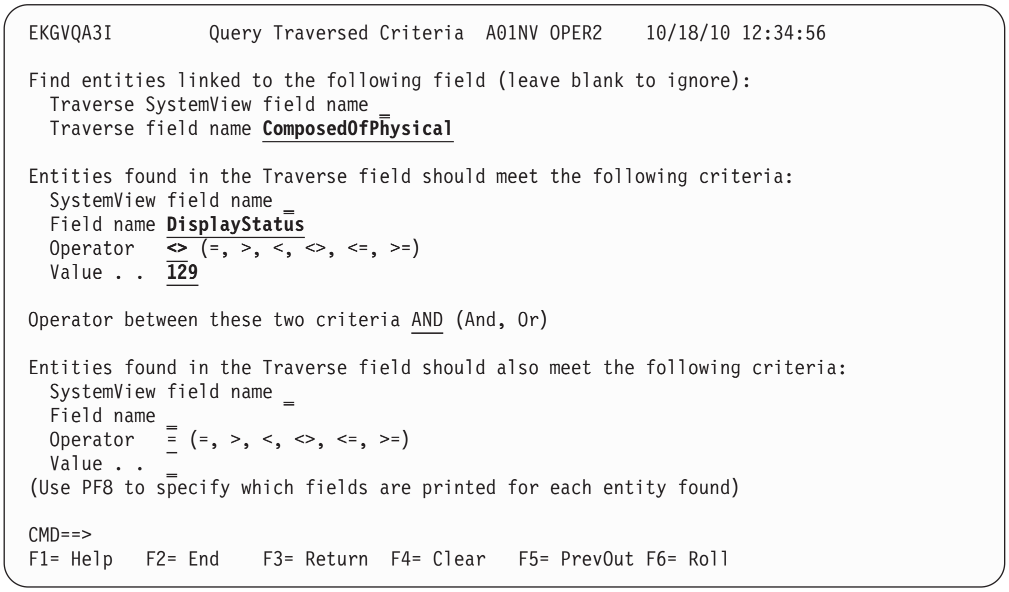
To select only those objects that have a non-satisfactory status, press PF8 on the first compound query panel to display the second compound query panel, EKGVQA2I. Specify that the DisplayStatus field is to have the value 129, as shown in Figure 112.



```
EKGVQA2I                 Query Criteria  A01NV OPER2    10/18/10 12:34:56

Entities from the previous panel should meet the following criteria:
  SystemView field name _
  Field name DisplayStatus
  Operator   = (=, >, <, <>, <=, >=)
  Value . .  129

Operator between these two criteria AND (And, Or)

Entities from the previous panel should also meet the following criteria:
  SystemView field name _
  Field name _
  Operator   = (=, >, <, <>, <=, >=)
  Value . .  _




(Use PF8 to further specify query)

CMD==>
F1= Help   F2= End    F3= Return  F4= Clear   F5= PrevOut F6= Roll
```

Figure 112. Selecting Only Those Entities Having a Satisfactory DisplayStatus

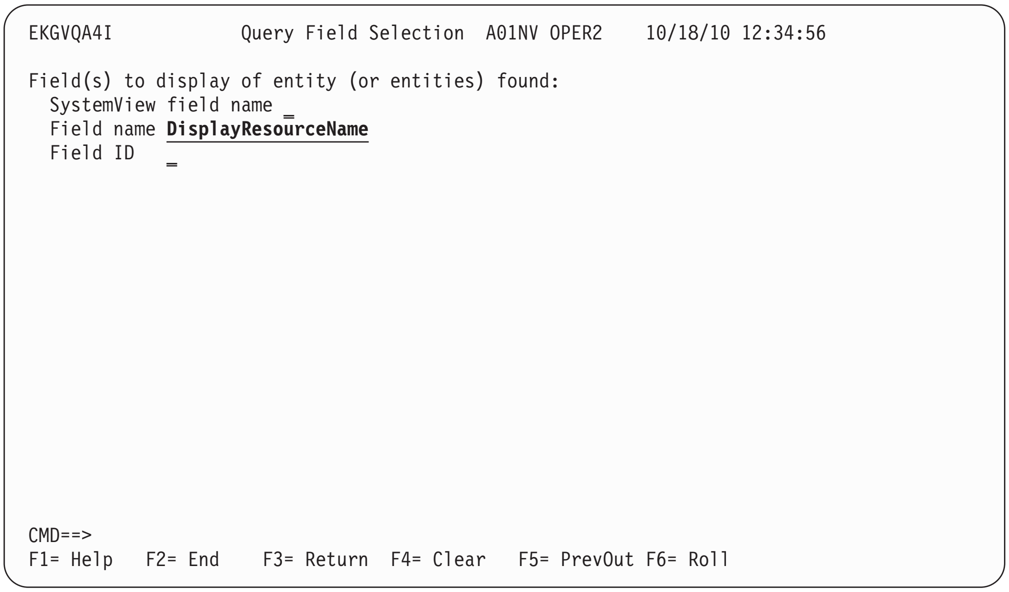
To specify that the query follows the ComposedOfPhysical link field and that those objects found on that link have an unsatisfactory DisplayStatus, press PF8 to scroll to the third compound query panel, EKGVQA3I. The panel is filled in as shown in Figure 113.



```
EKGVQA3I         Query Traversed Criteria  A01NV OPER2    10/18/10 12:34:56

Find entities linked to the following field (leave blank to ignore):
  Traverse SystemView field name _
  Traverse field name ComposedOfPhysical

Entities found in the Traverse field should meet the following criteria:
  SystemView field name _
  Field name DisplayStatus
  Operator   <> (=, >, <, <>, <=, >=)
  Value . .  129

Operator between these two criteria AND (And, Or)

Entities found in the Traverse field should also meet the following criteria:
  SystemView field name _
  Field name _
  Operator   = (=, >, <, <>, <=, >=)
  Value . .  _
(Use PF8 to specify which fields are printed for each entity found)

CMD==>
F1= Help   F2= End    F3= Return  F4= Clear   F5= PrevOut F6= Roll
```

Figure 113. Traversing Across the ComposedOfPhysical Link Field and Adding DisplayStatus Criteria

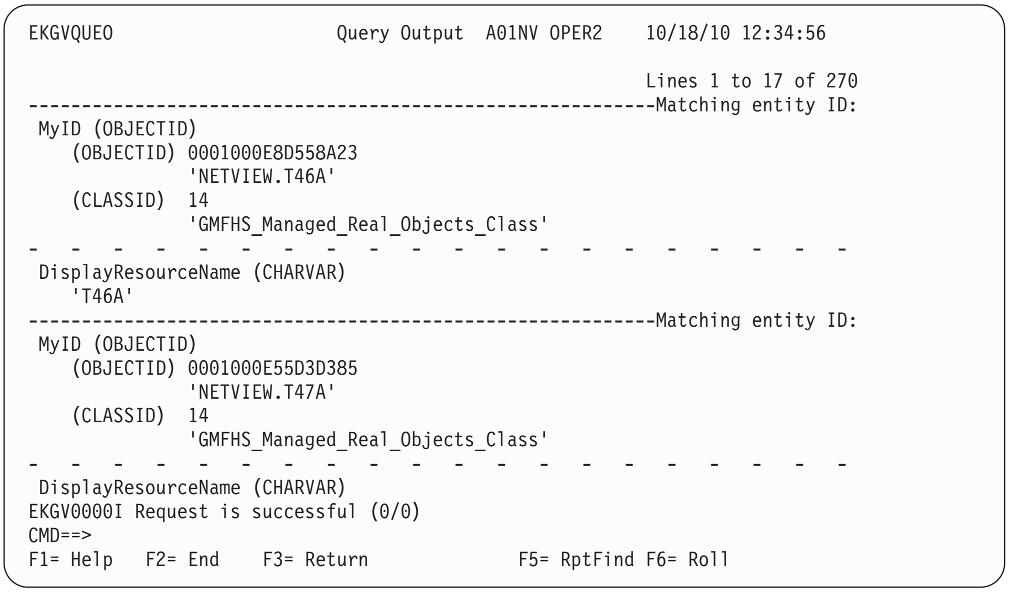
You can restrict the output for the entities displayed using the fourth panel, EKGVQA4I. For example, to display only the DisplayResourceName of the entities found, the fourth panel is filled in as shown in Figure 114.



```
EKGVQA4I            Query Field Selection  A01NV OPER2    10/18/10 12:34:56

Field(s) to display of entity (or entities) found:
  SystemView field name _
  Field name DisplayResourceName
  Field ID   _

CMD==>
F1= Help   F2= End    F3= Return  F4= Clear   F5= PrevOut F6= Roll
```

Figure 114. Selecting Only the DisplayResourceName Field to be Displayed

After the compound query specification has been completed, press Enter to run the query. If some aggregate network objects were in satisfactory status with some of their descendant objects defined to the ComposedOfPhysical link in non-satisfactory status, the output is displayed as shown in Figure 115.



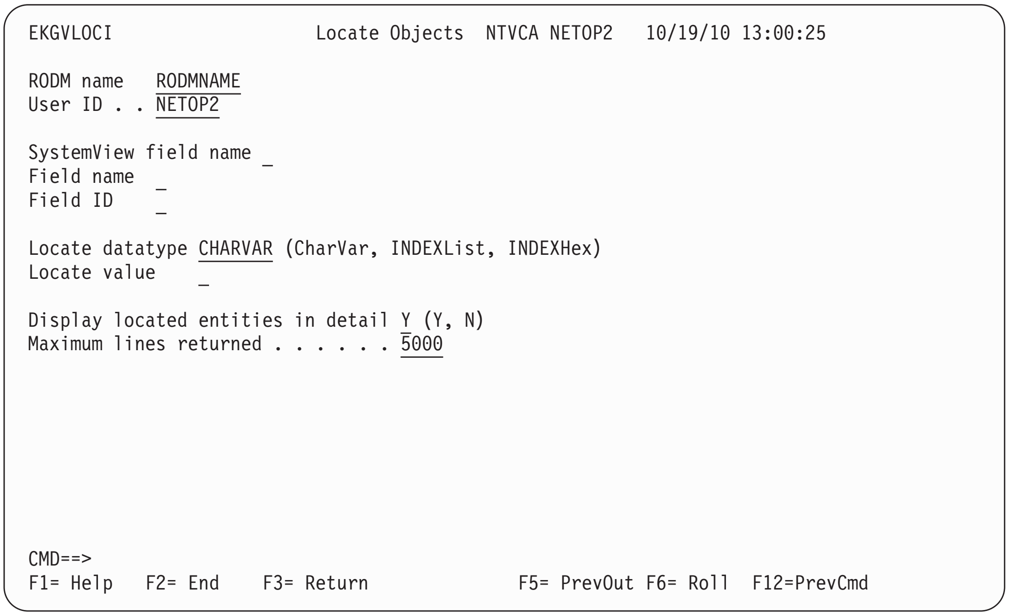
```
EKGVQUEO                     Query Output  A01NV OPER2    10/18/10 12:34:56

                                                          Lines 1 to 17 of 270
-----------------------------------------------------------Matching entity ID:
 MyID (OBJECTID)
    (OBJECTID) 0001000E8D558A23
               'NETVIEW.T46A'
    (CLASSID)  14
               'GMFHS_Managed_Real_Objects_Class'
-   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
 DisplayResourceName (CHARVAR)
    'T46A'
-----------------------------------------------------------Matching entity ID:
 MyID (OBJECTID)
    (OBJECTID) 0001000E55D3D385
               'NETVIEW.T47A'
    (CLASSID)  14
               'GMFHS_Managed_Real_Objects_Class'
-   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
 DisplayResourceName (CHARVAR)
EKGV0000I Request is successful (0/0)
CMD==>
F1= Help   F2= End    F3= Return              F5= RptFind F6= Roll
```

Figure 115. Query Output Example 2

Locate Objects Function

Use the Locate Objects function to search for objects with data defined in indexed (either CharVar or IndexList) fields:

From the RODMView main menu, select **4. Locate Objects**. The Locate Objects panel is displayed as shown in Figure 116.



```
EKGVLOCI                   Locate Objects  NTVCA NETOP2   10/19/10 13:00:25

RODM name   RODMNAME
User ID . . NETOP2

SystemView field name _
Field name  _
Field ID    _

Locate datatype CHARVAR (CharVar, INDEXList, INDEXHex)
Locate value    _

Display located entities in detail Y (Y, N)
Maximum lines returned . . . . . . 5000








CMD==>
F1= Help   F2= End    F3= Return              F5= PrevOut F6= Roll  F12=PrevCmd
```

Figure 116. Locate Objects Panel

Using the Locate Objects Panel, you can locate objects using the field name and data value, and you can specify whether you want to display the objects themselves or just the number of objects with this value that are located.

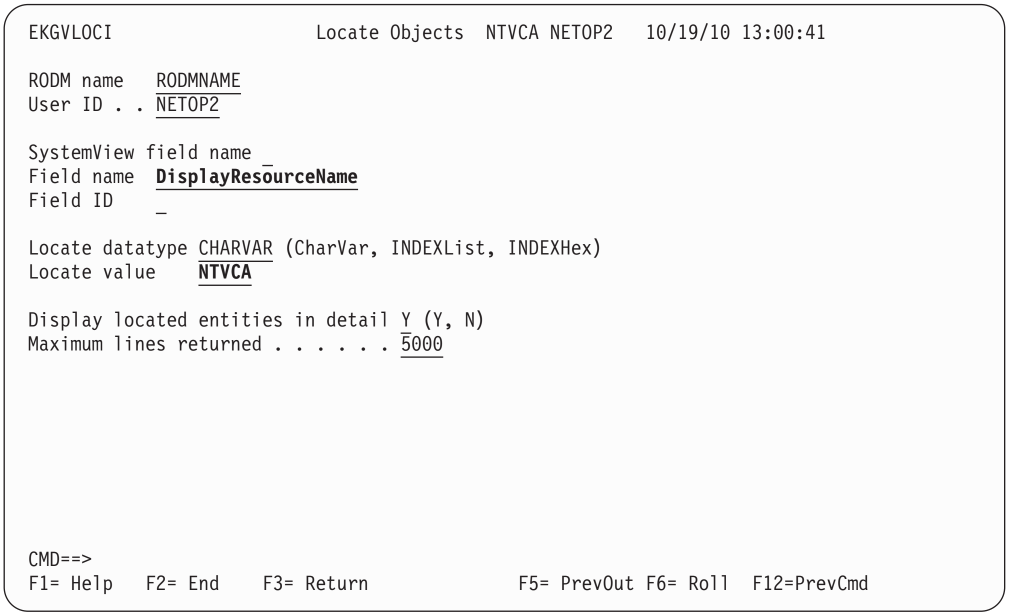
The field specified on this panel must have been created as indexed. For example, both CharVar and IndexList fields can be created as public or public indexed. Fields must be public indexed to use the indexing and locating capabilities.

To locate objects with a particular value in an indexed CharVar field, type **Locate value** as normal characters. To locate data with leading or trailing blanks, enclose the string in quotation marks.

There are two ways to specify the locate data to locate objects with a particular value in an IndexList field, If you specify **INDEXLIST**, you can enter a character string, and it is automatically converted to AnonymousVar data before it is passed to RODM. If you specify **INDEXHEX** as the data type, the data on the Locate value line must be an even number of hexadecimal digits representing the AnonymousVar value you want to locate. Character data can contain blanks. To include leading or trailing blanks, enclose the string in quotation marks.

Note: This data is case sensitive, except on the DisplayResourceName field in the GMFHS data model.

To locate all the objects that have a value of `NTVCA` on a field named DisplayResourceName field, fill in the panel as shown in Figure 117.

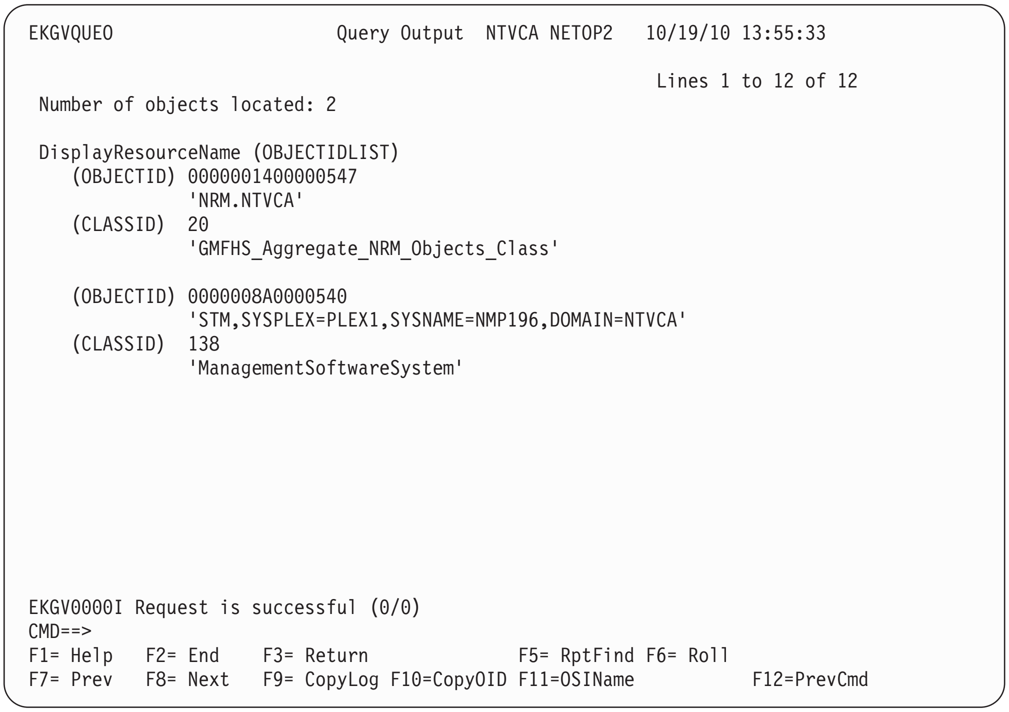


```
EKGVLOCI                  Locate Objects  NTVCA NETOP2   10/19/10 13:00:41

RODM name   RODMNAME
User ID . . NETOP2

SystemView field name _
Field name  DisplayResourceName
Field ID    _

Locate datatype CHARVAR (CharVar, INDEXList, INDEXHex)
Locate value    NTVCA

Display located entities in detail Y (Y, N)
Maximum lines returned . . . . . . 5000








CMD==>
F1= Help   F2= End    F3= Return             F5= PrevOut F6= Roll  F12=PrevCmd
```

Figure 117. Locating Objects with an Indexed CharVar Field

Because CHARVAR is specified as the field datatype, RODMView interprets the data entered on the Locate value field as character data.

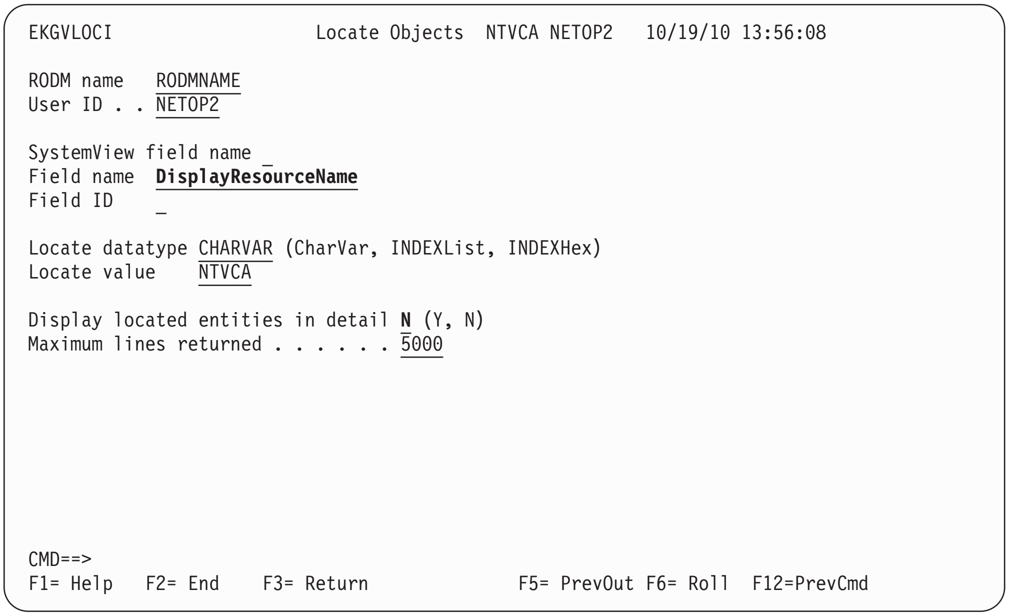
If RODM locates objects with the specified characteristics, the EKGVQUEO panel (Query Output) is displayed as shown in Figure 118 on page 523.



```
EKGVQUEO                     Query Output  NTVCA NETOP2   10/19/10 13:55:33
                                                           Lines 1 to 12 of 12
 Number of objects located: 2

 DisplayResourceName (OBJECTIDLIST)
    (OBJECTID) 0000001400000547
               'NRM.NTVCA'
    (CLASSID)  20
               'GMFHS_Aggregate_NRM_Objects_Class'

    (OBJECTID) 0000008A0000540
               'STM,SYSPLEX=PLEX1,SYSNAME=NMP196,DOMAIN=NTVCA'
    (CLASSID)  138
               'ManagementSoftwareSystem'









EKGV0000I Request is successful (0/0)
CMD==>
F1= Help   F2= End    F3= Return              F5= RptFind F6= Roll
F7= Prev   F8= Next   F9= CopyLog F10=CopyOID F11=OSIName           F12=PrevCmd
```

Figure 118. Locate Objects Output

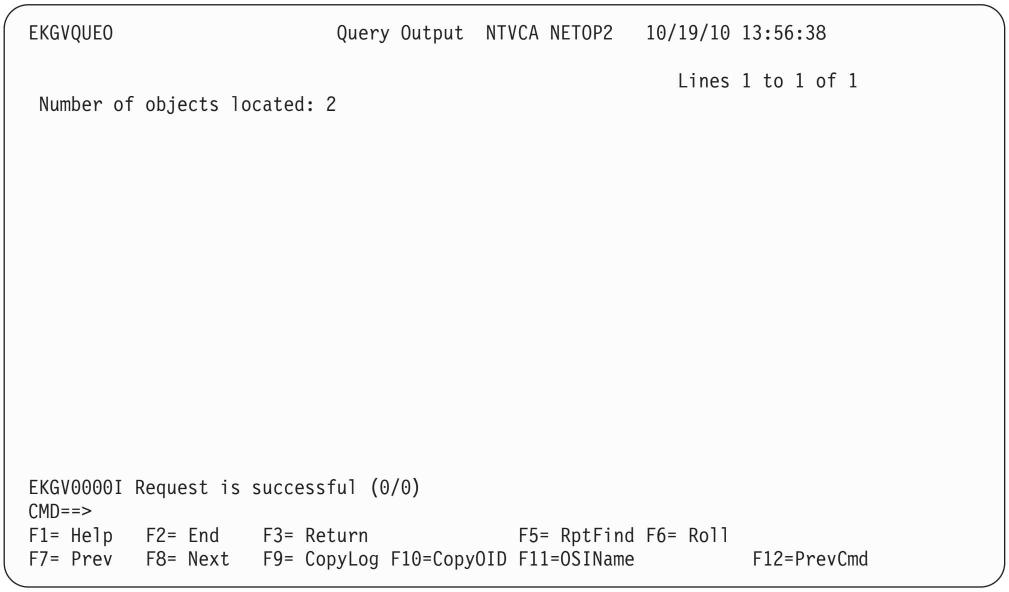
The next example, shown in Figure 119, shows the same locate function, except that N is specified in the Display located entities in detail input field to only report the number of entities that are found with matching data.



```
EKGVLOCI                   Locate Objects  NTVCA NETOP2   10/19/10 13:56:08

RODM name   RODMNAME
User ID . . NETOP2

SystemView field name _
Field name  DisplayResourceName
Field ID    _

Locate datatype CHARVAR (CharVar, INDEXList, INDEXHex)
Locate value    NTVCA

Display located entities in detail N (Y, N)
Maximum lines returned . . . . . . 5000








CMD==>
F1= Help   F2= End    F3= Return              F5= PrevOut F6= Roll  F12=PrevCmd
```

Figure 119. Locating Objects with Number of Objects and No Object Detail

Because `N` was specified in the Display located entities in detail field, the output are displayed as shown in Figure 120 on page 524.



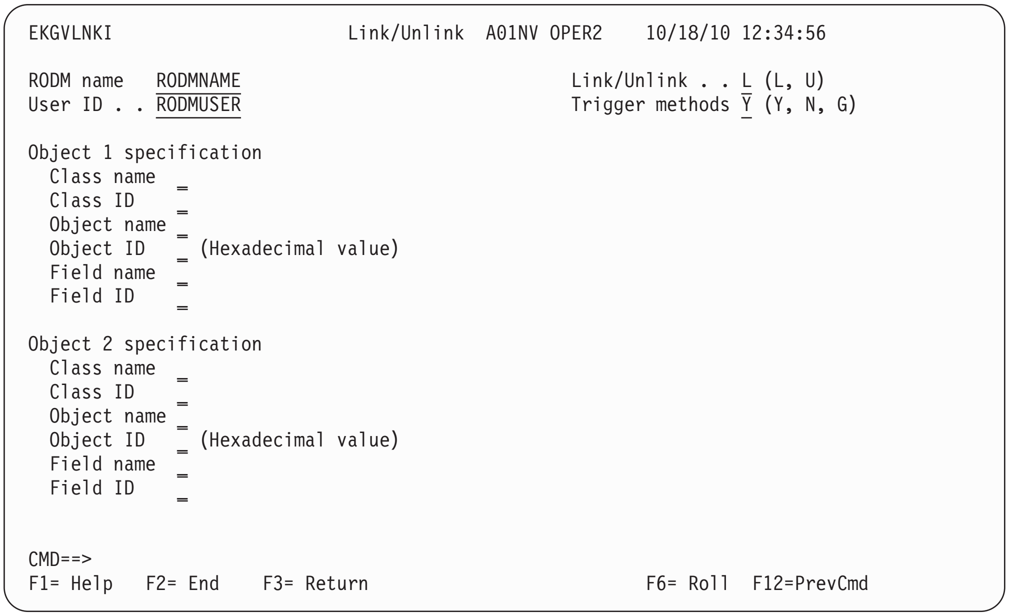
```
EKGVQUEO                 Query Output  NTVCA NETOP2   10/19/10 13:56:38
                                                         Lines 1 to 1 of 1
 Number of objects located: 2




















EKGV0000I Request is successful (0/0)
CMD==>
F1= Help   F2= End    F3= Return             F5= RptFind F6= Roll
F7= Prev   F8= Next   F9= CopyLog F10=CopyOID F11=OSIName           F12=PrevCmd
```

Figure 120. Locate Objects Output, No Object Detail

Link/Unlink Function

Use the Link/Unlink function to link or unlink the fields of two objects.

From the RODMView main menu, select **5. Link/Unlink**. The Link/Unlink panel is displayed as shown in Figure 121.



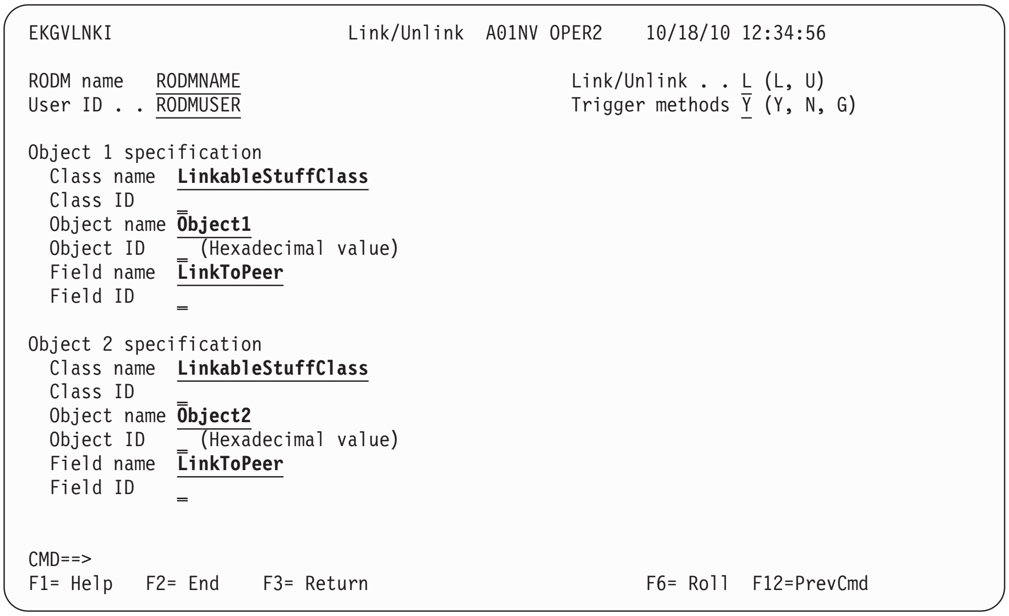
```
EKGVLNKI                  Link/Unlink  A01NV OPER2    10/18/10 12:34:56

RODM name   RODMNAME                         Link/Unlink . . L (L, U)
User ID . . RODMUSER                         Trigger methods Y (Y, N, G)

Object 1 specification
  Class name  _
  Class ID    _
  Object name _
  Object ID   _ (Hexadecimal value)
  Field name  _
  Field ID    _

Object 2 specification
  Class name  _
  Class ID    _
  Object name _
  Object ID   _ (Hexadecimal value)
  Field name  _
  Field ID    _


CMD==>
F1= Help   F2= End    F3= Return                         F6= Roll  F12=PrevCmd
```

Figure 121. RODMView Link Objects Panel — EKGVLNKI

Using the Link/Unlink panel, you can specify two objects to link or unlink, and whether you want associated change methods to be run when the link or unlink is performed.

You must specify enough information to uniquely identify two objects in RODM and the fields through which they are to be linked. For example, if you have a class named `LinkableStuffClass` that has a field called LinkToPeer of type

ObjectLinkList and two objects called Object1 and Object2, you can link them by entering the link request information as shown in Figure 122.



```
EKGVLNKI                      Link/Unlink  A01NV OPER2    10/18/10 12:34:56

RODM name   RODMNAME                               Link/Unlink . . L (L, U)
User ID . . RODMUSER                               Trigger methods Y (Y, N, G)

Object 1 specification
  Class name  LinkableStuffClass
  Class ID    _
  Object name Object1
  Object ID   _ (Hexadecimal value)
  Field name  LinkToPeer
  Field ID    _

Object 2 specification
  Class name  LinkableStuffClass
  Class ID    _
  Object name Object2
  Object ID   _ (Hexadecimal value)
  Field name  LinkToPeer
  Field ID    _


CMD==>
F1= Help   F2= End    F3= Return                          F6= Roll  F12=PrevCmd
```

Figure 122. RODMView Linking Two Objects

You can unlink the two objects by changing the Link/Unlink field from **L** to **U**. If you do not want to involve change methods that are defined to the link fields, change the Trigger methods from **Y** to **N**.

Notes:

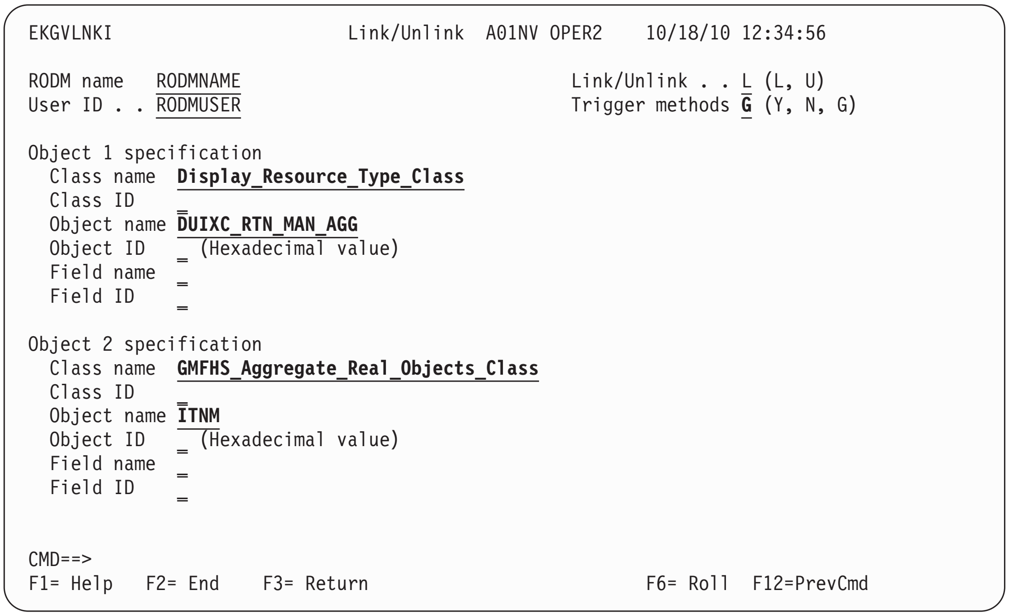
1. Objects can only be linked through fields of data types ObjectLink or ObjectLinkList.
2. Classes cannot be linked or unlinked.

The only output from this function is the return and reason codes displayed on the message line.

Linking with GMFHS Methods DUIFCLRT and DUIFCUAP

You can use the Link/Unlink function to run the GMFHS methods DUIFCLRT and DUIFCUAP. Method DUIFCLRT links a GMFHS displayable object to a GMFHS resource type object. See “DUIFCLRT: Link Resource Type Method” on page 489 for more information about method DUIFCLRT. Method DUIFCUAP creates an aggregation path from a parent to a child GMFHS displayable object. See “DUIFCUAP: Update Aggregation Path Method” on page 491 for more information about method DUIFCUAP. For more information about aggregate objects and aggregation, see “Defining GMFHS Aggregate Objects” on page 38.

To run these GMFHS methods, enter **G** in the Trigger methods input field of the Link/Unlink panel. Also specify whether the method links or unlinks the two objects by specifying either **L** or **U** in the Link/Unlink input field. Specify the class and object information for the two objects that are to be linked or unlinked. RODMView determines which method needs to be run. If either of the objects is in the GMFHS `Displayable_Objects_Class` class, method DUIFCLRT (link resource type) is triggered. Otherwise, method DUIFCUAP (update aggregation path) method is triggered. For example, to link the GMFHS aggregate object ITNM to the GMFHS display resource type object `DUIXC_RTN_MAN_AGG`, the Link/Unlink panel is filled in as shown in Figure 123.

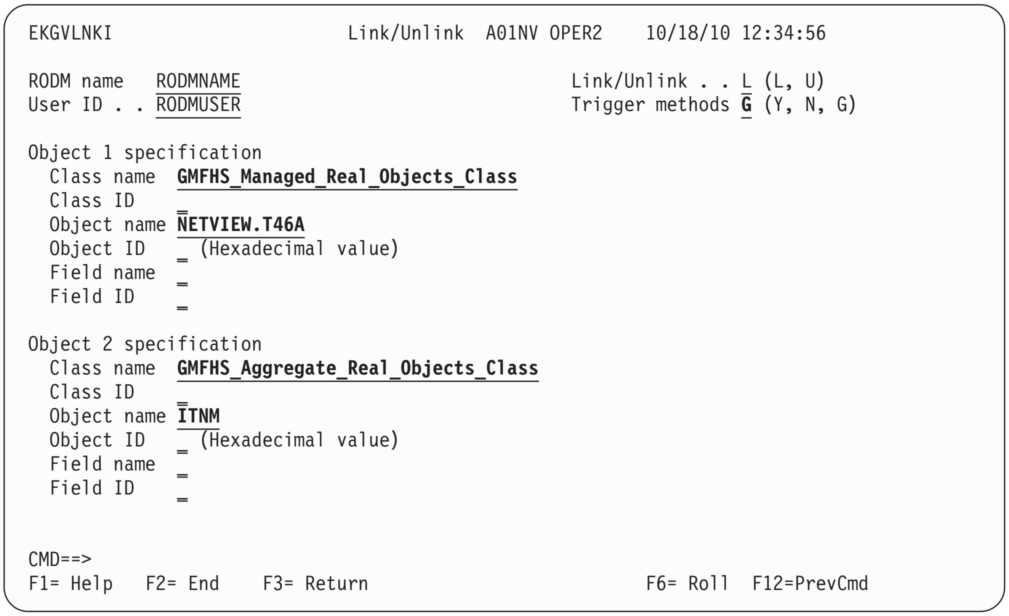


```
EKGVLNKI                  Link/Unlink  A01NV OPER2    10/18/10 12:34:56

RODM name   RODMNAME                            Link/Unlink . . L (L, U)
User ID . . RODMUSER                            Trigger methods G (Y, N, G)

Object 1 specification
  Class name  Display_Resource_Type_Class
  Class ID    _
  Object name DUIXC_RTN_MAN_AGG
  Object ID   _ (Hexadecimal value)
  Field name  _
  Field ID    _

Object 2 specification
  Class name  GMFHS_Aggregate_Real_Objects_Class
  Class ID    _
  Object name ITNM
  Object ID   _ (Hexadecimal value)
  Field name  _
  Field ID    _


CMD==>
F1= Help   F2= End    F3= Return                         F6= Roll  F12=PrevCmd
```

Figure 123. RODMView Linking a GMFHS Aggregate Object To Its Resource Type

Because one of the objects specified the `Displayable_Resource_Type_Class`, method DUIFCLRT is run. The order in which the objects are specified is not significant.

To establish an aggregation path between two objects, the DUIFCUAP is run, with one object specified as the aggregation parent and the other the aggregation child. An aggregation child is lower in the aggregation hierarchy than the aggregation parent. RODMView runs the DUIFCUAP method if the Trigger methods input field is set to **G** and the class specifications of both objects are GMFHS displayable object classes. The first object specification is assumed by RODMView to be the aggregation child, and the second is assumed to be the aggregation parent. GMFHS requires that an aggregate parent object is in the `GMFHS_Aggregate_Objects_Class` class. For example, to make the GMFHS managed real object NETVIEW.T46A an aggregation child of the GMFHS aggregate object ITNM, fill in the Link/Unlink panel as shown in Figure 124 on page 527.



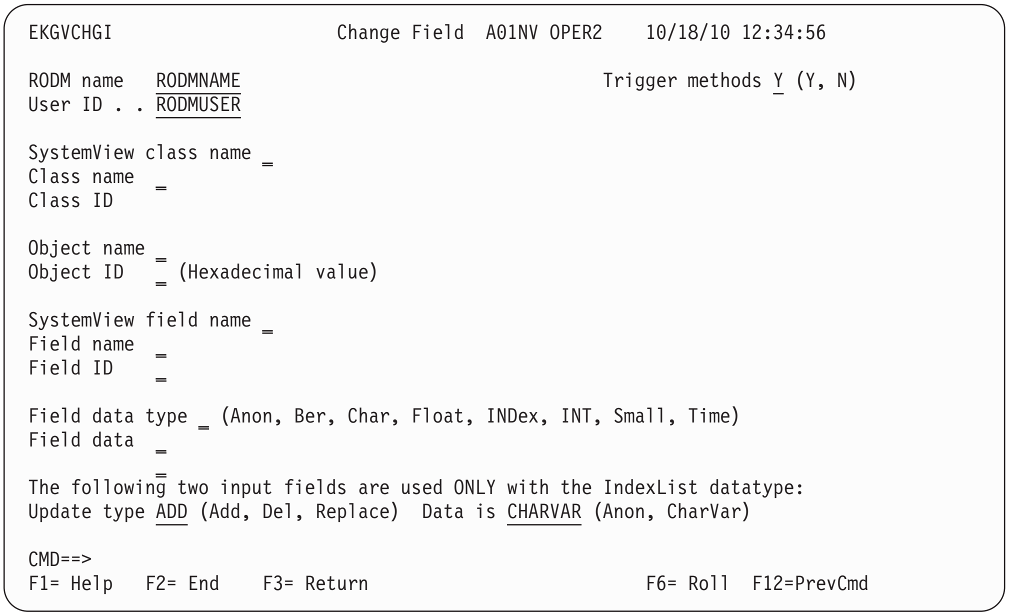
```
EKGVLNKI                  Link/Unlink  A01NV OPER2    10/18/10 12:34:56

RODM name   RODMNAME                                  Link/Unlink . . L (L, U)
User ID . . RODMUSER                                  Trigger methods G (Y, N, G)

Object 1 specification
  Class name  GMFHS_Managed_Real_Objects_Class
  Class ID    _
  Object name NETVIEW.T46A
  Object ID   _ (Hexadecimal value)
  Field name  _
  Field ID    _

Object 2 specification
  Class name  GMFHS_Aggregate_Real_Objects_Class
  Class ID    _
  Object name ITNM
  Object ID   _ (Hexadecimal value)
  Field name  _
  Field ID    _


CMD==>
F1= Help   F2= End    F3= Return                           F6= Roll  F12=PrevCmd
```

Figure 124. Updating the Aggregation Path Between NETVIEW.T46A and ITNM

Change Field Function

Use the change field function to change certain types of data stored in fields of classes or objects.

From the RODMView main menu, select **6. Change field**. The Change field panel is displayed as shown in Figure 125.



```
EKGVCHGI                 Change Field  A01NV OPER2    10/18/10 12:34:56

RODM name   RODMNAME                                     Trigger methods Y (Y, N)
User ID . . RODMUSER

SystemView class name _
Class name  _
Class ID

Object name _
Object ID   _ (Hexadecimal value)

SystemView field name _
Field name  _
Field ID    _

Field data type _ (Anon, Ber, Char, Float, INDex, INT, Small, Time)
Field data  _
            _
The following two input fields are used ONLY with the IndexList datatype:
Update type ADD (Add, Del, Replace)  Data is CHARVAR (Anon, CharVar)

CMD==>
F1= Help   F2= End    F3= Return                           F6= Roll  F12=PrevCmd
```

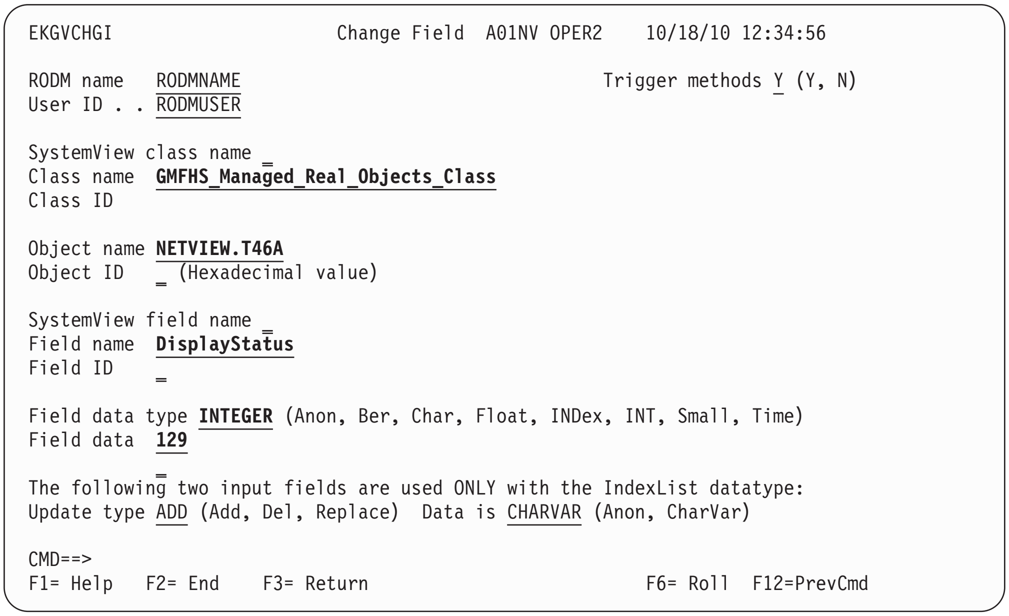
Figure 125. RODMView Change Field Panel — EKGVCHGI

You can change the value of a field of an entity by specifying either its name or ID along with the name or ID of the field, the field data type, and the new data to copy. You can also specify whether you want associated change methods to be triggered before the change takes place. Fields with the following data types can be changed:

- AnonymousVar

- BERVar
- CharVar
- Floating
- IndexList
- Integer
- Smallint
- TimeStamp

For example, the display status (the color) of a GMFHS managed object can be changed by filling in the class, the object and field to change, and the new value to copy to the field. To change display status of GMFHS managed real object NETVIEW.T46A to 129, fill in panel EKGVCHGI as shown in Figure 126.



```
EKGVCHGI                     Change Field  A01NV OPER2    10/18/10 12:34:56

RODM name   RODMNAME                                  Trigger methods Y (Y, N)
User ID . . RODMUSER

SystemView class name _
Class name  GMFHS_Managed_Real_Objects_Class
Class ID

Object name NETVIEW.T46A
Object ID   _ (Hexadecimal value)

SystemView field name _
Field name  DisplayStatus
Field ID    _

Field data type INTEGER (Anon, Ber, Char, Float, INDex, INT, Small, Time)
Field data  129
            _
The following two input fields are used ONLY with the IndexList datatype:
Update type ADD (Add, Del, Replace)  Data is CHARVAR (Anon, CharVar)

CMD==>
F1= Help   F2= End    F3= Return                          F6= Roll  F12=PrevCmd
```

Figure 126. RODMView Changing a Field

Notes:

1. The Field data input field is limited to a maximum length of 134 characters. The two lines of input are concatenated together when sending the data to RODM.
2. The input fields at the bottom of the panel, Update type and Data is, are used only for IndexList data type fields. These input fields are ignored for all other data types, even if they are specified.

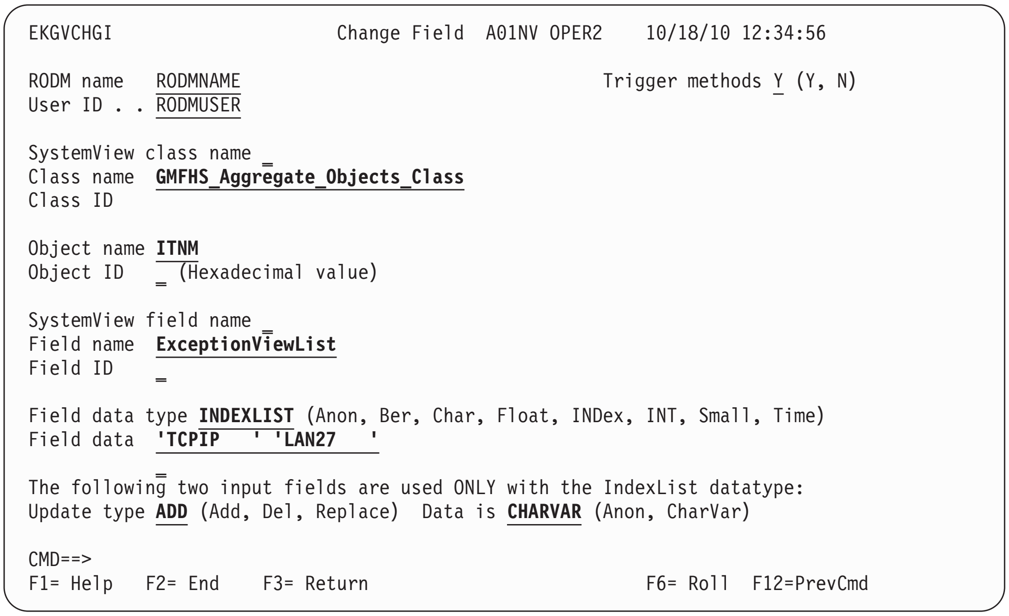
Table 222 on page 529 lists, by data type, the rules for changing fields.

Table 222. Rules for Changing Specific Data Type

Data Type	Rules
AnonymousVar and BERVar	• The field data entered is interpreted as hexadecimal. • The field data value is validated to ensure that it contains a hex string. If it does not contain a hex string, the following message is displayed: `EKGV8052E The Field data value is not a valid hex value` • When entering hexadecimal data, do not use any special notation like `X'001122'`, for example. It is sufficient to enter just the numeric portion `001122`. • AnonymousVar and BERVar field data types contain a 2-byte length before the actual data. Do not include the 2-byte length when you enter a value. RODMView calculates this value after parsing the data.
CharVar	Accepts characters.
Floating	Accepts real numbers.
IndexList	See “Changing IndexList Fields.”
Integer	Accepts integers.
TimeStamp	• The string is interpreted as an 8-byte (16 digit) hexadecimal value, which represents the number of Lillian seconds. • Query the EKG_Name field on the EKG_System class with the `HEX` level of subfield detail to see an example of this value.

Changing IndexList Fields

Use the Change Field function to add elements to or delete elements from an IndexList field. An example of an IndexList field is the ExceptionViewList field. Use the Change Field function of RODMView to dynamically change the value of an ExceptionViewList field. For example, to add views named 'TCPIP ' and 'LAN27 ' to the list of exception views for the aggregate object ITNM, fill in panel EKGVCHGI as shown in Figure 127.



```
EKGVCHGI                     Change Field  A01NV OPER2    10/18/10 12:34:56

RODM name   RODMNAME                                  Trigger methods Y (Y, N)
User ID . . RODMUSER

SystemView class name _
Class name  GMFHS_Aggregate_Objects_Class
Class ID

Object name ITNM
Object ID   _ (Hexadecimal value)

SystemView field name _
Field name  ExceptionViewList
Field ID    _

Field data type INDEXLIST (Anon, Ber, Char, Float, INDex, INT, Small, Time)
Field data  'TCPIP   ' 'LAN27   '
            _
The following two input fields are used ONLY with the IndexList datatype:
Update type ADD (Add, Del, Replace)  Data is CHARVAR (Anon, CharVar)

CMD==>
F1= Help   F2= End    F3= Return                          F6= Roll  F12=PrevCmd
```

Figure 127. Adding Multiple Values to an IndexList Field in Character Format

Notes:

1. The two view names are added to the list, even if the list does not contain other values.
2. If a value already exists in the list, it is not duplicated.
3. Multiple input values must be separated by spaces, for example, `'TCPIP   ' 'LAN27   '`.
4. When values contain spaces, enclose the value in single quotation marks, for example `'TCPIP   '`.

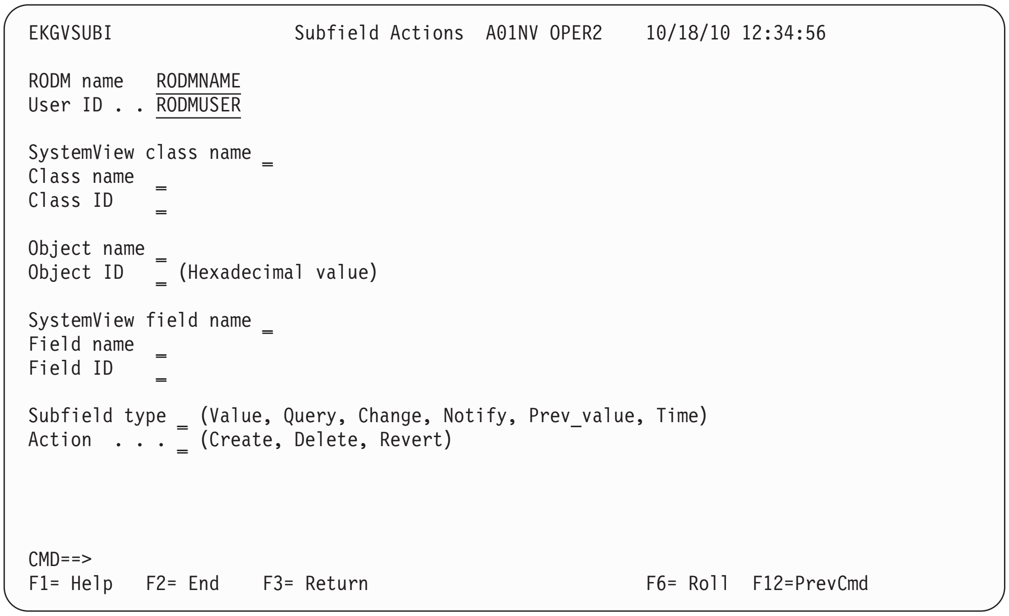
To replace the contents of an index list with the data you specify on the panel, change the **Update type** input field to `REPLACE`.

Subfield Actions Function

Use the Subfield Actions function to specify:

- The type of subfield (Value, Query, Change, Notify, Prev_value, or Timestamp)
- Which action you want to perform (create, delete, or revert to an inherited value)
- The field that the subfield is associated with

From the RODMView main menu, specify option 7, Subfield Actions. The Subfield Actions panel is displayed as shown in Figure 128.



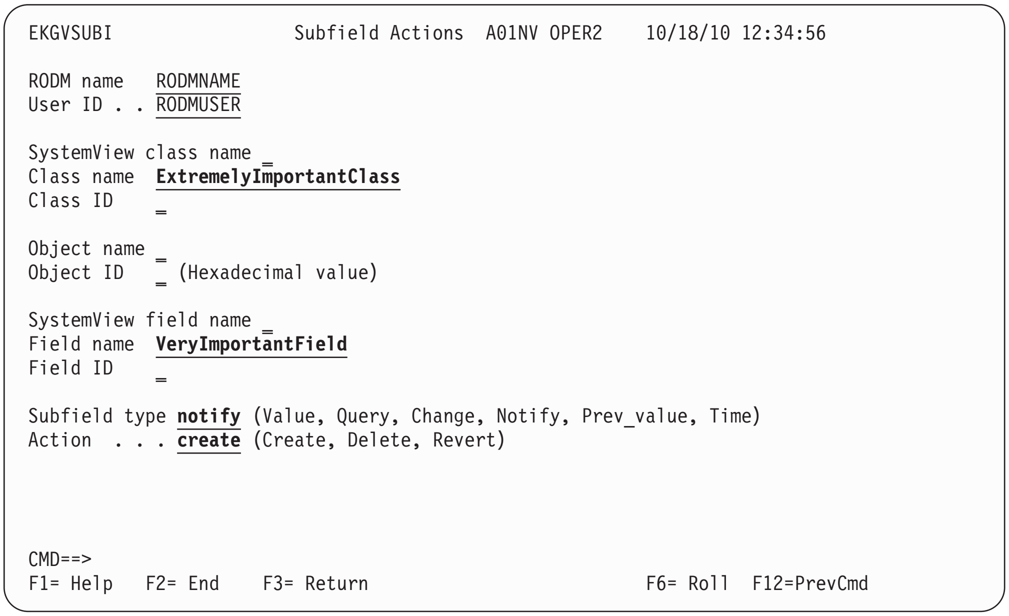
```
EKGVSUBI                 Subfield Actions  A01NV OPER2    10/18/10 12:34:56

RODM name   RODMNAME
User ID . . RODMUSER

SystemView class name _
Class name  _
Class ID    _

Object name _
Object ID   _ (Hexadecimal value)

SystemView field name _
Field name  _
Field ID    _

Subfield type _ (Value, Query, Change, Notify, Prev_value, Time)
Action  . . . _ (Create, Delete, Revert)



CMD==>
F1= Help   F2= End    F3= Return                          F6= Roll  F12=PrevCmd
```

Figure 128. RODMView Subfield Actions Panel — EKGVSUBI

Some actions are not permitted for certain subfields. For example, RODM does not permit a user to make a Timestamp subfield revert to an inherited value.

Subfields can only be created or deleted on fields of classes. For example, if you want to create a notify subfield on a field called VeryImportantField which exists on the `ExtrememlyImportantClass` class, enter the information in the Subfield Action panel as shown in Figure 129 on page 531.

]]>



```
EKGVSUBI                 Subfield Actions  A01NV OPER2    10/18/10 12:34:56

RODM name   RODMNAME
User ID . . RODMUSER

SystemView class name _
Class name  ExtremelyImportantClass
Class ID    _

Object name _
Object ID   _ (Hexadecimal value)

SystemView field name _
Field name  VeryImportantField
Field ID    _

Subfield type notify (Value, Query, Change, Notify, Prev_value, Time)
Action  . . . create (Create, Delete, Revert)




CMD==>
F1= Help   F2= End    F3= Return                         F6= Roll  F12=PrevCmd
```

Figure 129. RODMView Creating a Notify Subfield

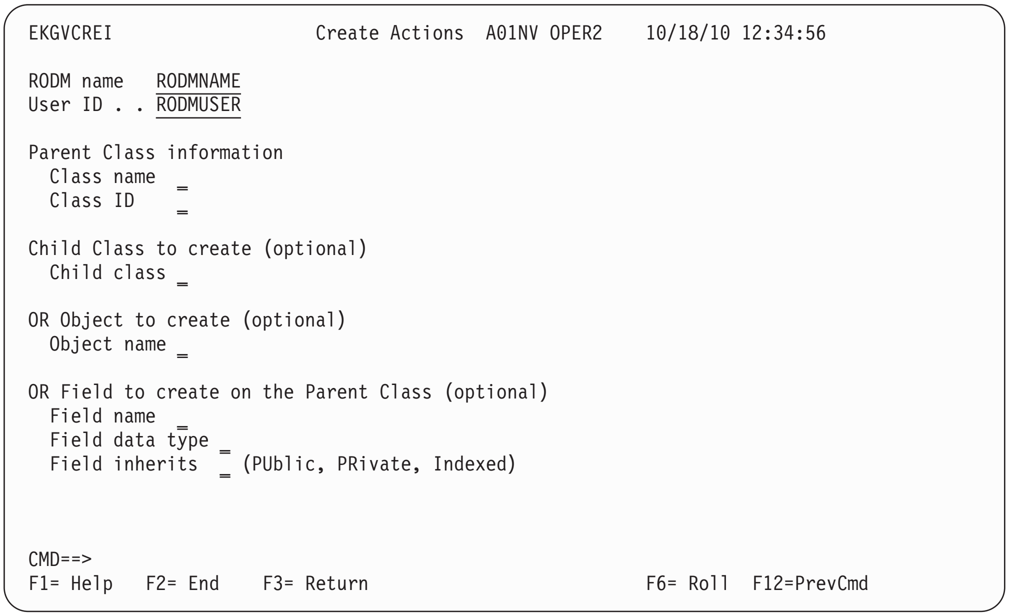
Notes:

1. You cannot use RODMView to change the value of a notify subfield, which is of the type MethodSpec.
2. Subfields must be created on the parent class of an object. The existence and initial contents of the subfield are inherited from the class to the object. For a Notify subfield, a null value is inherited.
3. Subfields cannot be deleted from class fields when that class has either class or object children.
4. A subfield must be deleted from the class on which it was defined.
5. The Notify, Prev_value, and Timestamp subfields cannot revert to an inherited value.

Create Actions Function

Use the Create Actions function to create classes, objects, or fields on classes. In each case, you must specify which class, called the *parent class*, you want to work with.

From the RODMView main menu, select **8. Create Actions**. The Create Actions panel is displayed as shown in Figure 130 on page 532.



```
EKGVCREI                   Create Actions  A01NV OPER2    10/18/10 12:34:56

RODM name   RODMNAME
User ID . . RODMUSER

Parent Class information
  Class name  _
  Class ID    _

Child Class to create (optional)
  Child class _

OR Object to create (optional)
  Object name _

OR Field to create on the Parent Class (optional)
  Field name  _
  Field data type _
  Field inherits  _ (PUblic, PRivate, Indexed)



CMD==>
F1= Help   F2= End    F3= Return                         F6= Roll  F12=PrevCmd
```

Figure 130. RODMView Create Actions Panel — EKGVCREI

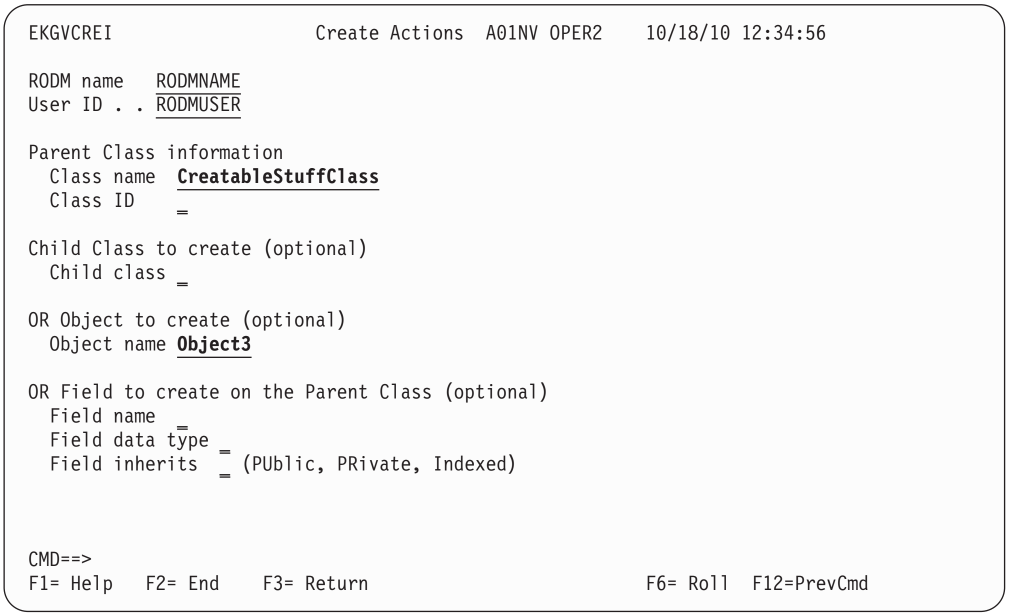
Table 223 lists the information that must be provided to create a child class, an object, or a field.

Table 223. Specifications to Create Entities.

To create this:	Fill in only these input fields:
Child Class	Class name or Class ID Child Class name
Object	Class name or Class ID Object name
Field	Class name or Class ID Field name or Field ID Field data type Field inherits

RODMView requests that RODM create the entity as specified on the panel. If RODM detects that you are trying to create something that is not possible (for example, create a field on an object) a message is displayed.

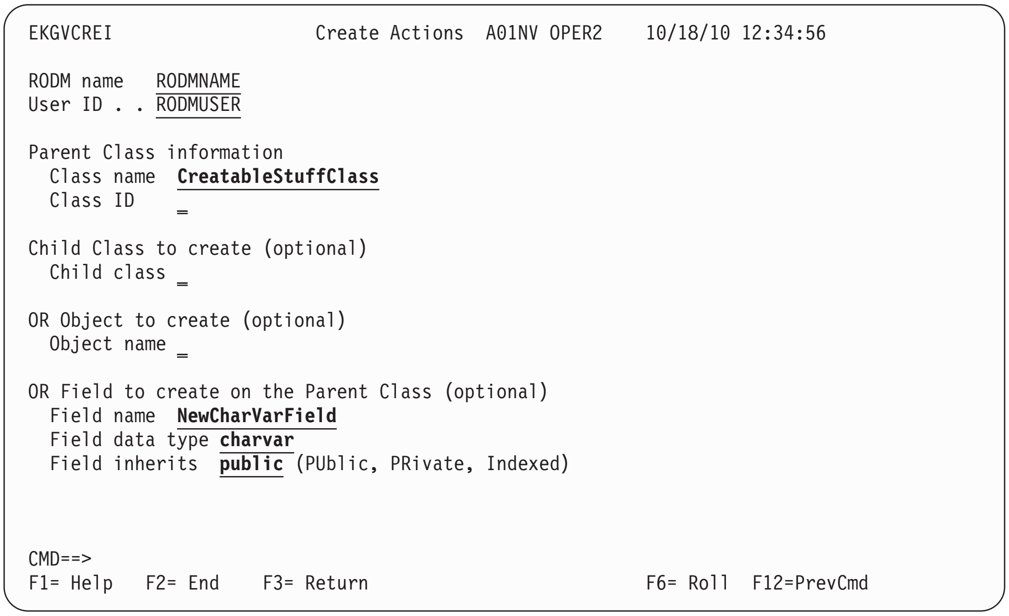
If you want to create an object on the `CreatableStuffClass` named Object3, enter the information on the Create Actions panel as shown in Figure 131 on page 533.



```
EKGVCREI                   Create Actions  A01NV OPER2    10/18/10 12:34:56

RODM name   RODMNAME
User ID . . RODMUSER

Parent Class information
  Class name  CreatableStuffClass
  Class ID    _

Child Class to create (optional)
  Child class _

OR Object to create (optional)
  Object name Object3

OR Field to create on the Parent Class (optional)
  Field name  _
  Field data type _
  Field inherits  _ (PUblic, PRivate, Indexed)



CMD==>
F1= Help   F2= End    F3= Return                          F6= Roll  F12=PrevCmd
```

Figure 131. RODMView Creating an Object

If you want to create a private field named NewCharVarField on the class `CreatableStuffClass`, enter the information in the Create Actions panel as shown in Figure 132. Note that no value is specified for the Object name field.



```
EKGVCREI                   Create Actions  A01NV OPER2    10/18/10 12:34:56

RODM name   RODMNAME
User ID . . RODMUSER

Parent Class information
  Class name  CreatableStuffClass
  Class ID    _

Child Class to create (optional)
  Child class _

OR Object to create (optional)
  Object name _

OR Field to create on the Parent Class (optional)
  Field name  NewCharVarField
  Field data type charvar
  Field inherits  public (PUblic, PRivate, Indexed)



CMD==>
F1= Help   F2= End    F3= Return                          F6= Roll  F12=PrevCmd
```

Figure 132. RODMView Creating a Field

Data in the Field data type and Field inherits input fields are ignored unless a field name has been specified to create them.

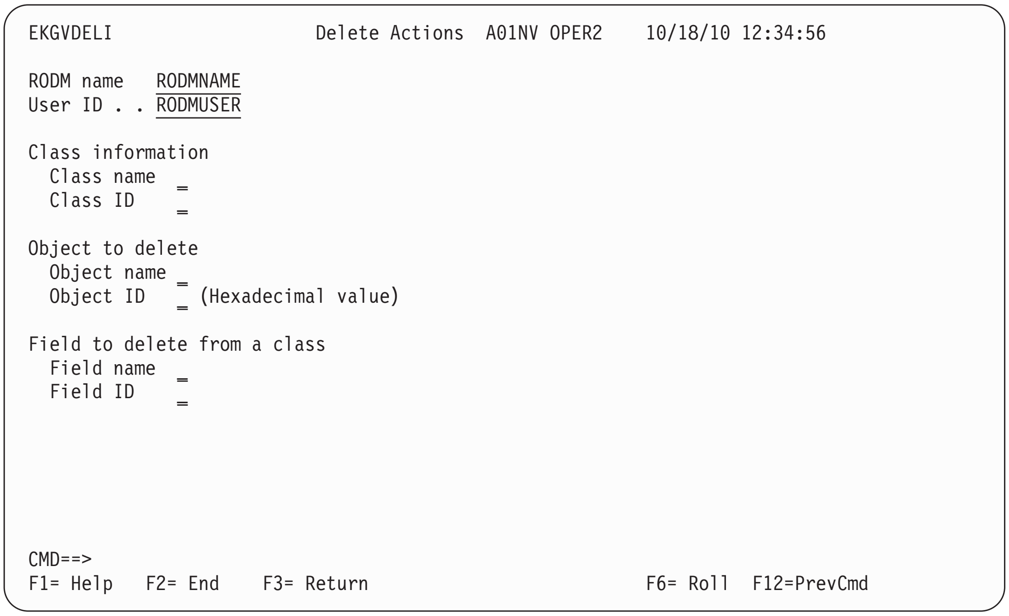
For the example shown in Figure 132, the only output from this request is the return and reason codes displayed on the message line.

Delete Actions Function

Use the Create Actions function to delete classes, objects, or fields on classes.

Delete Actions Function

From the RODMView main menu, select **9. Delete Actions**. The Delete Actions panel is displayed as shown in Figure 133.



```
EKGVDELI                   Delete Actions  A01NV OPER2    10/18/10 12:34:56

RODM name   RODMNAME
User ID . . RODMUSER

Class information
  Class name  _
  Class ID    _

Object to delete
  Object name _
  Object ID   _ (Hexadecimal value)

Field to delete from a class
  Field name  _
  Field ID    _






CMD==>
F1= Help   F2= End    F3= Return                          F6= Roll    F12=PrevCmd
```

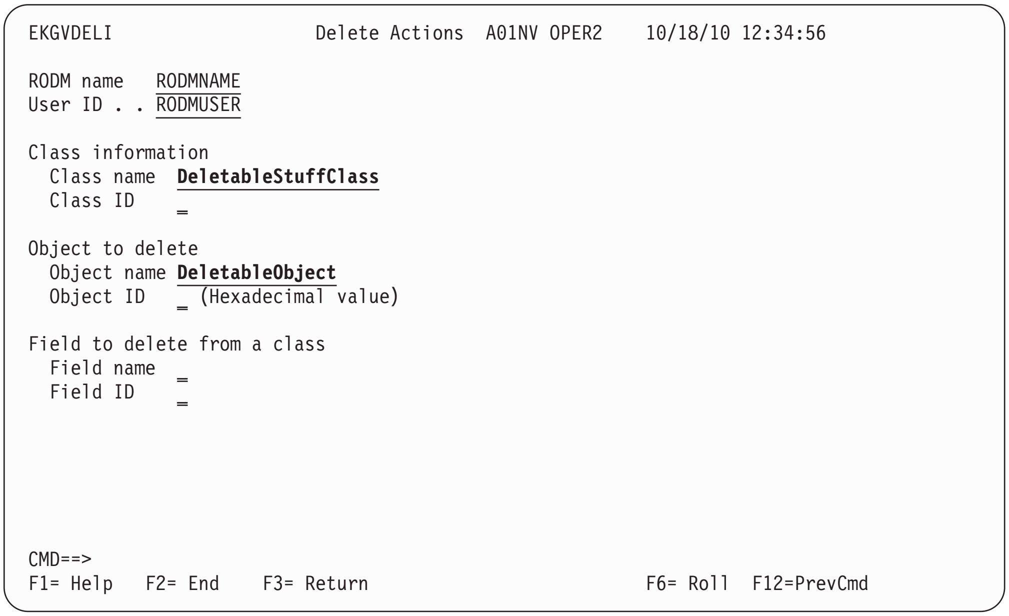
Figure 133. RODMView Delete Actions Panel — EKGVDELI

Table 224 lists the information that must be provided to delete a child class, an object, or a field.

Table 224. Specifications to Delete Entities.

To delete this:	Fill in only these input fields:
Class	Class name or Class ID
Object	Class name, Class ID, or Object name
Object	Object ID
Field	Class name, Class ID, Field name, or Field ID

If you want to delete an object named DeletableObject from the `DeletableStuffClass` class, enter the information on the Delete Actions panel as shown in Figure 134 on page 535.



```
EKGVDELI                Delete Actions  A01NV OPER2    10/18/10 12:34:56

RODM name   RODMNAME
User ID . . RODMUSER

Class information
  Class name  DeletableStuffClass
  Class ID    _

Object to delete
  Object name DeletableObject
  Object ID   _ (Hexadecimal value)

Field to delete from a class
  Field name  _
  Field ID    _






CMD==>
F1= Help   F2= End    F3= Return                           F6= Roll  F12=PrevCmd
```

Figure 134. RODMView Deleting a Field from a Class

Before RODMView sends the delete request, you are prompted to verify the delete request.

Notes:

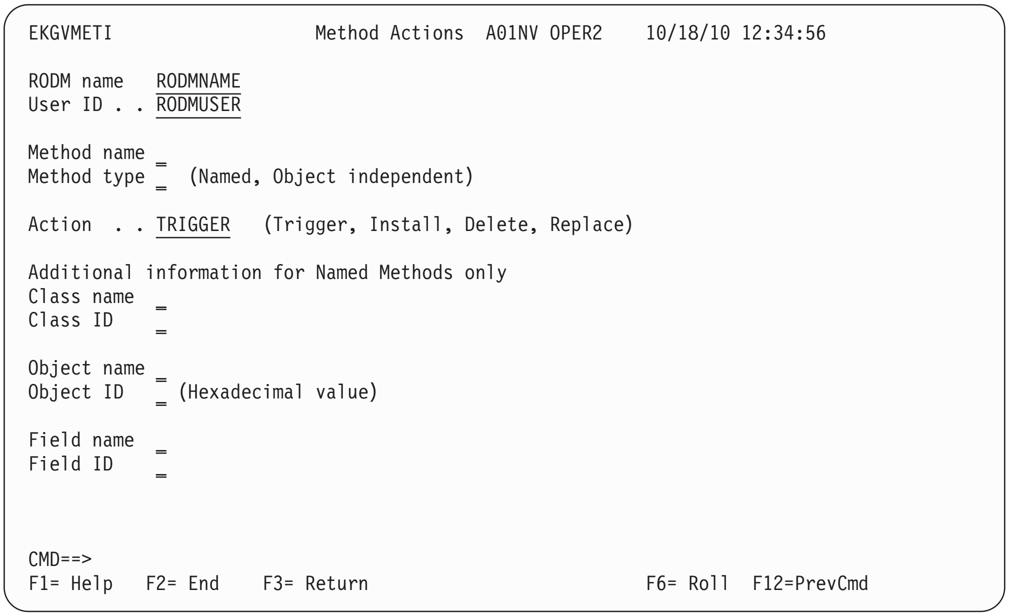
1. To delete a class, the class must not have class or object children.
2. To delete an object, the object must not contain links to other objects.
3. To delete a field from a class, that class can not have class or object children.
4. A field can not be deleted directly from an object. The field must be deleted from its parent class.

Method Actions Function

Use the Method Actions function to do the following:

- Trigger a method either as an object-independent or object-specific (named) method
- Install a method
- Delete a method
- Replace method code

From the RODMView main menu, select **10. Method Actions**. The Method Actions panel is displayed as shown in Figure 135 on page 536.

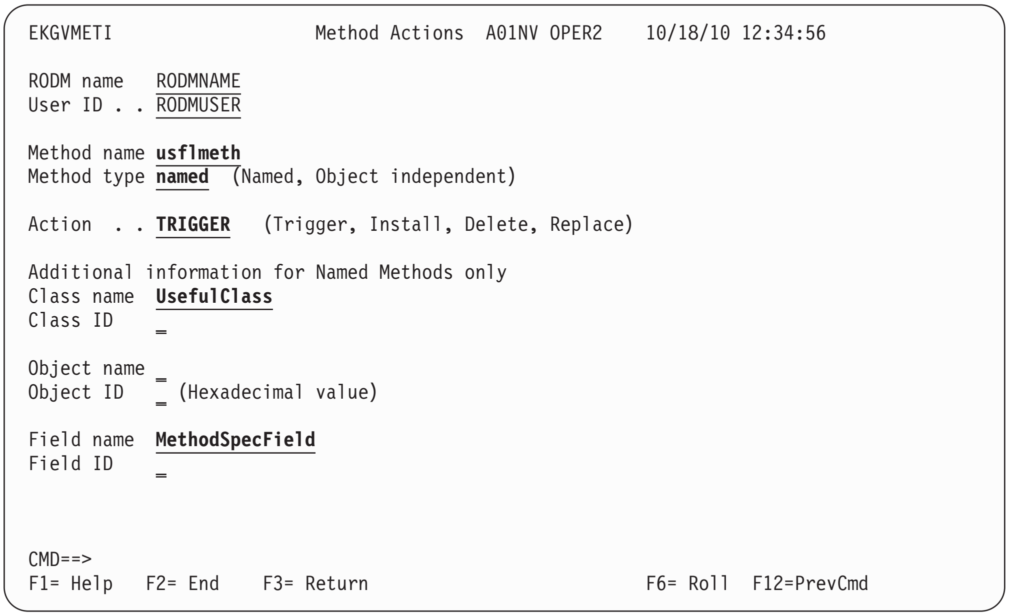


```
EKGVMETI                 Method Actions  A01NV OPER2    10/18/10 12:34:56

RODM name   RODMNAME
User ID . . RODMUSER

Method name _
Method type _  (Named, Object independent)

Action  . . TRIGGER   (Trigger, Install, Delete, Replace)

Additional information for Named Methods only
Class name  _
Class ID    _

Object name _
Object ID   _ (Hexadecimal value)

Field name  _
Field ID    _



CMD==>
F1= Help   F2= End    F3= Return                         F6= Roll  F12=PrevCmd
```

Figure 135. RODMView Method Actions Panel — EKGVMETI

Using RODMView, object-independent methods are run without short-lived parameters. Named methods, however, receive the short-lived parameters defined on the field (of data type MethodSpec) that you specify.

For example, assume that a field called MethodSpecField of type MethodSpec is defined on the class `UsefulClass`, and MethodSpecField has a value that includes a method called USFLMETH. To run the method, enter the information on the Method Actions panel as shown in Figure 136.

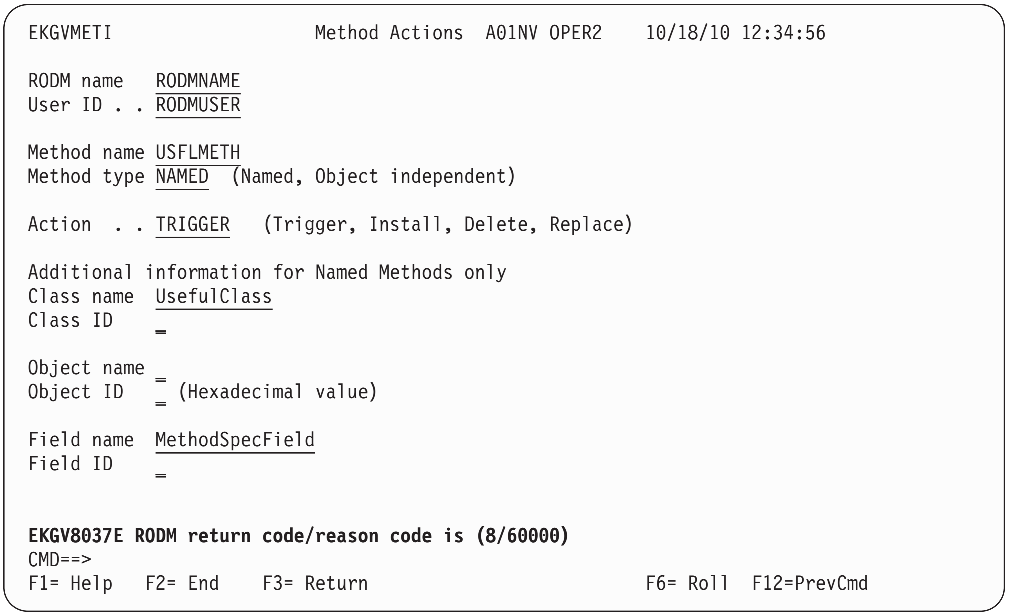


```
EKGVMETI                 Method Actions  A01NV OPER2    10/18/10 12:34:56

RODM name   RODMNAME
User ID . . RODMUSER

Method name usflmeth
Method type named  (Named, Object independent)

Action  . . TRIGGER   (Trigger, Install, Delete, Replace)

Additional information for Named Methods only
Class name  UsefulClass
Class ID    _

Object name _
Object ID   _ (Hexadecimal value)

Field name  MethodSpecField
Field ID    _



CMD==>
F1= Help   F2= End    F3= Return                         F6= Roll  F12=PrevCmd
```

Figure 136. RODMView Triggering a Named Method

The method USFLMETH is run with the short-lived parameters defined in the field MethodSpecField.

When the method has finished executing, the return and reason codes that RODMView displays on the message lines are from the method itself. The result of the example described is similar to the panel shown in Figure 137.



```
EKGVMETI                  Method Actions  A01NV OPER2    10/18/10 12:34:56

RODM name   RODMNAME
User ID . . RODMUSER

Method name USFLMETH
Method type NAMED  (Named, Object independent)

Action  . . TRIGGER   (Trigger, Install, Delete, Replace)

Additional information for Named Methods only
Class name  UsefulClass
Class ID    _

Object name _
Object ID   _ (Hexadecimal value)

Field name  MethodSpecField
Field ID    _


EKGV8037E RODM return code/reason code is (8/60000)
CMD==>
F1= Help   F2= End    F3= Return                          F6= Roll  F12=PrevCmd
```

Figure 137. RODMView Return and Reason Codes From a Triggered Method

In the prior example, the method that was triggered was user-written. After the method completes, it issues the return/reason code combination 8/60000. This combination is not translated into a specific RODMView message; therefore, RODMView displays the following message:

`EKGV8037E RODM return code/reason code is (`*`return_code/reason_code`*`)`

Note: The method name in Figure 136 on page 536 was typed in lowercase, but when the RODMView panel is refreshed in Figure 137, the method name is converted to uppercase. While it is true that the RODM-defined null method NullMeth has uppercase and lowercase letters in its name, all methods that exist as code in RODM must have uppercase names. RODMView automatically translates method names to uppercase.

RODM Unload Function

The RODM unload function can be used to unload classes, objects, and fields. For example, the RODM unload function can be used to migrate from one version of RODM to another by unloading an existing RODM and loading the newer version of RODM with the output from the RODM unload function.

The RODM unload function queries the class structure of RODM in a depth-first manner. For each class, a RODM high-level syntax statement is written to create the class along with its unique fields. All class-level creation statements are written to the CLASSES file. If any class field contains a locally defined value, that value is written to the CLASSVAL file.

The RODM unload function does not unload the values of system-defined fields on the system classes (UniversalClass and all EKG*xxxx* classes). If the RODM unload function finds a user-defined field, it writes a primitive to create the field, and a primitive to assign the field a value if a non-null value currently exists.

While unloading a class, a check is made to see if it has any object children. Each object child is in turn examined, and a RODM low-level primitive is written to the OBJECTS file to create it. All data contained in fields that have local values are written to the OBJVAL file.

To ensure that unloaded data sets load properly again, they must be concatenated in the RODM load function EKGIN3 statement in the following order:

1. CLASSES
2. OBJECTS
3. CLASSVAL
4. OBJECTVAL
5. LINKS

This order ensures that no data contained in subfields refers to something that has not been loaded.

Using the data set scheme as detailed in the sample EKGKUJCL, the EKGIN1 DD concatenation of the RODM load function that runs JCL shows as follows in Figure 138.

```
//EKGIN1     DD DSN=EKG.RODMUNLD.CLASSES,DISP=SHR
//           DD DSN=EKG.RODMUNLD.OBJECTS,DISP=SHR
//           DD DSN=EKG.RODMUNLD.CLASSVAL,DISP=SHR
//           DD DSN=EKG.RODMUNLD.OBJVAL,DISP=SHR
//           DD DSN=EKG.RODMUNLD.LINKS,DISP=SHR
```

Figure 138. Sample JCL for EKGIN1

Data types FieldID and Anonymous(*N*) cannot be unloaded using the RODM unload function.

The RODM unload function operates on the premise that RODM data is static and unchanging. RODM data might change while the RODM unload function is running. If this happens, the unloaded data sets might contain data that is inconsistent with the current RODM data. Therefore, run the RODM unload function at periods of low RODM activity.

Starting the RODM Unload Function

Submit job EKGKUJCL to start the RODM unload function.

Customizing the RODM Unload Function

This section contains the information that is needed to customize the RODM unload function.

1. Customize the EKGKUCDS job.

   The EKGKUCDS job allocates the output data sets for the RODM unload function. Edit the NETVIEW.V6R1M0.CNMSAMP (EKGKUCDS) job to indicate the location for the output data sets.
2. Run EKGKUCDS to allocate the RODM unload function output data set.
3. Modify the EKGKUJCL job.

   Modify the parameters as required by your installation. This job is found in the NETVIEW.V6R1M0.CNMSAMP data set.

The RODM unload function is run with JCL. Input parameters are passed to the RODM unload function in a file named by the SYSIN DD file of the JCL. Figure 139 on page 539 contains a section from the sample JCL. For simplicity, the

SYSIN DD file is placed in-line with the JCL.

```
 ...
//SYSIN     DD *
  RODM=
  CLASS=
  OBJECT=
  DEPTH=
  REPORTONLY=
  WRITEMODE=
  WHITESPACE=
 ...
```

Figure 139. Sample SYSIN DD file of the JCL.

Table 225 contains a description of the SYSIN DD parameters.

Table 225. SYSIN DD Parameter Descriptions

Parameter	Description
RODM	Specifies the name of the RODM to unload. This is usually the same as the z/OS procedure used to start RODM.
CLASS	• Specifies a class from which the unloading process is started. • If left blank, the UniversalClass is the starting point. • Multiple classes can be specified by repeating the parameter on multiple lines, specifying one class per line. • This parameter is case sensitive.
OBJECT	• Specifies a specific object to unload. • Multiple objects can be specified by repeating the parameter on multiple lines, specifying one object per line. • If left blank or omitted, all objects are unloaded. • This parameter is case sensitive.
DEPTH	• Specified as either ALL or ONE. • If DEPTH=ALL, the classes specified on the CLASS= parameters and all classes that descend from them are unloaded. • If DEPTH=ONE, only the individual classes specified on the CLASS= parameters are unloaded.
REPORTONLY	• Can be specified as either YES or NO. • If REPORTONLY=YES, a summary report of all classes, objects, fields, and links defined are produced, but no RODM load function compatible output is actually produced. This is useful for extracting current capacity information of a RODM. • If REPORTONLY=NO, the RODM load function compatible output is produced along with this summary report.
WRITEMODE	• Can be specified as either APPEND or OVERWRITE. • If WRITEMODE=APPEND, all output generated is appended to the end of the data sets specified in the start JCL. • If WRITEMODE=OVERWRITE, any data that previously existed in the data sets is destroyed, and any new output created by the RODM unload function is written in its place.

Table 225. SYSIN DD Parameter Descriptions (continued)

Parameter	Description
WHITESPACE	• This specifies the level of whitespace (blank lines) to be mixed in with the RODM load function compatible output. • Can be specified as either LOW or HIGH. Specifying WHITESPACE=HIGH gives the most readable output, but WHITESPACE=LOW reduces the lines of total output by approximately half. • The actual data content of the output is identical with either LOW or HIGH.

The 5 output data sets are specified in the JCL. The output data sets and content follow:

CLASSES	Contains the class structure creation high-level syntax
CLASSVAL	Contains the class subfield creation and value-setting primitives
OBJECTS	Contains the object-creation primitives
OBJVAL	Contains the object subfield value-setting primitives
LINKS	Contains the link primitives

The RODM unload function reads the DCB specifications of the data sets from the JCL and modifies itself. Use the DCB specifications in the sample as supplied. The RODM unload function always produces output that is a maximum of 80 characters wide, even if a wider DCB is specified.

Start the RODM unload function by running the EKGKUJCL job.

Running the RODM Unload Function

The RODM unload function can be used to migrate from one version of RODM to another. This is accomplished by unloading an existing RODM and loading the newer version of RODM with the output from the RODM unload function. To perform a complete unload of RODM, change the SYSIN parameters in the EKGKUJCL job as shown in Figure 140 and run the job. Note that the `OBJECT=` parameter has been deleted from the sample JCL.

```
RODM=(rodmname)
CLASS=UniversalClass
DEPTH=All
REPORTONLY=No
WRITEMODE=Overwrite
WHITESPACE=Low
```

Figure 140. EKGKUJCL SYSIN Parameters to Unload RODM Completely

To unload all the objects that represent network monitorable (real and aggregate) resources in the GMFHS data model, the SYSIN parameters to EKGUJCL are changed as shown in Figure 141 on page 541.

```
RODM=(rodmname)
CLASS=GMFHS_Monitorable_Objects_Class
DEPTH=All
REPORTONLY=No
WRITEMODE=Overwrite
WHITESPACE=Low
```

Figure 141. EKGKUJCL SYSIN Parameters to Unload Network Monitorable Objects

To get the RODM definitions for a particular object, when the class of the object is not known, change the SYSIN parameters EKGKUJCL job as shown in Figure 142.

```
RODM=(rodmname)
CLASS=UniversalClass
OBJECT=DesiredObject
DEPTH=All
REPORTONLY=No
WRITEMODE=Overwrite
WHITESPACE=High
```

Figure 142. EKGKUJCL SYSIN Parameters to Unload an Object When Class is Unknown

If the class that the object is defined under is known, it saves processing time to specify that class directly. Set the `CLASS=`, `OBJECT=` and the `DEPTH=` parameters as shown in Figure 143.

```
RODM=(rodmname)
CLASS=SpecificClass
OBJECT=DesiredObject
DEPTH=One
REPORTONLY=No
WRITEMODE=Overwrite
WHITESPACE=High
```

Figure 143. EKGKUJCL SYSIN Parameters to Unload an Object When Class is Known

To get the RODM definitions for all objects in two particular classes only, change the parameters in the EKGKUJCL job as shown in Figure 144.

```
RODM=(rodmname)
CLASS=SpecificClass1
CLASS=SpecificClass2
DEPTH=One
REPORTONLY=No
WRITEMODE=Overwrite
WHITESPACE=Low
```

Figure 144. EKGKUJCL SYSIN Parameters to Determine Object Definitions for Two Classes

FLCARODM

FLCARODM (RODM Access Facility) provides a fast and efficient REXX interface to RODM. (FLCARODM was formerly known as the RODM Access Facility or MultiSystem Manager Access.) With FLCARODM, you can create, update, and delete objects using a NetView CLIST written in REXX. FLCARODM provides a simple interface to RODM and you can use it to exploit the processing advantages of issuing batched requests to RODM.This section describes how to use FLCARODM.

The following topics are covered:
- Using stem building routines
- The FLCARODM command

- FLCARODM functions
- The result stem
- The object data stream

Overview

FLCARODM provides a REXX interface to the RODM user application programming interface (UAPI). FLCARODM performs multiple operations on one or more objects in a single invocation and removes many of the complexities of using the RODM UAPI. Use this high speed interface to create, update, query, locate, and delete objects in RODM.

The two ways to use FLCARODM are as follows:
- Specify the data and operations using a low-level data stream. See “Object Data Stream Detail” on page 581 for more information.
- Use the stem building subroutines that are provided by NetView to create a REXX stem variable.

Stem Building Subroutines

This section describes the subroutines that are provided to create the REXX object data stream in a REXX stem variable. These subroutines are called **stem building subroutines**, and they create the contents of a REXX stem variable that gets passed to FLCARODM using the FLCARODM command.

The stem building subroutines are provided in the FLCSSTEM sample. You can use FLCSSTEM by appending it to your procedure using an INCLUDE statement such as the following statement:

```
%INCLUDE FLCSSTEM
```

To use an INCLUDE statement, you must also enable the NetView program support of %INCLUDE by coding `/*%NETVINCL` at the beginning of the first line of your procedure.

Note: Instead of using an INCLUDE statement, you can copy the code from the FLCSSTEM sample into your procedure.

The subroutines that are provided manipulate REXX stem variables that are used with FLCARODM. These subroutines manipulate the following stem variables:
- RodmStem which is used as input to FLCARODM
- RodmResult which is used to hold the output from FLCARODM
- QueryStem which is used to hold queried information extracted from RodmResult

A variable called Retcode is used by all of the subroutines to indicate if any errors have occurred. A non-zero value in the Retcode variable indicates that processing stops.

FLCARODM supports class, object, and field IDs in the input stem variable. To specify a numeric ID instead of a name, prefix the ID with a #. For example, if you knew an object's class ID was 12, you can specify an element of the input stem variable as `input.x = '#12'`.

AddAttr Subroutine

Use the AddAttr subroutine to specify a new or existing field on the current object.

Specification:

`call AddAttr` *`fieldname fieldtype fieldvalue`*

Operand Descriptions: Where:

`fieldname`
The name of the field

`fieldtype`
The data type of the field

`fieldvalue`
The new or changed value of the field

Usage Notes:

- Use AddAttr with the BUILD and UPDATE functions.
- AddAttr must be specified before Addlink

Example: The following code from sample FLCSX7 calls the AddAttr subroutine that creates a field named DispStat that is of type Integer and that has a value of InActive:

```
call AddAttr DispStat Integer InActive
```

Note: DispStat is a shortened version of DisplayStatus that is defined in the FLCSSTEM sample by using the following assignment statement:

```
DispStat  = 'DisplayStatus'
```

AddAttrForQuery Subroutine

Use the AddAttrForQuery subroutine to specify either the field to be queried using the QUERY function, or the name of the first field when a function is specified with the XREF=1STFIELD parameter.

Specification:

`call AddAttrForQuery '`*`fieldname`*`'`

Operand Descriptions: Where:

`fieldname`
The name of the field to query or the name of the field referred to by the XREF=1STFIELD parameter

Usage Notes:

- Use the AddAttrForQuery subroutine with the QUERY function, or with the following functions when they are specified with the XREF=1STFIELD parameter.
  - DELINKA
  - DELOBJ
  - QUERY
  - UPDATE

Example: The following code from sample FLCSXS02 calls the AddAttrForQuery subroutine to specify four fields on the RealAgent object of the RAgeClass that are queried:

```
call StartObject RAgeClass RealAgent
call AddAttrForQuery MyName
call AddAttrForQuery DispName
call AddAttrForQuery RealAgeNam
call AddAttrForQuery RealSerNam
call MakeRODMCall 'QUERY'
```

The following code from sample FLCSX19 calls the AddAttrForQuery subroutine to specify two fields on the Demo_Lan object of the AGrphClass class that are used to identify object links that are to be removed:

```
call StartObject AGrphClass 'Demo_Lan'

call AddAttrForQuery  Member
call AddAttrForQuery  PhyConn
call MakeRODMCall 'DELINKA' 'XREF=1STFIELD'
```

`AddAttrForQuery  Member` specifies that all objects specified by the Member field are identified and `AddAttrForQueryPhyConn` specifies that all links specified by the PhyicalConPP field are removed.

AddLink Subroutine

Use the AddLink subroutine to specify a field to link to. The field must be one of the following data types:

- ObjectLink
- ObjectLinkList
- ObjectIdList

Specification:

```
call AddLink 'linkfldname' 'classofobj' 'nameofobj' 'fldofobj'
```

Operand Descriptions: Where:

linkfldname
The name of the field to be linked to

classofobj
The class of the object to be linked to

nameofobj
The name of the object to be linked to

fldofobj
The field on the object to be linked to

Usage Notes:

- Calls to the AddAttr subroutine must be specified before call to AddLink are specified

Example: The following code from sample FLCSX11 uses the AddLink subroutine to specify the PhysicalConnPP field of the Bridge_1 object and the PhysicalConnPP fields of the Segment_1 and Segment_2 objects. The DELINKAB function removes the links defined by the PhysicalConnPP fields.

```
call StartObject ABrgClass 'Bridge_1'

call AddLink PhyConn ASegClass 'Segment_1' PhyConn
call AddLink PhyConn ASegClass 'Segment_2' PhyConn

call MakeRODMCall 'DELINKAB'
```

AddLinkForDelete Subroutine

Use the AddLinkForDelete subroutine to specify a link on the specified object.

Specification:

```
call AddLinkForDelete fldname
```

Operand Descriptions: Where:

fldname
The name of the field on the specified object that defines the link that is to be deleted.

Example: The following code from sample FLCSX10 calls the AddLinkForDelete subroutine that specifies the PhysicalConnPP on the object of the ABrgClass class named Bridge_1. The DELINKA function removes the links defined by the PhysicalConnPP field.

```
 call StartObject ABrgClass 'Bridge_1'

 call AddLinkForDelete PhyConn
 call MakeRODMCall 'DELINKA'
```

CheckChildrenUpdate Subroutine

Use the CheckChildrenUpdate subroutine to remove acceptable return codes from the RodmResult stem variable when either the UPDATE or DELINKA function is specified with the CHILDREN=ONLY parameter.

Acceptable return codes indicate one of the following:
- An aggregate object does not exist.
- Child objects do not exist.
- Specified fields do not exist on the child object.

For unacceptable return codes:
- Message FLC070E is issued.
- The return codes are written to the log.
- The Retcode stem variable is set to 16.

Specification:

```
call CheckChildrenUpdate
```

- Use this subroutine only when you specify the UPDATE and DELINKA functions with the CHILDREN=ONLY parameter. Combinations of other functions and parameters are not supported.

CheckDelinkResponse Subroutine

Use the CheckDeLinkResponse subroutine to remove acceptable return codes from the RodmResult stem variable when either the DELOBJ or DELINKA function is specified.

Acceptable return codes indicate one of the following:

- An aggregate object does not exist.
- Child objects do not exist.
- Specified fields do not exist on the child object.

For unacceptable return codes:
- Message FLC070E is issued.
- The return codes are written to the log.
- The Retcode stem variable is set to 16.

Specification:

```
call CheckDelinkResponse
```

Usage Notes:

- Use this subroutine only when you specify the DELOBJ and DELINKA functions. Other functions are not supported.

InitRODMConstants Subroutine

Use the InitRODMConstants subroutine to initialize the constants specified in the FLCSSTEM sample.

Specification:

```
call InitRODMConstants
```

Usage Notes:

- You must read the code to see what variables are available for your use.

InitRODMStem Subroutine

Use the InitRODMStem subroutine to initialize the RODMStem variable.

Specification:

```
call InitRODMStem
```

Usage Notes:

- Specify InitRODMStem the first time you use FLCSSTEM. Subsequent calls to InitRODMStem are not required, because the MakeRODMCall subroutine calls InitRODMStem.

MakeRODMCall Subroutine

Use the MakeRODMCall subroutine to issue the FLCARODM command with the RODMStem variable as input.

Specification:

`call MakeRODMCall` *`function functparm1 functparm2`*

Operand Descriptions: Where:

`function`
Specifies the function to be performed. See “FLCARODM Functions” on page 553 for more information.

`functparm1`
Specifies the first function parameter.

functparm2
Specifies the second function parameter.

Example: The following code from sample FLCSXF1 calls the QUERY subroutine with the XREF and FILTER parameters.

```
call MakeRODMCall 'QUERY' 'XREF=2.9.3.2.7.42' 'FILTER=1STFIELD'
```

SetIndexList Subroutine

Use the SetIndexList subroutine to update the value of fields that are of type IndexList.

Specification:

`call SetIndexList` *fieldvalue fieldname*

Operand Descriptions: Where:

fieldvalue
Specifies the value of the field.

fieldname
Specifies the name of the field.

Usage Notes:

- Use SetIndexList to update the value of fields that are only of type IndexList.
- Use caution when using the SetIndexList function, because the value of the field is overwritten and the previous value cannot be recovered.

Example: The following code from sample FLCSX22 calls the SetIndexList subroutine to modify the ExceptionViewList field on the Demo_Lan object:

```
call StartObject AGrphClass 'Demo_Lan'

my_String = 'testing'
call SetIndexList my_String ExceptionViewList

call MakeRODMCall 'UPDATE'
```

StartObject Subroutine

Use the StartObject subroutine to specify a new or existing object. Subsequent subroutine specifications (for example, AddAttr) apply to the current object until either another object is specified by StartObject, or the MakeRODMCall subroutine is specified.

Specification:

`call StartObject` *classname objectname*

Operand Descriptions: Where:

classname
The name of the class for the object that is specified.

objectname
The name of the object that is specified.

Usage Notes:

- Classes cannot be created using StartObject.
- Use the StartObject subroutine with all of the FLCARODM functions.

- Object names must be specified between single quotation marks (' ').

Example: The following code from sample FLCSX09 calls the StartObject subroutine which creates an object of the AGrphClass named Demo_Lan:

```
call StartObject AGrphClass 'Demo_Lan'
```

Note that if no object named Demo_Lan exists when sample FLCSX09 is run, a new object is created. If an object named Demo_Lan already exists, the existing object is used.

About the Examples

The examples used in this appendix are provided by the NetView Product as sample code. Although the examples use the MultiSystem Manager and GMFHS data models, FLCARODM supports any data model that is loaded in RODM.

The examples create stem variables that are used as input to the FLCARODM command. The statement *call MakeRODMCall **function*** calls the FLCARODM command using the function specified. For example, the following statement issues the FLCARODM command with the BUILD function.

```
call MakeRODMCall 'BUILD'
```

Using the Samples

To use the sample code provided by the NetView product, perform the following tasks:

- In your REXX procedure, set a variable FLC_RODMNAME to the name of the RODM you are using.
- In your REXX procedure, set a variable FLC_RODMAPPL to your RODM application ID.
- Include the subroutines in FLCSSTEM by using an %INCLUDE statement in your REXX procedure immediately after your own subroutines. To use an INCLUDE statement, you must also enable the NetView program support of %INCLUDE by coding `/*%NETVINCL` at the beginning of the first line of your procedure. FLCSSTEM provides the subroutines and constant definitions that are used by the samples.

  Note: Instead of using an INCLUDE statement, you can copy the code from the FLCSSTEM sample into your procedure.

FLCARODM Command

Use the FLCARODM command to input data into and read data from RODM.

The FLCARODM command must be issued using the NETVIEW stage of the NetView PIPE command. Therefore, it receives information about the functions to be performed from two sources: the PIPE data stream and the parameters the command is issued with. Figure 145 shows an example of issuing the FLCARODM command:

```
PIPE STEM object_data
  | COLLECT
  | NETVIEW FLCARODM parameters
  | stem result
```

Figure 145. Issuing the FLCARODM Command

Where:

object_data
The REXX stem variable that is used as input.

parameters
The parameters of the FLCARODM command

result The REXX stem variable that receives the return codes or data from FLCARODM.

Use the format shown in Figure 145 on page 548 when you are specifying data using the object data stream described in “Object Data Stream Detail” on page 581.

The NetView product also provides another way to use the FLCARODM command. Instead of specifying the command directly, use the MakeRODMCall subroutine. See “Stem Building Subroutines” on page 542 for a description of the MakeRODMCall subroutine and the other subroutines you can use to create a REXX object data stream.

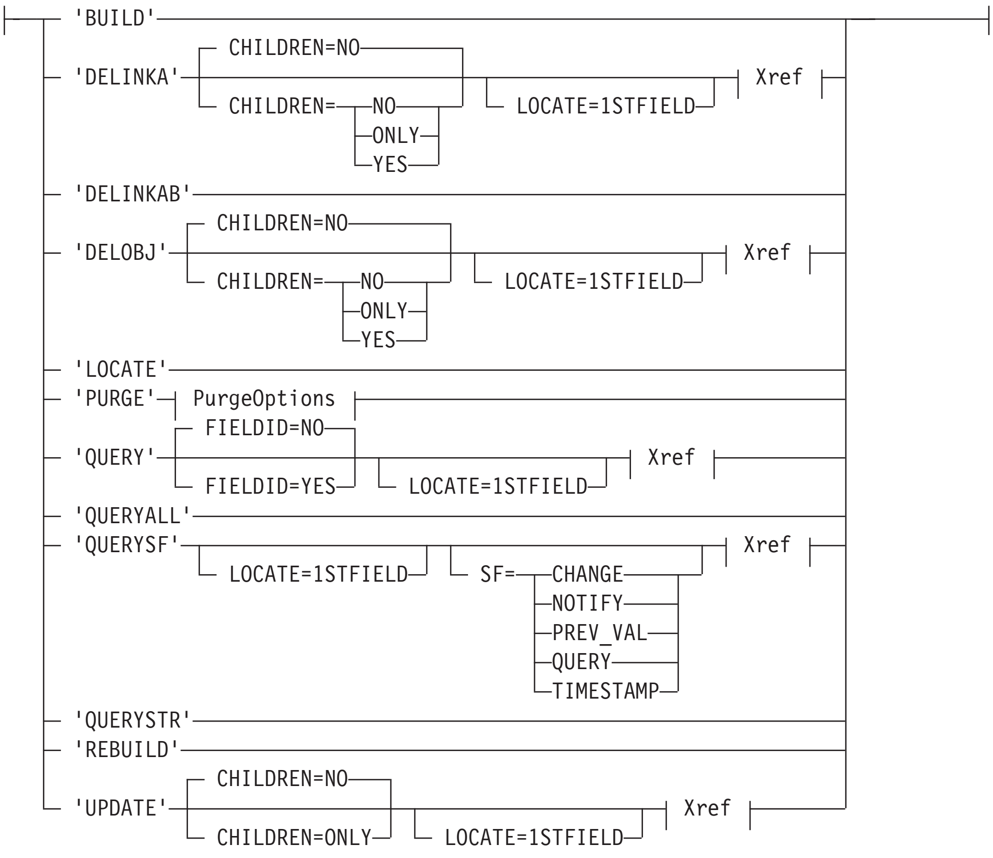
The following section describes the format of the FLCARODM command. The description includes the format and description of the operands and usage notes.

FLCARODM

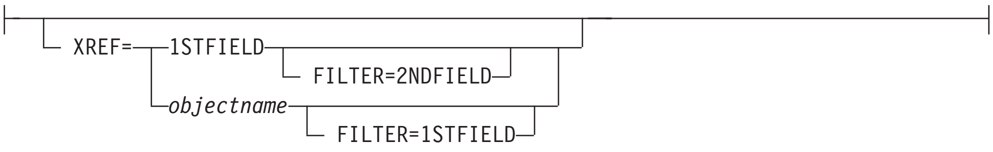
Syntax:

```
►►──FLCARODM── RODMNAME=name── RODMUSER=user── FUNCTION=─┤ FLCARodmFunctions ├────►

►─┬──────────────────────┬─┬─────────────────────────────────┬────────────────────►◄
  └─ RODMINT=interval──┘ └─ RODMRETRY=number_retries──┘
```

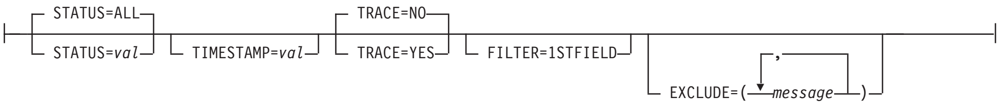
FLCARodmFunctions:



Xref:



PurgeOptions:



Operand Descriptions:

CHILDREN

Specifies whether the operation applies to the specified children of the object. The CHILDREN parameter cannot be specified if the XREF parameter is specified.

Use the CHILDREN parameter with the following functions:
- UPDATE
- DELINKA
- DELOBJ

NO Indicates that the function is performed on the specified object, but not on its children.

ONLY
Indicates that the function is performed on the specified children of the object, but not on the object itself.

YES
Indicates that the function is performed on the object specified and its children.

Notes:

1. YES is not valid with the UPDATE function.
2. For the UPDATE function, only the first level of children is updated.

EXCLUDE
Used with the PURGE function and can only be specified when TRACE=YES is specified. The EXCLUDE option indicates which purge messages (FLC040I, FLC041I, and FLC042I) must not be issued during purge processing. If you attempt to purge an aggregate object that has many objects beneath it, you might want to receive the FLC040I and FLC042I successful purge messages and suppress the FLC041I unsuccessful purge messages (for example, EXCLUDE=FLC041I). Otherwise, you might receive many unwanted FLC041I messages. One to three of these purge messages can be specified. No other messages are permitted.

FIELDID

Indicates whether the QUERY function returns field identifiers with the field names.

NO Indicates that the field identifiers are not returned.

YES
Indicates that the field identifiers are returned.

FILTER
Used with the XREF parameter to filter the list of objects that are operated on.

Use the FILTER parameter with the following functions:
- DELOBJ
- DELINKA
- PURGE
- QUERY
- QUERYSF
- UPDATE

The XREF parameter must be specified before the FILTER parameter is specified except for the PURGE function. For the PURGE function, FILTER can be specified without the XREF parameter.

The first field on each object specification must be the field name, type, and value of the filter criteria. The FILTER value is applied only after all other functions and parameters have been processed. FILTER returns values that are either exact matches or partial matches. For example, if the field value Segment is specified and an object exists that has the value Seg, the filter matches and the object is returned.

`FILTER=1STFIELD` must be specified unless `XREF=1STFILELD` is specified. If `XREF=1STFILELD` is specified, `FILTER=2NDFIELD` must be specified. The field description must specify the following information in the order shown:
1. Field name

2. Field data type
3. Field value

FUNCTION
Specifies the function that is performed. For a description of each function, see "FLCARODM Functions" on page 553.

LOCATE
Specifies that the first field definition is used as the criteria to create a list of objects.

`LOCATE=1STFIELD` must be specified, and the first field description must specify the following information in the order listed:
1. Field name
2. Field data type
3. Field value

Use the LOCATE parameter with the following functions:
- DELINKA
- DELOBJ
- QUERY
- QUERYSF
- UPDATE

RODMINT
The amount of time in seconds that FLCARODM waits between retrying requests when RODM is checkpointing. The default value is five seconds.

RODMRTRY
The number of times FLCARODM retries a request when RODM is checkpointing. The default value is three. If RODM is still checkpointing after FLCARODM has retried the request for the number of times specified, an error is returned to the application.

RODMNAME
The name of the RODM to be used.

RODMUSER
The application name that is used to connect to RODM. The same RODMUSER value can be used by multiple NetView operators executing REXX programs that call FLCARODM. However, Access cannot use the same RODMUSER value as other applications (for example, RODMVIEW) that connect to RODM.

Create a RODMUSER value by concatenating the NetView domain name with a three-character identifier. For example, MultiSystem Manager concatenates the NetView domain name CNMO1 with MultiSystem Manager to create the RODMUSER value. For example, if the NetView domain name is CNM01, MultiSystem Manager creates a RODMUSER value of CNM01MSM.

SF Indicates the subfield to be queried. Specify one of the following values:
- CHANGE
- NOTIFY
- PREV_VAL
- QUERY
- TIMESTAMP

STATUS
The DisplayStatus field value used to determine whether objects are purged by the PURGE function.

ALL
Indicates that an object are purged regardless of its DisplayStatus value. The TIMESTMP parameter cannot be specified when STATUS has a value of `ALL`.

val
The DisplayStatus field value of the objects that are to be deleted. The default value is 132 (unknown).

TIMESTAMP
The age criteria, specified in seconds, of objects to be purged. The default is 84400, which is the number of seconds in 24 hours.

TRACE
Specifies whether the PURGE function is run in trace mode. In trace mode, a message is issued for every object that is purged.

NO Indicates that the PURGE function is not run in trace mode.

YES
Indicates that the PURGE function is run in trace mode.

XREF
Specifies that a function is performed on a list of dynamically acquired objects. The list of objects is defined by the field that is specified. The field must be one of the following data types:

- ObjectIdList
- ObjectLink
- ObjectLinkList

Use the XREF parameter with the following functions:

- DELINKA
- DELOBJ
- QUERY
- UPDATE

The XREF parameter cannot be specified if the CHILDREN parameter is used.

Because the XREF parameter can contain mixed-case characters, `ADDRESS NETVASIS` must be specified.

1STFIELD
Specifies that the first field that is defined on an object is used.

objectname
Indicates the name of the field that is used. For objects that have dotted decimal notation names, you must use the dotted decimal name. For example, to specify the member field you must specify 2.9.3.2.7.42.

FLCARODM Functions

This section describes the functions provided by the FUNCTION parameter of the FLCARODM command.

The following information is provided for each function:

- A description of each function and when to use it.
- An example based on a set of samples that are provide by MultiSystem Manager.
- The results of the function are described, if applicable.

For information about using the samples described in this section, see “About the Examples” on page 548.

BUILD Function

Use the BUILD function to perform the following functions:
- Create new objects
- Modify existing objects
- Create fields and assign field values
- Define relationships between objects

The following data types are supported by the BUILD and UPDATE functions:

Date Type	Data Type Identifier
CHARVAR	4
INTEGER	10
SELFDEFINING	19
SMALLINT	21
FIELDID	26
ANONYMOUSVAR	30

The following code from sample FLCSX1 demonstrates how to use the BUILD function to create objects in RODM:

⋮

```
/*********************************************************************/
/* Start the first object.  This is the top object and is of type    */
/* Network_View_Class.  Its name is Hometown                       */
/*********************************************************************/
call StartObject NetClass 'Hometown'
                                  /* Start creating Hometown object  */

call AddAttr Annotate CharVar 'This is the Hometown City View'
                                  /* Add an Annotation or label      */

/*********************************************************************/
/* Add a second object to the list.  This object are inside the  */
/* Hometown class.  It is called Main_Street, is of type             */
/* GMFHS_Aggregate_Objects_Class.                                    */
/*********************************************************************/
call StartObject AggClass 'Main_Street'

     /********************************************************/
     /* Now add a label which says 'Constructed in 1889 to   */
     /* the object.                                          */
     /********************************************************/
call AddAttr DispOther CharVar 'Constructed in 1889'

     /********************************************************/
     /* Add a link to the object which tells the Display     */
     /* ResourceType and Display_Resource_Type_Class are     */
     /* linked to the DUIXC_RTN_HOST_AGG                     */
     /********************************************************/
call AddLink DispType DispClass HtAgg_Icon 'Resources'

     /********************************************************/
     /* Now add another link to link the object to the       */
     /* Hometown view                                        */
     /********************************************************/
call AddLink ConView NetClass 'Hometown' ConObjs

call MakeRODMCall 'BUILD'
/*********************************************************************/
/* Start the third object in the group.  This one is called          */
/* '1000_Main_Street' and is contained in the 'Main_Street' object   */
/*********************************************************************/
call InitRODMStem
```

```
call StartObject AggClass '1000_Main_Street'

     /********************************************************/
     /* Add some information to the object                   */
     /********************************************************/
call AddAttr DispOther CharVar '3 Bedroom Ranch'
call AddAttr DispStat Integer Active

      /*******************************************************/
      /* Now link it to its parent and to its class          */
      /*******************************************************/
call AddLink DispType DispClass HtAgg_Icon 'Resources'
call AddLink PartOf AggClass 'Main_Street' COMPPHY


call MakeRODMCAll 'BUILD'  /* make the FLCARODM call           */
⋮
```

Results of Executing the BUILD Function: The following objects were created in RODM by the BUILD function:

- A view object that represents a network view named Hometown
- An aggregate object that represents Main_Street
- A real object that represents a house on Main_Street named 1000_Main_Street

UPDATE Function

Use the UPDATE function to change the value of fields on existing objects. The UPDATE function does not create objects. If you attempt to update a field on an object that does not exist, an error is returned.

The following code from sample FLCSX2 demonstrates how to use the UPDATE function to change objects in RODM.

```
⋮

call StartObject AggClass '1000_Main_Street' /*Which object we are   */
                                             /*referring to.         */
call AddAttr DispStat Integer InActive       /*Update display status */
call MakeRODMCall 'UPDATE'                   /*Call RODM             */
⋮
```

Results of Executing the UPDATE Function: The value of the DisplayStatus field on real object that represents named 1000_Main_Street is changed to 132 (Unsatisfactory).

QUERY Function

Use the QUERY function to determine the value of one or more fields on one or more objects. If either the field or the object does not exist, an error is returned. The field type and the field value are returned for every field on each object.

Although the field type is not specified when querying a field, FLCARODM only returns values for the following data types:

CLASSID	1
CHARVAR	4
INTEGER	10
OBJECTID	14
OBJECTIDLIST	15
OBJECTLINK	16
OBJECTLINKLIST	17
OBJECTNAME	18
SELFDEFINING	19

SMALLINT	21
SMALLINT	23
FIELDID	26
ANONYMOUSVAR	30

Examples of Using the QUERY Function: This section contains several examples of using the query function.

The following code from sample FLCSX3 queries the DisplayResourceOtherData field on the Main_Street object:

⋮

```
call StartObject AggClass 'Main_Street'    /*Which object we are     */
                                           /*referring to.           */
call AddAttrForQuery DispOther             /*Query contents of       */
                                           /*DisplayResourceOtherData*/
call MakeRODMCall 'QUERY'                  /*Call RODM               */
```

⋮

The result stem from FLCSX3 contains the following information in the order specified:
- The number of elements in the stem
- The FLCARODM return code followed by the RODM return and reason code
- The value of the field

The following is a partial example of the result stem that is returned when sample FLCSX3 is run.

```
3
FLCARODM:0,0,0
4
Constructed In 1889
```

⋮

Table 226 describes the result stem that was returned for sample FLCSX3:

Table 226. FLCSX3 sample (result stem)

Element Number	Element Value	Explanation
0	3	Indicates that the result stem contains 3 elements
1	FLCARODM:0,0,0	The FLCARODM return code and the RODM return and reason code
2	4	Indicates the data type of the field (charvar)
3	Constructed In 1889	The value of the field

Sometimes is it useful to know the value of the field identifier for a specified field. For example, if you are saving fields in a table, you can save space by saving the four-byte field ID instead of the larger field name.

Specifying the FIELDID parameter with a value of `YES` causes FLCARODM to return the field identifier value for fields returned by query functions.

Notes:

1. The field identifiers can change when RODM is cold-started, so any previously stored information regarding field identifiers are not used.

2. The FIELDID parameter can not be used with the LOCATE, XREF, or CHILDREN parameter

The following code from sample FLCSX3 has been modified by specifying `FIELDID=YES` to return the field ID of the DisplayResourceOtherData field:

```
⋮

call StartObject AggClass 'Main_Street'    /*Which object we are     */
                                           /*referring to.           */
call AddAttrForQuery DispOther             /*Query contents of       */
                                           /*DisplayResourceOtherData*/
call MakeRODMCall 'QUERY' 'FIELDID=YES'   /*Call RODM               */
⋮
```

The following partial example shows the result stem that is returned when the modified sample FLCSX3 is run.

```
4
FLCARODM:0,0,0
4
60
Constructed In 1889
⋮
```

Table 227 describes the result stem that was returned for the modified sample FLCSX3:

Table 227. Modified FLCSX3 sample (result stem)

Element Number	Element Value	Explanation
0	3	Indicates that the result stem contains 3 elements
1	FLCARODM:0,0,0	The FLCARODM return code and the RODM return and reason code
2	4	Indicates the data type of the field (charvar)
3	60	Indicates the field ID of the field
4	Constructed In 1889	The value of the field

Run samples FLCSX1, FLCSX2, and FLCSX3 before you run sample FLCSX4.

Sample FLCSX4 provides an example of using two queries to accomplish a task, and demonstrates how to determine the field values on a class, which is useful for querying default field values or for acquiring all of the objects of a certain class. For this example, assume that RODM was empty before sample FLCSX1 was run. The first part of sample FLCXS4 queries all of GMFHS_Aggegrate_Objects_Class objects in RODM:

```
⋮

call StartObject AggClass '.'                /*Which object we are   */
                                            /*referring to.         */
call AddAttrForQuery 'MyObjectChildren'

Say ''
Say 'Result from MyObjectChildren query:'
call MakeRodmCall 'QUERY'
'PIPE STEM RodmResult. | CONSOLE'
```

⋮

The result stem from the first part of sample FLCSX4 contains the following information in the order specified:

- The number of elements in the stem
- The FLCARODM return code followed by the RODM return and reason code
- The data type of the field
- The number of object IDs in the list
- The object ID of the object

The following is an example of the result stem that is returned by the first part of sample FLCSX4

```
4
FLCARODM:0,0,0
15
1
00010012E05C2A1E
```

Table 228 describes the result stem that was returned for the first part of sample FLCSX4:

Table 228. FLCSX4 sample (result stem)

Element Number	Element Value	Explanation
0	4	Indicates that the result stem contains 4 elements
1	FLCARODM:0,0,0	The FLCARODM return code and the RODM return and reason code
2	15	Indicates the data type of the field (objectidlist)
3	1	The number of object IDs in the list
4	00010012E05C2A1E	The hexadecimal object ID of the object

The second part of sample FLCSX4 queries the name and status of the object ID that was returned from the first query:

⋮

```
/*********************************************************************/
/* Query the name and status of the object.                          */
/*********************************************************************/
call InitRODMStem             /*Get ready for next set of operations*/

call StartObject AggClass '.'  /*Use Object ID from previous call*/
call AddAttrForQuery 'MyName'
call AddAttrForQuery 'DisplayStatus'

Say ''
Say 'Result from MyName and DisplayStatus query:'
call MakeRodmCall 'QUERY'
```

⋮

The following is an example of the result stem that is returned by the second part of sample FLCSX4.

```
6
FLCARODM:0,0,0
18
```

```
Main_Street
FLCARODM:0,0,0
10
132
```

Table 229 describes the result stem that was returned for the second part of sample FLCSX4:

Table 229. FLCSX4 sample, second part (result stem)

Element Number	Element Value	Explanation
0	6	Indicates that the result stem contains 6 elements
1	FLCARODM:0,0,0	The FLCARODM return code and the RODM return and reason code for the first field that was queried
2	18	Indicates the data type of the field
3	Main_Street	The number of object IDs in the list
4	FLCARODM:0,0,0	The FLCARODM return code and the RODM return and reason code for the second field that was queried
5	10	The data type of the field (integer)
6	132	The value of the field

Note: The query functions in FLCSX4 were performed by two calls to FLCARODM using the MakeRODMCall subroutine. Both functions can be performed using one call to FLCARODM by using the XREF parameter. See "FLCARODM Command" on page 548 for more information.

DELOBJ Function

Use the DELOBJ to delete one or more objects. When an object is deleted, its links to all other objects are deleted. Note that fields and links cannot be specified with the DELOBJ function.

Use care when using the DELOBJ function, because objects that other applications or users require might be deleted. Consider using the PURGE function instead. It provides a way to remove objects while protecting objects that are associated with other applications from being deleted.

The following code from sample FLCSX5 uses the DELOBJ to delete the 1000 Main Street object.

```
⋮

call StartObject AggClass '1000_Main_Street' /*Which object we are   */
                                             /*referring to.         */
call MakeRODMCall 'DELOBJ'                   /*Call RODM             */
⋮
```

Results of Executing the DELOBJ Function: After running this program, the 1000_Main_Street object, its links to Main_Street and, the object, are removed.

DELINKA Function

Use the DELINKA function to delete all links to specified fields on an object. You do not have to specify the links, because the DELINKA function determines which links exist and removes all of them.

For an example of using the DELINKA function, see “Delinking Objects” on page 570.

DELINKAB Function

Use the DELINKAB function to delete the specified links between objects.

For most objects linked using fields of type ObjectLink, it is not necessary to remove a link between objects before defining a new link. Instead, use the UPDATE function, which first removes the old link and then defines the new link. However, for fields that require a method to perform the link removal, (for example, DisplayResourceType), you must use the DELINKAB function.

For links that are defined by fields of type ObjectLinkList (for example, Resources), you must use the DELINKAB function, because the UPDATE function only adds the new link, but it does not delete previously defined links.

For an example of using the DELINKAB function, see “Delinking Objects” on page 570.

PURGE Function

Use the PURGE function to remove objects from RODM. Consider using the **RemvObjs** command to remove objects from RODM instead of the PURGE function. See the *IBM Tivoli NetView for z/OS User's Guide: NetView Management Console* for more information about the **RemvObjs** command.

LOCATE Function

Use the LOCATE function to search all fields of type CharVar or IndexList which have been created as public_indexed for a specified string. An example of a publicly indexed field is DisplayResourceName.

The LOCATE function returns the object ID of objects that contain a value that matches the specified string. Note that the search is not case sensitive.

The following code from sample FLCSXL01 finds all of the objects in RODM whose DisplayResourceName field has a value of CPU_UTILIZATION.

```
⋮
call StartObject '' '' /*Can not specify a class or an object for    */
                        /*This function                               */
call AddAttr DispName CharVar 'CPU_utilization'  */
call MakeRODMCall 'LOCATE'                 /*Call RODM            */
⋮
```

Note that you cannot specify a class or object for the LOCATE function. Therefore, `StartObject '' ''` is specified, which means search all objects on all classes.

The result stem from FLCSXL01 contains a list of the object IDs of the objects whose DisplayResourceName matches the comparison string NOT_LOGGED_IN. For example, if one object matched this criteria, the following result stem is returned:

```
4
FLCARODM:0,0,0
15
1
000100012E05C2A1E
```

Table 230 on page 561 describes the result stem that was returned for sample FLCSXL01 if one object met the search criteria.

Table 230. FLCSXL01 sample, one object met the search criteria (result stem)

Element Number	Element Value	Explanation
0	4	Indicates that the result stem contains 4 elements
1	FLCARODM:0,0,0	The FLCARODM return code and the RODM return and reason code
2	15	Indicates the data type of the return data (objectidlist)
3	1	The number of matches found
4	000100012E05C2A1E	The object ID of the object that matched the search criteria

If there were no objects in RODM with a field that matched the comparison criteria, FLCARODM returns an Object ID List with zero elements as follows.

```
3
FLCARODM:0,0,0
15
0
```

Table 231 describes the result stem that is returned for sample FLCSXL01 if no objects met the search criteria.

Table 231. FLCSXL01 sample, no objects met the search criteria (result stem)

Element Number	Element Value	Explanation
0	3	Indicates that the result stem contains 3 elements
1	FLCARODM:0,0,0	The FLCARODM return code and the RODM return and reason code
2	15	Indicates the data type of the return data (objectidlist)
3	0	Indicates that no matches were found

RodmResult.0	3
RodmResult.1	FLCARODM:0,0,0
RodmResult.2	15
RodmResult.3	0

QUERYALL Function

The QUERYALL function returns the field name, field type, and value for all of the fields defined on the specified object. For example, the following example queries the fields on the Main_Street object.

```
⋮
call StartObject AggClass 'MainStreet'
call MakeRODMCall 'QUERYALL'
⋮
```

Results of Executing the QUERYALL Function: The result stem from FLCSXQ2 contains the following information in the order specified:

- The number of elements in the stem.
- The FLCARODM return code followed by the RODM return and reason code.
- The number of fields defined on the object.
- A sequence of field specifications. For each field, the field specification contains the following information in the order specified:

- Return Code
- Name
- Identifier
- Value

The field specification information is repeated for each field.

The result stem from FLCSXQ2 contains the number of elements in the stem, the return code, the number of fields defined on the object, and a sequence of field specifications. Each field specification contains the following information:

The following is a partial example of the result stem that is returned when sample FLCSXQ2 is run.

```
212
FLCARODM:0,0,0
51
FLCARODM:0,0,0
IsPartOf
17
0
FLCARODM:0,0,0
IsBusNode
17
0
⋮
```

Table 232 describes the result stem that was returned for sample FLCSXQ2.

Table 232. FLCSXQ2 sample (result stem)

Element Number	Element Value	Explanation
0	212	Indicates that the result stem contains 212 elements
1	FLCARODM:0,0,0	The FLCARODM return code and the RODM return and reason code
2	51	Indicates the number of fields defined on the object
3	FLCARODM:0,0,0	The FLCARODM return code and the RODM return and reason code
4	IsPartOf	The name of the first field defined on the object.
5	17	The data type of the IsPartOf field. (objectlinklist)
6	0	The value of the IsPartOf field
7	FLCARODM:0,0,0	The FLCARODM return code and the RODM return and reason code
8	IsBusNode	The name of the second field defined on the object
9	17	The data type of the IsBusNode field. (objectlink)
10	0	The value of the IsBusNode field

The previous example describes the first two fields in the result stem. Elements 11 through 212 describe the remaining fields using the same format.

QUERYSTR Function

Use the QUERYSTR function to determine the structure of object classes. For each class, the field names, the field identifier type, and inheritance status bitmap for each field defined on the class is returned. For example, the following sample queries the structure of the GMFHS_Aggregate_Objects_Class class.

```
⋮
call StartObject AggClass ''
call MakeRODMCall 'QUERYSTR'
⋮
```

Results of Executing the QUERYSTR Function: The result stem from FLCSXQ1 contains the following information in the order specified:

- The number of elements in the stem
- The FLCARODM return code followed by the RODM return and reason code
- The number of fields defined on the object
- A sequence of field specifications. For each field, the field specification contains the following information in the order specified:
  - Name
  - Identifier
  - Type
  - Inheritance Status Bitmap

  The field specification information is repeated for each field.

The following is a partial example of the result stem that is returned when sample FLCSXQ1 is run.

```
214
FLCARODM:0,0,0
53
AggrgationChild
121
17
00
UpdateAggregationCounters
122
13
00
⋮
```

Table 233 describes the result stem that was returned for sample FLCSXQ1.

Table 233. FLCSXQ1 sample (result stem)

Element Number	Element Value	Explanation
0	214	Indicates that the result stem contains 214 elements
1	FLCARODM:0,0,0	The FLCARODM return code and the RODM return and reason code.
2	53	Indicates the number of fields defined on the object
3	AggregationChild	The name of the first field defined on the object.
4	121	The field identifier
5	17	The data type of the field (objectlinklist)
6	00	The inheritance status bitmap
7	UpdateAggregationCounters	The name of the second field defined on the object
8	122	The field identifier
9	13	The data type of the field (methodspec)

Table 233. FLCSXQ1 sample (result stem) (continued)

Element Number	Element Value	Explanation
10	00	The inheritance status bitmap

The previous example describes the first two fields in the result stem. Elements 11 through 214 describe the remaining fields using the same format.

QUERYSF Function

Use QUERYSF to query the value of the specified subfield for a field on the specified objects. The following subfields can be queried:

- VALUE
- QUERY
- CHANGE
- NOTIFY
- TIMESTAMP
- PREV_VAL

The following code from sample FLCSXQ3 returns the value of the previous value subfield of the DisplayStatus field of the 1000 Main Street object:

```
⋮

call StartObject AggClass '1000_Main_Street' /*Which object we are   */
                                             /*referring to.         */
call AddAttrForQuery DispStat                /*Query this field      */
call MakeRODMCall 'QUERYSF' 'SF=PREV_VAL'    /*Call RODM             */
⋮
```

Results of Executing the QUERYSF Function: The result stem from FLCSXQ3 contains the following information in the order specified:

- The number of elements in the stem
- The FLCARODM return code followed by the RODM return and reason code
- The data type of the subfield
- The subfield value

Note: Run samples FLCSX1 and FLCSX2 before you run sample FLCSXQ3.

The following is an example of the result stem that is returned when sample FLCSXQ3 is run.

```
3
FLCARODM:0,0,0
10
129
```

Table 234 describes the result stem that was returned for sample FLCSXQ3.

Table 234. FLCSXQ3 sample (result stem)

Element Number	Element Value	Explanation
0	3	Indicates that the result stem contains three elements
1	FLCARODM:0,0,0	The FLCARODM return code and the RODM return and reason code
2	10	The data type of the subfield (integer)
3	129	The previous value of the field

Note: FLCSX1 set the value to 129 and then FLCSX2 changed the value to 130, so the previous value was 129.

REBUILD Function

Use the REBUILD function to change objects when the links between objects have changed. For every object specified on the REBUILD function, all specified fields are updated, all specified links are defined, and all previously defined links are removed, with the following exceptions:
- LayoutParmList
- DetailLayoutParmList
- 2.9.3.2.7.42 (member)
- 1.3.18.0.0.2217 (memberArcs)
- ComposedOfPhysical
- ComposedOfLogical
- AggregationChild

The relationships listed above are not removed to avoid having objects in RODM that have no parent objects defined.

Putting It All Together

This section describes sample files that provide examples of using functions and parameters.

For a description of the subroutines used in the samples, see “Stem Building Subroutines” on page 542.

FLCSOX02

IBM provides sample FLCSOX02 to demonstrate how to create more complex environments with FLCARODM. This sample creates a sysplex model in RODM that contains a sysplex, a system in the sysplex, and multiple stacks contained in the system. This sample demonstrates a complete application to load RODM with objects, and can be used to verify NetView management console installations.

See the header information in FLCSOX02 for more complete details on this sample.

Building Objects

The following sample uses the StartObject and AddLink routines to create and link the following objects:
- An aggregate object named Demo_Lan
- Two objects that represent LAN segments
- An object that represents a bridge that connects the segments

```
⋮

call StartObject NetClass 'Advanced'        /*Which object?         */
/*******************************************************************/
/* Start creating LAN object in the Advanced Operations View        */
/*******************************************************************/
call StartObject AGrphClass 'Demo_Lan'
call AddLink DispType DispClass 'DUIXC_RTN_LAN_AGG' 'Resources'
call AddLink ConView NetClass 'Advanced' ConObjs

/*******************************************************************/
/* Create the Segment_1 object                                      */
/*******************************************************************/
call StartObject RSegClass 'Segment_1'
call AddLink DispType DispClass 'DUIXC_RTN_TR_SEGMENT' 'Resources'
call AddLink MemberOf AGrphClass 'Demo_Lan' Member

/*******************************************************************/
/* Add a Bridge called Bridge_1                                     */
/* Add a link to hook it to Segment_1.                              */
/*******************************************************************/
Call StartObject ABrgClass 'Bridge_1'
Call AddLink DispType DispClass 'DUIXC_RTN_BRIDGE_APPL' 'Resources'
Call AddLink MemberOf AGrphClass 'Demo_Lan' Member
Call AddLink PhyConn RSegClass 'Segment_1' Phyconn

/*******************************************************************/
/* Create the second segment, called Segment_2                      */
/* Add a link to connect it to Bridge_1                             */
/*******************************************************************/
call StartObject RSegClass 'Segment_2'
call AddLink DispType DispClass 'DUIXC_RTN_TR_SEGMENT' 'Resources'
call AddLink MemberOf AGrphClass 'Demo_Lan' Member
call AddLink PhyConn ABrgClass 'Bridge_1' Phyconn

call MakeRODMCall 'BUILD'                   /*Call RODM             */
⋮
```

Figure 146. Sample FLCSX6

Updating Objects

The following samples provide examples of changing objects using the UPDATE function.

Using the UPDATE Function With the CHILDREN Parameter: Figure 147 uses the UPDATE function to change the display status of the Demo_Lan aggregate object. Note that because `CHILDREN=ONLY` is specified, all of the Demo_Lan children are updated. However, the CHILDREN parameter only updates the first level of children.

```
⋮

call StartObject AGrphClass 'Demo_Lan'       /*Which object we are   */
                                            /*referring to.         */
call AddAttr DispStat Integer InActive      /*Update display status */

call MakeRODMCall 'UPDATE' 'CHILDREN=ONLY' /*Call RODM              */
                                          /*Update only the children*/
⋮
```

Figure 147. Sample FLCSX7

Using the UPDATE Function With the XREF Parameter: The XREF parameter can be used to specify fields of the following types:
- ObjectLink
- ObjectLinkList
- ObjectIdList

The following samples demonstrate using fields of these types to locate and update objects.

Figure 148 uses the UPDATE function to accomplish the same task as Figure 147 on page 566; however, instead of specifying the CHILDREN parameter, the XREF parameter is used to specify the links defined by field 2.9.3.2.7.42 (member).

```
⋮

call StartObject AGrphClass 'Demo_Lan'        /*Which object we are   */
                                             /*referring to.         */
call AddAttr DispStat Integer InActive       /*Update display status */
call MakeRODMCall 'UPDATE' 'XREF=2.9.3.2.7.42'    /*Call RODM        */
⋮
```

Figure 148. Sample FLCSX14

Figure 149 uses the UPDATE function with the XREF parameter to specify that the links defined by the ComposedOfPhysical field are used to determine the list of objects to be updated.

```
⋮

call StartObject AggClass 'Main_Street'      /*Which object we are   */
                                             /*referring to.         */
call AddAttr DispStat Integer Active         /*Update display status */
call MakeRODMCall 'UPDATE' 'XREF=ComposedOfPhysical' /* Call RODM    */
⋮
```

Figure 149. Sample FLCSX15

Figure 150 on page 568 performs the same functions as samples FLCSX14 and FLCSX15, which demonstrates that you can perform multiple functions with a single function call. Sample FLCSX16 uses the UPDATE function with the XREF parameter to specify that the links defined by the first field specified are used to determine the list of objects to be updated. For example, sample FLCSX16 specifies the following:

```
call StartObject AGrphClass 'Demo_Lan'
call AddLink Member DispStat Integer InActive
```

Because the first field that is defined on the Demo_Lan object is the Member field, the links it defines are used to determine which objects are updated.

```
⋮

call StartObject AGrphClass 'Demo_Lan'         /*Which object we are   */
                                              /*referring to.        */
call AddLink Member DispStat Integer InActive /*Update display status*/
                                              /*Cross Reference Member*/
                                              /*Field. Anything that  */
                                              /*has is a Member of the*/
                                              /*Demo_Lan gets changed */

call StartObject AggClass 'Main_Street'       /*Which object we are   */
                                              /*referring to.        */
call AddLink COMPPHY DispStat Integer InActive/*Update display status*/
                                 /*Cross Reference the COMPPHY field */
                                 /*in the Main_street to find out    */
                                 /*which objects have their Display  */
                                 /*status changed.                   */

call MakeRODMCall 'UPDATE' 'XREF=1STFIELD'    /*Call RODM            */
⋮
```

Figure 150. Sample FLCSX16

Figure 151 demonstrates how to update all of the child objects of a class by using the MyObjectChildren field, which is of type ObjectIdList and contains a list of object IDs of a class.

```
⋮

call StartObject RealClass ''                 /*Which object we are   */
                                              /*referring to.        */
call AddAttr DispStat Integer InActive        /*Update display status */
call MakeRODMCall 'UPDATE' 'XREF=MyObjectChildren' /*Call RODM      */
⋮
```

Figure 151. Sample FLCSX17

Querying Objects

This section describes using the QUERY function. For each sample, the query specification is described and a sample result stem is provided. See “Result Stem” on page 571 for more information about result stems.

Figure 152 queries the names of all of the Demo_Lan objects. The names are contained in the MyName field and the list of objects to be queried is defined by field 2.9.3.2.7.42 (member).

```
⋮

call StartObject AGrphClass 'Demo_Lan'         /*Which object we are   */
                                              /*referring to.        */
call AddAttrForQuery MyName                   /*Update display status */
call MakeRODMCall 'QUERY' 'XREF=2.9.3.2.7.42'    /*Call RODM         */
⋮
```

Figure 152. Sample FLCSX18

The following result stem was returned:

RodmResult.0	11
RodmResult.1	FLCARODM:0,0,0
RodmResult.2	3
RodmResult.3	FLCARODM:0,0,0
RodmResult.4	18

RodmResult.5	Segment_1
RodmResult.6	FLCARODM:0,0,0
RodmResult.7	18
RodmResult.8	Bridge_1
RodmResult.9	FLCARODM:0,0,0
RodmResult.10	18
RodmResult.11	Segment_2

FLCARODM:0,0,0 indicates that querying the cross reference field 2.9.3.2.7.42 was successful.

Figure 153 queries all objects in RODM to determine which objects have a display name of IP_NETWORKS. Note that `call StartObject '' ''` means all objects in RODM.

```
⋮

call StartObject '' ''                      /*Which object we are     */
                                            /*referring to.           */
call AddAttr DispName CharVar 'IP_Networks'  /*Look at all objects,  */
call AddAttrForQuery 'MyName'               /*Return MyNames for all  */
                                            /*objects with            */
                                            /*DispName="IP_Networks"  */
⋮

call StartObject '' ''                      /*Which object we are     */
                                            /*referring to.           */
call AddAttr DispName CharVar 'IP_Networks' /*Look at all objects,  */
call AddAttrForQuery 'MyName'               /*Return all with MyName= */
                                            /*IP_Networks           */
call MakeRODMCall 'QUERY' 'LOCATE=1STFIELD' /*Call RODM               */
```

Figure 153. Sample FLCSXL02

The following result stem was returned:

RodmResult.0	5
RodmResult.1	FLCARODM:0,0,0
RodmResult.2	1
RodmResult.3	FLCARODM:0,0,0
RodmResult.4	18
RodmResult.5	2.9.3.2.7.4=IP_Networks

The second stem variable indicates that there was one object that matched the criteria. The fifth stem variable provides the name of the object.

Figure 154 queries the display names of all Demo_Lan objects that contain the word Segment. Note that the FILTER parameter is used with the XREF parameter to refine the query.

```
⋮
call StartObject AggClass 'Demo_Lan'

call AddAttr MyName ObjectName 'Segment'
call AddAttrForQuery MyName

call MakeRODMCall 'QUERY' 'XREF=2.9.3.2.7.42' 'FILTER=1STFIELD'
⋮
```

Figure 154. Sample FLCSXF1

The following result stem was returned:

RodmResult.0	8
RodmResult.1	FLCARODM:0,0,0
RodmResult.2	2
RodmResult.3	FLCARODM:0,0,0
RodmResult.4	18
RodmResult.5	Segment_1
RodmResult.6	FLCARODM:0,0,0
RodmResult.7	18
RodmResult.8	Segment_2

The second stem variable indicates that there were two resources that matched the XREF and FILTER criteria. The names are contained in RodmResult.5 and RodmResult.8.

Note: If the XREF value is specified using 1STFIELD, then the filter criteria must be `FILTER=2NDFIELD`

Delinking Objects

This section describes how to use the DELINKA and DELINKAB functions to remove links between objects.

Figure 155 also uses the DELINKA function to delete all of the links defined by the PhysicalConnPP field of the Bridge_1 object.

```
⋮

call StartObject ABrgClass 'Bridge_1'        /*Which object we are   */
                                             /*referring to.        */
call AddLinkForDelete PhyConn                /*Add the link to Delete*/
call MakeRODMCall 'DELINKA'                  /*Call RODM            */
⋮
```

Figure 155. Sample FLCSX10

Like Figure 155, Figure 156 uses the DELINKA function to delete all of the links defined by the PhysicalConnPP field of the Bridge_1 object. However, the CHILDREN=ONLY parameter is used to determine which links are deleted.

```
⋮

call StartObject AGrphClass 'Demo_Lan'        /*Which object we are   */
                                             /*referring to.        */
call AddLinkForDelete PhyConn
call MakeRODMCall 'DELINKA' 'CHILDREN=ONLY'  /*Call RODM            */
                                             /*Only do the CHILDREN  */
⋮
```

Figure 156. Sample FLCSX9

```
⋮

call StartObject AGrphClass 'Demo_Lan'        /*Which object we are   */
                                             /*referring to.        */
call AddAttrForQuery  Member
call AddAttrForQuery  PhyConn
call MakeRODMCall 'DELINKA' 'XREF=1STFIELD'  /*Call RODM            */
⋮
```

Figure 157. Sample FLCSX19

Figure 158 uses the DELINKAB function to remove specific links to the Bridge_1 object.

```
⋮

call StartObject ABrgClass 'Bridge_1'        /*Which object we are   */
                                             /*referring to.        */
call AddLink PhyConn ASegClass 'Segment_1' PhyConn
call AddLink PhyConn ASegClass 'Segment_2' PhyConn
                                  /* Remove PhyConn links between the*/
                                  /* Bridge and the 2 Segments       */
call MakeRODMCall 'DELINKAB'                 /*Call RODM            */
⋮
```

Figure 158. Sample FLCSX11

Deleting Objects

Figure 159 uses the DELOBJ function to delete the Demo_Lan object. The CHILDREN parameter specifies that the child objects of the Demo_Lan object are also deleted.

```
⋮

call StartObject AGrphClass 'Demo_Lan'        /*Which object we are   */
                                             /*referring to.        */
call MakeRODMCall 'DELOBJ' 'CHILDREN=YES'    /*Call RODM            */
⋮
```

Figure 159. Sample FLCSX8

Working with IndexList Fields

Use the SetIndexList subroutine to change IndexList fields.

Figure 160. provides an example of changing an IndexList type field. The ExceptionViewList field of the Demo_Lan object is updated with the value test.

Note: Use caution when updating IndexList type fields, because this function overwrites the previous value of the field and the previous value is lost.

```
⋮
call StartObject AGrphClass 'Demo_Lan'      /*Which object we are    */
                                           /*referring to.          */
my_String = 'testing'
call SetIndexList my_String ExceptionViewList

call MakeRODMCall 'UPDATE'                 /*Call RODM              */
⋮
```

Figure 160. Sample FLCSX22

Result Stem

A result stem is returned each time the FLCARODM command is run. The format of the result stem depends on the operation that is performed and whether the operation completed successfully.

The first two elements (0 and 1) of any result stem always contain the same information. The 0 element (RodmResult.0) contains the total number of elements in the stem. The 1 element contains the following information in the order specified:

1. FLCARODM return code
2. RODM return code
3. RODM reason code

For example, assume that the FLCARODM command was issued with the BUILD function specified and the command completed successfully with no errors. The following result stem is returned:

```
1
FLCARODM:0,0,0
```

1 indicates the result stem contains one element and `FLCARODM:0,0,0` indicates that the FLCARODM command completed with no FLCARODM or RODM errors.

For a description of the FLCARODM return codes, see "Return Codes" on page 579. For a description of the RODM return and reason codes, see "RODM Return and Reason Codes" on page 452.

The following sections describe result stems based on the success or failure of an operation.

Result Stems for Operations That Complete Successfully

This section describes operations that complete without errors. See "ERROR CONDITIONS" on page 575 for information about error conditions.

Result Stems for Successful BUILD, UPDATE, DELETE, and PURGE Operations: For the BUILD, UPDATE, DELETE, and PURGE operations without error. the format of the result stem is:

Element	Element Value
RodmResult.0	1
RodmResult.1	FLCARODM:0,0,0

1 indicates the result stem contains one element and `FLCARODM:0,0,0` indicates that the FLCARODM command completed with no FLCARODM or RODM errors.

Result Stems for Successful Query Operations: The structure of the result stem for successful query operations depends on the data type of the field that is queried and whether the XREF parameter was specified.

If no error occurs while executing the QUERY function, and the XREF parameter was not specified then the format of the result stem is:

Table 235. QUERY function, XREF parameter not specified (result stem)

Element	Element Value	Explanation
RodmResult.0	*x*	The number of elements in the result stem
RodmResult.1	FLCARODM:0,0,0	The FLCARODM return code and the RODM return and reason code
RodmResult.2	10	The data contained in the field

If no error occurs while running the Query function, and the XREF parameter was specified, then the format of the result stem is slightly different. For each object, an additional return code indicates the success or failure of the cross reference field query, followed by the number of objects that were cross referenced.

Where:

elements
The total number of elements in the result stem.

xref_field_info
The structure containing the return code data for the cross referenced field, the number of cross referenced objects, and the query results for each object. The format of the req_field_info structure is:

```
►►──Stem.x=xref_return_code_data──Stem.x+1=number_of_cross_referenced_objects────►

    ┌───────────┐
    ▼           │
►───field_info──┴──────────────────────────────────────────────────────────────►◄
```

Where:

xref_return_code_data
The return code data regarding the query of the cross reference field.

number_of_cross_referenced_objects
The number indicating the number of objects that resulted from querying the cross reference field.

field_info
The structure containing the return code data, field ID, and field value for each field queried on each cross referenced object. The format of the field_info structure depends on the field type of the fields that were queried. This field type can always be found in the second element of the field_info structure.

For numeric, and character data types, the field_info format is:

Numeric & Character:

```
►►──Stem.f=return_code_data──Stem.f+1=field_type──Stem.f+2=field_value──────────►◄
```

Where

return_code_data
Data indicating that no errors occurred

field_type
Decimal value indicating either a numeric type, such as INTEGER (10) or a character type, such as CHARVAR (4)

field_value
The numeric or character data contained in the field

For example, querying the other data field of an object can result in:

```
FLCARODM:0,0,0
4
Constructed In 1889
```

OBJECTLINK: For fields of OBJECTLINK data types, the format of the result stem is:

Result Stem

```
►►──Stem.f=return_code_data──Stem.f+1=field_type──Stem.f+2=object_ID──────────►

►─Stem.f+3=field_ID───────────────────────────────────────────────────────────►◄
```

Where:

return_code_data
Data indicating that no errors occurred.

field_type
Decimal value (16) indicating that the data type is OBJECTLINK.

object_ID
The object identifier, in hexadecimal, of the object to which the field is linked.

field_ID
The field identifier, in decimal, of the field to which the queried field is linked.

For example, querying an objectlink field of an object can result in:

```
FLCARODM:0,0,0
16
00010012E05C2A1E
5
```

OBJECTLINKLIST: For fields of OBJECTLINKLIST data types, the format of the result stem is:

```
►►──Stem.f=return_code_data──Stem.f+1=field_type──Stem.f+2=relations──────────►

►─Stem.f+3=relation_definition────────────────────────────────────────────────►◄
```

Where:

return_code_data
Data indicating that no errors occurred

field_type
Decimal value (17) indicating that the data type is OBJECTLINKLIST

relations
The number of relations to the field that was queried

relation definition
Information regarding which objects are linked to the object, using the field that was queried

The format is:

```
►►──Stem.l=object_ID──Stem.l+1=field_ID───────────────────────────────────────►◄
```

The object ID and field ID, can repeat until the number of relations indicated have been presented.

Where:

object_ID
The object identifier, in hexadecimal, of the object to which the field is related.

field_ID
The field identifier, in decimal, of the field to which the queried field is related.

For example, querying an ObjectLinkList field of an object can result in:

```
FLCARODM:0,0,0
17
2
00010012E05C2A1E
5
00010012E05C2A1F
6
```

ERROR CONDITIONS: For error conditions, the format of the result stem depends on the operation that was performed, and where the error occurred. Regardless of the error situation, the following five pieces of information are always be returned.

```
►►──Stem.r=return_code_data──Stem.r+1=operation_code──Stem.r+2=object_ID──────────►

►─Stem.r+3=object_class──Stem.r+4=object_name─────────────────────────────────────►◄
```

Where:

return_code_data
In the format:

```
FLCARODM FLCARODM_return_code RODM_return_code RODM_reason_code
```

FLCARODM_return_code is the return code from the FLCARODM command processor. A value of 2000 indicates the error occurred in RODM, and the RODM_return_code and RODM_reason_code must be inspected. See “Return Codes” on page 579 for other return code value definitions.. Refer to the *IBM Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer's Guide* for more information.

operation_code
The operation that FLCARODM was attempting to perform when the error occurred. FLCARODM might perform several different operations, per function requested. The FLCARODM operations are discussed later.

object_ID
The RODM object identifier, in hexadecimal, of the object that the FLCARODM operation failed for. If it is not known it is null.

object_class
The RODM object class of the object for which the FLCARODM operation failed. If it is not known it is null.

object_name
The RODM object name of the object for which the FLCARODM operation failed. If it is not known it is null.

Locate: The format of the result stem for Locate is identical to that of the Query function for an Object ID List. Error conditions for Locate are the same as Query except that the Object Class and Object Name have null values.

MultiSystem Manager Operations

The operations that FLCARODM performs are:

```
Operation Id                        Operation
------------         ---------------------------------------------------
    000              No Operation Determined
    100              Create An Object
    101              Delete An Object
    102              Delete An Object And Its Children
    103              Delete An Object's Children
    104              Execute Purge Against An Object
    200              Change A Field, Creating The Object If Necessary
    201              Change A Field, Only If The Object Exists
    202              Query A Field On An Object
    203              Change A Field On A Child Object
    300              Define A Relation, Creating The Object If Necessary
    301              Define A Relation, Only If The Object Exists
    302              Delete A Relation
    303              Delete All Relations To A Field On Children Objects
    304              Delete All Relations To A Field On An Object
    401              Locate
```

For operation ids 000,100,101,102,103, and 104 no additional information other than what was previously discussed is present. For example, the following attempts to Build a single object in an object class that doesn't exist (FLCSX12).

```
⋮

call StartObject 'NoClass' 'Dave'           /*Which object we are   */
                                            /*referring to.         */
call MakeRODMCall 'BUILD'                   /*Call RODM             */
⋮
```

The following error stem is returned:

```
FLCARODM:2000,8,52
100
0000000000000000
No_Class
Dave
```

The information returned indicates that an error occurred (2000,8,52) while attempting to create (100) an object named (Dave) in the (No_Class) class. The return code 2000 indicates that the error was a RODM error. The description for the RODM return code/reason code (8/52) states that the referenced object class No_Class does not exist. Thus, a complete description of the error that occurred is returned. For this simple example. this might seem to be more information than is needed, but since FLCARODM supports multiple operations on multiple objects, with multiple fields and relations, this level of detail becomes necessary for more complex invocations.

For operation ids 200, 201, and 203, details regarding the field that was operated on is also returned. The format of the field information is:

►►—Stem.f=*field_name*—Stem.f+1=*field_type*—Stem.f+2=*field_value*——————————►◄

Where:

field_name
The field name or field identifier where the operation is performed

field_type
The data type for the field where the operation is performed

field_value
The specified field value for the field where the operation is performed

An example error can be:

```
FLCARODM:1048,0,0
200
0000000000000000
GMFHS_Managed_Real_Objects_Class
1000_Main_Street
DisplayStatus
Integer
129
```

The information returned indicates that an error occurred (1048,0 0) while attempting to change (200) the field (DisplayStatus) which is of type (Integer) to a value of (129) on an object named (1000_Main_Street) in the (GMFHS_Managed_Real_Objects_Class) class. The return code 1048 indicates that the field type specified is not valid. The field type must be a decimal value representing the data type and the word Integer was specified, which is incorrect. Use the decimal value 10.

For operation ids 202, 303, and 304, the field that was being operated on is also returned. The only additional data is the field name or field ID, the field type and field value are not present, because they do not apply to these operations. The following is what can be returned when you try to query a field that does not exist.

```
FLCARODM:2000,4,56
202
0000000000000000
GMFHS_Managed_Real_Objects_Class
1000_Main_Street
My_New_Field
```

The information returned indicates that an error occurred (2000,4 56) while attempting to query (202) the field (My_New_Field) on an object named (1000_Main_Street) in the (GMFHS_Managed_Real_Objects_Class) class. A RODM error occurred, because the field is not defined to the GMFHS_Managed_Real_Objects_Class class.

For operation ids 300, 301, and 302, the relation that was being operated on is also returned. The format of the relation data is:

```
►►──Stem.r=field_name──Stem.r+1=object_ID──Stem.r+2=object_class──────────────►

►─Stem.r+3=object_name──Stem.r+4=linked_field_name──────────────────────────►◄
```

Where:

field_name
The field name or field identifier that is being used to relate to another object

object_ID
The object identifier of the object that is related to the previous object

object_class
The class of the object that is related to the previous object

object_name
The name of the object that is related to the previous object

linked_field_name
The name of the field on the object that is being used to relate to the previous object

The following can be returned if an attempt was made to relate an object to another object that did not exist.

```
FLCARODM:2000,?,??
301
00010012E05C2A1E
GMFHS_Managed_Real_Objects_Class
1000_Main_Street
PhysicalConnPP
0000000000000000
GMFHS_Managed_Real_Objects_Class
Not_Defined_Yet
PhysicalConnPP
```

The information returned indicates that an error occurred (2000,?,??) while attempting to link (301) two real objects (1000_Main_Street) and (Not_Defined_Yet), defining a physical relation (PhysicalConnPP).

As stated before, the reason that the error information is so detailed is that FLCARODM proceeds when it encounters RODM errors (FLCARODM return codes between 2000 and 2999). It does not proceed if FLCARODM itself determines that the input data is corrupt, or an internal error occurs. So the following error output can result from one FLCARODM invocation:

```
FLCARODM:2000,8,52
100
0000000000000000
No_Class
Dave
FLCARODM:2000,4,56
202
0000000000000000
GMFHS_Managed_Real_Objects_Class
1000_Main_Street
My_New_Field
FLCARODM:2000,?,??
301
00010012E05C2A1E
GMFHS_Managed_Real_Objects_Class
1000_Main_Street
PhysicalConnPP
0000000000000000
GMFHS_Managed_Real_Objects_Class
Not_Defined_Yet
PhysicalConnPP
```

This can indicate that three errors occurred while processing the FLCARODM request. The calling application is able to decode this information because the FLCARODM operation code defines the format of the data that follows.

When no errors occur, FLCARODM only sends one return code FLCARODM:0,0,0 as stated before, for all operations except for Query. For Query, an individual return code is sent for every field queried, either indicating success and containing the data, or indicating failure with the cause of the failure. This enables the calling application to determine which fields where queried successfully and which ones failed. The application can then extract the information for the successful queries, and handle the unsuccessful queries as appropriate. For example:

```
FLCARODM:0,0,0
4
Constructed In 1889
FLCARODM:0,0,0
16
00010012E05C2A1E
5
FLCARODM:2000,4,56
202
0000000000000000
GMFHS_Managed_Real_Objects_Class
1000_Main_Street
My_New_Field
FLCARODM:0,0,0
17
2
00010012E05C2A1E
5
00010012E05C2A1F
6
```

This indicates that the first field was successfully queried, and that it has a character field with a value of `Constructed In 1889`. The second field queried was an object link, and the object ID and field ID are returned. The third field queried resulted in an error (2000,4,56), and the error information is returned. The fourth field queried was an object link list, and the information regarding the objects is returned. Note that even though querying the third field resulted in an error, FLCARODM continued on and sent back the data regarding the fourth field.

Return Codes

The FLCARODM return codes are documented below.

1000 No object data was found. Either the command was not issued using the NetView PIPE command, or nothing was found in the PIPE data stream.

1004 An incorrect function was requested. Valid functions are
- BUILD
- DELINKA
- DELINKAB
- DELOBJ
- PURGE
- QUERY
- UPDATE

1012 The RODM name specified was either null, or its length was greater than eight characters.

1016 The application name specified was either null, or its length was greater than eight characters.

1020 The class specified was not valid, possible reasons are:
- For class names, the length was greater than 64 characters, or the length was zero and an object ID was not specified.
- For class ids, the value following the #, was either non numeric, or the value was too large to be stored in four bytes.

1024 The object specified was not valid, possible reasons are:

	• For object names, the length was greater than 254 characters, or was zero, and no object class was specified. • For object ids, the value of the data following the #, was not 16 EBCDIC characters representing a hexadecimal value.
1028	The number of objects specified was either an incorrect number, or was too large.
1032	The number of fields specified was either an incorrect number, or was too large.
1036	The number of relations specified was either an incorrect number or was too large.
1044	The field specified was not valid, possible reasons are: • For field names, the length was greater than 64 characters, or was zero. • For field ids, the value following the #, was either non numeric, or the value was too large to be stored in four bytes.
1048	The field type specified was either an incorrect number, or was too large.
1052	The field value specified was not valid. If the field type indicates that the field value is numeric, then the field value was either an incorrect number, or was too large. If the field type indicates that the field value is character data, then the field value is greater than 254 characters in length.
1056	The value of fields and relations were both zero on an Update or Query operation. Update requires at least one field or link to update, and Query requires exactly one field to query.
1060	The specified field name to link to was either null, or its length was greater than 64 characters.
1064	The specified class name to link to was either null, or its length was greater than 64 characters.
1068	The specified object name to link to was either null, or its length was greater than 254 characters.
1072	The specified field name to link to was either null, or its length was greater than 64 characters.
1076	For the function specified, no fields are allowed.
1080	For the function specified, no relations are allowed.
1084	The data type returned for the field that was queried is not supported by FLCARODM.
1088	The value supplied for the RODMRTRY parameter is not valid.
1092	The value supplied for the RODMINT parameter is not valid.
1096	The value supplied for the CHILDREN parameter is not valid.
1100	The value supplied for the STATUS parameter is not valid.
1104	The value supplied for the TIME parameter is not valid.
1108	The value supplied for the TRACE parameter is not valid.
1112	A parameter specified is not valid or unauthorized for the function specified.
1116	The number of object definitions found was less than the number of objects specified.

1120	All expected data has been processed, but more data still exists.
1124	The object definition was not complete.
1128	The number of field definitions found was less than the number of fields specified.
1132	The field definition was not complete.
1136	The number of relations found was less than the number specified.
1140	The relation definition was incomplete.
1144	The number of fields specified was incorrect for the XREF function.
1148	The value supplied for the LOCATE parameter is not valid.
1152	The value supplied for the SF parameter is not valid.
1156	The value supplied for the FIELDID parameter is not valid.
1160	The value supplied for the FILTER parameter is not valid.
1164	Too many field definitions were specified for the function specified.
19XX	All error codes from 1900 to 1999 indicates that an internal error occurred in FLCARODM while processing the object data. Please report this return code to the appropriate service representative, along with the associated error information.
2000	An error occurred in RODM while processing a request. The RODM return code and reason code provide more detailed information.
2004	There were no children on the object specified. For a function with the XREF option, this return code means that there were no relationships to traverse.
2008	The field indicated to be changed on an object's children does not exist on a child object. For a function with the XREF option, this return code means the field did not exist on any of the objects that were cross-referenced.
4000	An internal error has occurred in FLCARODM while attempting to perform the indicated operations. Please report this return code to the appropriate service representative, along with the associated error information.
4004	FLCARODM is unable to get necessary storage.
4008	FLCARODM has detected a condition that must not occur. Please report this return code to the appropriate service representative, along with the associated error information.
4012	An attempt was made to delete a link, but the data type of the specified field was not of type ObjectLink or ObjectLinkList.
4016	No Member or MemberArcs field is defined on the specified object, so the function can not be performed on the object's children.
4020	Filter error.

Object Data Stream Detail

The data stream is a low-level means of specifying data to RODM for creation and update of objects. Developers that use the Stem Building Routines do not need to specify the Data Stream at this low level.

Data Stream Explanation

The format of the data stream consists of the total number of records in the REXX stem (X.0), followed by the number of objects to be defined, followed by each object definition.

Format of the Data Stream

```
Number Of Stem Records
Number Of Objects
Object Definition # 1
Object Definition # 2
          .
          .
          .
Object Definition # N
```

Each object definition consists of the name of the object class, the object name, the number of fields and relations to be defined, followed by the field and relation definitions.

Format Of Each Object Definition

```
Object Class
Object Name
Number Of Fields
Number Of Relations
Field Definition #1
Field Definition #2
          .
          .
          .
Field Definition #M
Relation Definition #1
Relation Definition #2
          .
          .
          .
Relation Definition #P
```

Each field definition consists of the name of the field, the data type of the field value, and the field value.

Format Of Each Field Definition

Field Name
Field Value Data Type
Field Value

Each relation definition consists of the name of the field present on this object that is related to another object, the class and object name the field is related to, followed by the field on the related object.

Format Of Each Relation Definition

Field Used For Relation
Class Of Related Object
Name Of Related Object
Field On Related Object

A data stream consists of individual data stems.

Data Stem Detail

This section details the format of the REXX object data stem. It is structured in the following format:

```
                                      ┌──────────────────┐
                                      ▼                  │
►►──Stem.0=elements──Stem.1=objects────object_definition─┴───────────────────►◄
```

Where:

elements
The total number of elements defined for the stem variable.

objects
The number of objects where the operation is performed. This value must be at least one.

object_definition
Defines the objects to be modified. The object definitions can be repeated, and the number of object definitions must be equal to the number indicated by *objects*.

Object Definition: The format of object_definition follows. Note: The letter 'o' is used in the stem variable since the actual stem value varies.

```
►►──Stem.o=object_class──Stem.o+1=object──Stem.o+2=fields──Stem.o+3=relations────►
```

```
   ┌──────────────────┐  ┌─────────────────────┐
   ▼                  │  ▼                     │
►──┬──────────────────┴──┬─────────────────────┴──────────────►◄
   └─field_definition─┘  └─relation_definition─┘
```

Where:

object_class
The object class on which to be operated. This must be blank if an object ID is specified. This must be null if the Locate function is specified.

object The name or object ID of the object on which to be operated. The object ID is specified by prefixing it with the #, followed by the hexadecimal object ID value. If the first character is not a #, then the data is interpreted as an object name. If an object ID is specified then the object class is ignored. If a null is specified '', then the operation is performed on the class. This is only valid for Query operations. This must be null if the Locate function is specified.

fields The number of fields on the object to be modified or queried.

relations
The number of relations on the object to be created or removed.

field_definition
Defines the fields to be modified or queried. The field definitions can be repeated, and the number of field definitions must be equal to the number indicated by *fields*.

relation_definition
Defines the relations to be created/deleted between objects. The relation definitions can be repeated, and the number of definitions must be equal to the number indicated by *relations*.

Field Definition: The format of field_definition follows. Note: The letter f is used in the stem variable since the actual stem value varies.

```
►►──Stem.f=field──┬──────────────────────┬──┬───────────────────────┬──────────►◄
                  └─Stem.f+1=field_type──┘  └─Stem.f+2=field_value──┘
```

Where:

field The name or field ID of the field to be modified or queried. The field ID is specified by prefixing it with the #, followed by the decimal numeric field ID value. If the first character is not a #, then the data is interpreted as a field name.

field_type
A decimal integer value corresponding to the data type identifier of the field. The following data types are supported for Build and Update.

```
  Data Type        Data Type Identifier
--------------     --------------------
CHARVAR                     4
INTEGER                    10
SELFDEFINING               19
SMALLINT                   21
FIELDID                    26
ANONYMOUSVAR               30
```

For a list of data types supported by the BUILD and UPDATE functions, see“BUILD Function” on page 554.

For a list of data types supported by the QUERY function, see "QUERY Function" on page 555.

field_value
The value that is assigned to a field.

Field type and field value are required components of a field definition for the Build, Update and Locate functions. They must not be specified for the other functions. When the XREF parameter is specified (Build and Update functions only) with a value of 1STFIELD, the field type and field value must not be specified for the first field on each object. For the Locate function, field_value is the comparison string.

Relation Definition: The format of relation_definition follows. Note: The letter r is used in the stem variable since the actual stem value varies.

```
►►──Stem.r=field_to_link──Stem.r+1=object_class_to_link_to──────────────────►

►─Stem.r+2=object_to_link_to──Stem.r+3=field_to_link_to─────────────────────►◄
```

Where:

field_to_link
The name or field ID on *object* to be related to another field. The field ID is specified by prefixing it with the #, followed by the decimal numeric field ID value. If the first character is not a #, then the data is interpreted as a field name.

object_class_to_link_to
The name of the object class of the object to be related to the object being defined.

object_to_link_to
The name or object ID of the object to be related to the object being defined. The object ID is specified by prefixing it with the #, followed by the hexadecimal object ID value. If the first character is not a #, then the data is interpreted as an object name. If an object ID is specified then the object_class_to_link_to is ignored.

field_to_link_to
The name or field ID on *object_to_link_to* to be related to *field_to_link* on *object* The field ID is specified by prefixing it with the #, followed by the decimal numeric field ID value. If the first character is not a #, then the data is interpreted as a field name.

RODM Collection Manager

The RODM collection manager actively manages the contents of views and aggregates based on criteria that you set. These criteria can be a naming convention, a set of statuses, or both. A view that is managed by the RODM collection manager can have characteristics of both network and exception views. NetView management console administrators have access to the RODM collection manager on their desktops. You can also manage views with complex criteria beyond names and statuses.

RODM Collection Manager dynamically manages the views or aggregates. This means that RODM Collection Manager continually updates the views or aggregates, so that you can add, change, or delete collections.

For information about using collection definition objects, see "Using the Collection Definition Objects" on page 143. For information about using RODM Collection Manager with NetView management console, see the *IBM Tivoli NetView for z/OS User's Guide: NetView Management Console*.

Visual BLDVIEWS

Visual BLDVIEWS (VBV) is a graphical front end to BLDVIEWS; it simplifies the use of BLDVIEWS by eliminating the need to know the syntax of the BLDVIEWS input file language.

Visual BLDVIEWS provides a way to gather a set of objects (based on a naming convention) and to modify a field for each of these objects. This is useful for making batch updates to many objects. With Visual BLDVIEWS, you can query and display objects as they appear in RODM by double-clicking them. Immediate updates can be made to individual objects.

Visual BLDVIEWS consists of host and workstation applications that communicate through TCP/IP. To enable the VBV server on the NetView host, perform the following steps. For information about changing CNMSTYLE statements, see *IBM Tivoli NetView for z/OS Installation: Getting Started*.

1. Copy the AUTOTASK.AUTOVBV statement from the CNMSTYLE member to the CNMSTUSR or C*xx*STGEN member, and uncomment it.
2. Ensure that the TOWER MSM statement is uncommented in the CNMSTYLE member.

For more information about Visual BLDVIEWS, see the VBV online help. For information about using Visual BLDVIEWS with topology correlation, see the *IBM Tivoli NetView for z/OS User's Guide: NetView Management Console*.

BLDVIEWS

BLDVIEWS is a REXX exec that you can use to create aggregate objects and customized views. It works with objects of the GMFHS, SNA topology manager, and MultiSystem Manager data models. BLDVIEWS also provides an easy way to map a default set of commands to generic commands for key MultiSystem Manager resources by enabling generic command support from a NetView management console for MultiSystem Manager discovered network resource objects. Use BLDVIEWS to create the following types of views:
- Configuration backbone
- Configuration logical
- Configuration physical
- Configuration peer
- Exception
- More detail logical
- More detail physical
- Network

BLDVIEWS uses control statements to specify the names of the views and aggregates you want to create and the resources that you want the views and aggregates to contain. Control statements use keywords and values to specify the parameters. When specifying resources, you do not need to know the RODM classes or formats of the RODM names. To specify a resource, type the name of the resource that is displayed (the value of the RODM DisplayResourceName field). You can also specify ALL or a wild card name.

Use BLDVIEWS to link existing resources (objects) in RODM to views and aggregate objects, or to modify a subset of the more commonly used fields on existing resources. You can create new views and aggregates or update existing views and aggregates. BLDVIEWS supports RODM objects created by MultiSystem Manager and SNA topology manager. However, BLDVIEWS does not create objects on those classes. Use BLDVIEWS to create resources on GMFHS classes.

The control statements are passed to BLDVIEWS using one of the following methods:
- DSIPARM member (for example, BLDVIEWS MYMEMBER)
- A fully qualified cataloged sequential data set (for example, BLDVIEWS ESP.GAF.DATA(MYDEFS)
- A stem array, collected and passed using the PIPE command (for example, MyStem.0=2; MyStem.1=VIEW ....; MyStem.2=BRIDGE ...; 'PIPE STEM MyStem. | COLLECT | NETV BLDVIEWS | CONSOLE')

BLDVIEWS also provides a REXX exec called DELVIEWS that you can use to delete views or groups of views with a specified prefix.

Before You Begin

You can use Visual BLDVIEWS (VBV) to generate the BLDVIEWS control statements. VBV is an application that simplifies the management of RODM views and information. VBV provides a graphical, drag-and-drop interface to BLDVIEWS and RODMView. Note that your existing BLDVIEWS files can be imported into VBV. For more information about VBV, See the VBV online help.

Sample BLDVIEWS control statements are contained in member FLCVBLDS which resides in the CNMSAMP data set. FLCVBLDS has examples of coding control statements using various parameters.

BLDVIEWS Processing

BLDVIEWS queries RODM for specified objects and then links these objects to the views or aggregate objects that you specify. BLDVIEWS can modify certain fields on objects in any class in RODM, and can create objects on GMFHS classes.

Any processing performed by BLDVIEWS is static. Only the resources that were in RODM at the time you run BLDVIEWS are processed. If resources are later added or deleted from RODM, rerun BLDVIEWS to incorporate the changes into your views.

The RODM Collection Manager provides fully dynamic view creation and maintenance, and it is compatible with the BLDVIEWS control statements. Refer to the NetView Command online help for FLCV2RCM and the NetView management console online help for more information about the RODM Collection Manager.

All combinations of classes are supported.

Views

BLDVIEWS supports the following types of views:
- Network
- Configuration Peer
- Configuration Backbone
- Configuration Connectivity
- More Detail

With BLDVIEWS, you can specify any supported view layout type, but BLDVIEWS uses only the following view layout types:

- Hierarchical
- Ellipse
- Grid

Aggregate Objects

Use the AGGregate control statement and AGGChild control statements to create your own aggregate resources and specify which objects you want linked to the aggregate. BLDVIEWS links the AGGChild resources to the AGGregate resource by linking both the AggregationParent and AggregationChild field and the ComposedOfLogical and IsPartOf fields.

BLDVIEWS Control Statements

The following control statements are supported:

AGG	Specifies the aggregate resources (GMFHS aggregate objects). The aggregate resources that you specify can be existing resources, or you can create an aggregate and link the resources to the aggregate resource on the AGGChild control statements that follow.
AGGCHILD	Specifies the aggregation children that you want linked to the aggregate resource that was previously defined.
BBVIEW	Defines a configuration backbone view, which contains the resources on the control statements that follow it.
CDRM	Specifies VTAM CDRM resources.
CDRSC	Specifies VTAM CDRSC resources.
CIRCUIT	Specifies the APPN transmission group circuits, and subarea circuits.
CLASS	Specifies the global RODM class that contains the resources on the OTHER control statements that follow it. This control statement is used only for the OTHER control statement and you can use it to specify the RODM class globally without having to specify it on each OTHER control statement.
CLUSTER	Specifies the MultiSystem Manager or APPNTAM cluster aggregate resource.
DOMAIN	Specifies APPN domains.
ENODE	Specifies APPN end nodes.
EVIEW	Defines an exception view. **Note:** Objects specified after the EVIEW statement only participate in the exception view if CREATE=Y or CREATE=B is specified on the EVIEW statement. If CREATE=N is specified, these objects are ignored, and do not participate in the exception view.
GW_NCP	Specifies NCP gateway resources.

HOST_NODE	Specifies the host PUs (PU Type 5 nodes).
IC_NODE	Specifies APPN interchange nodes.
INTERFACE	Specifies TCP/IP adapter resources.
IP_BRIDGE	Specifies TCP/IP bridge aggregate resources.
IP_HOST	Specifies TCP/IP host aggregate resources.
IP_HUB	Specifies TCP/IP hub aggregate resources.
IP_LINK	Specifies TCP/IP interface link resources.
IP_LOCATION	Specifies TCP/IP location aggregate resources.
IP_ROUTER	Specifies TCP/IP router aggregate resources.
IP_SEGMENT	Specifies TCP/IP segment aggregate resources.
IP_SUBNET	Specifies TCP/IP subnetwork aggregate resources.
IPSPname	Specifies the VTAM PU, LU, or CP name for the service point that manages the resources on the control statements that follow it.
LCVIEW	Defines a configuration logical connectivity view, which contains the resources on the control statements that follow it.
LINE	Specifies VTAM lines.
LLINK	Specifies logical links.
LNODE	Specifies APPN len nodes.
LU	Specifies VTAM logical units.
LU_GROUP	Specifies VTAM logical unit groups.
MAJNODE	Specifies VTAM major nodes.
MDLVIEW	Defines a more detailed logical view, which contains the resources on the control statements that follow it.
MDPVIEW	Defines a more detailed physical view, which contains the resources on the control statements that follow it.
MIG_DATA_HOST	Specifies Migration Data Hosts.
NCP	Specifies NCP resources.
NETWORK	Specifies the MultiSystem Manager or APPNTAM network aggregate resource.
NNODE	Specifies APPN network nodes.
NONSNA	Specifies Non-SNA (GMFHS managed real) resources.
OTHER	Specifies a resource from a user-created or MultiSystem Manager open class.
PCVIEW	Defines a configuration physical connectivity view, which contains the resources on the control statements that follow it.
PU	Specifies VTAM physical units.

PVIEW	Defines a configuration peer view, which contains the resources on the control statements that follow it.
SNA	Specifies VTAM SNA shadow resources.
SNA_DOMAIN	Specifies the global VTAM domain which owns the resources on the control statements that follow it.
SNA_PORT	Specifies the SNA port.
SNALOCALTOPO	Specifies the APPN SNA local topology resources.
SYSTEM	Specifies system aggregate resources.
TG	Specifies APPN transmission groups.
VIEW	Defines a network view, which contains the resources on the control statements that follow it.
VRN	Specifies APPN virtual routing nodes.
WILDCARD	Defines wildcard characters to use when coding wild card names on the control statements.

Control Statement Syntax

BLDVIEWS control statements have a free-form syntax which uses keywords and values. You can start coding in any column. Leading and trailing blanks are ignored. A specific control statement can span 1 or more lines. Two types of continuation are available:

- A control statement separated into multiple statements with the break occurring after a `keyword=value`. This is done by coding a comma after the `keyword=value` and continuing with the remaining parameters on the next statements. For example:

  ```
  BRIDGE=ALL,
         TYPE=AGG,AGGTHRESH=(20%,60%,80%),
         SP=A19SRVCP
  ```

- A control statement separated into multiple statements with the break occurring anywhere in the coding. (This type of continuation is required when an entire `keyword=value` cannot be coded on one statement). The break can occur in the middle of a keyword or value by coding the following characters: **||,**. For example:

  ```
  BRIDGE=ALL,
         TYPE=||,
         AGG,
         AGG||,
         THRESH=(20%,||,
         60%,80%),
         SP=A19||,
         SRVCP
  ```

  The statements are concatenated and the characters are removed.

Note: The RODM Collection Manager interpreter supports the use of double equal signs (==) to distinguish between using MyName-based names or DisplayResourceName-based names as they appear on a view. For example, the following control statement creates a view and adds an object based on its DisplayResourceName:

```
VIEW=NewView,CREATE=Yes
GENERIC==CommonName,CLASS=My_Object_Class
```

Control statements can be coded in the following items:

- NetView DSIPARM member
- Fully qualified cataloged data set
- A REXX stem array, which is collected in a MLWTO and passed to BLDVIEWS using the NetView PIPE command

Note: You can use z/OS system symbolics in control statements processed by BLDVIEWS.

Keywords can be specified in any case (upper, lower or mixed) and they can be abbreviated. The abbreviated syntax is denoted in uppercase letters defined on each control statement.

If the control statements are coded in a NetView DSIPARM member or a fully qualified data set, the maximum length of each record is 80 characters. Columns 73-80 are ignored. If the control statements are passed to BLDVIEWS using the NetView PIPE command, there is no limit to the size of the records and no columns are ignored

The following resource names are always translated to upper case:
- AGGCHILD (All resources except for NONSNA, AGG, CLUSTER, and MultiSystem Manager TCP/IP resources)
- All APPNTAM resources except for nnDomainNetworkCluster
- All SNA topology manager resources
- IPSPname
- SNA
- SNA_DOMAIN

For all other resource names, code them in the same case, because that is how they are displayed by NetView management console or by the various element managers.

Keyword values can be coded in mixed case. In some instances the values are respected and in other instances the values are translated to upper case. The values for the following keywords are not translated to upper case:
- CONSOLE
- CORRELATER (NetView V1R3 and above)
- DISPLAY_NAME
- DOMAIN
- Generic Commands (ACTIVATE, DEACTIVATE, RECYCLE, DISPLAY)
- OTHER_DATA
- USER_DATA

Comments can be used, but only on separate statements. Code a comment statement by coding an * in column 1.

```
* NETA NCPs
NCP=NETA*
```

When you want to link resources to a view, code a VIEW statement followed by the resource statements that you want linked to the view.

```
VIEW=NEWVIEW,CREATE=YES
IP_ROUTER=rtr1.company.com
IP_ROUTER=rtr2.company.com
```

When you want to link resources to an aggregate, code an AGGregate statement followed by the AGGChild resource statements that you want linked to the aggregate.

```
AGGREGATE=NEWAGG,CREATE=YES
AGGCHILD=rtr1.company.com,type=IP_ROUTER
AGGCHILD=rtr2.company.com,type=IP_ROUTER
```

Common Control Statement Parameters

The following parameters are common to many of the BLDVIEWS control statements and are documented here and referenced later in the documentation by the control statements that support them:

- AGGPRI
- AGGTHRESH
- COLUMN
- CONSOLE
- CORRELATER
- DISPLAY_NAME
- DISPLAY_STATUS
- OTHER_DATA
- ROW
- TYPE
- UNLINK
- USER_DATA
- User Status
  - MARK
  - AUTO_IN_PROGRESS
  - SUSPEND
  - SUSPEND_WITH_AUTO_CLEAR
- Generic Commands:
  - ACTIVATE
  - DEACTIVATE
  - DISPLAY
  - RECYCLE

AGGPRI:

Description: The AGGPRI keyword is used to set or change the aggregation priority for real resources. The aggregation priority is the number of levels of aggregate resources whose status immediately changes to degraded when the real resource becomes unsatisfactory (regardless of aggregation threshold values). Use this to give higher priority to critical resources.

Syntax:

`AGGPRI=x`

-2	Use the DisplayResourceType default value
-1	Do not aggregate
0	Aggregate, but immediately degrade 0 levels
1	Immediately degrade 1 level
2	Immediately degrade 2 levels
3	Immediately degrade 3 levels
4	Immediately degrade 4 levels
5	Immediately degrade 5 levels
6	Immediately degrade 6 levels
7	Immediately degrade 7 levels
8	Immediately degrade 8 levels
9	Immediately degrade 9 levels

Example: `AGGPRI=2`

AGGTHRESH:

Description: The AGGTHRESH keyword is used to set the aggregation thresholds for aggregate resources. The aggregation thresholds are used to determine when the status of aggregates are changed to reflect the status of the underlying resources. There are three aggregation thresholds:
- ThresholdDegraded (status color is yellow)
- ThresholdSeverelyDegraded (status color is pink)
- ThresholdUnsatisfactory (status color is red)

Thresholds are specified on aggregate resources and are the minimum number of unsatisfactory, real resources underneath the aggregate which cause the aggregate to change status.

If you specify percentages, BLDVIEWS queries the TotalRealResourceCount field of the aggregate and multiplies it by the specified percentages to calculate the new values for the thresholds.

Syntax:

```
AGGTHRESH=(xxx,yyy,zzz)
```

xxx	1-3 digit ThresholdDegraded
yyy	1-3 digit ThresholdSeverelyDegraded
zzz	1-3 digit ThresholdUnsatisfactory

Example: AGGTHRESH=(10#,25%,75%)

Usage Notes:
- To specify a threshold value as a percentage, prefix or suffix the number with a %. BLDVIEWS multiplies it by the total number of real resources linked to the aggregate to come up with the threshold.

  To specify a threshold value as an actual number, prefix or suffix the number with a #. If the specified threshold is larger than the total number of real resources beneath the aggregate, then the threshold is set to the total number of real resources beneath the aggregate.

  You can mix actual values and percentages in the AGGTHRESH keyword.
- If resources are added or deleted from an aggregate object after BLDVIEWS is run, it is necessary to rerun BLDVIEWS to readjust the thresholds.

COLUMN:

Description: When building a grid view (layout type of 9), you can specify the specific column on the screen where you want a resource to be placed. The COLUMN keyword is used to specify the column.

Syntax: COLUMN=column_on_screen

Example: COLUMN=3

Usage Notes:
- The COLUMN keyword is only supported if specified on a resource control statement that follows a view control statement with a layout type of 9 (grid).
- ROW must be specified if COLUMN is specified.

CONSOLE:

Description: With remote console support, you can click on a resource and then issue a command, such as TELNET, or remotely log on to a resource. Although this is referred to as remote console support, any command can be specified. The command runs on the NetView management console workstation. BLDVIEWS envelopes the specified command with RemoteConsole `= # and #` and then sets the DisplayResourceUserData field. The user only has to specify the command.

You can set the remote console field for any resource in RODM that has the DisplayResourceUserData field defined. See the *IBM Tivoli NetView for z/OS User's Guide: NetView Management Console* for more information.

Syntax:

```
CONSOLE='command'
```

Example:

```
CONSOLE='TELNET.EXE %name%'
```

Usage Notes:

- You can set Remote Console support for any object which has the DisplayResourceUserData field defined.
- BLDVIEWS envelopes the specified command with the appropriate control information that is required for the command to be run correctly. The command must be specified either with a fully qualified name (drive and path) or the PATH must be set so the command can be located.
- CONSOLE is mutually exclusive with USER_DATA
- BLDVIEWS provides control variables that can be coded anywhere in the command text. The variables are:

%NAME%	Is substituted with the name of the resource. This variable is supported for all resources.
%RANDOM%	Is substituted with a 1-5 digit random number. This variable is supported for all resources.
%SEGMENT%	Is substituted with the segment number where the resource resides. This variable is supported only for the resources that are identified by the following control statements: – IP_HOST – INTERFACE
%IPADDRESS%	Is substituted with the internet address of the resource.

- BLDVIEWS enables the following NetView variables to be coded anywhere in the command text:

netid()	VTAM network identifier
domain()	Current NetView domain
opid()	NetView operator or task ID
cursys()	Current operating system name
ecvtpseq()	Level of operating system
vtam()	VTAM version and release
netview()	NetView version and release
mvslevel()	z/OS version and release
opsystem()	Type of operating system
sysplex()	Name of the MVS sysplex

- Single quotation marks (') or double quotation marks (") can be used as a delimiter.

Correlater:

Description: The CORRELATER keyword is used to set the Correlater field for an object.

BLDVIEWS enables the following NetView variables to be coded anywhere in the correlater text:

netid()	VTAM network identifier
domain()	Current NetView domain
opid()	NetView operator or task ID
cursys()	Current operating system name
ecvtpseq()	Level of operating system
vtam()	VTAM version and release
netview()	NetView version and release
mvslevel()	MVS version and release
opsystem()	Type of operating system
sysplex()	Name of the MVS sysplex

Syntax: CORRELATER='USA VA RICHMOND'

```
CORRELATER='text'
```

Usage Notes: Single quotation marks (') or double quotation marks (") can be used as a delimiter.

DISPLAY_NAME:

Description: Set the DisplayResourceName field for resources coded on SNA, NONSNA and AGGREGATE statements. The DisplayResourceName field is used to define a more descriptive and useful name to the resources. DisplayResourceName, if defined for a resource, is displayed on the workstation instead of the actual RODM name (MyName) of the resource.

Use the BLDVIEWS substitution variable %NAME% as part of the new DisplayResourceName value. The %NAME% variable is substituted with the name of the resource. You can use this to reformat the names of multiple resources at once with one control statement. You can prefix or suffix the names with additional text.

Syntax:

```
DISPLAY_NAME=xxx
```

Example:

```
DISPLAY_NAME=NCP_1
```

DISPLAY_STATUS:

Description: The DISPLAY_STATUS keyword is used to set the status of an object.

Syntax:

```
DISPLAY_STATUS=xxx
```

129	Satisfactory
144	Medium satisfactory
145	Low satisfactory
130	Unsatisfactory
160	Medium unsatisfactory
161	Low unsatisfactory

131	Intermediate
132	Unknown
133	Degraded
134	Severely degraded
136–143	User status
152–159	User status

Example:

```
DISPLAY_STATUS=130
```

Usage Notes: Display status value 131 is not supported for aggregate objects.

Display status values 133 and 134 are not supported for real objects.

OTHER_DATA:

Description: The OTHER_DATA keyword is used to set the RODM DisplayResourceOtherData field for an object. The DisplayResourceOtherData field can be set to any value. The value in this field is displayed in the NetView management console Data1 field.

Syntax:

```
OTHER_DATA='other_data'
```

Example:

```
OTHER_DATA='Call 1-800-IBM-HELP for support'
```

Usage Notes: BLDVIEWS enables the following NetView variables to be coded anywhere in the other data text. The variables are:

netid()	VTAM network identifier
domain()	Current NetView domain
opid()	NetView operator or task ID
cursys()	Current operating system name
vtam()	VTAM version and release
netview()	NetView version and release
mvslevel()	MVS version and release
opsystem()	Ttype of operating system
sysplex()	Name of the MVS sysplex

Single quotation marks (') or double quotation marks (") can be used as a delimiter.

ROW:

Description: When building a hierarchical view (layout type of 6) or a grid view (layout type of 9), you can specify the specific row on the screen where you want a resource to be placed. Use the ROW keyword to specify the row on a screen where you want a resource to be positioned.

The ROW keyword is only supported if specified on a resource control statement that follows a view control statement with a layout type of 6 (hierarchical) or 9 (grid).

Syntax:

```
ROW=row_on_screen
```

Example:

```
ROW=2
```

UNLINK:

Description: Use the UNLINK keyword to remove a resource from a view or from an aggregate object without having to delete the view or aggregate and rebuild them.

Syntax:

```
Syntax: UNLINK
```

Example:

```
View=myview
Agg=myagg,unlink
```

USER_DATA:

Description: The USER_DATA keyword is used to set the DisplayResourceUserData field for an object. The contents of this field is displayed in the NetView management console Data2 field. You can set the User Data field for any resource which has the DisplayResourceUserData field defined, and you can set the DisplayResourceUserData field to any value.

BLDVIEWS enables the following NetView variables to be coded anywhere in the user data text. The variables are:

netid()	VTAM network identifier
domain()	Current NetView domain
opid()	NetView operator ID or task ID
cursys()	Current operating system name
vtam()	VTAM version and release
netview()	NetView version and release
mvslevel()	MVS version and release
opsystem()	Type of operating system
sysplex()	Name of the MVS sysplex

Syntax:

```
Syntax: USER_DATA='user_data'
```

Example:

```
USER_DATA=Call x45108 for support
```

Usage Notes:

- Single quotation marks (') or double quotation marks (") can be used as a delimiter.
- This function cannot be used if Remote Console support is used, because they occupy the DisplayResourceUserData field in RODM.

User Statuses:

Description: Use the following user status keywords to set the corresponding bits in the UserStatus field:

- Mark
- Automation in progress
- Suspend

The MARK keyword is used to set or clear the mark bit in the UserStatus field for any resource which has the UserStatus field defined. The resource must already exist in RODM. If you want to create an object, you must first code the control statements to create the resource and then code the control statements to update the resource.

The AUTO_IN_PROGRESS keyword is used to set or clear the Automation in Progress bit in the UserStatus field for any resource in RODM that has the UserStatus field defined. The resource must already exist in RODM. If you want to create an object, you must first code the control statements to create the resource and then code the control statements to update the resource.

The SUSPEND and SUSPEND_WITH_AUTO_CLEAR keywords can be used to set the Suspend bit in the UserStatus field for any resource in RODM that has the UserStatus field defined. The resource must already exist in RODM. If you want to create an object, you must first code the control statements to create the resource and then code the control statements to update the resource.

Setting the suspend User status flag disables the resource from aggregation and participation in exception views. If you set the Suspend bit in the UserStatus field with the SUSPEND_WITH_AUTO_CLEAR keyword, GMFHS automatically clears the Suspend bit when the resource returns to a satisfactory state. If you set the Suspend bit in the UserStatus field with the SUSPEND keyword, you must manually clear the Suspend bit from NetView management console or use BLDVIEWS.

The state of the UserStatus bits can be displayed in the Resource Information pop-up window.

Generic Commands:

Description: BLDVIEWS can be used to set generic commands for objects. With the NetView management console generic commands function, a NetView management console operator can select a resource and issue one of the following generic commands:
- Current Status (DisplayStatusCommandText)
- Activate (ActivateCommandText)
- Inactivate (DeactivateCommandText)
- Recycle (RecycleCommandText)

The actual command to be issued is retrieved from fields on the object. For example, the command text for the Activate command is retrieved from the ActivateCommandText field.

Syntax:

```
ACTivate='activate_command'
DEACTivate'=deactivate_command'
RECYcle='recycle_command'
DISPlay='display_command'
```

Example:

```
ACTIVATE='BRG LINK NAME=%NAME%'
DISPLAY=BRG QUERY NAME=%NAME%
```

Usage Notes:
- For MultiSystem Manager token ring resources, BLDVIEWS appends the commands with an operator ID of FLCVBLDV and a unique correlator value.

- BLDVIEWS provides the following control variables that can be coded anywhere in the command text:

%NAME%	Substituted with the name of the resource. This variable is supported for all resources.
%RANDOM%	Substituted with a 1-5 digit random number. This variable is supported for all resources.
%SEGMENT%	Substituted with the segment number where the resource resides. This variable is supported only for the resources that are identified by the following control statements: – IP_HOST – INTERFACE
%IPADDRESS%	Substituted with the internet address of the resource. This variable is only supported for the NWSERVER control statement.

- BLDVIEWS enables the following NetView variables to be coded anywhere in the command text:

netid()	VTAM network identifier
domain()	Current NetView domain
opid()	NetView operator or task ID
cursys()	Current operating system name
vtam()	VTAM version and release
netview()	NetView version and release
mvslevel()	MVS version and release
opsystem()	Type of operating system
sysplex()	Name of the MVS sysplex

- Single quotation marks (') or double quotation marks (") can be used as a delimiter.

Defining Wildcard Characters

Use the WILDCARD control statement to define wildcard characters.

WILDCARD Control Statement:

Description: The WILDCARD control statement defines wildcard characters to use when coding a wild card pattern matching name on a RESOURCE control statement.

wildcard_single_character and *wildcard_string* are special characters used to define a wild card pattern matching name.

THe default value of *wildcard_single_character* and *wildcard_string* is an *

WildCard

```
                                        ┌─,──────────────────┐
                                        ▼                    │
►►──WildCard=(wildcard_single_character───┬─────────────────┬─┴─)──────────────►◄
                                          └─wildcard_string─┘
```

Parameters:

`wildcard single_character and wildcard_string`
Special characters used to define a wild card pattern matching name.

Any character can be specified, except for a comma ',' or an equal sign '='. The default character for both *wildcard_single_character* and *wildcard_string* an asterisk (*).

wildcard_single_character

Used when you want to perform a wild card match on 1 character. The *wildcard_single_character* coded in a position in the wild card pattern matching name always matches the character in the corresponding position in the resource name.

If *wildcard_string* is specified in a wild card pattern matching name in any position but the last position of the wild card pattern matching name, it is treated as a *wildcard_single_character* (perform a wild card match on 1 character in the position specified).

wildcard_string

Used when you wish to perform a wild card match on a remaining string of characters at the end of a resource name. The *wildcard_string* coded at the end of a wild card pattern matching name always matches the string of characters at the corresponding position in the resource name.

If *wildcard_string* is specified in a wild card pattern matching name in any position except the last position of the wild card pattern matching name, it is treated as a *wildcard_single_character* (perform a wild card match on 1 character in the position specified).

Examples: Assume WILDCARD=(?,*) for the following pattern matching examples:

Pattern Match Example	Matches Found
BRIDGE=A001B*	Matches all bridge resources whose names begin with A001B.
BRIDGE=????B001	Matches all bridge resources whose names are eight characters in length and end with B001).
SEGMENT=?C?0	Matches all segment resources whose names have a C in position 2 and a 0 in position 4).
ADP=??SERV*	Matches all adapter resources whose names are 6 or more characters in length, and have SERV in positions 3 through 6).
ADP=??PRINTER0?	Matches all adapter resources whose names are 10 or 11 characters in length, and have PRINTER0 in positions 3 through 10).

Assume WILDCARD = *,* (the default), for the following pattern matching examples:

Pattern Match Example	Matches Found
BRIDGE=A001B*	Matches all bridge resources whose names begin with A001B.
BRIDGE=**B001**	Matches all bridge resources whose names are eight characters in length and end with B001.
SEGMENT=*C*0	Matches all segment resources whose names have a C in position 2 and a 0 in position 4.

ADP=SERV***	Matches all adapter resources whose names are 6 or more characters in length, and have SERV in positions 3 through 6.
ADP=PRINTER0***	Matches all adapter resources whose names are 10 or more characters in length, and have PRINTER0 in positions 3 through 10.

Selective Control Statements

You can use the following selective control statements to be more selective in specifying resources to be processed by BLDVIEWS, or to specify common information to be used to locate certain resources in RODM. Wildcard is not valid for these types of control statements.

Service Point Control Statement:

Description: The service point control statement specifies the service point that manages the resources on the control statements following the service point control statement. With the service point control statement, you can be more selective in specifying resources to be processed by BLDVIEWS. This service point name can be overridden on individual control statements using the SPname keyword.

The following service point control statements are enabled:
- LANSPname
- IPSPname

Syntax:

ATMSPname

```
              ┌─ALL───────────┐
►►──ATMSPname=─┴─service_point─┴──────────────────────────────────────────►◄
```

Parameters:

service_point
The 1-8 character VTAM PU, LU, CP name, or the IP host name.

All
Include resources from ALL service points. All is the default

Usage Notes:
- If you code control statements with a name of ALL or a resource name, the resources that get processed depend on whether a service point control statement was previously specified.
- If no prior service point statement was specified and a resource control statement was coded with ALL for a resource name, all resources are processed.
- If no prior service point statement was specified and a resource control statement was coded with a wild card resource name, all resources that match the wild card name are processed.
- If a prior service point statement was specified and a resource control statement was coded with ALL for a resource name, all resources managed by that service point are processed.
- If a prior service point statement was specified and a resource control statement was coded with a wild card resource name, all resources that match the wild card name, and are managed by that service point, are processed.

SNA_DOMAIN Control Statement:

Description: The SNA_DOMAIN control statement specifies the SNA domain that owns the SNA topology manager resources on the control statements following the SNA_DOMAIN control statement. The SNA domain is used to locate the SNA topology manager resources in RODM. The default is ALL. This value can be overridden on individual control statements using the SNA_DOMAIN keyword.

Syntax:

SNA_DOMAIN

```
►►──SNA_DOMAIN=sna_domain_name──────────────────────────────────────────────►◄
```

Parameters:

sna_domain_name

The 1–17 character SNA domain name in the format of network.host_pu_name.

network

VTAM network name 1–8 characters (NETID parameter in VTAM start list ATCSTRxx)

host_pu_name

VTAM host PU name 1–8 characters (HOSTPU parameter in VTAM start list ATCSTRxx)

If sna_domain_name is not specified, then the local SNA domain is used (domain where BLDVIEWS is run).

The following SNA Topology Resources require an SNA Domain:
- VTAM Major Node (MAJNODE control statement)
- CDRMs (CDRM control statement)
- CDRSCs (CDRSC control statement)
- Logical Units (LU control statement)
- Logical Unit Groups (LU_GROUP control statement)

The SNA Domain Name can also be specified on those control statements using the SNA_DOMAIN keyword in which case it overrides the SNA_DOMAIN control statement.

View Control Statements

The following view control statements define the types of views to be created.

VIEW Control Statement:

Description: The VIEW control statement defines a network view which contains the resources on the control statements that follow it.

Syntax:

VIEW

```
►►──VIEW=view_name──┬───────────────────────┬──┬────────────────────┬────────►
                    └─,ANNOTATION=annotation─┘  └─,LAYOUT=layout_type─┘
```

```
                                              ┌─,CREATE=YES─────────┐
►─┬──────────────────────────────┬────────────┼─────────────────────┼──────────────────────►◄
  └─,LAYOUT_WIDTH=layout_width───┘            └─,CREATE=──┬─BUILD─┬─┘
                                                          ├─NO────┤
                                                          └─YES───┘
```

Parameters:

`view_name`
The 1–32 character name of the view. It is the MyName of the network view object.

annotation
The 1–32 character view annotation.

layout
The 1 digit layout type specification which determines the layout algorithm to use when building the view. BLDVIEWS supports layout types; however, only the following values are used:
- 6 - hierarchical (default for CREATE=YES)
- 7 - ellipse
- 9 - grid

layout_width
An integer which specifies how many resource objects appear horizontally on one line in the view. The default value is 0, which results in a grid closely resembling a square. This is only applicable for layout type 9.

CREATE
Specifies which action to perform on the resource specified.

YES Create a new object for this view. The old object is deleted, if it exists.

YES is the default.

NO Do not create a new object for this view. Update the existing object. If the object does not exist, an error occurs.

BUILD
Create a new object for this view if it does not exist. If it does exist, update the object.

EVIEW Control Statement:

Description: The EVIEW control statement defines an exception view.

Syntax:

EVIEW

```
►►──EVIEW=view_name──┬─────────────────────────────┬──────────────────────────────────►
                     └─,ANNOTATION=annotation──────┘

►─┬──────────────────────────────┬─┬──────────────────────────────────┬───────────────►
  └─,LAYOUT_WIDTH=layout_width───┘ └─,EVIEW_NAME=exceptionviewname────┘
```

```
►──┬────────────────────┬──┬──────────────────┬──┬────────────────────┬────────────►
   └─,DSF=──┬─IGNORE─┬──┘  └─,MF=──┬─IGNORE─┬─┘  └─,SAPF=──┬─IGNORE─┬─┘
            └─NOXCPT─┘             ├─OFF────┤              ├─OFF────┤
                                   └─ON─────┘              └─ON─────┘

                                                   ┌─,CREATE=YES─────────┐
►──┬────────────────────┬──┬────────────────────┬──┴─────────────────────┴─────►◄
   └─,TIF=──┬─IGNORE─┬──┘  └─,AIPF=──┬─IGNORE─┬─┘  └─,CREATE=──┬─BUILD─┬─┘
            ├─OFF────┤               ├─OFF────┤                ├─NO────┤
            └─ON─────┘               └─ON─────┘                └─YES───┘
```

Parameters:

`view_name`
The 1–32 character name of the view. It is the MyName of the Exception View object.

`annotation`
The 1–32 character view annotation.

`layout_width`
An integer which specifies how many resource objects appear horizontally on one line in the view. The default value is 0 which results in a grid closely resembling a square. This is only applicable for layout type 9.

`exceptionviewname`
The 1–8 character name associated with the exception view. Resource objects that have this name in their ExceptionViewList field are considered candidates for display in the associated exception view. This field must be unique for all exception views. If not specified, BLDVIEWS creates a unique exceptionviewname.

`DSF`
Specifies the DisplayStatus filter options for the exception view.

IGNORE
No filtering is done and the DisplayStatus is ignored. Objects with a mapped display status of XCPT or NOXCPT are candidates for this view.

NOXCPT
Filter out all objects that do *not* map to an exception status.

`MF` Specifies the UserStatus Mark filter options for the exception view.

IGNORE
No filtering. UserStatus Mark is ignored.

ON Filters out objects that have the UserStatus bit for Mark ON. If an object has this UserStatus bit on, it is not in the view.

OFF Filters out objects that have the UserStatus bit for Mark OFF. If an object has this UserStatus bit off, it is not in the view.

`SAPF`
Specifies the UserStatus SNA Alert Pending filter options for the exception view.

IGNORE
No filtering. UserStatus SNA Alert Pending is ignored.

ON Filters out objects that have the UserStatus bit for SNA Alert Pending ON. If an object has this UserStatus bit on, it is not in the view.

OFF Filters out objects that have the UserStatus bit for SNA Alert Pending OFF. If an object has this UserStatus bit off, it is not in the view.

`TIF`

Specifies the UserStatus Threshold Inconsistency filter options for the Exception View.

IGNORE
No filtering. UserStatus Threshold Inconsistency is ignored.

ON Filters out objects that have the UserStatus bit for Threshold Inconsistency ON. If an object has this UserStatus bit on, it is not in the view.

OFF Filters out objects that have the UserStatus bit Threshold Inconsistency OFF. If an object has this UserStatus bit off, it is not in the view.

`AIPF`

Specifies the UserStatus Automation In Progress filter options for the Exception View.

IGNORE
No filtering. UserStatus Automation In Progress is ignored.

ON Filters out objects that have the UserStatus bit for Automation In Progress ON. If an object has this UserStatus bit on, it is not in the view.

OFF Filters out objects that have the UserStatus bit Automation In Progress OFF. If an object has this UserStatus bit off, it is not in the view.

`CREATE`

Specifies which action to perform on the resource specified.

YES Create a new object for this view. The old object is deleted, if it exists.

YES is the default.

NO Do not create a new object for this view. Uupdate the existing object. If the object does not exist, an error occurs.

BUILD
Create a new object for this view if it does not exist. If it does exist, update the object.

PVIEW Control Statement:

Description: The PVIEW control statement defines a configuration peer view, which contains the resources on the control statements that follow it.

Syntax:

PVIEW

```
►►──PVIEW=view_name──┬──────────────────────┬──┬──────────────────────────────────┬──────────►
                     └─,LAYOUT=layout_type──┘  └─,LAYOUT_WIDTH=layout_width─────────┘
```

```
   ┌─,CREATE=YES──────────┐
►──┼──────────────────────┼──────────────────────────────────────────────────►◄
   └─,CREATE=──┬─BUILD─┬──┘
               ├─NO────┤
               └─YES───┘
```

Parameters:

view_name
The 1–32 character name of the view. It is the MyName of the configuration peer view object.

layout
The 1 digit layout type specification which determines the layout algorithm to use when building the view. BLDVIEWS supports all layout types; however, only the following values are used:

- 6 - hierarchical (default for CREATE=YES)
- 7 - ellipse
- 9 - grid

layout_width
An integer which specifies how many resource objects appear horizontally on one line in the view. The default value is 0 which results in a grid closely resembling a square. This is only applicable for layout type 9.

CREATE
Specifies which action to perform on the resource specified.

YES Create a new object for this view. The old object is deleted, if it exists.

YES is the default.

NO Do not create a new object for this view. Update the existing object. If the object does not exist, an error occurs.

BUILD
Create a new object for this view if it does not exist. If it does exist, update the object.

BBVIEW Control Statement:

Description: The BBVIEW control statement defines a configuration backbone view, which contains the resources on the control statements that follow it.

Syntax:

BBVIEW

```
►►──BBVIEW=view_name──┬────────────────────────┬──┬──────────────────────┬──────►
                      └─,ANNOTATION=annotation─┘  └─,LAYOUT=layout_type──┘

                                            ┌─,CREATE=YES──────────┐
►──┬────────────────────────────────┬──┼──────────────────────┼─────────────►◄
   └─,LAYOUT_WIDTH=layout_width─────┘  └─,CREATE=──┬─BUILD─┬──┘
                                                   ├─NO────┤
                                                   └─YES───┘
```

Parameters:

`view_name`
The 1–32 character name of the view. It is the MyName of the configuration backbone view object.

`annotation`
The 1–32 character view annotation.

`layout`
The 1 digit layout type specification which determines the layout algorithm to use when building the view. BLDVIEWS supports all layout types; however, only the following values are used:
- 1 - Radial Layout by link type (default for CREATE=YES)
- 6 - hierarchical
- 7 - ellipse
- 9 - grid

`layout_width`
An integer which specifies how many resources appear horizontally on one line in the view. The default value is 0, which results in a grid closely resembling a square. This is only applicable for layout type 9.

CREATE
Specifies which action to perform on the resource specified.

YES Create a new object for this view. The old object is deleted, if it exists.

YES is the default.

NO Do not create a new object for this view. Update the existing object. If the object does not exist, an error occurs.

BUILD
Create a new object for this view if it does not exist. If it does exist, update the object.

LCVIEW Control Statement:

Description: The LCVIEW control statement defines a Configuration Logical Connectvity View which contains the resources on the control statements that follow it.

Syntax:

LCVIEW

```
►►──LCVIEW=view_name──┬─────────────────────┬──┬─────────────────────────────────┬──────────►
                      └─,LAYOUT=layout_type─┘  └─,LAYOUT_WIDTH=layout_width─┘

   ┌─,CREATE=YES──────────┐
►──┼──────────────────────┼──────────────────────────────────────────────────────────────►◄
   └─,CREATE=──┬─BUILD─┬──┘
               ├─NO────┤
               └─YES───┘
```

Parameters:

`view_name`
The 1–32 character name of the view. It is the MyName of the configuration logical connectivity view object.

`layout`
The 1 digit layout type specification which determines the layout algorithm to use when building the view. BLDVIEWS supports all layout types; however, only the following values are used:

- 1 - Radial Layout by link type (default for CREATE=YES)
- 6 - hierarchical
- 7 - ellipse
- 9 - grid

`layout_width`
An integer which specifies how many resources appear horizontally on one line in the view. The value is 0, which results in a grid closely resembling a square. This is only applicable for layout type 9.

`CREATE`
Specifies which action to perform on the resource specified.

YES Create a new object for this view. The old object is deleted, if it exists.

YES is the default.

NO Do not create a new object for this view. Update the existing object. If the object does not exist, an error occurs.

BUILD
Create a new object for this view if it does not exist. If it does exist, update the object.

PCVIEW Control Statement:

Description: The PCVIEW control statement defines a configuration physical connectivity view, which contains the resources on the control statements that follow it.

Syntax:

PCVIEW

```
►►──PCVIEW=view_name──┬──────────────────────┬──┬──────────────────────────────────┬──────────►
                      └─,LAYOUT=layout_type──┘  └─,LAYOUT_WIDTH=layout_width──────┘

   ┌─,CREATE=YES──────────┐
►──┼──────────────────────┼───────────────────────────────────────────────────────────────►◄
   └─,CREATE=──┬─BUILD─┬──┘
               ├─NO────┤
               └─YES───┘
```

Parameters:

`view_name`
The 1–32 character name of the view. It is the MyName of the configuration physical connectivity view object.

`layout`
The 1 digit layout type specification which determines the layout algorithm to use when building the view. BLDVIEWS supports all layout types; however, only the following values are used:

- 1 - Radial Layout by link type (default for CREATE=YES)
- 6 - hierarchical

- 7 - ellipse
- 9 - grid

layout_width
An integer which specifies how many resources appear horizontally on one line in the view. The value is 0, which results in a grid closely resembling a square. This is only applicable for layout type 9.

CREATE
Specifies which action to perform on the resource specified.

YES Create a new object for this view. The old object is deleted, if it exists.

YES is the default.

NO Do not create a new object for this view. Update the existing object. If the object does not exist, an error occurs.

BUILD
Create a new object for this view if it does not exist. If it does exist, update the object.

MDLVIEW Control Statement:

Description: The MDLVIEW control statement defines a more detail logical view, which contains the resources on the control statements that follow it.

Syntax:

MDLVIEW

```
►►──MDLVIEW=view_name──┬─────────────────────┬──┬───────────────────────────────┬──────►
                       └─,LAYOUT=layout_type─┘  └─,LAYOUT_WIDTH=layout_width─┘


   ┌─,CREATE=YES─────────┐
►──┼─────────────────────┼──────────────────────────────────────────────────────────►◄
   └─,CREATE=──┬─BUILD─┬─┘
               ├─NO────┤
               └─YES───┘
```

Parameters:

view_name
The 1–32 character name of the view. It is the MyName of the more detail logical view object.

layout
The 1 digit layout type specification which determines the layout algorithm to use when building the view. BLDVIEWS supports all layout types; however, only the following values are used:
- 1 - Radial Layout by link type (default for CREATE=YES)
- 6 - hierarchical
- 7 - ellipse
- 9 - grid

layout_width
An integer which specifies how many resources appear horizontally on one line in the view. The value is 0, which results in a grid closely resembling a square. This is only applicable for layout type 9.

`CREATE`
Specifies which action to perform on the resource specified.

YES Create a new object for this view. The old object is deleted, if it exists.

YES is the default.

NO Do not create a new object for this view. Update the existing object. If the object does not exist, an error occurs.

BUILD
Create a new object for this view if it does not exist. If it does exist, update the object.

MDPVIEW Control Statement:

Description: The MDPVIEW control statement defines a more detail physical view, which contains the resources on the control statements that follow it.

Syntax:

MDPVIEW

```
►►──MDPVIEW=view_name──┬──────────────────────┬──┬──────────────────────────────┬──────►
                       └─,LAYOUT=layout_type──┘  └─,LAYOUT_WIDTH=layout_width──┘

   ┌─,CREATE=YES──────────┐
►──┼──────────────────────┼──────────────────────────────────────────────────────────►◄
   └─,CREATE=──┬─BUILD─┬──┘
               ├─NO────┤
               └─YES───┘
```

Parameters:

`view_name`
The 1–32 character name of the view. It is the MyName of the more detail physical view object.

`layout`
The 1 digit layout type specification which determines the layout algorithm to use when building the view. BLDVIEWS supports all layout types; however, only the following values are used:
- 1 - Radial Layout by link type (default for CREATE=YES)
- 6 - hierarchical
- 7 - ellipse
- 9 - grid

`layout_width`
An integer which specifies how many resources appear horizontally on one line in the view. The value is 0, which results in a grid closely resembling a square. This is only applicable for layout type 9.

`CREATE`
Specifies which action to perform on the resource specified.

YES Create a new object for this view. The old object is deleted, if it exists.

YES is the default.

NO Do not create a new object for this view. Update the existing object. If the object does not exist, an error occurs.

BUILD
Create a new object for this view if it does not exist. If it does exist, update the object.

Resource Control Statements

The following resource control statements specify the resources to be processed by BLDVIEWS.

CDRM Control Statement:

Description: The CDRM control statement specifies the VTAM CDRM resource to be processed.

Syntax:

CDRM

```
►►──CDRM=──┬─name─┬──┬────────────────────────────┬──────────────────────────►
           └─ALL──┘  └─,SNA_DOMAIN=sna_domain_name─┘

►─┬──────────────────────────────────┬─┬─────────────────────┬───────────────►
  └─,AGGPRI=aggregation_priority─────┘ ├─,CONSOLE=command────┤
                                       └─,USER_DATA=user_data┘

►─┬───────────────────┬──┬──────────────────────────────┬──┬─────────┬───────►
  └─,CORRELATER=text──┘  └─,DISPLAY_STATUS=status_integer┘  └─,UNLINK─┘

►─┬───────────────────┬──┬─────────────────────────┬──┬──────────────────┬───►
  └─,ROW=row_on_view──┘  └─,COLUMN=column_on_view──┘  └─,MARK=──┬─OFF─┬─┘
                                                               └─ON──┘

►─┬─────────────────────────────────┬──┬──────────────────────────┬──────────►
  └─,AUTO_IN_PROGRESS=──┬─OFF─┬─────┘  └─,SUSPEND=──┬─OFF─┬───────┘
                        └─ON──┘                     └─ON──┘

►─┬──────────────────────────────────────┬───────────────────────────────────►◄
  └─,SUSPEND_WITH_AUTO_CLEAR=──┬─OFF─┬───┘
                               └─ON──┘
```

Parameters:

`name`
The 1–17 character VTAM CDRM name in the format of: snaNetID.snaNodeName. ALL or a wild card name can be specified.

`sna_domain_name`
specifies the VTAM SNA domain that owns the CDRM resource. This overrides the value specified on the SNA_DOMAIN control statement. The format of the name is network.domain.

See "Common Control Statement Parameters" on page 592 for a description of the other supported keywords.

CDRSC Control Statement:

Description: The CDRSC control statement specifies the VTAM CDRSC resource to be processed.

Syntax:

CDRSC

```
►►──CDRSC=──┬─name─┬──┬──────────────────────────────┬──────────────────────────►
            └─ALL──┘  └─,SNA_DOMAIN=sna_domain_name─┘

►──┬────────────────────────────────────┬──┬─────────────────────────┬──────────►
   └─,AGGPRI=aggregation_priority─┘       ├─,CONSOLE=command────────┤
                                          └─,USER_DATA=user_data──┘

►──┬──────────────────────┬──┬─────────────────────────────────┬──┬──────────┬──►
   └─,CORRELATER=text─┘     └─,DISPLAY_STATUS=status_integer─┘  └─,UNLINK─┘

►──┬─────────────────────┬──┬─────────────────────────────┬──┬─────────────────┬─►
   └─,ROW=row_on_view─┘     └─,COLUMN=column_on_view─┘      └─,MARK=──┬─OFF─┬─┘
                                                                       └─ON──┘

►──┬─────────────────────────────────┬──┬─────────────────────────┬─────────────►
   └─,AUTO_IN_PROGRESS=──┬─OFF─┬─┘      └─,SUSPEND=──┬─OFF─┬─┘
                         └─ON──┘                     └─ON──┘

►──┬──────────────────────────────────────────┬─────────────────────────────────►◄
   └─,SUSPEND_WITH_AUTO_CLEAR=──┬─OFF─┬─┘
                                └─ON──┘
```

Parameters:

name
The 1–17 character VTAM CDRSC name in the format of snaNetID.snaNodeName. The network portion of the CDRSC name might be omitted for those CDRSCS which were not defined with a NETID parameter. ALL or a wild card name can be specified.

sna_domain_name
Specifies the VTAM SNA domain that owns the CDRM resource. This overrides the value specified on the SNA_DOMAIN control statement. The format of the name is *network.domain*.

See "Common Control Statement Parameters" on page 592 for a description of the other supported keywords.

CIRCUIT Control Statement:

Description: The CIRCUIT control statement specifies the Circuit resource to be processed. This includes APPN Transmission Group circuits connected to Type 2.1 nodes, APPN Transmission Group circuits connected to Composite Nodes, APPN Transmission Group circuits connected to NTRI-like nodes, APPN Transmission Group interdomain circuits, APPN Transmission Group intersubnetwork circuits, and Subarea Transmission Group Circuits.

Syntax:

CIRCUIT

```
►►──CIRCUIT=──┬─name─┬──┬──────────────────────────┬─────────────────────────────────►
              └─ALL──┘  └─,TYPE=──┬─APPN_TG──────┬─┘
                                  ├─CN───────────┤
                                  ├─INTER_DOMAIN─┤
                                  ├─INTER_SUBNET─┤
                                  ├─NTRI─────────┤
                                  └─SUBAREA_TG───┘

►──┬─────────────────────────────────────┬──┬───────────────────────────┬────────────►
   ├─,AGGPRI=aggregation_priority────────┤  ├─,CONSOLE=command──────────┤
   └─,AGGTHRESH=(xxx,yyy,zzz)────────────┘  └─,USER_DATA=user_data──────┘

►──┬──────────────────────┬──┬────────────────────────────────┬──┬──────────┬────────►
   └─,CORRELATER=text─────┘  └─,DISPLAY_STATUS=status_integer─┘  └─,UNLINK──┘

►──┬──────────────────────┬──┬──────────────────────────┬──┬─────────────────────┬───►
   └─,ROW=row_on_view─────┘  └─,COLUMN=column_on_view───┘  └─,MARK=──┬─OFF─┬────┘
                                                                    └─ON──┘

►──┬──────────────────────────────────┬──┬─────────────────────────┬─────────────────►
   └─,AUTO_IN_PROGRESS=──┬─OFF─┬──────┘  └─,SUSPEND=──┬─OFF─┬──────┘
                         └─ON──┘                      └─ON──┘

►──┬─────────────────────────────────────────┬───────────────────────────────────────►◄
   └─,SUSPEND_WITH_AUTO_CLEAR=──┬─OFF─┬──────┘
                                └─ON──┘
```

Parameters:

`name`
The SNA Circuit name in the format of snaNetID.circuitID. The name is in the same format that is displayed in NetView management console for the resource (DisplayResourceName). ALL or a wild card name can be specified.

TYPE
Specifies the type of circuit.

APPN_TG
circuit connected to Type 2.1 nodes

CN circuit connected to Composite Nodes

NTRI circuit connected to NTRI-like Nodes

INTER_SUBNET
intersubnetwork circuits

INTER_DOMAIN
interdomain circuits

SUBAREA_TG
subarea transmission group circuits

See “Common Control Statement Parameters” on page 592 for a description of the other supported keywords.

CLUSTER Control Statement:

Description: The CLUSTER control statement specifies the MultiSystem Manager or APPN Cluster aggregate resource to be processed. This aggregate can contain 1 or more network aggregates.

Syntax:

CLUSTER

```
►►──CLUSTER=──┬─cluster_name─┬──┬───────────┬──┬────────────────────────────┬──────►
              └─ALL──────────┘  └─TYPE=APPN─┘  └─,AGGTHRESH=(xxx,yyy,zzz)─┘

►──┬──────────────────────────┬──┬───────────────────┬────────────────────────────►
   ├─,CONSOLE=command─────────┤  └─,CORRELATER=text─┘
   └─,USER_DATA=user_data─────┘

►──┬──────────────────────────────────┬──┬──────────┬──┬─────────────────────┬────►
   └─,DISPLAY_STATUS=status_integer─┘  └─,UNLINK─┘  └─,ROW=row_on_view─┘

►──┬─────────────────────────────┬──┬────────────────┬────────────────────────────►
   └─,COLUMN=column_on_view─┘  └─,MARK=──┬─OFF─┬─┘
                                          └─ON──┘

►──┬───────────────────────────────────┬──┬──────────────────────────┬────────────►
   └─,AUTO_IN_PROGRESS=──┬─OFF─┬─┘  └─,SUSPEND=──┬─OFF─┬─┘
                          └─ON──┘                 └─ON──┘

►──┬─────────────────────────────────────────┬────────────────────────────────────►◄
   └─,SUSPEND_WITH_AUTO_CLEAR=──┬─OFF─┬─┘
                                 └─ON──┘
```

Parameters:

cluster_name
The name of the CLUSTER aggregate resource.

For TYPE=APPN, the name is in the format of snaNetid.systemId which is the network identifier of the NetView domain where the topology manager is located.

ALL or a wild card name can be specified.

TYPE
Specifies the type of CLUSTER aggregate resource. The following value is valid:

APPN	APPN

See "Common Control Statement Parameters" on page 592 for a description of the other supported keywords.

DOMAIN Control Statement:

Description: The DOMAIN control statement specifies the APPN Domain resource to be processed.

Syntax:

DOMAIN

```
►►──DOMAIN=──┬─name─┬──┬────────────────────────────┬──┬──────────────────────────┬──►
             └─ALL──┘  └─,AGGTHRESH=(xxx,yyy,zzz)─┘  ├─,CONSOLE=command─────────┤
                                                      └─,USER_DATA=user_data─────┘
```

```
►─┬──────────────────┬──┬──────────────────────────────────┬──┬─────────┬──────────────────►
  └─,CORRELATER=text─┘  └─,DISPLAY_STATUS=status_integer─┘  └─,UNLINK─┘

►─┬──────────────────────┬──┬────────────────────────────┬──┬─────────────────┬───────────►
  └─,ROW=row_on_view─────┘  └─,COLUMN=column_on_view─────┘  └─,MARK=─┬─OFF─┬─┘
                                                                     └─ON──┘

►─┬─────────────────────────────────┬──┬──────────────────────┬──────────────────────────►
  └─,AUTO_IN_PROGRESS=─┬─OFF─┬──────┘  └─,SUSPEND=─┬─OFF─┬────┘
                       └─ON──┘                     └─ON──┘

►─┬─────────────────────────────────────────┬─────────────────────────────────────────────►◄
  └─,SUSPEND_WITH_AUTO_CLEAR=─┬─OFF─┬───────┘
                              └─ON──┘
```

Parameters:

name
The 1–17 character APPN network node domain name in the format: snaNetID.snaNodeName. ALL or a wild card name can be specified.

See "Common Control Statement Parameters" on page 592 for a description of the other supported keywords.

ENODE Control Statement:

Description: The ENODE control statement specifies the APPN End Node resource to be processed.

Syntax:

ENODE

```
►►──ENODE=─┬─name─┬──┬───────────────────────────────────┬────────────────────────────────►
           └─ALL──┘  └─,AGGPRI=aggregration_priority─────┘

►─┬─────────────────────────┬──┬──────────────────┬──────────────────────────────────────►
  ├─,CONSOLE=command────────┤  └─,CORRELATER=text─┘
  └─,USER_DATA=user_data────┘

►─┬──────────────────────────────────┬──┬─────────┬──┬──────────────────┬────────────────►
  └─,DISPLAY_STATUS=status_integer───┘  └─,UNLINK─┘  └─,ROW=row_on_view─┘

►─┬────────────────────────────┬──┬─────────────────┬────────────────────────────────────►
  └─,COLUMN=column_on_view─────┘  └─,MARK=─┬─OFF─┬─┘
                                           └─ON──┘

►─┬─────────────────────────────────┬──┬──────────────────────┬──────────────────────────►
  └─,AUTO_IN_PROGRESS=─┬─OFF─┬──────┘  └─,SUSPEND=─┬─OFF─┬────┘
                       └─ON──┘                     └─ON──┘

►─┬─────────────────────────────────────────┬─────────────────────────────────────────────►◄
  └─,SUSPEND_WITH_AUTO_CLEAR=─┬─OFF─┬───────┘
                              └─ON──┘
```

Parameters:

name
The 1–17 character SNA end node resource name in the format: snaNetID.snaNodeName. ALL or a wild card name can be specified.

See "Common Control Statement Parameters" on page 592 for a description of the other supported keywords.

GENERIC Control Statement:

Description: The GENERIC control statement specifies a Real or Aggregate resource from a user-defined class to be processed.

Note: The BLDVIEWS interpreter (FLCVBLDV) and the RODM Collection Manager interpreter (FLCV2RCM) treat the *name* parameter slightly differently. See the following description of the *name* parameter.

Syntax:

GENERIC

```
►►──GENERIC=name──┬──────────────────┬──┬─────────────────────────────────┬──────────────►
                  └─,CLASS=classname─┘  └─,AGGPRI=aggregation_priority─┘

►──┬──────────────────────────┬──┬────────────────────────────────┬──────────────────────►
   └─,AGGTHRESH=(xxx,yyy,zzz)─┘  └─,ACTivate=activate_command─┘

►──┬──────────────────────────────────┬──┬──────────────────────────┬────────────────────►
   └─,DEACTivate=deactivate_command─┘  └─,RECYcle=recycle_command─┘

►──┬──────────────────────────┬──┬──────────────────────┬──┬────────────────────┬───────►
   └─,DISPlay=display_command─┘  ├─,CONSOLE=command─────┤  └─,CORRELATER=text─┘
                                 └─,USER_DATA=user_data─┘

►──┬──────────────────────────────────┬──┬─────────┬──┬──────────────────┬──────────────►
   └─,DISPLAY_STATUS=status_integer─┘  └─,UNLINK─┘  └─,ROW=row_on_view─┘

►──┬───────────────────────────┬──┬─────────────────┬───────────────────────────────────►
   └─,COLUMN=column_on_view─┘  └─,MARK=──┬─OFF─┬─┘
                                         └─ON──┘

►──┬──────────────────────────────┬──┬─────────────────────┬────────────────────────────►
   └─,AUTO_IN_PROGRESS=──┬─OFF─┬─┘  └─,SUSPEND=──┬─OFF─┬─┘
                         └─ON──┘                 └─ON──┘

►──┬────────────────────────────────────┬──────────────────────────────────────────────►◄
   └─,SUSPEND_WITH_AUTO_CLEAR=──┬─OFF─┬─┘
                                └─ON──┘
```

Parameters:

name
The BLDVIEWS interpreter (FLCVBLDV) searches both the RODM MyName and the DisplayResourceName attributes for matching object names. The RODM Collection Manager interpreter (FLCV2RCM) searches only the RODM DisplayResourceName attribute for matching names.

classname
The name of the RODM class containing the object.

See “Common Control Statement Parameters” on page 592 for a description of the other supported keywords.

GW_NCP Control Statement:

Description: The GW_NCP control statement specifies the SNA Communication Controller node resource functioning as gateways to be processed.

Syntax:

GW_NCP

```
►►──GW_NCP=──┬─name─┬──┬──────────────────────────────────┬──────────────────►
             └─ALL──┘  └─,AGGPRI=aggregration_priority─┘

►──┬───────────────────────┬──┬─────────────────────┬─────────────────────────►
   ├─,CONSOLE=command──────┤  └─,CORRELATER=text─┘
   └─,USER_DATA=user_data─┘

►──┬─────────────────────────────────┬──┬─────────┬──┬──────────────────┬─────►
   └─,DISPLAY_STATUS=status_integer─┘  └─,UNLINK─┘  └─,ROW=row_on_view─┘

►──┬────────────────────────┬──┬───────────────────┬──────────────────────────►
   └─,COLUMN=column_on_view─┘  └─,MARK=──┬─OFF─┬─┘
                                         └─ON──┘

►──┬─────────────────────────────────┬──┬─────────────────────┬───────────────►
   └─,AUTO_IN_PROGRESS=──┬─OFF─┬─┘  └─,SUSPEND=──┬─OFF─┬─┘
                         └─ON──┘                 └─ON──┘

►──┬───────────────────────────────────────┬──────────────────────────────────►◄
   └─,SUSPEND_WITH_AUTO_CLEAR=──┬─OFF─┬─┘
                                └─ON──┘
```

Parameters:

name

The 1–17 character SNA Communication Controller node in the format of: snaNetID.snaNodeName. ALL or a wild card name can be specified.

See “Common Control Statement Parameters” on page 592 for a description of the other supported keywords.

HOST_NODE Control Statement:

Description: The HOST_NODE control statement specifies the SNA Type 5 Node resource to be processed. A Type 5 node is a subarea node containing an SSCP and having hierarchical control of Type 4 nodes and peripheral nodes.

Syntax:

HOST_NODE

```
►►──HOST_NODE=──┬─name─┬──┬──────────────────────────────────┬────────────────►
                └─ALL──┘  └─,AGGPRI=aggregration_priority─┘
```

```
►─┬──────────────────────┬──┬───────────────────┬─────────────────────►
  ├─,CONSOLE=command─────┤  └─,CORRELATER=text──┘
  └─,USER_DATA=user_data─┘

►─┬────────────────────────────────┬──┬─────────┬──┬───────────────────┬──►
  └─,DISPLAY_STATUS=status_integer─┘  └─,UNLINK─┘  └─,ROW=row_on_view──┘

►─┬──────────────────────────┬──┬──────────────────┬──────────────────►
  └─,COLUMN=column_on_view───┘  └─,MARK=──┬─OFF─┬──┘
                                          └─ON──┘

►─┬────────────────────────────────┬──┬─────────────────────┬─────────►
  └─,AUTO_IN_PROGRESS=──┬─OFF─┬────┘  └─,SUSPEND=──┬─OFF─┬──┘
                        └─ON──┘                    └─ON──┘

►─┬─────────────────────────────────────────┬────────────────────────►◄
  └─,SUSPEND_WITH_AUTO_CLEAR=──┬─OFF─┬──────┘
                               └─ON──┘
```

Parameters:

name
The 1–17 character SNA Host Node name in the format of: snaNetID.snaNodeName. ALL or a wild card name can be specified.

See "Common Control Statement Parameters" on page 592 for a description of the other supported keywords.

IC_NODE Control Statement:

Description: The IC_NODE control statement specifies the SNA Interchange Node resources to be processed.

Syntax:

IC_NODE

```
                          ┌─,TYPE=ALL─────────┐
►►──IC_NODE=──┬─name─┬────┼───────────────────┼──┬──────────────────────────────────┬──►
              └─ALL──┘    └─,TYPE=──┬─ALL─┬───┘  └─,AGGPRI=aggregration_priority────┘
                                    ├─CDS─┤
                                    ├─EBN─┤
                                    ├─GWS─┤
                                    ├─ICN─┤
                                    ├─IRS─┤
                                    └─PBN─┘

►─┬──────────────────────┬──┬───────────────────┬─────────────────────►
  ├─,CONSOLE=command─────┤  └─,CORRELATER=text──┘
  └─,USER_DATA=user_data─┘

►─┬────────────────────────────────┬──┬─────────┬──┬───────────────────┬──►
  └─,DISPLAY_STATUS=status_integer─┘  └─,UNLINK─┘  └─,ROW=row_on_view──┘
```

```
►─┬──────────────────────┬─┬────────────────┬───────────────────────────────────►
  └─,COLUMN=column_on_view─┘ └─,MARK=─┬─OFF─┬─┘
                                      └─ON──┘

►─┬──────────────────────────────┬─┬────────────────────┬───────────────────────►
  └─,AUTO_IN_PROGRESS=─┬─OFF─┬─┘ └─,SUSPEND=─┬─OFF─┬─┘
                       └─ON──┘               └─ON──┘

►─┬─────────────────────────────────────┬──────────────────────────────────────►◄
  └─,SUSPEND_WITH_AUTO_CLEAR=─┬─OFF─┬─┘
                              └─ON──┘
```

Parameters:

name
The 1–17 character SNA Interchange Node name in the format of: snaNetID.snaNodeName. ALL or a wild card name can be specified.

TYPE
specifies the type of network node resource. The values are :

GWS	Nodes with gateway services
CDS	Nodes with central directory services
IRS	Nodes with intermediate routing services
PBN	Nodes which are peripheral border nodes
EBN	Nodes which are extended border nodes
ALL	all IC_NODE types (default)

TYPE
Is ignored when you specify an exact resource name. It is supported only for a name of ALL or a wild card name.

See “Common Control Statement Parameters” on page 592 for a description of the other supported keywords.

INTERFACE Control Statement:

Description: The INTERFACE control statement specifies the MultiSystem Manager TCP/IP adapter resource to be processed.

Syntax:

INTERFACE

```
►►──INTERFACE=─┬─adapter_name─┬─┬──────────────────────────────┬────────────────►
               └─ALL──────────┘ └─,SPname=─┬─ALL───────────┬─┘
                                           └─service_point─┘

   ┌─,SEGMENT=ALL─────┐
►──┼──────────────────┼─┬────────────────────────────────┬─────────────────────►
   └─,SEGMENT=segment─┘ └─,AGGPRI=aggregation_priority─┘
```

```
►─┬──────────────────────────────────┬──┬──────────────────────────────────────┬──────────────►
  └─,ACTivate=activate_command───────┘  └─,DEACTivate=deactivate_command───────┘

►─┬──────────────────────────────┬──┬──────────────────────────┬───────────────────────────────►
  └─,RECYcle=recycle_command─────┘  └─,DISPlay=display_command─┘

►─┬─────────────────────────┬──┬───────────────────────┬──────────────────────────────────────►
  ├─,CONSOLE=command────────┤  └─,CORRELATER=text──────┘
  └─,USER_DATA=user_data────┘

►─┬──────────────────────────────────┬──┬──────────┬──┬──────────────────────┬────────────────►
  └─,DISPLAY_STATUS=status_integer───┘  └─,UNLINK──┘  └─,ROW=row_on_view─────┘

►─┬──────────────────────────────┬──┬──────────────────┬──────────────────────────────────────►
  └─,COLUMN=column_on_view───────┘  └─,MARK=──┬─OFF─┬──┘
                                              └─ON──┘

►─┬─────────────────────────────────┬──┬─────────────────────┬────────────────────────────────►
  └─,AUTO_IN_PROGRESS=──┬─OFF─┬─────┘  └─,SUSPEND=──┬─OFF─┬──┘
                        └─ON──┘                     └─ON──┘

►─┬─────────────────────────────────────────┬─────────────────────────────────────────────────►◄
  └─,SUSPEND_WITH_AUTO_CLEAR=──┬─OFF─┬──────┘
                               └─ON──┘
```

Parameters:

`adapter_name`
The TCP/IP interface adapter name.

ALL or a wild card name can be specified.

`segment_name`
The segment name.

ALL can be specified and is the default.

See "Common Control Statement Parameters" on page 592 for a description of the other supported keywords.

IP_BRIDGE Control Statement:

Description: The IP_BRIDGE control statement specifies the MultiSystem Manager TCP/IP bridge aggregate resource to be processed.

Syntax:

IP_BRIDGE

```
►►──IP_BRIDGE=──┬─name─┬──┬──────────────────────────────┬────────────────────────────────────►
                └─ALL──┘  └─,SPname=──┬─ALL───────────┬──┘
                                      └─service_point─┘
```

```
►──┬──────────────────────────┬──┬──────────────────────────────┬──────────────────►
   └─,AGGTHRESH=(xxx,yyy,zzz)─┘  └─,ACTivate=activate_command─┘

►──┬──────────────────────────────────┬──┬──────────────────────────┬──────────────►
   └─,DEACTivate=deactivate_command─┘  └─,RECYcle=recycle_command─┘

►──┬──────────────────────────┬──┬────────────────────────┬──┬─────────────────────┬──►
   └─,DISPlay=display_command─┘  ├─,CONSOLE=command───────┤  └─,CORRELATER=text──┘
                                 └─,USER_DATA=user_data───┘

►──┬─────────────────────────────────┬──┬──────────┬──┬────────────────────┬──────►
   └─,DISPLAY_STATUS=status_integer─┘  └─,UNLINK─┘  └─,ROW=row_on_view──┘

►──┬──────────────────────────┬──┬─────────────────┬───────────────────────────────►
   └─,COLUMN=column_on_view─┘  └─,MARK=──┬─OFF─┬─┘
                                          └─ON──┘

►──┬─────────────────────────────────┬──┬────────────────────────┬────────────────►
   └─,AUTO_IN_PROGRESS=──┬─OFF─┬───┘  └─,SUSPEND=──┬─OFF─┬────┘
                          └─ON──┘                    └─ON──┘

►──┬──────────────────────────────────────┬──────────────────────────────────────►◄
   └─,SUSPEND_WITH_AUTO_CLEAR=──┬─OFF─┬──┘
                                 └─ON──┘
```

Parameters:

name
The TCP/IP bridge name. ALL or a wild card name can be specified.

See "Common Control Statement Parameters" on page 592 for a description of the other supported keywords.

IP_HOST Control Statement:

Description: The IP_HOST control statement specifies the MultiSystem Manager TCP/IP host aggregate resource to be processed.

Syntax:

IP_HOST

```
►►──IP_HOST=──┬─name─┬──┬────────────────────────────┬────────────────────────────►
              └─ALL──┘  └─,SPname=──┬─ALL───────────┬─┘
                                     └─service_point─┘

►──┬──────────────────────────┬──┬──────────────────────────────┬──────────────────►
   └─,AGGTHRESH=(xxx,yyy,zzz)─┘  └─,ACTivate=activate_command─┘

►──┬──────────────────────────────────┬──┬──────────────────────────┬──────────────►
   └─,DEACTivate=deactivate_command─┘  └─,RECYcle=recycle_command─┘

►──┬──────────────────────────┬──┬────────────────────────┬──┬─────────────────────┬──►
   └─,DISPlay=display_command─┘  ├─,CONSOLE=command───────┤  └─,CORRELATER=text──┘
                                 └─,USER_DATA=user_data───┘
```

```
►─┬────────────────────────────────┬──┬──────────┬──┬───────────────────┬──────────►
  └─,DISPLAY_STATUS=status_integer─┘  └─,UNLINK─┘  └─,ROW=row_on_view─┘

►─┬───────────────────────────┬──┬─────────────────┬───────────────────────────────►
  └─,COLUMN=column_on_view─┘  └─,MARK=──┬─OFF─┬─┘
                                        └─ON──┘

►─┬─────────────────────────────────┬──┬─────────────────────────┬─────────────────►
  └─,AUTO_IN_PROGRESS=──┬─OFF─┬─┘  └─,SUSPEND=──┬─OFF─┬─┘
                        └─ON──┘                 └─ON──┘

►─┬────────────────────────────────────────┬──────────────────────────────────────►◄
  └─,SUSPEND_WITH_AUTO_CLEAR=──┬─OFF─┬─┘
                               └─ON──┘
```

Parameters:

name
The TCP/IP host name. ALL or a wild card name can be specified.

See "Common Control Statement Parameters" on page 592 for a description of the other supported keywords.

IP_HUB Control Statement:

Description: The IP_HUB control statement specifies the MultiSystem Manager TCP/IP hub aggregate resource to be processed.

Syntax:

IP_HUB

```
►►──IP_HUB=──┬─name─┬──┬────────────────────────────────┬─────────────────────────►
             └─ALL──┘  └─,SPname=──┬─ALL───────────┬─┘
                                   └─service_point─┘

►─┬───────────────────────────┬──┬──────────────────────────────┬─────────────────►
  └─,AGGTHRESH=(xxx,yyy,zzz)─┘  └─,ACTivate=activate_command─┘

►─┬───────────────────────────────────┬──┬──────────────────────────┬─────────────►
  └─,DEACTivate=deactivate_command─┘  └─,RECYcle=recycle_command─┘

►─┬───────────────────────────┬──┬─────────────────────────┬──┬──────────────────┬─►
  └─,DISPlay=display_command─┘  ├─,CONSOLE=command────────┤  └─,CORRELATER=text─┘
                                └─,USER_DATA=user_data──┘

►─┬────────────────────────────────┬──┬──────────┬──┬───────────────────┬──────────►
  └─,DISPLAY_STATUS=status_integer─┘  └─,UNLINK─┘  └─,ROW=row_on_view─┘

►─┬───────────────────────────┬──┬─────────────────┬───────────────────────────────►
  └─,COLUMN=column_on_view─┘  └─,MARK=──┬─OFF─┬─┘
                                        └─ON──┘

►─┬─────────────────────────────────┬──┬─────────────────────────┬─────────────────►
  └─,AUTO_IN_PROGRESS=──┬─OFF─┬─┘  └─,SUSPEND=──┬─OFF─┬─┘
                        └─ON──┘                 └─ON──┘
```

```
►─┬────────────────────────────────────┬────────────────────────────────────►◄
  └─,SUSPEND_WITH_AUTO_CLEAR=─┬─OFF─┬─┘
                              └─ON──┘
```

Parameters:

name
The TCP/IP hub name. ALL or a wild card name can be specified.

See "Common Control Statement Parameters" on page 592 for a description of the other supported keywords.

IP_LINK Control Statement:

Description: The IP_LINK control statement specifies the MultiSystem Manager TCP/IP interface link aggregate resource to be processed.

Syntax:

IP_LINK

```
►►─IP_LINK=─┬─name─┬──┬─────────────────────────────┬───────────────────────►
            └─ALL──┘  └─,SPname=─┬─ALL───────────┬─┘
                                 └─service_point─┘

►─┬───────────────────────────┬──┬─────────────────────────────┬────────────►
  └─,AGGTHRESH=(xxx,yyy,zzz)─┘  └─,ACTivate=activate_command─┘

►─┬───────────────────────────────────┬──┬─────────────────────────┬────────►
  └─,DEACTivate=deactivate_command─┘  └─,RECYcle=recycle_command─┘

►─┬───────────────────────────┬──┬──────────────────────────┬──┬────────────────────┬──►
  └─,DISPlay=display_command─┘  ├─,CONSOLE=command─────────┤  └─,CORRELATER=text─┘
                                └─,USER_DATA=user_data─────┘

►─┬────────────────────────────────┬──┬──────────┬──┬────────────────────┬──►
  └─,DISPLAY_STATUS=status_integer─┘  └─,UNLINK─┘  └─,ROW=row_on_view─┘

►─┬──────────────────────────┬──┬─────────────────────┬─────────────────────►
  └─,COLUMN=column_on_view─┘  └─,MARK=─┬─OFF─┬─┘
                                       └─ON──┘

►─┬─────────────────────────────┬──┬─────────────────────┬──────────────────►
  └─,AUTO_IN_PROGRESS=─┬─OFF─┬─┘  └─,SUSPEND=─┬─OFF─┬─┘
                       └─ON──┘                └─ON──┘

►─┬────────────────────────────────────┬────────────────────────────────────►◄
  └─,SUSPEND_WITH_AUTO_CLEAR=─┬─OFF─┬─┘
                              └─ON──┘
```

Parameters:

name
The TCP/IP Link name. ALL or a wild card name can be specified.

See "Common Control Statement Parameters" on page 592 for a description of the other supported keywords.

IP_LOCATION Control Statement:

Description: The IP_LOCATION control statement specifies the MultiSystem Manager TCP/IP location resource to be processed.

Syntax:

IP_LOCATION

```
►►──IP_LOCATION=──┬─name─┬──┬─────────────────────────────┬──────────────────────►
                  └─ALL──┘  └─,SPname=──┬─ALL───────────┬─┘
                                        └─service_point─┘

►──┬──────────────────────────┬──┬─────────────────────────┬──┬──────────────────┬──►
   └─,AGGTHRESH=(xxx,yyy,zzz)─┘  ├─,CONSOLE=command────────┤  └─,CORRELATER=text─┘
                                 └─,USER_DATA=user_data────┘

►──┬─────────────────────────────────┬──┬─────────┬──┬───────────────────┬───────►
   └─,DISPLAY_STATUS=status_integer──┘  └─,UNLINK─┘  └─,ROW=row_on_view──┘

►──┬─────────────────────────┬──┬──────────────────┬─────────────────────────────►
   └─,COLUMN=column_on_view──┘  └─,MARK=──┬─OFF─┬──┘
                                          └─ON──┘

►──┬──────────────────────────────┬──┬─────────────────────┬─────────────────────►
   └─,AUTO_IN_PROGRESS=──┬─OFF─┬──┘  └─,SUSPEND=──┬─OFF─┬──┘
                         └─ON──┘                  └─ON──┘

►──┬────────────────────────────────────┬──────────────────────────────────────►◄
   └─,SUSPEND_WITH_AUTO_CLEAR=──┬─OFF─┬──┘
                                └─ON──┘
```

Parameters:

name
The TCP/IP location name. ALL or a wild card name can be specified.

See "Common Control Statement Parameters" on page 592 for a description of the other supported keywords.

IP_ROUTER Control Statement:

Description: The IP_ROUTER control statement specifies the MultiSystem Manager TCP/IP router aggregate resource to be processed.

Syntax:

IP_ROUTER

```
►►──IP_ROUTER=──┬─name─┬──┬─────────────────────────────┬────────────────────────►
                └─ALL──┘  └─,SPname=──┬─ALL───────────┬─┘
                                      └─service_point─┘

►──┬──────────────────────────┬──┬─────────────────────────────┬─────────────────►
   └─,AGGTHRESH=(xxx,yyy,zzz)─┘  └─,ACTivate=activate_command──┘
```

```
►─┬──────────────────────────────────┬──┬────────────────────────────┬──────────►
  └─,DEACTivate=deactivate_command───┘  └─,RECYcle=recycle_command───┘

►─┬────────────────────────────┬──┬──────────────────────────┬──┬──────────────────────┬─►
  └─,DISPlay=display_command───┘  ├─,CONSOLE=command─────────┤  └─,CORRELATER=text─────┘
                                  └─,USER_DATA=user_data─────┘

►─┬──────────────────────────────────┬──┬──────────┬──┬───────────────────┬──────►
  └─,DISPLAY_STATUS=status_integer───┘  └─,UNLINK──┘  └─,ROW=row_on_view──┘

►─┬────────────────────────────┬──┬───────────────────┬──────────────────────────►
  └─,COLUMN=column_on_view─────┘  └─,MARK=──┬─OFF─┬───┘
                                            └─ON──┘

►─┬─────────────────────────────────────┬──┬───────────────────────┬─────────────►
  └─,AUTO_IN_PROGRESS=──┬─OFF─┬─────────┘  └─,SUSPEND=──┬─OFF─┬────┘
                        └─ON──┘                         └─ON──┘

►─┬──────────────────────────────────────────┬──────────────────────────────────►◄
  └─,SUSPEND_WITH_AUTO_CLEAR=──┬─OFF─┬───────┘
                               └─ON──┘
```

Parameters:

name
The TCP/IP router name. ALL or a wild card name can be specified.

See "Common Control Statement Parameters" on page 592 for a description of the other supported keywords.

IP_SEGMENT Control Statement:

Description: The IP_SEGMENT control statement specifies the MultiSystem Manager TCP/IP Segment aggregate resource to be processed.

Syntax:

IP_SEGMENT

```
►►─IP_SEGMENT=──┬─name─┬──┬───────────────────────────────────┬──────────────────►
                └─ALL──┘  └─,SPname=──┬─ALL───────────────┬───┘
                                      └─service_point─────┘

►─┬────────────────────────────────┬──┬──────────────────────────┬──┬──────────────────────┬─►
  └─,AGGTHRESH=(xxx,yyy,zzz)───────┘  ├─,CONSOLE=command─────────┤  └─,CORRELATER=text─────┘
                                      └─,USER_DATA=user_data─────┘

►─┬──────────────────────────────────┬──┬──────────┬──┬───────────────────┬──────►
  └─,DISPLAY_STATUS=status_integer───┘  └─,UNLINK──┘  └─,ROW=row_on_view──┘

►─┬────────────────────────────┬──┬───────────────────┬──────────────────────────►
  └─,COLUMN=column_on_view─────┘  └─,MARK=──┬─OFF─┬───┘
                                            └─ON──┘

►─┬─────────────────────────────────────┬──┬───────────────────────┬─────────────►
  └─,AUTO_IN_PROGRESS=──┬─OFF─┬─────────┘  └─,SUSPEND=──┬─OFF─┬────┘
                        └─ON──┘                         └─ON──┘
```

```
►─┬──────────────────────────────────────┬─────────────────────────────────►◄
  └─,SUSPEND_WITH_AUTO_CLEAR=─┬─OFF─┬─┘
                               └─ON──┘
```

Parameters:

name
The TCP/IP segment name. ALL or a wild card name can be specified.

See "Common Control Statement Parameters" on page 592 for a description of the other supported keywords.

IP_SUBNET Control Statement:

Description: The IP_SUBNET control statement specifies the MultiSystem Manager TCP/IP Subnetwork aggregate resource to be processed.

Syntax:

IP_SUBNET

```
►►──IP_SUBNET=─┬─name─┬──┬──────────────────────────────┬───────────────────►
               └─ALL──┘  └─,SPname=─┬─ALL───────────┬─┘
                                     └─service_point─┘

►─┬──────────────────────────┬──┬──────────────────────────┬──┬──────────────────┬─►
  └─,AGGTHRESH=(xxx,yyy,zzz)─┘  ├─,CONSOLE=command─────────┤  └─,CORRELATER=text─┘
                                └─,USER_DATA=user_data─────┘

►─┬─────────────────────────────────┬──┬─────────┬──┬─────────────────────┬──►
  └─,DISPLAY_STATUS=status_integer──┘  └─,UNLINK─┘  └─,ROW=row_on_view────┘

►─┬─────────────────────────────┬──┬────────────────┬────────────────────────►
  └─,COLUMN=column_on_view──────┘  └─,MARK=─┬─OFF─┬─┘
                                             └─ON──┘

►─┬──────────────────────────────┬──┬─────────────────────┬──────────────────►
  └─,AUTO_IN_PROGRESS=─┬─OFF─┬─┘  └─,SUSPEND=─┬─OFF─┬─┘
                        └─ON──┘                 └─ON──┘

►─┬──────────────────────────────────────┬─────────────────────────────────►◄
  └─,SUSPEND_WITH_AUTO_CLEAR=─┬─OFF─┬─┘
                               └─ON──┘
```

Parameters:

name
The TCP/IP Subnetwork name. ALL or a wild card name can be specified.

See "Common Control Statement Parameters" on page 592 for a description of the other supported keywords.

LINE Control Statement:

Description: The LINE control statement specifies the SNA Line resource to be processed.

Syntax:

LINE

```
►►──LINE=──┬─name─┬──┬──────────────────────────────┬──┬───────────────────────┬──────►
           └─ALL──┘  └─,AGGPRI=aggregation_priority─┘  ├─,CONSOLE=command──────┤
                                                        └─,USER_DATA=user_data─┘

►──┬────────────────────┬──┬─────────────────────────────────┬──┬─────────┬──────────►
   └─,CORRELATER=text───┘  └─,DISPLAY_STATUS=status_integer──┘  └─,UNLINK─┘

►──┬───────────────────┬──┬─────────────────────────┬──┬──────────────────┬──────────►
   └─,ROW=row_on_view──┘  └─,COLUMN=column_on_view──┘  └─,MARK=──┬─OFF─┬──┘
                                                                 └─ON──┘

►──┬────────────────────────────────┬──┬──────────────────────────┬──────────────────►
   └─,AUTO_IN_PROGRESS=──┬─OFF─┬────┘  └─,SUSPEND=──┬─OFF─┬───────┘
                         └─ON──┘                    └─ON──┘

►──┬──────────────────────────────────────┬────────────────────────────────────────►◄
   └─,SUSPEND_WITH_AUTO_CLEAR=──┬─OFF─┬───┘
                                └─ON──┘
```

Parameters:

name
The 1–17 character SNA line name in the format of: snaNetID.snaNodeName. ALL or a wild card name can be specified.

See "Common Control Statement Parameters" on page 592 for a description of the other supported keywords.

LLINK Control Statement:

Description: The LLINK control statement specifies the Logical Link resource to be processed.

Syntax:

LLINK

```
►►──LLINK=──┬─name─┬──┬───────────────────────┬──┬──────────────────────────────┬──────►
            └─ALL──┘  └─,TYPE=──┬─ALL──────┬──┘  └─,AGGPRI=aggregation_priority─┘
                                ├─ENODE────┤
                                ├─LNODE────┤
                                ├─NNODE────┤
                                └─UNKNOWN──┘

►──┬───────────────────────┬──┬────────────────────┬─────────────────────────────────►
   ├─,CONSOLE=command──────┤  └─,CORRELATER=text───┘
   └─,USER_DATA=user_data──┘

►──┬─────────────────────────────────┬──┬─────────┬──┬───────────────────┬───────────►
   └─,DISPLAY_STATUS=status_integer──┘  └─,UNLINK─┘  └─,ROW=row_on_view──┘

►──┬─────────────────────────┬──┬──────────────────┬─────────────────────────────────►
   └─,COLUMN=column_on_view──┘  └─,MARK=──┬─OFF─┬──┘
                                          └─ON──┘
```

```
►──┬──────────────────────────────┬──┬─────────────────────┬──────────────────►
   └─,AUTO_IN_PROGRESS=─┬─OFF─┬──┘  └─,SUSPEND=─┬─OFF─┬──┘
                        └─ON──┘                  └─ON──┘

►──┬──────────────────────────────────┬───────────────────────────────────────►◄
   └─,SUSPEND_WITH_AUTO_CLEAR=─┬─OFF─┬─┘
                               └─ON──┘
```

Parameters:

name
The SNA Logical Link resource name in the format: network.resource.link. ALL or a wild card name can be specified.

TYPE
Specifies the type of Logical LInk. TYPE is ignored when you specify an exact resource name. It is supported only for a name of ALL or a wild card name. The values are :

NNODE	Network Node
ENODE	End Node
LNODE	Len Node
UNKNOWN	Logical Link type is unknown
ALL	All logical links

See “Common Control Statement Parameters” on page 592 for a description of the other supported keywords.

LNODE Control Statement:

Description: The LNODE control statement specifies the APPN Len Node resource to be processed.

Syntax:

LNODE

```
►►──LNODE=─┬─name─┬──┬──────────────────────────────────┬────────────────────►
           └─ALL──┘  └─,AGGPRI=aggregation_priority─────┘

►──┬────────────────────────────┬──┬─────────────────────┬───────────────────►
   ├─,CONSOLE=command───────────┤  └─,CORRELATER=text────┘
   └─,USER_DATA=user_data───────┘

►──┬───────────────────────────────────┬──┬──────────┬──┬─────────────────────┬──►
   └─,DISPLAY_STATUS=status_integer────┘  └─,UNLINK──┘  └─,ROW=row_on_view────┘

►──┬───────────────────────────┬──┬─────────────────┬─────────────────────────►
   └─,COLUMN=column_on_view────┘  └─,MARK=─┬─OFF─┬──┘
                                           └─ON──┘

►──┬──────────────────────────────┬──┬─────────────────────┬──────────────────►
   └─,AUTO_IN_PROGRESS=─┬─OFF─┬──┘  └─,SUSPEND=─┬─OFF─┬──┘
                        └─ON──┘                  └─ON──┘
```

```
►─┬──────────────────────────────────┬──────────────────────────────────►◄
  └─,SUSPEND_WITH_AUTO_CLEAR=─┬─OFF─┬─┘
                              └─ON──┘
```

Parameters:

name
: The 1–17 character SNA LEN node resource name in the format: snaNetID.snaNodeName. ALL or a wild card name can be specified.

See "Common Control Statement Parameters" on page 592 for a description of the other supported keywords.

LU Control Statement:

Description: The LU control statement specifies the SNA Logical Unit resource to be processed.

Syntax:

LU

```
►►──LU=─┬─name─┬──┬──────────────────────────────┬─────────────────────►
        └─ALL──┘  └─,SNA_DOMAIN=sna_domain_name─┘

►─┬─────────────────────────────────┬──┬─────────────────────────┬─────►
  └─,AGGPRI=aggregation_priority────┘  ├─,CONSOLE=command────────┤
                                       └─,USER_DATA=user_data────┘

►─┬─────────────────────┬──┬────────────────────────────────┬──┬─────────┬──►
  └─,CORRELATER=text────┘  └─,DISPLAY_STATUS=status_integer─┘  └─,UNLINK─┘

►─┬─────────────────────┬──┬─────────────────────────┬──┬───────────────────┬──►
  └─,ROW=row_on_view────┘  └─,COLUMN=column_on_view──┘  └─,MARK=─┬─OFF─┬───┘
                                                                 └─ON──┘

►─┬───────────────────────────┬──┬──────────────────────┬──────────────►
  └─,AUTO_IN_PROGRESS=─┬─OFF─┬┘  └─,SUSPEND=─┬─OFF─┬────┘
                       └─ON──┘               └─ON──┘

►─┬──────────────────────────────────┬──────────────────────────────────►◄
  └─,SUSPEND_WITH_AUTO_CLEAR=─┬─OFF─┬─┘
                              └─ON──┘
```

Parameters:

name
: The 1–17 character SNA logical unit name in the format of: snaNetID.snaNodeName. ALL or a wild card name can be specified.

sna_domain_name
: Specifies the VTAM SNA domain that owns the Logical Unit resource. This overrides the value specified on the SNA_DOMAIN control statement. The format of the name is network.domain.

See "Common Control Statement Parameters" on page 592 for a description of the other supported keywords.

LU_GROUP Control Statement:

Description: The LU_GROUP control statement specifies the SNA Logical Unit group resources to be processed.

Syntax:

LU_GROUP

```
►►──LU_GROUP=──┬─name─┬──┬─────────────────────────────┬──────────────────────────►
               └─ALL──┘  └─,SNA_DOMAIN=sna_domain_name─┘

►──┬──────────────────────────────┬──┬──────────────────────────┬──────────────────►
   └─,AGGPRI=aggregation_priority─┘  ├─,CONSOLE=command─────────┤
                                     └─,USER_DATA=user_data─────┘

►──┬───────────────────┬──┬───────────────────────────────────┬──┬──────────┬──────►
   └─,CORRELATER=text──┘  └─,DISPLAY_STATUS=status_integer────┘  └─,UNLINK──┘

►──┬──────────────────┬──┬────────────────────────┬──┬────────────────┬────────────►
   └─,ROW=row_on_view─┘  └─,COLUMN=column_on_view─┘  └─,MARK=──┬─OFF─┬┘
                                                              └─ON──┘

►──┬─────────────────────────────┬──┬──────────────────────┬───────────────────────►
   └─,AUTO_IN_PROGRESS=──┬─OFF─┬─┘  └─,SUSPEND=──┬─OFF─┬───┘
                         └─ON──┘                 └─ON──┘

►──┬────────────────────────────────────┬─────────────────────────────────────────►◄
   └─,SUSPEND_WITH_AUTO_CLEAR=──┬─OFF─┬─┘
                                └─ON──┘
```

Parameters:

name
The 1–17 character SNA logical unit group name the format of: luGroupName. ALL or a wild card name can be specified.

sna_domain_name
Specifies the VTAM SNA domain that owns the Logical Unit Group resource. This overrides the value specified on the SNA_DOMAIN control statement. The format of the name is network.domain.

See "Common Control Statement Parameters" on page 592 for a description of the other supported keywords.

MAJNODE Control Statement:

Description: The MAJNODE control statement specifies the VTAM Major Node resource to be processed.

Syntax:

MAJNODE

```
                          ┌─,TYPE=ALL────────┐
►►──MAJNODE=──┬─name─┬────┼──────────────────┼──┬─────────────────────────────┬────►
              └─ALL──┘    └─,TYPE=node_type──┘  └─,SNA_DOMAIN=sna_domain_name─┘
```

```
►─┬──────────────────────────────┬──┬──────────────────────┬──────────────────────────►
  └─,AGGPRI=aggregation_priority─┘  ├─,CONSOLE=command─────┤
                                    └─,USER_DATA=user_data─┘

►─┬──────────────────┬──┬─────────────────────────────────┬──┬─────────┬───────────────►
  └─,CORRELATER=text─┘  └─,DISPLAY_STATUS=status_integer─┘  └─,UNLINK─┘

►─┬─────────────────┬──┬───────────────────────┬──┬─────────────────┬──────────────────►
  └─,ROW=row_on_view─┘  └─,COLUMN=column_on_view─┘  └─,MARK=──┬─OFF─┬─┘
                                                             └─ON──┘

►─┬──────────────────────────────┬──┬──────────────────────┬───────────────────────────►
  └─,AUTO_IN_PROGRESS=──┬─OFF─┬─┘  └─,SUSPEND=──┬─OFF─┬─┘
                        └─ON──┘                 └─ON──┘

►─┬──────────────────────────────────────┬────────────────────────────────────────────►◄
  └─,SUSPEND_WITH_AUTO_CLEAR=──┬─OFF─┬─┘
                               └─ON──┘
```

Parameters:

name

The 1–8 character VTAM Major node name in the format of: snaNodeName. ALL or a wild card name can be specified.

sna_domain_name

specifies the VTAM SNA domain that owns the Major Node resource. This overrides the value specified on the SNA_DOMAIN control statement. The format of the name is network.domain.

TYPE

Specifies the type of VTAM Major Node. The values are :

APPL	Application Major Node
CA	Channel Major Node
CDRM	CDRM Major Node
CDRSC	CDRSC Major Node
LAN	Local Area Network Major Node
LCLNONSNA	Local Non SNA Major Node
LOCALSNA	Local SNA Major Node
LUGROUP	LU Group Major Node
NCP	NCP Major Node
PACKET	Packet Major Node
SWITCHED	Switched Major Node
TRL	Token Ring Lan Major Node
XCA	XCA Major Node
ALL	All Major Node types (default)

See “Common Control Statement Parameters” on page 592 for a description of the other supported keywords.

MIG_DATA_HOST Control Statement:

Description: The MIG_DATA_HOST control statement specifies the SNA Migration Data Host node resource to be processed.

Syntax:

MIG_DATA_HOST

```
►►──MIG_DATA_HOST=──┬─name─┬──┬─────────────────────────────────┬──────────►
                    └─ALL──┘  └─,AGGPRI=aggregation_priority─┘

►──┬─────────────────────────┬──┬────────────────────┬─────────────────────►
   ├─,CONSOLE=command────────┤  └─,CORRELATER=text─┘
   └─,USER_DATA=user_data────┘

►──┬─────────────────────────────────┬──┬─────────┬──┬──────────────────┬──►
   └─,DISPLAY_STATUS=status_integer─┘  └─,UNLINK─┘  └─,ROW=row_on_view─┘

►──┬───────────────────────────┬──┬─────────────────┬──────────────────────►
   └─,COLUMN=column_on_view─┘  └─,MARK=──┬─OFF─┬─┘
                                          └─ON──┘

►──┬─────────────────────────────────┬──┬────────────────────────┬─────────►
   └─,AUTO_IN_PROGRESS=──┬─OFF─┬─┘  └─,SUSPEND=──┬─OFF─┬─┘
                          └─ON──┘                  └─ON──┘

►──┬────────────────────────────────────────┬───────────────────────────►◄
   └─,SUSPEND_WITH_AUTO_CLEAR=──┬─OFF─┬─┘
                                 └─ON──┘
```

Parameters:

name
The 1–17 character SNA Migration Data Host node in the form of network.name. ALL or a wild card name can be specified.

See "Common Control Statement Parameters" on page 592 for a description of the other supported keywords.

NCP Control Statement:

Description: The NCP control statement specifies the SNA Communication Controller node resource to be processed.

Syntax:

NCP

```
►►──NCP=──┬─name─┬──┬─────────────────────┬──┬─────────────────────────────────┬──►
          └─ALL──┘  └─,TYPE=──┬─GW─────┬─┘  └─,AGGPRI=aggregation_priority─┘
                               └─NON_GW─┘

►──┬─────────────────────────┬──┬────────────────────┬─────────────────────►
   ├─,CONSOLE=command────────┤  └─,CORRELATER=text─┘
   └─,USER_DATA=user_data────┘

►──┬─────────────────────────────────┬──┬─────────┬──┬──────────────────┬──►
   └─,DISPLAY_STATUS=status_integer─┘  └─,UNLINK─┘  └─,ROW=row_on_view─┘
```

```
►──┬──────────────────────────┬──┬──────────────────┬──────────────────────────────►
   └─,COLUMN=column_on_view───┘  └─,MARK=──┬─OFF─┬──┘
                                           └─ON──┘

►──┬───────────────────────────────┬──┬─────────────────────────┬──────────────────►
   └─,AUTO_IN_PROGRESS=──┬─OFF─┬───┘  └─,SUSPEND=──┬─OFF─┬──────┘
                         └─ON──┘                   └─ON──┘

►──┬──────────────────────────────────────┬───────────────────────────────────────►◄
   └─,SUSPEND_WITH_AUTO_CLEAR=──┬─OFF─┬───┘
                                └─ON──┘
```

Parameters:

name
The 1–17 character SNA Communication Controller node in the format of: snaNetID.snaNodeName. ALL or a wild card name can be specified.

TYPE
Specifies the type of SNA Communication Controller. TYPE is a required keyword. The values are :

GW	Gateway Communications Controller
NON_GW	Non-Gateway Communications Controller

See "Common Control Statement Parameters" on page 592 for a description of the other supported keywords.

NETWORK Control Statement:

Description: The NETWORK control statement specifies the MultiSystem Manager or APPN Network aggregate resource to be processed. This aggregate represents the network managed by one service point.

Syntax:

NETWORK

```
                          ┌─,SPname=ALL─────────────────────┐
►►──NETWORK=network_name──┼─────────────────────────────────┼──┬─────────────┬────►
                          └─,SPname=──┬─ALL───────────┬─────┘  └─TYPE=APPN───┘
                                      └─service_point─┘

►──┬───────────────────────────┬──┬──────────────────────────┬──┬──────────────────┬──►
   └─,AGGTHRESH=(xxx,yyy,zzz)──┘  ├─,CONSOLE=command─────────┤  └─,CORRELATER=text─┘
                                  └─,USER_DATA=user_data─────┘

►──┬─────────────────────────────────┬──┬──────────┬──┬──────────────────┬─────────►
   └─,DISPLAY_STATUS=status_integer──┘  └─,UNLINK──┘  └─,ROW=row_on_view──┘

►──┬──────────────────────────┬──┬──────────────────┬──────────────────────────────►
   └─,COLUMN=column_on_view───┘  └─,MARK=──┬─OFF─┬──┘
                                           └─ON──┘
```

```
►──┬──────────────────────────────┬──┬────────────────────┬──────────────────────────►
   └─,AUTO_IN_PROGRESS=──┬─OFF─┬──┘  └─,SUSPEND=──┬─OFF─┬─┘
                         └─ON──┘                  └─ON──┘

►──┬──────────────────────────────────┬───────────────────────────────────────────────►◄
   └─,SUSPEND_WITH_AUTO_CLEAR=──┬─OFF─┬┘
                                └─ON──┘
```

Parameters:

network_name
The name of the network aggregate resource.

For TYPE=APPN the name is in the format of snaNetid.n where n is a numeric increment. ALL or a wild card name can be specified.

service_point
The VTAM PU, LU, or CP name for the agent. It is not supported for TYPE=APPN and is ignored.

ALL is the default.

TYPE
Specifies the type of NETWORK aggregate resource. The following value is valid:

APPN	APPN

See "Common Control Statement Parameters" on page 592 for a description of the other supported keywords.

NNODE Control Statement:

Description: The NNODE control statement specifies the APPN Network Node resource to be processed.

Syntax:

NNODE

```
►►──NNODE=──┬─name─┬──┬─────────────┬──┬─────────────────────────────────┬──────────►
            └─ALL──┘  └─,TYPE=──┬─CDS─┤  └─,AGGPRI=aggregation_priority─┘
                                ├─EBN─┤
                                ├─GWS─┤
                                ├─ICN─┤
                                ├─IRS─┤
                                └─PBN─┘

►──┬───────────────────────────┬──┬───────────────────┬────────────────────────────►
   ├─,CONSOLE=command──────────┤  └─,CORRELATER=text──┘
   └─,USER_DATA=user_data──────┘

►──┬─────────────────────────────────┬──┬─────────┬──┬───────────────────┬─────────►
   └─,DISPLAY_STATUS=status_integer──┘  └─,UNLINK─┘  └─,ROW=row_on_view──┘
```

```
►─┬─────────────────────────┬──┬────────────────┬──────────────────────────────────►
  └─,COLUMN=column_on_view──┘  └─,MARK=──┬─OFF─┬─┘
                                         └─ON──┘

►─┬──────────────────────────────┬──┬───────────────────┬──────────────────────────►
  └─,AUTO_IN_PROGRESS=──┬─OFF─┬──┘  └─,SUSPEND=──┬─OFF─┬─┘
                        └─ON──┘                  └─ON──┘

►─┬────────────────────────────────────┬──────────────────────────────────────────►◄
  └─,SUSPEND_WITH_AUTO_CLEAR=──┬─OFF─┬─┘
                               └─ON──┘
```

Parameters:

name
The 1–17 character SNA network node resource name in the format: snaNetID.snaNodeName. ALL or a wild card name can be specified.

TYPE
Specifies the type of network node resource. TYPE is ignored when you specify an exact resource name. It is only supported for a name of ALL or a wild card name. The values are :

GWS	Nodes with gateway services
CDS	Nodes with central directory services
IRS	Nodes with intermediate routing services
PBN	Nodes which are peripheral border nodes
ICN	Nodes which are interchange nodes
EBN	Nodes which are extended border nodes

See “Common Control Statement Parameters” on page 592 for a description of the other supported keywords.

NONSNA Control Statement:

Description: The NONSNA control statement specifies the Non-SNA (GMFHS Managed Real) resource to be processed. You can set the Non-SNA Domain for any resource coded on a NONSNA statement. This links the non-SNA resource to that Non-SNA Domain. The Non-SNA Domain object must exist before the link is created.

Syntax:

NONSNA

```
►►──NONSNA=──┬─nonsna_resource_name─┬──┬─────────────────────────────────┬────────►
             └─ALL──────────────────┘  └─,AGGPRI=aggregation_priority────┘

►─┬──────────────────────────────┬──┬──────────────────────────────────┬──────────►
  └─,ACTivate=activate_command───┘  └─,DEACTivate=deactivate_command───┘

►─┬────────────────────────────┬──┬──────────────────────────┬────────────────────►
  └─,RECYcle=recycle_command───┘  └─,DISPlay=display_command─┘
```

```
►──┬──────────────────────┬──┬──────────────────┬──┬─────────────────────────┬──►
   ├─,CONSOLE=command─────┤  └─,CORRELATER=text─┘  └─,OTHER_DATA=other_data─┘
   └─ ,USER_DATA=user_data┘

►──┬───────────────────────────────┬──┬───────────────────────────────────────┬──►
   └─,DISPLAY_STATUS=status_integer┘  └─,DISPLAY_NAME=display_resource_name─┘

                                     ┌─,QUERYFIELD=DRN────────┐
►──┬────────────────────────────┬──┼────────────────────────┼──────────────────►
   └─,TYPE=display_resource_type┘  └─,QUERYFIELD=──┬─DRN────┤
                                                     └─MYNAME─┘

                                    ┌─,CREATE=NO──────────┐
►──┬────────────────────────┬──┼─────────────────────┼──┬──────────┬──────────►
   └─,DOMAIN=nonsna_domain─┘  └─,CREATE=──┬─BUILD─┤  └─,UNLINK─┘
                                               ├─NO────┤
                                               └─YES───┘

►──┬─────────────────────┬──┬───────────────────────────┬──┬───────────────────┬──►
   └─,ROW=row_on_view───┘  └─,COLUMN=column_on_view───┘  └─,MARK=──┬─OFF─┤
                                                                      └─ON──┘

►──┬──────────────────────────────┬──┬───────────────────┬─────────────────────►
   └─,AUTO_IN_PROGRESS=──┬─OFF─┤  └─,SUSPEND=──┬─OFF─┤
                           └─ON──┘                 └─ON──┘

►──┬──────────────────────────────────┬────────────────────────────────────────►◄
   └─,SUSPEND_WITH_AUTO_CLEAR=──┬─OFF─┤
                                  └─ON──┘
```

Parameters:

nonsna_resource_name
The Non SNA resource name. ALL or a wild card name can be specified for CREATE=NO

DISPLAY_NAME
Specifies the RODM DisplayResourceName for the object. This value is displayed on the NetView management console workstation for the resource instead of the RODM resource_name.

Note: BLDVIEWS provides the %NAME% substitution variable that can be coded anywhere in the value. This can be used to reformat the DisplayResourceName for multiple resources with one control statement.

TYPE
Specifies the type of non-SNA resource. TYPE is required for CREATE=YES and ignored for other values. The TYPE value determines the DisplayResourceType value to set in RODM for the non-SNA object. You can specify any valid non-SNA DisplayResourceType value documented in the *IBM Tivoli NetView for z/OS Data Model Reference.*

QUERYFIELD
Specifies the field to use for RODM object queries from the NONSNA resource class(GMFHS_Managed_Real_Objects_Class). Specifying QUERYFIELD=DRN

retrieves objects using the DisplayResourceName field. Specifying QUERYFIELD=MYNAME retrieves objects using the MyName field. DRN is the default if QUERYFIELD is not specified on the NONSNA control statement.

`DOMAIN`
Specifies the name of the non-SNA Domain resource that you want to link to this resource. The non-SNA Domain resource must exist in RODM.

`CREATE`
Specifies which action to perform on the resource specified.

YES	Create a new object for this resource. The old object is deleted, if it exists.
NO	Do not create a new object for this resource. Instead, update the object. If the object does not exist, an error occurs. NO is the default.
BUILD	Create a new object for this resource if it does not exist. If it does exist, update the object.

See “Common Control Statement Parameters” on page 592 for a description of the other supported keywords.

OTHER Control Statement:

Description: The OTHER control statement specifies a Real or Aggregate resource from a user-defined class to be processed.

Note: The BLDVIEWS interpreter (FLCVBLDV) and the RODM Collection Manager interpreter (FLCV2RCM) treat the *name* parameter slightly differently. See the following description of the *name* parameter.

Syntax:

OTHER

```
►►──OTHER=name──┬───────────────────┬──┬────────────────────────────────┬───────────►
                └─,CLASS=classname──┘  └─,AGGPRI=aggregation_priority──┘

►─┬──────────────────────────┬──┬──────────────────────────────┬────────────────────►
  └─,AGGTHRESH=(xxx,yyy,zzz)─┘  └─,ACTivate=activate_command───┘

►─┬───────────────────────────────────┬──┬───────────────────────────┬──────────────►
  └─,DEACTivate=deactivate_command────┘  └─,RECYcle=recycle_command──┘

►─┬──────────────────────────┬──┬──────────────────────────┬──┬──────────────────┬──►
  └─,DISPlay=display_command─┘  ├─,CONSOLE=command─────────┤  └─,CORRELATER=text─┘
                                └─,USER_DATA=user_data─────┘

►─┬─────────────────────────────────┬──┬─────────┬──┬───────────────────┬───────────►
  └─,DISPLAY_STATUS=status_integer──┘  └─,UNLINK─┘  └─,ROW=row_on_view──┘

►─┬─────────────────────────┬──┬──────────────┬────────────────────────────────────►
  └─,COLUMN=column_on_view──┘  └─,MARK=─┬─OFF─┤
                                        └─ON──┘
```

```
►──┬──────────────────────────────────┬──┬──────────────────────┬──────────────────────►
   └─,AUTO_IN_PROGRESS=──┬─OFF─┬──────┘  └─,SUSPEND=──┬─OFF─┬───┘
                         └─ON──┘                      └─ON──┘

►──┬──────────────────────────────────────┬────────────────────────────────────────────►◄
   └─,SUSPEND_WITH_AUTO_CLEAR=──┬─OFF─┬───┘
                                └─ON──┘
```

Parameters:

name
The BLDVIEWS interpreter (FLCVBLDV) searches both the RODM MyName and the DisplayResourceName attributes for matching object names. The RODM Collection Manager interpreter (FLCV2RCM) searches the RODM MyName attribute only for matching names.

classname
The name of the RODM class containing the object.

See “Common Control Statement Parameters” on page 592 for a description of the other supported keywords.

PU Control Statement:

Description: The PU control statement specifies the SNA Physical Unit resource to be processed.

Syntax:

PU

```
                    ┌─,TYPE=ALL──────────────────┐
►►──PU=──┬─name─┬───┼────────────────────────────┼──┬───────────────────────────────────┬──►
         └─ALL──┘   └─,TYPE=──┬─ALL──────┬───────┘  └─,AGGPRI=aggregation_priority──────┘
                              ├─UNKNOWN──┤
                              └─pu_type──┘

►──┬──────────────────────────┬──┬──────────────────────┬──────────────────────────────────►
   ├─,CONSOLE=command─────────┤  └─,CORRELATER=text─────┘
   └─,USER_DATA=user_data─────┘

►──┬──────────────────────────────────┬──┬──────────┬──┬──────────────────────┬────────────►
   └─,DISPLAY_STATUS=status_integer───┘  └─,UNLINK──┘  └─,ROW=row_on_view─────┘

►──┬──────────────────────────────┬──┬─────────────────────┬───────────────────────────────►
   └─,COLUMN=column_on_view───────┘  └─,MARK=──┬─OFF─┬─────┘
                                               └─ON──┘

►──┬──────────────────────────────────┬──┬──────────────────────┬──────────────────────►
   └─,AUTO_IN_PROGRESS=──┬─OFF─┬──────┘  └─,SUSPEND=──┬─OFF─┬───┘
                         └─ON──┘                      └─ON──┘
```

```
►──┬─────────────────────────────────────┬────────────────────────────────────►◄
   └─,SUSPEND_WITH_AUTO_CLEAR=──┬─OFF─┬──┘
                                └─ON──┘
```

Parameters:

name

The 1–17 character SNA physical unit name in the format of: snaNetID.snaNodeName. ALL or a wild card name can be specified.

TYPE

Specifies the type of SNA Physical Unit. The values are :

1	PU Type 1
2	PU Type 2
2.1	PU Type 2.1
4	PU Type 4
5	PU Type 5
UNKNOWN	PU type is unknown
ALL	all PU types (default)

TYPE

Ignored when you specify an exact resource name. It is only supported for a name of ALL or a wild card name.

See "Common Control Statement Parameters" on page 592 for a description of the other supported keywords.

SNA Control Statement:

Description: The SNA control statement specifies the SNA (GMFHS Shadow) resource to be processed.

Syntax:

SNA

```
►►──SNA=──┬─sna_resource_name─┬──┬────────────────────────┬──┬────────────────────┬──►
          └─ALL───────────────┘  ├─,CONSOLE=command───────┤  └─,CORRELATER=text───┘
                                 └─,USER_DATA=user_data───┘

►──┬────────────────────────┬──┬────────────────────────────────────┬──────────────►
   └─,OTHER_DATA=other_data─┘  └─,TYPE=──┬─sna_resource_type─────┬─┘
                                         └─display_resource_type─┘

                                                    ┌─,CREATE=NO──────────┐
►──┬────────────────────────────────────────────┬───┼─────────────────────┼──┬──────────┬──►
   └─,DISPLAY_NAME=display_resource_name────────┘   └─,CREATE=──┬─BUILD─┬─┘  └─,UNLINK──┘
                                                                ├─NO────┤
                                                                └─YES───┘

►──┬──────────────────┬──┬────────────────────────┬──┬──────────────────┬─────────►
   └─,ROW=row_on_view─┘  └─,COLUMN=column_on_view─┘  └─,MARK=──┬─OFF─┬──┘
                                                               └─ON──┘
```

```
►──┬────────────────────────────────────┬──┬──────────────────────────┬──────────────►
   └─,AUTO_IN_PROGRESS=──┬─OFF─┬────────┘  └─,SUSPEND=──┬─OFF─┬──────┘
                         └─ON──┘                        └─ON──┘

►──┬────────────────────────────────────────┬───────────────────────────────────────►◄
   └─,SUSPEND_WITH_AUTO_CLEAR=──┬─OFF─┬─────┘
                                └─ON──┘
```

Parameters:

`sna_resource_name`

The 1–17 character SNA resource name in the format: network.resource. ALL or a wild card name can be specified for CREATE=NO

`TYPE`

Specifies the type of SNA resource. TYPE is required for CREATE=YES and ignored for other values. The TYPE value determines what DisplayResourceType value to set in RODM for the SNA object. You can specify one of the following values or specify any valid DisplayResourceType value documented in the *IBM Tivoli NetView for z/OS Data Model Reference*.

HOST	DUIXC_RTS_HOST
GATEWAY_NCP	DUIXC_RTS_GATEWAY_NCP
NCP	DUIXC_RTS_PU4
PU4	DUIXC_RTS_PU4
APPL	DUIXC_RTS_APPL
CDRM	DUIXC_RTS_CDRM
CDRSC	DUIXC_RTS_CDRSC
LINK	DUIXC_LTS_GENERIC_LINK
PU21	DUIXC_RTS_PU21
PU20	DUIXC_RTS_PU20
PU1	DUIXC_RTS_PU1
PU	DUIXC_RTS_GENERIC_PU
LU	DUIXC_RTS_LU

`DISPLAY_NAME`

Specifies the RODM DisplayResourceName for the object. This value is displayed on the NetView management console workstation for the resource instead of the sna_resource_name.

Note: BLDVIEWS provides the %NAME% substitution variable which can be coded anywhere in the value. This can be used to reformat the DisplayResourceName for multiple resources with one control statement.

`CREATE`

Specifies which action to perform on the resource specified.

YES	Create a new object for this resource. The object is deleted first if it exists.
NO	Do not create a new object for this resource. Instead, update the object. If the object does not exist, an error occurs. NO is the default.
BUILD	Create a new object for this resource if it does not exist. If it does exist, update the object.

See “Common Control Statement Parameters” on page 592 for a description of the other supported keywords.

SNA_PORT Control Statement:

Description: The SNA_PORT control statement specifies the SNA resource to be processed.

Syntax:

SNA_PORT

```
►►──SNA_PORT=──┬─name─┬──┬──────────────────────────────┬──────────────────►
               └─ALL──┘  └─,AGGPRI=aggregation_priority─┘

►──┬─────────────────────────┬──┬──────────────────┬─────────────────────────►
   ├─,CONSOLE=command────────┤  └─,CORRELATER=text─┘
   └─,USER_DATA=user_data────┘

►──┬───────────────────────────────┬──┬─────────┬──┬──────────────────┬──────►
   └─,DISPLAY_STATUS=status_integer─┘  └─,UNLINK─┘  └─,ROW=row_on_view─┘

►──┬────────────────────────┬──┬────────────────┬─────────────────────────────►
   └─,COLUMN=column_on_view─┘  └─,MARK=──┬─OFF─┬─┘
                                         └─ON──┘

►──┬──────────────────────────────┬──┬─────────────────────┬─────────────────►
   └─,AUTO_IN_PROGRESS=──┬─OFF─┬──┘  └─,SUSPEND=──┬─OFF─┬──┘
                         └─ON──┘                  └─ON──┘

►──┬────────────────────────────────────┬────────────────────────────────────►◄
   └─,SUSPEND_WITH_AUTO_CLEAR=──┬─OFF─┬─┘
                                └─ON──┘
```

Parameters:

name
: The SNA Port resource name in the format: snaNetID.portId. ALL or a wild card name can be specified.

SNALOCALTOPO Control Statement:

Description: The SNALOCALTOPO control statement specifies the APPN SNA Local Topology resource to be processed.

Syntax:

SNALOCALTOPO

```
►►──SNALOCALTOPO=──┬─name─┬──┬──────────────────────────┬────────────────────►
                   └─ALL──┘  └─,AGGTHRESH=(xxx,yyy,zzz)─┘

►──┬─────────────────────────┬──┬──────────────────┬─────────────────────────►
   ├─,CONSOLE=command────────┤  └─,CORRELATER=text─┘
   └─,USER_DATA=user_data────┘
```

```
►─┬─────────────────────────────────┬──┬──────────┬──┬─────────────────────┬──────────►
  └─,DISPLAY_STATUS=status_integer──┘  └─,UNLINK──┘  └─,ROW=row_on_view────┘

►─┬───────────────────────────┬──┬─────────────────┬───────────────────────────────────►
  └─,COLUMN=column_on_view────┘  └─,MARK=──┬─OFF─┬─┘
                                           └─ON──┘

►─┬─────────────────────────────────┬──┬────────────────────┬──────────────────────────►
  └─,AUTO_IN_PROGRESS=──┬─OFF─┬─────┘  └─,SUSPEND=──┬─OFF─┬─┘
                        └─ON──┘                     └─ON──┘

►─┬─────────────────────────────────────────┬──────────────────────────────────────────►◄
  └─,SUSPEND_WITH_AUTO_CLEAR=──┬─OFF─┬──────┘
                               └─ON──┘
```

Parameters:

name
Is the APPN SNA Local Topology resource name in the format of: snaNetID.snaNodeName. ALL or a wild card name can be specified.

See “Common Control Statement Parameters” on page 592 for a description of the other supported keywords.

SYSTEM Control Statement:

Description: The SYSTEM control statement specifies the workstation System aggregate resource to be processed.

Syntax:

SYSTEM

```
►►──SYSTEM=──┬─name─┬──┬──────────────────────────────┬────────────────────────────────►
             └─ALL──┘  └─,AGGTHRESH=(xxx,yyy,zzz)─────┘

►─┬─────────────────────────────────┬──┬───────────────────────────────────────┬───────►
  └─,ACTivate=activate_command──────┘  └─,DEACTivate=deactivate_command────────┘

►─┬───────────────────────────────┬──┬─────────────────────────────────┬───────────────►
  └─,RECYcle=recycle_command──────┘  └─,DISPlay=display_command────────┘

►─┬──────────────────────────────┬──┬──────────────────────┬───────────────────────────►
  ├─,CONSOLE=command─────────────┤  └─,CORRELATER=text─────┘
  └─,USER_DATA=user_data─────────┘

►─┬─────────────────────────────────┬──┬──────────┬──┬─────────────────────┬──────────►
  └─,DISPLAY_STATUS=status_integer──┘  └─,UNLINK──┘  └─,ROW=row_on_view────┘

►─┬───────────────────────────┬──┬─────────────────┬───────────────────────────────────►
  └─,COLUMN=column_on_view────┘  └─,MARK=──┬─OFF─┬─┘
                                           └─ON──┘

►─┬─────────────────────────────────┬──┬────────────────────┬──────────────────────────►
  └─,AUTO_IN_PROGRESS=──┬─OFF─┬─────┘  └─,SUSPEND=──┬─OFF─┬─┘
                        └─ON──┘                     └─ON──┘
```

```
►─┬──────────────────────────────────────┬────────────────────────────────────►◄
  └─,SUSPEND_WITH_AUTO_CLEAR=─┬─OFF─┬─┘
                              └─ON──┘
```

Parameters:

`name`
: The name of the System. The name can be one of the following depending upon the type of workstation:

- Nickname
- Computer name (physical name found in IBMLAN.INI file)
- Mac address
- IPX address

ALL or a wild card name can be specified.

See "Common Control Statement Parameters" on page 592 for a description of the other supported keywords.

TG Control Statement:

Description: The TG control statement specifies the APPN Transmission Group resource to be processed.

Syntax:

TG

```
►►──TG=─┬─name─┬──┬──────────────────────────────────┬──┬──────────────────────────┬────►
        └─ALL──┘  └─,AGGPRI=aggregation_priority─┘  ├─,CONSOLE=command───────┤
                                                    └─,USER_DATA=user_data─┘

►─┬──────────────────┬──┬──────────────────────────────────┬──┬─────────┬──────────►
  └─,CORRELATER=text─┘  └─,DISPLAY_STATUS=status_integer─┘  └─,UNLINK─┘

►─┬──────────────────┬──┬─────────────────────────┬──┬───────────────────┬─────────►
  └─,ROW=row_on_view─┘  └─,COLUMN=column_on_view─┘  └─,MARK=─┬─OFF─┬─┘
                                                            └─ON──┘

►─┬───────────────────────────────┬──┬────────────────────────┬───────────────────►
  └─,AUTO_IN_PROGRESS=─┬─OFF─┬─┘  └─,SUSPEND=─┬─OFF─┬─┘
                       └─ON──┘                 └─ON──┘

►─┬──────────────────────────────────────┬────────────────────────────────────►◄
  └─,SUSPEND_WITH_AUTO_CLEAR=─┬─OFF─┬─┘
                              └─ON──┘
```

Parameters:

`name`
: Is the APPN Transmission Group resource name in one of the following formats:

- snaNetID.snaNodeName.tgn{.adj_snaNetID}.adj_snaNodeName
- snaNetID.vrnNodeName.tgn{.adj_snaNetID}.adj_snaNodeName
- snaNetID.snaNodeName.tgn{.adj_snaNetID}.adj_vrnNodeName

The name is in the same format as displayed from NetView management console for the resource (DisplayResourceName). ALL or a wild card name can be specified.

See "Common Control Statement Parameters" on page 592 for a description of the other supported keywords.

VRN Control Statement:

Description: The VRN control statement specifies the APPN Virtual Routing Node resource to be processed.

Syntax:

VRN

```
►►──VRN=──┬─name─┬──┬──────────────────────────────┬──┬──────────────────────┬──────►
          └─ALL──┘  └─,AGGPRI=aggregation_priority─┘  ├─,CONSOLE=command─────┤
                                                      └─,USER_DATA=user_data─┘

►──┬──────────────────┬──┬────────────────────────────────┬──┬─────────┬──────────────►
   └─,CORRELATER=text─┘  └─,DISPLAY_STATUS=status_integer─┘  └─,UNLINK─┘

►──┬───────────────────┬──┬─────────────────────────┬──┬─────────────────┬────────────►
   └─,ROW=row_on_view──┘  └─,COLUMN=column_on_view──┘  └─,MARK=──┬─OFF─┬─┘
                                                                 └─ON──┘

►──┬─────────────────────────────────┬──┬─────────────────────┬──────────────────────►
   └─,AUTO_IN_PROGRESS=──┬─OFF─┬─────┘  └─,SUSPEND=──┬─OFF─┬──┘
                         └─ON──┘                     └─ON──┘

►──┬──────────────────────────────────────┬────────────────────────────────────────►◄
   └─,SUSPEND_WITH_AUTO_CLEAR=──┬─OFF─┬───┘
                                └─ON──┘
```

Parameters:

name
The 1–17 character SNA Virtual Routing Node resource name in the format: snaNetID.snaNodeName. ALL or a wild card name can be specified.

See "Common Control Statement Parameters" on page 592 for a description of the other supported keywords.

Aggregation Control Statements

The following control statements specify the aggregate resources to be created or updated and the resources that compose the aggregate resource.

AGGregate Control Statement:

Description: The AGGregate control statement specifies the Aggregate (GMFHS Aggregate) resource to be processed.

Syntax:

AGGregate

```
►►──AGGregate=──┬─aggregate_name─┬──┬──────────────────────────────────────┬───────────►
                └─ALL────────────┘  └─,TYPE=──┬─aggregate_resource_type─┬─┘
                                              └─display_resource_type───┘

►──┬──────────────────────────┬──┬─────────────────────────────┬────────────────────────►
   └─,AGGTHRESH=(xxx,yyy,zzz)─┘  └─,ACTivate=activate_command─┘

►──┬─────────────────────────────────┬──┬──────────────────────────┬────────────────────►
   └─,DEACTivate=deactivate_command─┘  └─,RECYcle=recycle_command─┘

►──┬───────────────────────────┬──┬───────────────────────┬──┬─────────────────────┬────►
   └─,DISPlay=display_command─┘  ├─,CONSOLE=command──────┤  └─,CORRELATER=text─┘
                                 └─,USER_DATA=user_data──┘

►──┬─────────────────────────┬──┬──────────────────────────────────┬────────────────────►
   └─,OTHER_DATA=other_data─┘  └─,DISPLAY_STATUS=status_integer─┘

►──┬──────────────────────────────────────────┬──┬──────────┬──┬───────────────────┬────►
   └─,DISPLAY_NAME=display_resource_name─┘  └─,UNLINK─┘  └─,ROW=row_on_view─┘

►──┬───────────────────────────┬──┬──────────────────┬──────────────────────────────────►
   └─,COLUMN=column_on_view─┘  └─,MARK=──┬─OFF─┬─┘
                                          └─ON──┘

                                        ┌─,CREATE=NO──────────┐
►──┬────────────────────────────────┬──┼─────────────────────┼──────────────────────────►
   └─,AUTO_IN_PROGRESS=──┬─OFF─┬─┘  └─,CREATE=──┬─BUILD─┬─┘
                         └─ON──┘                 ├─NO────┤
                                                 └─YES───┘

►──┬─────────────────────┬──┬─────────────────────────────────────┬─────────────────────►◄
   └─,SUSPEND=──┬─OFF─┬─┘  └─,SUSPEND_WITH_AUTO_CLEAR=──┬─OFF─┬─┘
                └─ON──┘                                 └─ON──┘
```

Parameters:

aggregate_name

The aggregate resource name.

ALL or a wild card name can be specified for CREATE=NO

TYPE

Specifies the type of aggregate resource. TYPE is required for CREATE=YES and ignored for other values. The TYPE value determines what DisplayResourceType value to set in RODM for the aggregate object. You can specify one of the following values or specify any valid DisplayResourceType value documented in the RODM Programming Guide.

LAN_CLUSTER

DUIXC_RTN_LAN_NETWORK_AGG

LAN_NETWORK	DUIXC_RTN_LAN_AGG
IP_CLUSTER	DUIXC_RTN_INTERNET_CLUSTER
IP_NETWORK	DUIXC_RTN_INTERNET_MGMT_DOMAIN_AGG
IP_SUBNET	DUIXC_RTN_INTERNET_SUBNET_AGG
IP_SEGMENT	DUIXC_RTN_INTERNET_SEGMENT_AGG'
IP_LOCATION	DUIXC_RTN_INTERNET_LOCATION_AGG
IP_ROUTER	DUIXC_RTN_INTERNET_ROUTER_AGG
IP_HUB	- DUIXC_RTN_INTERNET_HUB_AGG
IP_BRIDGE	- DUIXC_RTN_INTERNET_BRIDGE_AGG
IP_HOST	DUIXC_RTN_INTERNET_HOST_AGG
IP_LINK	DUIXC_RTN_LTN_IP_LINK_AGG
SYSTEM	DUIXC_RTN_OPEN_SYSTEM_AGG
APPN_DOMAIN	DUIXC_RTN_NN_DOMAIN_AGG
APPN_NETWORK	DUIXC_RTN_NN_DOMAIN_NETWORK
APPN_CLUSTER	DUIXC_RTN_NN_DOM_NET_CLUSTER
SNALOCALTOPO	DUIXC_RTN_NN_LOCAL_TOP_AGG
USER	DUIXC_RTN_NODE_AGG_USER1

CREATE

Specifies which action to perform on the resource specified.

YES	Create a new object for this resource. The old object is deleted, if it exists.
NO	Do not create a new object for this resource. Instead update the object. If the object does not exist, an error occurs. NO is the default.
BUILD	Create a new object for this resource if it does not exist. If it does exist, update the object.

See “Common Control Statement Parameters” on page 592 for a description of the other supported keywords.

Note: The AGGregate control statement creates new aggregates or references existing aggregates which belong to the GMFHS_Aggregate_Objects_Class class.

If any AGGChild control statements follow the AGGregate control statement, the resources specified on the AGGChild control statements are linked to the aggregate specified on the AGGregate control statement, unless the AGGCHILD control statements specify UNLINK=YES.

AGGChild Control Statement:

Description: The AGGChild control statement specifies the aggregation children resource that you want linked or unlinked to the aggregate resource on the AGGregate statement that precedes the AGGChild control statements.

Syntax:

AGGChild

```
►►──AGGChild=──┬─child_name─┬──┬──────────────────────────────────┬──────────────►
               └─ALL────────┘  └─,TYPE=──┬─resource──────────────┬─┘
                                         └─OTHER,CLASS=classname─┘

   ┌─,DETAIL_LINK=LOGICAL─────────┐   ┌─,SEGMENT=ALL──────────┐
►──┼──────────────────────────────┼───┼───────────────────────┼──────────────────►
   └─,DETAIL_LINK=──┬─LOGICAL──┬──┘   └─,SEGMENT=──┬─ALL─────┬┘
                    └─PHYSICAL─┘                   └─segment─┘

   ┌─,QUERYFIELD=DRN────────────┐   ┌─,PARENT_CHILD_LINK=NO──────────┐
►──┼────────────────────────────┼───┼────────────────────────────────┼───────────►
   └─,QUERYFIELD=──┬─DRN────┬───┘   └─,PARENT_CHILD_LINK=──┬─NO──┬───┘
                   └─MYNAME─┘                              └─YES─┘

►──┬───────────────────────────────┬──┬────────────────────────────────┬──┬──────────┬──►◄
   └─,SYStem=──┬─ALL─────────┬─────┘  └─,SNA_DOMAIN=sna_domain_name────┘  └─,UNLINK──┘
               └─system_name─┘
```

Parameters:

name
: The name of the resource. The name formats and lengths depend upon the type of resource.

ALL or a wild card name can be specified.

TYPE
: Specifies the type of resource. The types correspond to the specific resource control statements.

- STATION_ADAPTER
- IP_CLUSTER
- IP_NETWORK
- IP_SUBNET
- IP_LOCATION
- INTERFACE
- SYSTEM
- NONSNA
- APPN_CLUSTER
- APPN_NETWORK
- SNALOCALTOPO
- NNODE
- ENODE
- LNODE
- LINE
- SNA_PORT
- DOMAIN
- LLINK
- TG
- APPN_VRN

- APPN_TG_CIRCUIT
- INTER_DOMAIN_CIRCUIT
- INTER_SUBNETWORK_CIRCUIT
- CN_CIRCUIT
- NTRI_CIRCUIT
- SUBAREA_TG_CIRCUIT
- AGG
- APPL_MAJNODE
- CDRSC_MAJNODE
- CDRM_MAJNODE
- LAN_MAJNODE
- LCLNONSNA_MAJNODE
- LOCALSNA_MAJNODE
- LUGROUP_MAJNODE
- NCP_MAJNODE
- PACKET_MAJNODE
- SWITCHED_MAJNODE
- TRL_MAJNODE
- XCA_MAJNODE
- HOST_NODE
- IC_NODE
- MIG_DATA_HOST
- GW_NCP
- NCP_GW
- NCP_NON_GW
- CDRM
- CDRSC
- PU
- LU
- LU_GROUP
- CA_MAJNODE

DETAIL_LINK

Specifies which type of connection to establish between the aggregation child and the aggregate.

LOGICAL	Link the aggregation child to the aggregate with a logical connection (DEFAULT).
PHYSICAL	Link the aggregation child to the aggregate with a physical connection.

segment_name

(STATION_ADAPTER, BRIDGE_ADAPTER, CAU_ADAPTER, or LAN_ADAPTER) segment number (3–4 characters) or segment name (for example, SEGxxxx). ALL can be specified and is the default.

segment_name

(INTERFACE) segment name (1–64 characters) ALL can be specified and is the default.

sna_domain_name
Specifies the VTAM SNA domain that owns the Major Node resource. This overrides the value specified on the SNA_DOMAIN control statement. The format of the name is network.domain.

QUERYFIELD
Specifies the field to use for RODM object queries from the NONSNA resource class(GMFHS_Managed_Real_Objects_Class). Specifying QUERYFIELD=DRN retrieves objects using the DisplayResourceName field. Specifying QUERYFIELD=MYNAME retrieves objects using the MyName field. DRN is the default if QUERYFIELD is not specified on the NONSNA control statement.

PARENT_CHILD_LINK
Enables the option of linking aggregate children to an aggregate parent using null links. The parameter is coded as follows:

PARENT_CHILD_LINK=YES (NO is the default)

See “Common Control Statement Parameters” on page 592 for a description of the other supported keywords.

Running BLDVIEWS

Code the BLDVIEWS control statements which direct BLDVIEWS to build the views and aggregates you specify. The control statements can be coded in a NetView DSIPARM member, a fully qualified cataloged sequential data set (includes PDS specified with a member), or in a REXX stem array and passed to BLDVIEWS using the NetView PIPE command.

Coding Control Statements in a NetView DSIPARM Member

If the control statements are coded in a DSIPARM member, the syntax is:

```
BLDVIEWS dsiparm_member {RODM=rodmname}
                        {TEST=YES|NO}
                        {ECHO=YES|NO}
                        {QUIET=YES|NO}
                        {OPTIMIZE=CPU|STORage}
```

dsiparm_member
The NetView DSIPARM member name which contains the BLDVIEWS control statements.

rodmname
The name of the RODM with which you want to connect. *rodmname* is optional. If it is not specified, the MultiSystem Manager common global FLC_RODMNAME is used.

TEST=YES
Results in BLDVIEWS only syntax checking the control statements. No actions are performed. RODM does not need to be active. The default is TEST=NO.

ECHO=YES
Results in BLDVIEWS displaying the control statements one at a time as they are read, and before they are processed. The default is ECHO=NO.

QUIET=YES
Results in BLDVIEWS suppressing all messages except for error messages. The default is QUIET=NO.

OPTIMIZE

CPU Results in BLDVIEWS saving the results of querying entire classes, in REXX arrays in storage. This is done to reduce cycles that are required to query the classes multiple times during a BLDVIEWS execution. This saves cycles at the expense of using additional storage to keep the data in storage. This is the default. If your storage is constrained, you might have to specify OPTIMIZE=STORage.

STORage Results in BLDVIEWS NOT saving the results of querying entire classes, in REXX arrays in storage. This saves storage at the expense of using more CPU if the resources in those classes are again needed later during the same BLDVIEWS execution.

Coding Control Statements in a fully Qualified Data set

If the control statements are coded in a cataloged data set then the syntax is:

```
BLDVIEWS data_set {RODM=rodmname}
                  {TEST=YES|NO}
                  {ECHO=YES|NO}
                  {QUIET=YES|NO}
                  {OPTIMIZE=CPU|STORage}
```

data_set
The name of a fully qualified cataloged data set which contains the BLDVIEWS control statements. The data set can be a sequential file or a partitioned data set specified with a member.

rodmname
The name of the RODM with which you want to connect. It is optional, if not specified the MultiSystem Manager common global FLC_RODMNAME are used.

TEST=YES
Results in BLDVIEWS only syntax checking the control statements. No actions are performed. RODM does not need to be active. The default is TEST=NO.

ECHO=YES
Results in BLDVIEWS displaying the control statements one at a time as they are read, and before they are processed. The default is ECHO=NO.

QUIET=YES
Results in BLDVIEWS suppressing all messages except for error messages. The default is QUIET=NO.

OPTIMIZE

CPU Results in BLDVIEWS saving the results of querying entire classes, in REXX arrays in storage. This is done to reduce cycles that are required to query the classes multiple times during a BLDVIEWS execution. This saves cycles at the expense of using additional storage to keep the data in storage. This is the default. If you are storage constrained you might have to specify OPTIMIZE=STORage.

STORage Results in BLDVIEWS NOT saving the results of querying entire classes, in REXX arrays in storage. This saves storage at the expense of using more cpu if the resources in those classes are again needed later during the same BLDVIEWS execution.

Examples:

```
BLDVIEWS ESP.NV24.BLDVIEWS(MYVIEWS)

BLDVIEWS ESP.NV24.BLDVIEWS.DATA1
```

Coding Control Statements in REXX Stem Arrays

If the control statements are coded in a REXX stem array, the syntax is:

```
'PIPE STEM stem_array. | COLLECT | NETV BLDVIEWS',
                                   '{RODM=rodmname}',
                                   '{TEST=YES|NO}',
                                   '{ECHO=YES|NO}',
                                   '{QUIET=YES|NO}',
                                   '{OPTIMIZE=CPU|STORage} | ....'
```

`stem_array`
The name of the REXX stem array variable that contains the BLDVIEWS control statements. *stem.array.0* must contain the number of entries in the array.

`rodmname`
The name of the RODM you with which want to connect. It is optional. If not specified, the MultiSystem Manager common global FLC_RODMNAME is used for the rodmname and the common global FLC_RODMAPPL is used for the RODM userid.

If rodmname is specified, then the NetView operator ID of the task running BLDVIEWS is used as the RODM user ID. This user ID must have the appropriate SAF access to RODM.

`TEST=YES`
Results in BLDVIEWS only syntax checking the control statements. No actions are performed. RODM does not need to be active. The default is TEST=NO.

`ECHO=YES`
Results in BLDVIEWS displaying the control statements one at a time as they are read, and before they are processed. The default is ECHO=NO.

`QUIET=YES`
Results in BLDVIEWS suppressing all messages except for error messages. The default is QUIET=NO.

`OPTIMIZE`

CPU
Results in BLDVIEWS saving the results of querying entire classes, in REXX arrays in storage. This is done to reduce cycles that are required to query the classes multiple times during a BLDVIEWS execution. This saves cycles at the expense of using additional storage to keep the data in storage. This is the default. If you are storage constrained you might have to specify OPTIMIZE=STORage.

STORage
Results in BLDVIEWS NOT saving the results of querying entire classes, in REXX arrays in storage. This saves storage at the expense of using more CPU if the resources in those classes are again needed later during the same BLDVIEWS execution.

Example:

```
/* REXX  */

statement.1="VIEW=My_View,ANNOTATION='This is my View',",
statement=2=' CREATE=YES'
statement.3='NONSNA='resource',CREATE=YES,',
       ||'TYPE=DUIXC_RTN_HOST'


statement.0=3
```

```
'PIPE STEM statement. | COLLECT | NETV FLCVBLDV | CONSOLE'
exit
```

BLDVIEWS Control Statement Examples

This section contains examples of coding BLDVIEWS control statements.

BLDVIEWS Example 1:

This example changes the aggregation thresholds for all the MultiSystem Manager cluster and network aggregates for TCP/IP resources. The aggregation thresholds are changed to 25%, 50% and 75%.

```
NETWORK=ALL,AGGTHRESH=(25%,50%,75%),TYPE=IP
CLUSTER=ALL,AGGTHRESH=(25%,50%,75%),TYPE=IP
```

BLDVIEWS Example 2:

This example sets the generic commands in RODM for the MultiSystem Manager TCP/IP routers, hubs, bridges, hosts and adapters. The DisplayStatusCommandText (generic display command) field is set to do an rping. The DisplayResourceUserData (Remote Console) is set to do a TELNETPM.

BLDVIEWS envelopes the commands with `RemoteConsole = # and #`, which correctly sets the DisplayResourceUserData field so that the remote console support can work correctly.

```
IP_ROUTER=ALL,
              DISPLAY='asis rping -n 2 %NAME%',
              CONSOLE='TELNETPM.EXE %NAME%'

IP_HUB=ALL,
              DISPLAY='asis rping -n 2 %NAME%',
              CONSOLE='TELNETPM.EXE %NAME%'

IP_BRIDGE=ALL,
              DISPLAY='asis rping -n 2 %NAME%',
              CONSOLE='TELNETPM.EXE %NAME%'

IP_HOST=ALL,
            DISPLAY='asis rping -n 2 %NAME%',
            CONSOLE='TELNETPM.EXE %NAME%'

INTERFACE=ALL,
           DISPLAY='asis rping -n 2 %NAME%',
           CONSOLE='TELNETPM.EXE %NAME%'
```

BLDVIEWS Example 3:

This example sets the DisplayResourceName for the non-SNA resource mercury.raleigh.ibm.com to Router1.

```
NONSNA=mercury.raleigh.ibm.com,
       DISPLAY_NAME='Router1'
```

BLDVIEWS Example 4:

This example creates a view that contains all bridge aggregate resources managed by service point A19SRVCP.

```
VIEW=GAF_ALLBridgesA,ANNOTATION='All Bridge Aggregates'
LANSPNAME=A19SRVCP
BRIDGE=ALL,TYPE=AGG
```

BLDVIEWS Example 5:

This example creates a view that contains specific bridge and segment resources managed by service point A19SRVCP. This example also sets the aggregation thresholds for the segment aggregates to 20%, 60% and 80%.

```
VIEW=GAF_BLDG_500,ANNOTATION='Building 500'
LANSPNAME=A19SRVCP

BRIDGE=A085C17,TYPE=AGG
BRIDGE=A082C17,TYPE=AGG
BRIDGE=AC15C17,TYPE=AGG
BRIDGE=A056C17,TYPE=AGG
BRIDGE=AC15C16,TYPE=AGG
BRIDGE=A056C16,TYPE=AGG
BRIDGE=AC16B00,TYPE=AGG
BRIDGE=A032C01,TYPE=AGG
BRIDGE=A03B032,TYPE=AGG

SEGMENT=0C16,TYPE=AGG,AGGTHRESH=(20%,60%,80%)
SEGMENT=0056,TYPE=AGG,AGGTHRESH=(20%,60%,80%)
SEGMENT=0C15,TYPE=AGG,AGGTHRESH=(20%,60%,80%)
SEGMENT=0C17,TYPE=AGG,AGGTHRESH=(20%,60%,80%)
SEGMENT=0082,TYPE=AGG,AGGTHRESH=(20%,60%,80%)
SEGMENT=0085,TYPE=AGG,AGGTHRESH=(20%,60%,80%)
SEGMENT=0C01,TYPE=AGG,AGGTHRESH=(20%,60%,80%)
SEGMENT=0032,TYPE=AGG,AGGTHRESH=(20%,60%,80%)
SEGMENT=003B,TYPE=AGG,AGGTHRESH=(20%,60%,80%)
```

BLDVIEWS Example 6:

This example creates a view that contains two new aggregate resources with specific resources.

```
VIEW=GAF_Key_Bridges,ANNOTATION='Key Bridges'
LANSPNAME=A19SRVCP

AGG=GAF_B500_Bridges,type=BRIDGE,
                     AGGTHRESH=(40%,60%,75%),CREATE=YES
AGGCHILD=A085C17,TYPE=BRIDGE_AGG
AGGCHILD=A082C17,TYPE=BRIDGE_AGG
AGGCHILD=AC15C17,TYPE=BRIDGE_AGG
AGGCHILD=A056C17,TYPE=BRIDGE_AGG
AGGCHILD=AC15C16,TYPE=BRIDGE_AGG
AGGCHILD=A056C16,TYPE=BRIDGE_AGG
AGGCHILD=AC16B00,TYPE=BRIDGE_AGG
AGGCHILD=A032C01,TYPE=BRIDGE_AGG
AGGCHILD=A03B032,TYPE=BRIDGE_AGG

AGG=GAF_MS_Bridges,type=BRIDGE,
                   AGGTHRESH=(40%,60%,75%),CREATE=YES
AGGCHILD=AC01B00,TYPE=BRIDGE_AGG
AGGCHILD=AB01B00,TYPE=BRIDGE_AGG
AGGCHILD=AC03B00,TYPE=BRIDGE_AGG
AGGCHILD=AC24B00,TYPE=BRIDGE_AGG
AGGCHILD=AC03B01,TYPE=BRIDGE_AGG
AGGCHILD=AC24B01,TYPE=BRIDGE_AGG
AGGCHILD=AC05B00,TYPE=BRIDGE_AGG
AGGCHILD=AC06B01,TYPE=BRIDGE_AGG
AGGCHILD=A059C05,TYPE=BRIDGE_AGG
AGGCHILD=A059C06,TYPE=BRIDGE_AGG
AGGCHILD=A061C05,TYPE=BRIDGE_AGG
AGGCHILD=A062C05,TYPE=BRIDGE_AGG
AGGCHILD=A062C06,TYPE=BRIDGE_AGG
```

BLDVIEWS Example 7:

This example creates a view with a layout type of 6 (hierarchical) and puts specific resources in the view on the rows that are specified:

```
VIEW=GAF_View_Hier,ANNOTATION='Resources on specific rows',
     LAYOUT=6

LANSPNAME=A19SRVCP
NWSPNAME=A19NWAPU

NONSNA=ITNM.CODEBUST.BUDDY,ROW=1

BRIDGE=A059C06,TYPE=AGG,ROW=2

SEGMENT=0C16,TYPE=AGG,ROW=3

CAU=5A982D60,TYPE=AGG,ROW=4

ADP=GARY,ROW=5

NWSERVER=ESP_A86A,TYPE=IBM_AGENT,ROW=5
```

BLDVIEWS Example 8:

This example unlinks a bridge resource from a view.

```
VIEW=GAF_BLDG_500,CREATE=NO
LANSPNAME=A19SRVCP

BRIDGE=A085C17,TYPE=AGG,UNLINK
```

Deleting Views

This section describes how to delete a view or a group of views beginning with a specified prefix using DELVIEWS.

DELVIEWS Syntax

```
DELVIEWS view_name|view_name_prefix
         {TYPE=NETWORK|PEER|EXCP|BACKBONE|LC|PC|MDL|MDP}
         {RODM=rodmname}
```

view_name is the name of the view to be deleted from RODM.

To delete a group of views beginning with a prefix, specify the prefix with the wildcard character *.

TYPE specifies the type of views to delete as follows:

NETWORK	Network views (default)
PEER	Configuration peer views
EXCP	Exception views
BACKBONE	Configuration backbone views
LC	Logical connectivity views
PC	Physical connectivity views
MDL	More detailed logical views
MDP	More detailed physical views

RODM specifies the RODM name. The RODM name does not have to be specified if MultiSystem Manager is initialized, because DELVIEWS retrieves the RODM name from the MultiSystem Manager common global variable for RODM name.

Examples of Deleting Views

This section provides examples of using DELVIEWS to delete views.

To delete a network view with the name of MY_LAN_VIEW:

```
DELVIEWS MY_LAN_VIEW
```

To delete a group of network views beginning with the prefix RTP_ :

```
DELVIEWS RTP_*
```

To delete a configuration peer view with the name of MY_PEER_VIEW:

```
DELVIEWS MY_PEER_VIEW TYPE=PEER
```

To delete views with names that contain lower case characters, prefix the DELVIEWS REXX clist with the NetView NETVASIS command:

```
NETVASIS DELVIEWS Raleigh_Site_LAN
```

Refer to the *IBM Tivoli NetView for z/OS Data Model Reference* for more information.

DELVIEWS

You can use DELVIEWS to delete the following customized views from RODM:
- Network views
- Configuration peer views
- Configuration backbone views
- Exception views

For example, to delete a specific exception view (NET_EX_VIEW), type:

```
DELVIEWS NETA_EX_VIEW TYPE=EXCP
```

You can also use an asterisk (*) as a wild card at the end of a view name. For example, to delete all network views with names starting with LAN, enter:

```
DELVIEWS LAN*
```

An optional parameter is RODM=*rodmname*. This is not necessary if MultiSystem Manager is initialized and connected to RODM.

Appendix B. View Layout Facility

The view layout facility provides services that the NetView management console uses when laying out views. The input to the view layout facility consists of the view information stored in RODM as well as views that were created by the view preprocessor and downloaded from the host.

This appendix provides the following information for each layout type:
- A graphic example
- Advantages and disadvantages
- An explanation of how each layout type is affected by the GMFHS fields that it uses

View Layout Examples

For representing different aspects of a network, some views of a network model might be easier to visually interpret than others. Therefore, the view layout facility can produce many types of views:
- Radial layout for clustering by link (see Figure 161)
- Radial layout for user-defined clusters by cluster ID (see Figure 161)
- Radial layout for broadband networks (see Figure 161)
- Radial layout for token-ring networks (see Figure 162 on page 658)
- Radial layout for local area networks (see Figure 163 on page 658)
- Radial layout for local area networks with a central bus (see Figure 164 on page 658)
- Elliptical layout with a single ellipse (see Figure 165 on page 659)
- Hierarchical layout (see Figure 166 on page 659)
- Connectivity tree layout (see Figure 167 on page 659)
- Grid layout for exception, configuration, and network views (see Figure 168 on page 660)

For a list of the advantages and disadvantages of each layout type, see Table 236 on page 660.

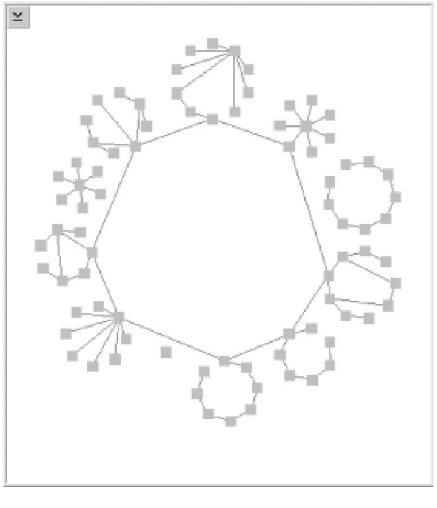


Figure 161. Radial Layout Example

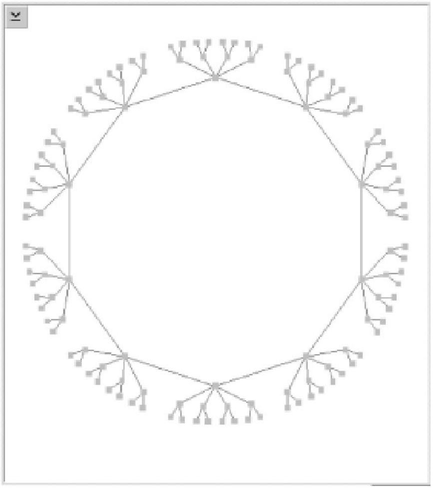


Figure 162. Token-Ring Layout Example

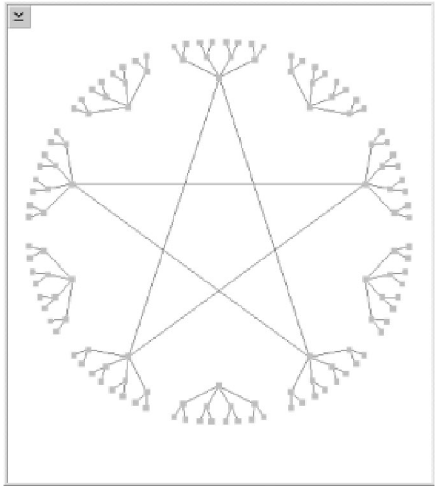


Figure 163. LAN Net Layout Example

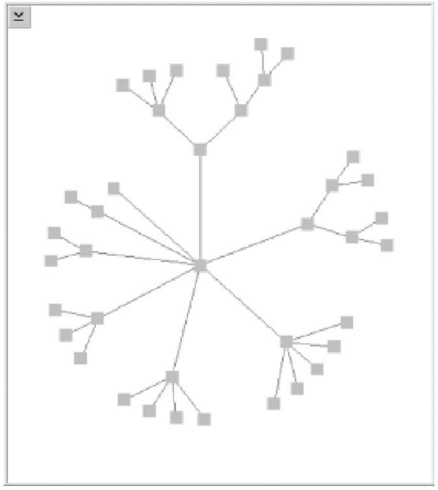


Figure 164. LAN Bus Layout Example

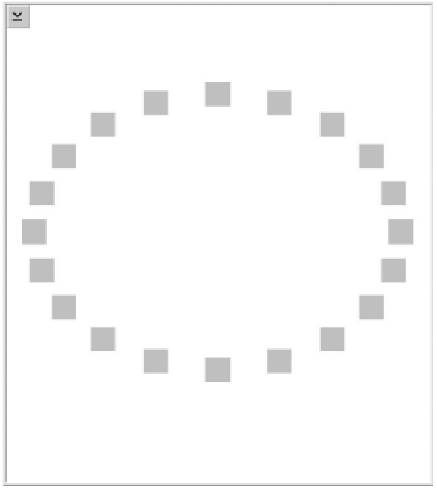


Figure 165. Ellipse Layout Example

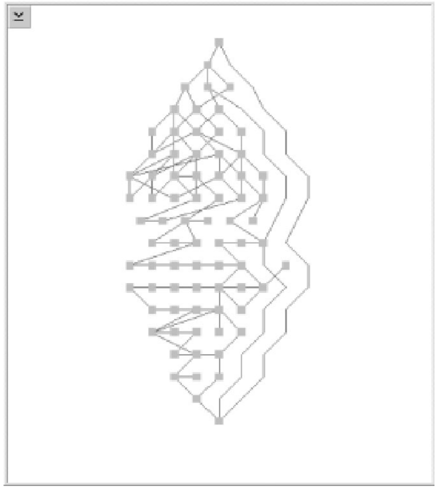


Figure 166. Hierarchical Graph Layout Example

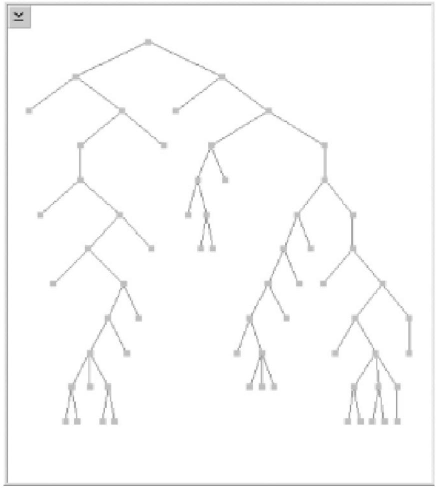


Figure 167. Connectivity Tree Layout Example

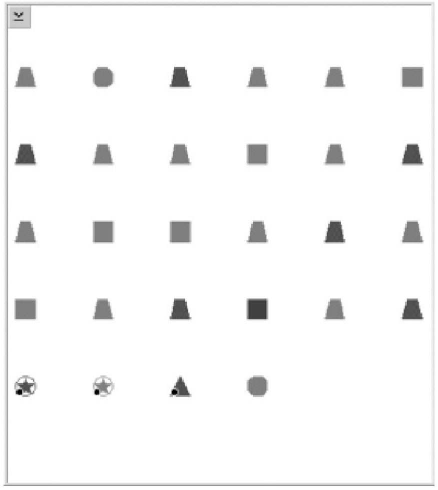


Figure 168. Grid Layout Example

Choosing a View Layout Type

Table 236 describes some of the advantages and disadvantages for each layout type.

Table 236. Advantages and Disadvantages of View Layout Types

View Layout Type	Advantages	Disadvantages
Radial by link type	Efficiently uses presentation space on workstation. Can effectively show groupings of resources at physical sites. Can lay out any view regardless of connectivity.	The mental picture of the user might not correspond to the view layout. Does not convey parent-child relationships well.
Radial by cluster ID	Same advantages as radial layout by link type. Gives you complete control of how nodes are grouped.	Requires you to assign a cluster ID to each node in the view.
Single ellipse	Makes optimal use of the presentation space.	Can only represent a single site or grouping. You must set sequence numbers for link-crossing reduction.
LAN network layout	Well suited to laying out views containing a broad band LAN.	The view must meet connectivity requirements for a LAN view as defined by the view layout facility.
LAN token-ring layout	Well suited to laying out views containing a token-ring LAN.	The view must meet connectivity requirements for a token-ring view as defined by the view layout facility.

Table 236. Advantages and Disadvantages of View Layout Types (continued)

View Layout Type	Advantages	Disadvantages
LAN bus layout	Well suited to laying out views containing a LAN with a central bus.	The view must meet connectivity requirements for a LAN bus view as defined by the view layout facility.
Connectivity tree layout.	Quick layout. Shows the parent-child relationships among resources.	The view must meet connectivity requirements for a connectivity tree view as defined by the view layout facility.
Hierarchical graph by node priority	Shows the parent-child relationships among network resources. Can lay out any view regardless of connectivity.	You must assign a hierarchical priority to each node in the view.
Grid layout	Quick layout. Good for displaying lists of related or unrelated network objects.	Does not display network topology unless you define the rows and columns. Does not show connectivity.

GMFHS Fields Used By the View Layout Facility

The following GMFHS fields supply data that is used by the view layout facility:
- BinPackingFlag
- BusNode
- ClusterIDValue
- DefaultRowSpacing
- EllipseAspectRatioHeight
- EllipseAspectRatioWidth
- FirstNode
- HierarchicalPriority
- LayoutOrientation
- LayoutSequence
- LayoutType
- LayoutWidth
- LinkCrossOptionValue
- ResourceLayoutCharacteristics
- RootNode
- SecondNode

See the following section for a description of how the view layout facility uses these fields.

Layout Type Descriptions

This section describes the view layout types. For each view layout type, a description is provided and the fields used with each view layout type is described.

Note: Setting the SymbolRadiusValue field in RODM no longer has any effect on the appearance of a view. Control of this aspect of view appearance has been moved to the NetView management console, which allows users to change the appearance of a view. For NMC, refer to the online help for more information.

Radial Layout View by Link Type

The radial layout view by link type is a radial layout with clustering based on link type. Nodes that are connected by a link whose ResourceLayoutCharacteristics bit 3 is turned on are put in the same cluster (circle).

Field Descriptions

The following fields are associated with the view and affect how the Radial Layout View by Link Type function will lay out the view:

LayoutType
Set the value of the LayoutType field to 1 to specify this type of view.

BinPackingFlag
If the BinPackingFlag field is set to 1, the Radial Layout View by Link Type function rearranges sites of the same level and weight attempting to obtain an even distribution of nodes.

LinkCrossOptionValue
This field controls the link-crossing optimization level. The greater this number is, the more time the view layout facility will spend attempting to reduce the number of link-crossings in the view. The range for values is 0–6.

The following field is associated with each node in the view and affects how the Radial Layout View by Link Type function will lay out the view:

ResourceLayoutCharacteristics
If bit 2 of this field for a node is turned on, and that node is a single node that is attached to a node in a cluster (circle) but is not attached to any other nodes, the node will be merged into the cluster (circle) to which it is attached.

The following field is associated with each link in the view and affects how the Radial Layout View by Link Type function will lay out the view:

ResourceLayoutCharacteristics
Nodes that are connected by a link with the ResourceLayoutCharacteristics bit 3 turned on will be placed in the same cluster (circle). You can use this bit in any way that is appropriate for you. For example, you can turn the bit on for all links whose link types indicate that they are high speed links. Devices that are attached by high speed links are often at the same site, so this results in devices that are probably at the same site being placed in the same circle.

Radial Layout View by Cluster ID

The radial layout view by cluster ID is a radial layout with clustering based on the ClusterIDValue fields of the nodes in the view. Nodes that have the same cluster IDs will be clustered together in the same site circle.

Field Descriptions

The following fields are associated with the view and affect how the Radial Layout View by Cluster ID function will lay out the view:

LayoutType
Set the value of the LayoutType field to 2 to specify this type of view.

BinPackingFlag
If the BinPackingFlag field is set to 1, the Radial Layout View by Cluster ID function will rearrange sites on the same level and of the same weight to attempt to obtain a homogenous distribution of nodes.

LinkCrossOptionValue
This field controls the link-crossing optimization level. The greater this number is, the more time the view layout facility will spend attempting to reduce the number of link-crossings in the view. The range for valid values is 0–6.

The following field is associated with each node in the view and affects how the Radial Layout View by Cluster ID function will lay out the view:

ResourceLayoutCharacteristics
If bit 2 of this field for a node is turned on, and that node is a single node that is attached to a node in a cluster (circle) but is not attached to any other nodes, the node will be merged into the cluster (circle) to which it is attached.

ClusterIDValue
This field allows the user to indicate how the nodes are grouped (clustered). Nodes that have the same ClusterIDValue will be grouped (clustered) together in the same circle.

Local Area Network Layout View

The local area network layout is a variation of the radial layout that is tailored to local area network views.

Field Descriptions

The following fields are associated with the view and affect how the Local Area Network Layout function will lay out the view:

LayoutType
Set the value of the LayoutType field to 3 to specify this type of view.

BinPackingFlag
If the BinPackingFlag field is set to 1, the Local Area Network Layout function will rearrange sites on the same level and of the same weight to attempt to obtain a homogenous distribution of nodes.

LinkCrossOptionValue
This field controls the link-crossing optimization level. The greater this number is, the more time the view layout facility will spend attempting to reduce the number of link-crossings in the view. The range for valid values is 0–6.

The following field is associated with each node in the view and affects how the Local Area Network Layout function will lay out the view:

LayoutSequence
In views where there are multiple children of the same parent on the subsite and sub-subsite circles, the ordering of the children will be based on the value in the LayoutSequence field for each node. The children will be ordered so that their LayoutSequence fields will be in ascending order when travelling in a clockwise direction around the circle. If you do not

want to control the sequence in which the nodes are placed, set the LayoutSequence field of each of the nodes in the view to 0, which is the default.

Token-Ring Network Layout View Interface

The token-ring network layout is a variation of the radial layout that is tailored to token-ring network views.

Field Descriptions

The following fields are associated with the view and affect how the Token-Ring Network Layout function will lay out the view:

LayoutType
Set the value of the LayoutType field to 4 to specify this type of view.

FirstNode
The ID of the node on the main site circle that is to be placed at the top of the circle (the twelve o'clock position).

SecondNode
The ID of the node on the main site circle that is to be placed immediately adjacent to (in a clockwise direction) the node with the ID of FirstNode.

The following field is associated with each node in the view and affects how the Token-Ring Network Layout function will lay out the view:

LayoutSequence
In views where there are multiple children of the same parent on the subsite and sub-subsite circles, the ordering of the children will be based on the value in the LayoutSequence field for each node. The children will be ordered so that their LayoutSequence fields will be in ascending order when travelling in a clockwise direction around the circle. If you do not want to control the sequence in which the nodes are placed, set the LayoutSequence field of each of the nodes in the view to 0, which is the default.

Bus Network Layout View Interface

The bus network layout is a variation of the radial layout that is tailored to bus network views.

Field Descriptions

The following fields are associated with the view and affect how the Bus Network Layout function will lay out the view:

LayoutType
Set the value of the LayoutType field to 5 to specify this type of view.

BusNode
The object ID of the central bus node for the view. This node will be the parent node of all the nodes on the main site circle of the view.

The following field is associated with each node in the view and affects how the Bus Network Layout function will lay out the view:

LayoutSequence
In views where there are multiple children of the same parent on the subsite and sub-subsite circles, the ordering of the children will be based on the value in the LayoutSequence field for each node. The children will be ordered so that their LayoutSequence fields will be in ascending order

when travelling in a clockwise direction around the circle. If you do not want to control the sequence in which the nodes are placed, set the LayoutSequence field of each of the nodes in the view to 0, which is the default.

Hierarchical Graph Layout View

The Hierarchical Graph Layout function is a layout with each level of a hierarchy occupied by nodes of equivalent specified priority.

This type of layout requires that no node be connected to a node or tackpoint that is more than 1 level away. However, you can build a view that does not satisfy this requirement. If this happens, the view layout facility will add as many additional tackpoints and links as necessary to meet this requirement.

Field Descriptions

The following fields are associated with the view and affect how the Hierarchical Graph Layout function will lay out the view:

LayoutType

Set the value of the LayoutType field to 6 to specify this type of view.

LayoutOrientation

When this field is set to 0, the view layout facility lays out the graph from top to bottom. When this field is set to 1, the view layout facility lays out the graph from left to right.

DefaultRowSpacing

This value indicates the default distance between rows in the connectivity tree. If this field is set to 0 or to any value not in the range from 1–50, the rows will be spaced the distance necessary to make the view square. If you need to explicitly control the distance between rows, set this field to any value in the range of 1–50. This value indicates a multiple of the symbol radius. For example, a value of 3 indicates that the rows are to be a distance equal to three times the symbol radius apart.

The following field is associated with each node in the view and affects how the Hierarchical Graph Layout function will lay out the view:

HierarchicalPriority

This field is used to specify the hierarchical priority of the node. Nodes are placed in the various levels of the hierarchical graph such that their priority values are in ascending order as the graph is traversed from top to bottom, or from left to right if a left to right orientation was specified for the view. All nodes with the same hierarchical priority are placed on the same row in the view. You can assign the hierarchical priority field of each node in any way that suits your needs. For example, one method is to set the hierarchical priority according to the node object type, so that all nodes of a type are on the same row.

Note that for this type of layout, the hierarchical priority is used as a relative value. For example, if all of the nodes in a view are assigned hierarchical priority values of either 1, 2, or 12, the distance between row 1 and row 2 is the same as the distance between row 2 and row 12. Note also that 0 is not a valid value for this field.

Elliptical Layout View

The Elliptical Layout Function lays out a view as a single ellipse.

Field Descriptions

The following fields are associated with the view and affect how the Elliptical Layout function will lay out the view:

LayoutType
Set the value of the LayoutType field to 7 to specify this type of view.

EllipseAspectRatioHeight
EllipseAspectRatioHeight and EllipseAspectRatioWidth will be used as the aspect ratio for the ellipse. An EllipseAspectRatioHeight of 1, and an EllipseAspectRatioWidth of 1 will result in a circle. An EllipseAspectRatioWidth of 640 and an EllipseAspectRatioHeight of 480 will result in an ellipse that approximates the height to width ratio of a standard VGA monitor in 640 × 480 mode.

EllipseAspectRatioWidth
See the definition of EllipseAspectRatioHeight.

The following field is associated with each node in the view and affects how the Elliptical Layout function will lay out the view:

LayoutSequence
Starting at the top of the ellipse, nodes will be arranged in a clockwise sequence, so that the LayoutSequence values for each node are in ascending order. If you do not want to control the sequence in which the nodes are placed, set the LayoutSequence field of each of the nodes in the view to 0, which is the default.

Connectivity Tree Layout View

The Connectivity Tree Layout function lays out a view as a simple connectivity tree. The view must be composed of 1 or more true trees. Except for root nodes, each node must be connected to exactly 1, parent. Nodes can be connected to multiple child nodes. Child nodes cannot be connected.

Field Descriptions

The following fields are associated with the view and affect how the Connectivity Tree Layout function will lay out the view:

LayoutType
Set the value of the LayoutType field to 8 to specify this type of view.

LayoutOrientation
When this field is set to 0 the view layout facility lays out the graph from top to bottom. When this field is set to 1 the view layout facility lays out the graph from left to right.

DefaultRowSpacing
This value indicates the default distance between rows in the connectivity tree. If this field is set to 0, or to any value not in the range from 1–50. the rows will be spaced the distance necessary to make the view square. If you need to explicitly control the distance between rows, you can set this field to any value in the range of 1–50, This value indicates a multiple of the symbol radius. For example, a value of 3 indicates that the rows are to be a distance equal to 3 times the symbol radius apart.

The following fields are associated with each node in the view and affect how the Connectivity Tree Layout function will lay out the view:

RootNode
Setting this field to 0x80 indicates to the view layout facility that the node is a root node. All nodes other than root nodes have a root node as their ancestor. Nodes that are not root nodes and that do not have a root node as their ancestor, will be laid out in a rectangular grid at the bottom of the view.

LayoutSequence
Nodes that are connected to a common parent node will be ordered such that the values in their LayoutSequence fields will be in ascending order from left to right, or from bottom to top depending on the orientation of the view. If you do not want to control the sequence in which the nodes are placed you can set the LayoutSequence field of each of the nodes in the view to 0, which is the default.

Grid Layout

The grid layout function aligns the view objects into a grid of rows and columns. The object locations can be specified by the row number, the column number, or both. If no coordinates are specified, the nodes are randomly placed in a grid formation.

The grid layout can be used with the following types of views:
- Exception
- Network
- Configuration

For exception views, the grid layout is the only layout that can be used, and you cannot specify row and column parameters.

For network or configuration peer views, it is suggested that you specify row and column values for all the objects in the view. The row and column values determine the placement of objects within the view.

Field Descriptions

The following fields are associated with the view and affect how the Grid Layout function will lay out the view:

LayoutType
Set the value of the LayoutType field to 9 to specify this type of view.

LayoutOrientation
When this field is set to 0, the view layout facility lays out the grid from top to bottom. That is the upper left corner is row 1 column 1, with the row numbers increasing as you move from top to bottom and the column numbers increasing as you move from left to right. When this field is set to 1 the view layout facility lays out the grid from left to right. That is the lower-left corner is row 1 column 1, with the row numbers increasing as you move from left to right and the column numbers increasing as you move from bottom to top.

LayoutWidth
The maximum column number to be used by the view layout facility when assigning nodes to columns. The view layout facility only makes column assignments for nodes whose column number was zero. If the LayoutWidth field is zero, the view layout facility will set the LayoutWidth to a value that will make the view square.

The following fields are associated with each node in the view and affect how the Grid Layout function will lay out the view:

HierarchicalPriority
This field is used to assign an absolute row number to the node. Absolute means that if you were to assign three different nodes row numbers of 1, 2, and 12 respectively, the distance between the rows on which nodes 1 and 2 are placed is one-tenth of the distance between the rows on which nodes 2 and 3 are placed. If you do not want to control the row on which the node is placed, set this field to 0 and the view layout facility assigns it to the next available unfilled row. This is the default.

LayoutSequence
This field is used to assign an absolute column number to the node. The meaning of absolute in this context is the same as for the HierarchicalPriority field. If you do not want to control the column in which the node is placed, set this field to 0 and the view layout facility will assign it to the next available column. This is the default. The value in the LayoutWidth field indicates the largest column number to which nodes are assigned. Note that this field only affects values that are assigned by the view layout facility, so it is valid to explicitly specify a column number greater than the LayoutWidth.

The following fields are associated with each link in the view and affect how the Grid Layout function will lay out the view:

HierarchicalPriority
This field is used to assign an absolute row number to the link. Links are drawn by the view layout facility between end-point nodes. The row value for a link is inherited by these end-point nodes, if they were not assigned to a row, that is, if their HierarchicalPriority field is set to 0. If you do not want to control the row on which the link is placed, set this field to zero and the view layout facility will assign it to the next available unfilled row. This is the default.

LayoutSequence
This field is used to assign an absolute column number to the link. Links are drawn by the view layout facility between end-point nodes. The column value for a link is inherited by these end-point nodes, if they were not assigned to a column, that is, if their LayoutSequence field is set to 0. If you do not want to control the column in which the node is placed, set this field to 0 and the view layout facility will assign it to the next available column. This is the default.

Grid Layout Notes

If a link is defined without end points, null end points are created for the link, so it can be placed in the view. Note that for grid layouts, when null nodes are created as end points for a link, they inherit the row and column fields for the link. If these fields are not specified for the link, the link and its null nodes are drawn at a random location in the view.

Table 237 on page 669 lists examples of differently defined links and the results of each definition:

Table 237. Link Definitions and Results

A link is defined with row and column layout parameters. No end points are defined for the link.	The link is drawn with two null nodes at the coordinates specified by the link. In this case, the layout parameters for the link are transferred to the layout parameters of both nodes.
A link is defined without row and column layout parameters. No end points are defined for the link.	The link is drawn with two null nodes at random locations. To control the location of the node, specify coordinates on the link.
A link is defined with row and column layout parameters. Only 1 end point is defined with row and column layout parameters.	The defined end point is drawn at the specified coordinates. A null node is created with the coordinates of the link. A link is drawn between the defined end point and the newly created null node.
A link is defined with row and column layout parameters. Only 1 end point is defined, but without row and column layout parameters.	A null node is created with the coordinates of the link. The defined end point is drawn at a random location and a link is drawn between the defined end point and the newly created null node.
A link is defined with row and column layout parameters. Two end points are defined with row and column layout parameters specified for both.	Both end points are drawn at their specified coordinates. The link is drawn between the two end points. The row and column layout parameters for the link are not used.

Grid Layout

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.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement might 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.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
2Z4A/101
11400 Burnet Road
Austin, TX 78758
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

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.

Programming Interfaces

This publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of Tivoli NetView for z/OS.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml .

Adobe is a trademark of Adobe Systems Incorporated in the United States, and/or other countries.

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

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

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

Other product and service names might be trademarks of IBM or other companies.

Index

Special characters

%APPL% replacement parameter 62
%DOMAIN% replacement parameter 63
%RESOURCE% replacement parameter 63
%SPNAME% replacement parameter 63
%TYPE% replacement parameter 63

A

abstract data types 221, 223
access block
  definition 8
  description 305
  RODM_name parameter 306
  Sign_on_token parameter 306
  User_appl_ID parameter 306
access functions 367
accessibility xix
accessing and changing GMFHS-defined fields 182
action functions 368
adding NMGs and domains, GMFHS 60
adding objects, GMFHS 57
administrative functions 367
aggregate objects
  defining 38
  definition 24
aggregate, suspending aggregation 128
aggregation
  aggregation child description 130
  aggregation hierarchy description 131
  aggregation hierarchy loop description 132
  aggregation hierarchy, building 132
  aggregation hierarchy, creating 131
  aggregation level 130
  aggregation parent description 130
  aggregation path description 130
  aggregation priority 137
  aggregation thresholds 136
  DisplayStatus field, role in 134
  DUIFCUAP method, role in 133
  events that start process 139
  introduction 130
  parent status, calculating 136
  problems 139
  process overview 130
  ResourceTraits field 130
  rules 138
  status group 142
  status group customization 137
  status, affects on aggregation 134
  suspending resources from 135
  updating status 133
aggregation limits 25
AggregationChild connectivity relationship 26
AggregationParent connectivity relationship 26
AGGRST parameter 495
alert processing, DUIFEDEF 172
alert translation tables 176
alerts
  down 80

alerts *(continued)*
  GMFHS data model 167
  INIT, DOMS010 protocol 78
  monitoring 167
  receiving 74
  resolving 167
  session termination 80
  timing 74
ANONYMOUSVAR
  load function data type 299
  null value 260
API_version parameter, transaction information block 307
application programs
  API query field control block sample 305
  asynchronous error notification 325
  calls, RODM 301
  compiling programs 303
  control block relationships 305
  EKGUAPI module 302
  error conditions, user API transactions 317
  languages, RODM user applications 220
  link-editing programs 303
  object deletion notification 326
  parameters, user API calls to RODM 304
  program calls 301
  programming reference 367
  register conventions 302
  RODM system (z/OS), illustration 221
  user API calls to RODM 304
  using control blocks 304
  using user APIs 302
  writing RODM application programs 301
APPLICATIONID
  load function data type 299
  null value 260
applying policy to views 122
ASSIST_CHARVAR data item 486
asterisk 282
asynchronous error
  definition 200
  notification 325
ATTRLIST high-level syntax keyword 277
authorization
  function calls 372
  load function statements 252
automation
  accessing and changing GMFHS-defined fields 182
  advantages 181
  CNMSNIFF sample application 186
  EKGSNIFF sample method 186
  GMFHS 181
  GMFHS example 185
  notifying applications, field changes 181
  overview 181
  RODM 189
  sample application 186
  sample method 186
  using GMFHS methods 183
  writing automation code, data model 181

B

backbone 31
BackboneConnPP 28
BASED attribute 228
batch job, RODM load function 250
BERVAR
  load function data type 299
  null value 260
Bit_map function parameter 445
BLDVIEWS
  Visual 586
books
  see publications xv
building views
  See GMFHS, view building process

C

C, definition 6
calling load function 251
calls
  call statement format 354
  program 301, 354
cause undetermined subvector, INIT alert 78
CE keyword, DOMP010 protocol 66
change method
  change subfield 343
  description 342
  parameters 342
  procedure interface 344
  restrictions 362
change subfield
  definition 214
  description 343
Change_status function parameter 445
changes, non-SNA domain 23
changing
  views 41
changing objects, GMFHS 57
char_literal data type 292
characters 195, 208, 210
  class name 195
  field name 210
  object name 208
characters, allowed 195
  class name 195
  field name 210
  object name 208
characters, double-byte 229
chars data type 291
CHARVAR
  load function data type 299
  null value 260
CHARVARADDR
  load function data type 299
  null value 260
checking output listings 253
checkpoint
  coding checkpoint control 382
  definition 5
  process 380
  TRANSPARENT_CHECKPOINT keyword 382
ChildAccess connectivity relationship 27
class
  names, in RODM 195
class locking, RODM 220
class structure definitions 245
Class_access_info_ptr function parameter 445
Class_access_info_ptr parameter
  EKG_CreateClass function 386
  EKG_CreateField function 387
  EKG_CreateSubfield function 390
  EKG_DeleteClass function 391
  EKG_DeleteField function 392
  EKG_DeleteSubfield function 396
Class_ID function parameter 445
Class_ID parameter
  EKG_QueryNotifyQueue function 421
  EKG_WhereAmI function 444
  entity access-information block 310
class_list, common syntactic element 292
Class_name function parameter 445
Class_name parameter, EKG_QueryObjectName function 423
Class_name_length parameter, entity access-information block 310
Class_name_ptr parameter, entity access-information block 310
class, common syntactic element 292
classes 195
CLASSID
  load function data type 299
  using data type 259
CLASSIDLIST load function data type 299
CLASSLINK load function data type 299
classlink_list, common syntactic element 292
CLASSLINKLIST load function data type 299
CM keyword, DOMP010 protocol 66
CMD_CHARVAR data item 487
CMD_DESC_CHARVAR data item 487
CNMQAPI service routine, description 189
CNMS4402 sample application 186
CNMS4406 sample 76
CNMSJH12, sample 57
CNMSNIFF sample application 186
CODEPAGE parameter 270
coding high-level load function statements 273
coding installation-written methods 358
coding primitive statements 282
cold start, definition 5
collection definition object fields 144
collection definition objects
  fields 144
  using 143
collection definition objects, examples 155
collection manager
  RODM 585
collection specification syntax 149
collection specification values 150
collection specification, using 145
command responses, timing 75
command session 75
command transport envelope, PPI 84
command, protocol
  INIT_ACCEPT 68
  INIT_ACCEPT_ACCEPT 68
  SESSION_REQUEST 68
  SESSION_REQUEST_ACCEPT 68
  SET_CLOCK 68
  SET_CLOCK_ACCEPT 68
common operations services (COS) NMGs 83
common syntactic elements
  class 292
  class_list 292

common syntactic elements *(continued)*
  classlink_list 292
  field 294
  method_spec 295
  object 296
  objectid_list 296
  objectlink_list 296
  recipient_spec 297
  sd_parm 297
  subfield 297
  subs_spec 298
  subs_spec_list 298
  type 298
  typed_value 299
communicating, NMGs 61
compiling application programs 303
compiling programs 359
complex conditional statements 147
ComposedOfLogical connectivity relationship 26
ComposedOfPhysical connectivity relationship 26
Concat_of_strings function parameter 445
Concat_of_strings parameter
  EKG_TriggerNamedMethod function 439
  EKG_TriggerOIMethod function 440
conditional statements 145
CONFIG DOMAIN command, GMFHS 58
CONFIG NETWORK command, GMFHS 58
CONFIG VIEW command, GMFHS 58
configuration view
  description 31
  types of 31
configuration views
  defining
    backbone 43
    logical 43
    peer 43, 44
    physical 43
configuration, RODM 33
connecting, RODM 327
connectivity relationships
  defining 40
  identifying 25
connectors
  NSL_B202 32
  NSL_ENET 32
  OEMLAB 32
control blocks
  access block 305
  API query field control block, sample 305
  entity access information block 309
  field access information block 312
  function block 308
  relationships 305
  response block 314
  transaction information block 307
  using 304
control functions 367
control table
  modifying 260
  sample 260
conventions
  typeface xxi
correlation
  aggregate object names 333
  concepts 330
  customization
    changing display name priority 337
correlation *(continued)*
  customization *(continued)*
    disabling correlation for specific resources 338
  enabling 329
  extending for MultiSystem Manager and SNA topology manager objects 335
  methods 330
  object display labels 333
  object field values 334
  object relationships 333
  objects enabled 331
  types
    free-form 331
    network 331
  using objects you created 335
Correlation_ID function parameter 445
Correlation_ID parameter, EKG_QueryResponseBlockOverflow function 424
COS NMGs 83
COS transport protocol, definition 81
CP keyword, DOMP010 protocol 67
CREATE high-level statement 278
creating
  class structure and object definitions 245
  high-level load function statements 273
  primitive statements 282
  RODM data models 239
creating a notification queue 320
crossing user APIs 302
customizing fast path to failing resource views 95
customizing locate failing resource views 95

D

data definitions
  initialization 266
  object load 266
  statements 264
  structure load 266
Data function parameter 445
data model, system class definitions 196
Data parameter
  EKG_QueryField function 411
  EKG_QueryResponseBlockOverflow function 424
  EKG_QuerySubfield function 426
data types
  abstract data types 223
  fields 222
  identifiers 222
  null values 222
  reserved data types 223
  subfields 215
data types, values and 153
Data_to_be_returned function parameter 445
Data_to_be_returned parameter, EKG_ResponseBlock function 428
Data_type function parameter 446
Data_type parameter
  EKG_ChangeField function 376
  EKG_ChangeMultipleFields function 377
  EKG_ChangeSubfield function 379
  EKG_CreateField function 387
  EKG_Locate function 404
  EKG_QueryEntityStructure function 409
  EKG_QueryField function 411
  EKG_QueryFieldStructure function 415
  EKG_QueryMultipleSubfields function 419

Data_type parameter *(continued)*
  EKG_QuerySubfield function 426
  EKG_SwapField function 435
  EKG_SwapSubfield function 437
Data_value parameter, EKG_QueryMultipleSubfields function 419
Date/Time subvector, INIT alert 79
dbcs_literal data type 293
DD List Structure 267
deciding load type 246
deciding method type 352
defining
  configuration, RODM 33
  network elements 22
  non-SNA domain 35
DELETE high-level statement 279
deleting notification queues 325
deleting objects, GMFHS 57
delimiters, load function statements 273
DELVIEWS 655
designing RODM data models 239
digits data type 293
directory names, notation xxi
disconnecting, RODM 328
Display_Resource_Type_Class 90
DisplayStatus field
  defining exception criteria 101
  ExceptionViewFilter field 103
DisplayStatus method, creating
  DUIFVCFT method 112
  sample method DUIFCUX2 112
  sample method DUIFCUXM 111
  USRXMETH keyword 108
DM keyword, DOMP010 protocol 68
domains 33, 35
DOMP010 presentation protocol 62
DOMP010 protocol
  CE keyword 66
  CM keyword 66
  CP keyword 67
  DM keyword 68
  packet definition 65
  packet format 65
  PT keyword 68
  RN keyword 69
  RP keyword 70
  SN keyword 70
  ST keyword 70
  TM keyword 71
  TX keyword 71
DOMP020 presentation protocol 63
DOMS010 protocol
  CNMS4406 sample 76
  establishing sessions 76
double-byte characters 229, 293
down alert 80
DSINOR service routine, description 189
DUIFCAAP method 488
DUIFCADT method 488
DUIFCAPC method 488
DUIFCASB method 488
DUIFCATC method 488
DUIFCATC method, description 183
DUIFCCAN method
  description 489
DUIFCCAN method, description 183
DUIFCCAP method 488
DUIFCDTC method 488
DUIFCDUC method 488
DUIFCGR2 method 488
DUIFCGR3 method 488
DUIFCGRA method 488
DUIFCGRT method 488
DUIFCLRT method
  description 489
  overview 183
DUIFCLS2 method 488
DUIFCLS3 method 488
DUIFCLSR method 488
DUIFCMUU method 488
DUIFCRDC method 488
DUIFCRTP method 488
DUIFCRTU method 488
DUIFCRUC method 488
DUIFCSRT method 488
DUIFCUAP method
  description 491
  overview 183
DUIFCURA method 488
DUIFCUTC method 488
DUIFCUUS method
  description 492
  overview 184
DUIFECDS method
  description 494
  overview 184
DUIFECMV 167
DUIFEDEF 171
DUIFEDEF AlertProc 172
DUIFEDST, DUIFEIBM, and DUIFEUSR alert translation tables 176
DUIFEGSN method 488
DUIFFAWS method
  description 495
  overview 184
DUIFFIRS method
  description 496
  overview 184
DUIFFRAS method
  description 497
  overview 184
DUIFFSUS method
  description 497
  overview 184
DUIFITKN method 488
DUIFRAIP method 488
DUIFRFDS method
  description 498
  not triggering DisplayStatus recalculation 105
  overview 185
DUIFRRTC method 488
DUIFSMT
  DUIFSMTE statement syntax 104
DUIFSMTE macro
  keywords
    CLASS 105
    CLASS, alias values for 105
    MYNAME 108
    RESOURCE 108
    USRXMETH 108
    XCPT 106
  sample table DUIFSMT 105
  syntax for 105

DUIFVCFT method
  description 498
  overview 185
  using 112
DUIFVCVT method 488
DUIFVDRT method 488
DUIFVEFC method 488
DUIFVEVF method 489
DUIFVEXV method 489
DUIFVFPV method 489
DUIFVGET method 489
DUIFVIEW method 489
DUIFVINS method
  description 499
  overview 185
DUIFVLST method 489
DUIFVLTT method 489
DUIFVMDR method 489
DUIFVNGI method 489
DUIFVNGN method 489
DUIFVNOI method 489
DUIFVNOT method 489
DUIFVPFR method 489
DUIFVSUB method 489
DUIFVTKN method 489
DUIFVUNS method 489
DUIFVUPD method 489
DUIFVVLC method 489
dynamically built views 89
  defining to spans 115
  object discovery 89

E

ECBADDRESS
  load function data type 299
  null value 260
education
  see Tivoli technical training xix
EKG_AddNotifySubscription function 373
EKG_AddObjDelSubs function 374
EKG_APIVersion field, EKG_System class 199
EKG_AsyncTasks field, EKG_System class 200
EKG_BOUNDARY 228
EKG_boundary macro substitution variable 371
EKG_ChangeField function 376
EKG_ChangeMultipleFields function 377
EKG_ChangeSubfield function 378
EKG_Checkpoint function 380
EKG_ConcurrentUsers field, EKG_System class 201
EKG_Connect function 383
EKG_ConnectLong function 384
EKG_CreateClass function 386
EKG_CreateField function 387
EKG_CreateObject function 388
EKG_CreateSubfield function 390
EKG_DeleteClass function 391
EKG_DeleteField function 392
EKG_DeleteNotifySubscription function 393
EKG_DeleteObject function 395
EKG_DeleteSubfield function 396
EKG_DelObjDelSubs function 397
EKG_Disconnect function 399
EKG_ECBAddress field, EKG_NotificationQueue class 205
EKG_ECBPostedStatus field, EKG_NotificationQueue class 205
EKG_ExecuteFunctionList function 400
EKG_ExternalLogState field, EKG_System class 199
EKG_InstallerID field, EKG_Method class 207
EKG_LastAsyncError field
  EKG_System class 200
  EKG_User class 203
EKG_LastCheckpointID field, EKG_System class 200
EKG_LastCheckpointResult field, EKG_System class 200
EKG_LinkNoTrigger function 218, 402
EKG_LinkTrigger function 218, 402
EKG_Locate function 404
EKG_LockObjectList function 405
EKG_LogLevel field, EKG_User class 203
EKG_Maximum_Q_Entries field, EKG_NotificationQueue class 206
EKG_MessagesOnQueue field, EKG_NotificationQueue class 206
EKG_MessageTriggeredAction function 406
EKG_Method class 206
EKG_MLogLevel field, EKG_User class 203
EKG_MTraceFlag field, EKG_Method class 208
EKG_MTraceType field, EKG_User class 204
EKG_Name field, EKG_System class 199
EKG_NotificationQueue class 204
EKG_OutputToLog function 408
EKG_PLI_ISA field, EKG_System class 201
EKG_QueryEntityStructure function 409
EKG_QueryField function 411
EKG_QueryFieldID function 412
EKG_QueryFieldName function 413
EKG_QueryFieldStructure function 415
EKG_QueryFunctionBlockContents function 416
EKG_QueryMultipleSubfields function 418
EKG_QueryNotifyQueue function 421
EKG_QueryObjectName function 423
EKG_QueryResponseBlockOverflow function 424
EKG_QuerySubfield function 426
EKG_RBOverflowAction field, EKG_User class 203
EKG_Refresh field, EKG_Method class 207
EKG_ReleaseID field, EKG_System class 199
EKG_ResponseBlock function 428
EKG_RevertToInherited function 429
EKG_SendNotification function 431
EKG_SetReturnCode function 432
EKG_SSBChain field, EKG_System class 201
EKG_Status field
  EKG_NotificationQueue class 205
  EKG_User class 202
EKG_Stop function 434
EKG_StopMode field, EKG_User class 202
EKG_SubscribedForDelete field, EKG_NotificationQueue class 206
EKG_SubscribedFromClass field, EKG_NotificationQueue class 205
EKG_SubscribedFromObject field, EKG_NotificationQueue class 206
EKG_SwapField function 435
EKG_SwapSubfield function 437
EKG_System class 198
EKG_SystemDataParent class 198
EKG_TransSegment field, EKG_System class 201
EKG_TriggerNamedMethod function 438
EKG_TriggerOIMethod function 440
EKG_UnlinkNoTrigger function 441
EKG_UnlinkTrigger function 441
EKG_UnlockAll function 443
EKG_UsageCount field, EKG_Method class 207
EKG_UsedBy field, EKG_NotificationQueue class 205

EKG_User class 201
EKG_Uses_Q field, EKG_User class 203
EKG_WhereAmI function 444
EKG_WindowSize field, EKG_System class 201
EKG1ACCB access block sample 306
EKG1ENTB entity access information block sample 310
EKG1FLDB field access information block sample 313
EKG1TRAB transaction information block sample 308
EKG3ACCB access block sample 306
EKG3ENTB entity access information block sample 310
EKG3FLDB field access information block sample 313
EKG3TRAB transaction information block sample 308
EKG5VDCL sample variable declarations 223
EKG5WAIT sample PL/I call, EKGWAIT 321
EKG6VDCL sample variable declarations 223
EKG6WAIT sample C call, EKGWAIT 321
EKGCPPI method 485
EKGCTABL control table 260
EKGCTIM method 484, 485
EKGIN1 DD Statement 265
EKGIN2 DD Statement 265
EKGIN3 DD Statement 265
EKGINMTB method name table 245, 261
EKGLANG DD statement 265
EKGLIILM method 249
EKGLISLM method 249
EKGLLOAD sample job, load function 250
EKGLOADP sample procedure, load function 250
EKGLUTB DD statement 265
EKGMANC 357
EKGMIMV method 485
EKGNEQL notification method 482
EKGNLST notification method 483
EKGNOTF notification method 482
EKGNTHD notification method 483
EKGOPPI method 485
EKGPRINT DD statement 265
EKGSNIFF sample method 186
EKGSPPI method 485
EKGSPPI method, description 189
EKGUAPI module 302
EKGWAIT 321
element management system
  communicating 62
  session 75
entity access information block
  Class_ID parameter 310
  Class_name_length parameter 310
  Class_name_ptr parameter 310
  definition 8
  description 309
  Naming_count parameter 310
  Object_ID parameter 310
  Object_name_length parameter 310
  Object_name_ptr parameter 310
Entity_access_info_ptr function parameter 446
Entity_access_info_ptr parameter
  EKG_AddNotifySubscription function 373
  EKG_AddObjDelSubs function 375
  EKG_ChangeField function 376
  EKG_ChangeMultipleFields function 377
  EKG_ChangeSubfield function 378
  EKG_CreateObject function 388
  EKG_DeleteNotifySubscription function 393
  EKG_DeleteObject function 395
  EKG_DelObjDelSubs function 397
  EKG_QueryEntityStructure function 409
Entity_access_info_ptr parameter *(continued)*
  EKG_QueryField function 411
  EKG_QueryFieldName function 414
  EKG_QueryFieldStructure function 415
  EKG_QueryMultipleSubfields function 419
  EKG_QuerySubfield function 426
  EKG_RevertToInherited function 429
  EKG_SwapField function 435
  EKG_SwapSubfield function 437
  EKG_TriggerNamedMethod function 438
Entity_access_info_ptr_1 function parameter 446
Entity_access_info_ptr_1 parameter
  EKG_LinkNoTrigger function 402
  EKG_LinkTrigger function 402
  EKG_UnlinkNoTrigger function 441
  EKG_UnlinkTrigger function 441
Entity_access_info_ptr_2 function parameter 446
Entity_access_info_ptr_2 parameter
  EKG_LinkNoTrigger function 402
  EKG_LinkTrigger function 402
  EKG_UnlinkNoTrigger function 441
  EKG_UnlinkTrigger function 441
envelope, PPI command transport 84
environment variables, notation xxi
errors
  asynchronous error notification 325
  reporting error conditions 317
  user API transactions 317
ESTAE routines, method restrictions 363
ESTAX routines, method restrictions 363
Ethernet network 20
example
  layout parameters, detailed views 51
  layout parameters, objects 54
examples of collection definition objects 155
exception state
  affect on exception views 104
  defining exception criteria 101
  defining ExceptionViewFilter field 103
  examples, mapping DisplayStatus to 110
  fast path to failing resource view 95
  user methods 112
exception view
  customizing DisplayStatus mapping table DUIFSMT
    CNMSJH13 104
    DUIFSMTE statements 104
    examples of 110
    syntax of DUIFSMTE macro 105
  Customizing DisplayStatus mapping table DUIFSMT
    See also DisplayStatus method, creating
    creating DisplayStatus method 111
  defining 41, 100
  defining candidates for 103
  defining the ExceptionViewFilter field 103
  description 28
  DUIFDEXV
    example of 41
    using 100
  illustration 29
  layout parameter used 46
  mapping display status
    using sample table DUIFSMT 101
  object connectivity process 100
  object discovery process 99
  open view, creating objects for 103
  open view, deleting objects from 103
  sample DUIFDEXV 100

exception view *(continued)*
  user method, not triggered 111
  using table DUIFSMT 104
exception views
  implementing 112
ExceptionViewFilter field
  defining 103
  defining exception criteria 101
  DisplayStatus filter 103
  role in exception view object discovery process 100
  UserStatus filter 104
ExceptionViewList field
  in sample DUIFDEXV 101
  role in exception view object discovery process 100
ExceptionViewName field
  role in exception view object discovery process 99
execute command major vector 84

F

fast path to failing resource views, customizing 95
field
  identifiers, RODM 211
  names, RODM 210
field access information block
  definition 8
  description 312
  Field_ID parameter 313
  Field_name_length parameter 313
  Field_name_ptr parameter 313
  Naming_count parameter 313
Field_access_info_ptr function parameter 446
Field_access_info_ptr parameter
  EKG_AddNotifySubscription function 373
  EKG_ChangeField function 376
  EKG_ChangeMultipleFields function 377
  EKG_ChangeSubfield function 378
  EKG_CreateField function 387
  EKG_CreateSubfield function 390
  EKG_DeleteField function 392
  EKG_DeleteNotifySubscription function 393
  EKG_DeleteSubfield function 396
  EKG_Locate function 404
  EKG_QueryField function 411
  EKG_QueryFieldID function 412
  EKG_QueryFieldName function 414
  EKG_QueryFieldStructure function 415
  EKG_QueryMultipleSubfields function 419
  EKG_QuerySubfield function 426
  EKG_RevertToInherited function 429
  EKG_SwapField function 435
  EKG_SwapSubfield function 437
  EKG_TriggerNamedMethod function 438
Field_access_info_ptr_1 function parameter 446
Field_access_info_ptr_1 parameter
  EKG_LinkNoTrigger function 402
  EKG_LinkTrigger function 402
  EKG_UnlinkNoTrigger function 441
  EKG_UnlinkTrigger function 441
Field_access_info_ptr_2 function parameter 446
Field_access_info_ptr_2 parameter
  EKG_LinkNoTrigger function 402
  EKG_LinkTrigger function 402
  EKG_UnlinkNoTrigger function 441
  EKG_UnlinkTrigger function 441
Field_ID function parameter 446
Field_ID parameter
  EKG_QueryEntityStructure function 409
  EKG_QueryFieldID function 412
  EKG_QueryNotifyQueue function 421
  EKG_WhereAmI function 444
Field_ID parameter, field access information block 313
Field_info_array function parameter 446
Field_info_array parameter
  EKG_QueryEntityStructure function 409
  EKG_QueryMultipleSubfields function 419
Field_info_count function parameter 446
Field_info_count parameter, EKG_QueryEntityStructure function 409
Field_info_element_size function parameter 446
Field_info_element_size parameter, EKG_QueryEntityStructure function 409
Field_name function parameter 446
Field_name parameter
  EKG_QueryEntityStructure function 409
  EKG_QueryFieldName function 414
Field_name_length parameter, field access information block 313
Field_name_ptr parameter, field access information block 313
Field_type_flag function parameter 446
Field_type_flag parameter, EKG_CreateField function 387
field, common syntactic element 294
FIELDID load function data type 299
fields
  customizing, performance 480
  RODM classes and objects 210
first product set ID subvector, INIT alert 79
FLCSEXV sample, exception view statements 113
FLCSSMT table 112
float_constant data type 294
FLOATING load function data type 299
FORCE_HAS_NO_INSTANCE load function primitive 283
FORCE_NOT_A_CLASS load function primitive 283
freeing methods 356
function block
  definition 8
  method API 355
  user API 308
function ID 478
function parameters
  Bit_map 445
  Change_status 445
  Class_access_info_ptr 445
  Class_ID 445
  Class_name 445
  Concat_of_strings 445
  Correlation_ID 445
  Data 445
  Data_to_be_returned 445
  Data_type 446
  Entity_access_info_ptr 446
  Entity_access_info_ptr_1 446
  Entity_access_info_ptr_2 446
  Field_access_info_ptr 446
  Field_access_info_ptr_1 446
  Field_access_info_ptr_2 446
  Field_ID 446
  Field_info_array 446
  Field_info_count 446
  Field_info_element_size 446
  Field_name 446
  Field_type_flag 446
  Function_block_copy 446

function parameters *(continued)*
  Function_block_origin 446
  Function_block_ptr 447
  Function_ID 447
  Function_info_array 447
  Indexed_data_length 447
  Indexed_data_ptr 447
  Inheritance_state 447
  Last_checkpoint_ID 447
  Local_copy_map 447
  Local_inherited_flag 447
  Log_message 447
  Long_lived_parm 447
  Message_CCSID 447
  Method_name 448
  Method_output_message 448
  Method_parms 448
  New_char_data_length 448
  New_data_ptr 448
  Notification_queue 448
  Notification_queue_count 448
  Notify_method 448
  Number_of_fields 448
  Number_of_functions 448
  Number_of_subfields 448
  Object_array 448
  Object_ID 448
  Object_list_length 448
  Object_name 448
  Old_char_data_length 449
  Old_data_ptr 449
  Parent_access_info_ptr 449
  Private_public_flag 449
  Reason_code 449
  Requesting_method_ID 449
  Response_block_length 449
  Response_block_reference 449
  Response_block_type 449
  Response_block_used 449
  Response_data 450
  Return_code 450
  Stop_ECB 450
  Stop_type 450
  Subfield 450
  Subfield_map 450
  Subscription_info 451
  User_appl_ID 451
  User_area 451
  User_password 451
  User_word 451
  Value_for_reason_code 452
  Value_for_return_code 452
Function_block_copy function parameter 446
Function_block_copy parameter, EKG_QueryFunctionBlockContents function 417
Function_block_origin function parameter 446
Function_block_origin parameter, EKG_QueryFunctionBlockContents function 417
Function_block_ptr function parameter 447
Function_block_ptr parameter
  EKG_ExecuteFunctionList function 400
  EKG_MessageTriggeredAction function 407
Function_ID function parameter 447
Function_ID parameter
  EKG_AddNotifySubscription function 373
  EKG_AddObjDelSubs function 374
  EKG_ChangeField function 376
Function_ID parameter *(continued)*
  EKG_ChangeMultipleFields function 377
  EKG_ChangeSubfield function 378
  EKG_Checkpoint function 380
  EKG_Connect function 383
  EKG_ConnectLong function 384
  EKG_CreateClass function 386
  EKG_CreateField function 387
  EKG_CreateObject function 388
  EKG_CreateSubfield function 390
  EKG_DeleteClass function 391
  EKG_DeleteField function 392
  EKG_DeleteNotifySubscription function 393
  EKG_DeleteObject function 395
  EKG_DeleteSubfield function 396
  EKG_DelObjDelSubs function 397
  EKG_Disconnect function 399
  EKG_ExecuteFunctionList function 400
  EKG_LinkNoTrigger function 402
  EKG_LinkTrigger function 402
  EKG_Locate function 404
  EKG_LockObjectList function 405
  EKG_MessageTriggeredAction function 407
  EKG_OutputToLog function 408
  EKG_QueryEntityStructure function 409
  EKG_QueryField function 411
  EKG_QueryFieldID function 412
  EKG_QueryFieldName function 413
  EKG_QueryFieldStructure function 415
  EKG_QueryFunctionBlockContents function 417
  EKG_QueryMultipleSubfields function 419
  EKG_QueryNotifyQueue function 421
  EKG_QueryObjectName function 423
  EKG_QueryResponseBlockOverflow function 424
  EKG_QuerySubfield function 426
  EKG_ResponseBlock function 428
  EKG_RevertToInherited function 429
  EKG_SendNotification function 431
  EKG_SetReturnCode function 432
  EKG_Stop function 434
  EKG_SwapField function 435
  EKG_SwapSubfield function 437
  EKG_TriggerNamedMethod function 438
  EKG_TriggerOIMethod function 440
  EKG_UnlinkNoTrigger function 441
  EKG_UnlinkTrigger function 441
  EKG_UnlockAll function 443
  EKG_WhereAmI function 444
Function_info_array function parameter 447
Function_info_array parameter, EKG_ExecuteFunctionList function 400
functions
  access functions 367
  action functions 368
  administrative functions 367
  control functions 367
  EKG_AddNotifySubscription function 373
  EKG_AddObjDelSubs function 374
  EKG_ChangeField function 376
  EKG_ChangeMultipleFields function 377
  EKG_ChangeSubfield function 378
  EKG_Checkpoint function 380
  EKG_Connect function 383
  EKG_ConnectLong function 384
  EKG_CreateClass function 386
  EKG_CreateField function 387
  EKG_CreateObject function 388

functions *(continued)*
  EKG_CreateSubfield function 390
  EKG_DeleteClass function 391
  EKG_DeleteField function 392
  EKG_DeleteNotifySubscription function 393
  EKG_DeleteObject function 395
  EKG_DeleteSubfield function 396
  EKG_DelObjDelSubs function 397
  EKG_Disconnect function 399
  EKG_ExecuteFunctionList function 400
  EKG_LinkNoTrigger function 218, 402
  EKG_LinkTrigger function 218, 402
  EKG_Locate function 404
  EKG_LockObjectList function 405
  EKG_MessageTriggeredAction function 406
  EKG_OutputToLog function 408
  EKG_QueryEntityStructure function 409
  EKG_QueryField function 411
  EKG_QueryFieldID function 412
  EKG_QueryFieldName function 413
  EKG_QueryFieldStructure function 415
  EKG_QueryFunctionBlockContents function 416
  EKG_QueryMultipleSubfields function 418
  EKG_QueryNotifyQueue function 421
  EKG_QueryObjectName function 423
  EKG_QueryResponseBlockOverflow function 424
  EKG_QuerySubfield function 426
  EKG_ResponseBlock function 428
  EKG_RevertToInherited function 429
  EKG_SendNotification function 431
  EKG_SetReturnCode function 432
  EKG_Stop function 434
  EKG_SwapField function 435
  EKG_SwapSubfield function 437
  EKG_TriggerNamedMethod function 438
  EKG_TriggerOIMethod function 440
  EKG_UnlinkNoTrigger function 441
  EKG_UnlinkTrigger function 441
  EKG_UnlockAll function 443
  EKG_WhereAmI function 444
  method API services 370
  query functions 369
  reason codes 470, 472
  reference 371
  user API services 370

G

gateways 61
GENALERT command, monitoring non-network devices 83
generic alert data subvector, INIT alert 78
generic commands using DOMP010 protocol 62
global character 282
GMFHS 25, 114
  accessing and changing fields 182
  adding NMGs and domains 60
  adding, changing, deleting objects 57
  aggregate objects, defining 38
  aggregate objects, definition 24
  aggregation process 130
  automation 181
  automation example 185
  CONFIG DOMAIN command 58
  CONFIG NETWORK command 58
  CONFIG VIEW command 58
  configuration views 31
  defining network 17

GMFHS *(continued)*
  exception views 28
  identifying network elements 22
  initialization process 87
    aggregation warm start 87
    normal 88
    resource status warm start 87
  loading the data model, RODM 57
  monitoring topology managers 89
  more detail views 33
  network views 29
  sample network 18
  span-of-control processing 114
  use of resource and view names 114
  view building process 89
    configuration backbone views 97
    configuration child II view 98
    configuration child III view 99
    configuration children views 95
    configuration logical views 96
    configuration parent views 95
    configuration peer views 94
    configuration physical views 96
    exception views 99
    fast path to failing resource 94
    general description 89
    locate failing resources 94
    more detail logical 98
    more detail physical 98
    network views 94
    object connectivity process 100
    object discovery process 89
    object discovery process for dynamically built views 89
    object discovery process for predefined 89
    refreshing open views 114
    restricting recursive views 113
    using Display_Resource_Type_Class objects 90
    using View_Information_Class objects 91
  views 28
GMFHS fields used by the view layout facility 661
gmfhs methods, restricted
  DUIFCAAP method 488
  DUIFCADT method 488
  DUIFCAPC method 488
  DUIFCASB method 488
  DUIFCATC method 488
  DUIFCCAP method 488
  DUIFCDTC method 488
  DUIFCDUC method 488
  DUIFCGR2 method 488
  DUIFCGR3 method 488
  DUIFCGRA method 488
  DUIFCGRT method 488
  DUIFCLS2 method 488
  DUIFCLS3 method 488
  DUIFCLSR method 488
  DUIFCMUU method 488
  DUIFCRDC method 488
  DUIFCRTP method 488
  DUIFCRTU method 488
  DUIFCRUC method 488
  DUIFCSRT method 488
  DUIFCURA method 488
  DUIFCUTC method 488
  DUIFEGSN method 488
  DUIFITKN method 488

gmfhs methods, restricted *(continued)*
  DUIFRAIP method 488
  DUIFRRTC method 488
  DUIFVCVT method 488
  DUIFVDRT method 488
  DUIFVEFC method 488
  DUIFVEVF method 489
  DUIFVEXV method 489
  DUIFVFPV method 489
  DUIFVGET method 489
  DUIFVIEW method 489
  DUIFVLST method 489
  DUIFVLTT method 489
  DUIFVMDR method 489
  DUIFVNGI method 489
  DUIFVNGN method 489
  DUIFVNOI method 489
  DUIFVNOT method 489
  DUIFVPFR method 489
  DUIFVSUB method 489
  DUIFVTKN method 489
  DUIFVUNS method 489
  DUIFVUPD method 489
  DUIFVVLC method 489
GMFHS parameter, AGGRST 495
GMFHS_Aggregate_Objects_Class objects 24
GMFHS_Managed_Real_Objects_Class objects 24
GMFHS_Shadow_Objects_Class objects 23
GMT offset 74, 80
GRAPHICVAR
  load function data type 299
  null value 260
grouping of method API services 353

H

HAS_FIELD load function primitive 284
HAS_INDEXED_FIELD load function primitive 284
HAS_INSTANCE load function primitive 284
HAS_NO_FIELD load function primitive 285
HAS_NO_INSTANCE load function primitive 285
HAS_NO_SUBFIELD load function primitive 286
HAS_PARENT load function primitive 286
HAS_PRV_FIELD load function primitive 286
HAS_SUBFIELD load function primitive 287
HAS_VALUE load function primitive 287
hex_chars data type 294
hex_literal data type 295
hierarchy resource list subvector, INIT alert 79
high-level load function statements
  CREATE 278
  definition 10
  DELETE 279
  description 242
  MANAGED OBJECT CLASS 276
  SET 280
  syntax 275
  syntax rules 273
how GMFHS builds views
  See GMFHS, view building process

I

identifiers, data type fields 222
identifying installation methods 245
il_parm data type 295
indexed fields 220, 480
Indexed_data_length function parameter 447
Indexed_data_length parameter, EKG_Locate function 404
Indexed_data_ptr function parameter 447
Indexed_data_ptr parameter, EKG_Locate function 404
INDEXLIST load function data type 299
inheritance in methods 350
Inheritance_state function parameter 447
Inheritance_state parameter, EKG_QueryFieldStructure function 415
INHERITS load function primitive 288
INIT alert
  cause undetermined subvector 78
  Date/Time subvector 79
  DOMS010 protocol 78
  first product set ID subvector 79
  generic alert data subvector 78
  hierarchy resource list subvector 79
  probable cause subvector 78
  second product set ID subvector 79
  self-defining text message subvector 80
INIT high-level syntax keyword 277
INIT_ACCEPT protocol command 68
INIT_ACCEPT_ACCEPT protocol command 68
INITIAL high-level syntax keyword 277
initialization load
  cold start 249
  description 246
  warm start 249
initialization method
  coding 341
  definition 6
  services available 364
installing methods 356
INTEGER load function data type 299
interfaces 220
INVOKED_WITH load function primitive 288
INVOKER high-level syntax keyword 278
IS_LINKED_TO load function primitive 289
IS_NOT_LINKED_TO load function primitive 289
IsPartOf connectivity relationship 26

L

languages, methods 221
languages, RODM user applications 220
Last_checkpoint_ID function parameter 447
Last_checkpoint_ID parameter, EKG_Connect function 383
Last_checkpoint_ID parameter, EKG_ConnectLong function 384
layout algorithms 46
layout parameters
  defining detailed views 49
  defining network and configuration views 46
  exception views 46
  objects, detailed views 51
library, RODM method 365
link action functions 218
link-editing application programs 303
link-editing RODM programs 360
linkage conventions 266
links between objects 216, 217
LISTLEVEL parameter 270
load function
  ATTRLIST high-level syntax keyword 277
  authorization level 252
  batch job 250

load function *(continued)*
  calling load function, module 251
  checking output listings 253
  class structure definitions 245
  coding high-level load function statements 273
  coding primitive statements 282
  common syntactic elements 291
  CREATE high-level statement 278
  creating class structure and object definitions 245
  data definitions necessary
    initialization 266
    object load 266
    structure load 266
  data types 299
  deciding load type 246
  DELETE high-level statement 279
  delimiters 273
  EKGLLOAD sample job 250
  EKGLOADP sample procedure 250
  FORCE_HAS_NO_INSTANCE primitive 283
  FORCE_NOT_A_CLASS primitive 283
  HAS_FIELD primitive 284
  HAS_INDEXED_FIELD primitive 284
  HAS_INSTANCE primitive 284
  HAS_NO_FIELD primitive 285
  HAS_NO_INSTANCE primitive 285
  HAS_NO_SUBFIELD primitive 286
  HAS_PARENT primitive 286
  HAS_PRV_FIELD primitive 286
  HAS_SUBFIELD primitive 287
  HAS_VALUE primitive 287
  identifying installation methods 245
  INHERITS primitive 288
  INIT high-level syntax keyword 277
  INITIAL high-level syntax keyword 277
  initialization load
    cold start 249
    description 246
    warm start 249
  input columns 273
  introduction 240
  INVOKED_WITH primitive 288
  INVOKER high-level syntax keyword 278
  IS_LINKED_TO primitive 289
  IS_NOT_LINKED_TO primitive 289
  link-edit restriction, calling module 251
  loading class structure and method names 250
  loading data cache 244
  loading data models, RODM 244
  loading definitions and method names 251
  loading modules 251
  loading object definitions 250
  loading RODM data cache 241
  MANAGED OBJECT CLASS high-level statement 276
  MODE high-level syntax keyword 280
  MODLIST high-level syntax keyword 280
  NOT_A_CLASS primitive 290
  OBJCLASS high-level syntax keyword 278
  object definitions 245
  object load 248
  OBJINST high-level syntax keyword 278
  operations 240
  parameters
    CODEPAGE 270
    LISTLEVEL 270
    LOAD 270
    NAME 271

load function *(continued)*
  parameters *(continued)*
    OPERATION 271
    ROUTECODE 272
    SEVERITY 273
  parameters, invoking 269
  PARENT IS high-level syntax keyword 277
  PL/I and C 251
  primitive statements 242
  primitive statements, definition 10
  PRIVATE high-level syntax keyword 277
  processing logic 282
  PUBLIC high-level syntax keyword 277
  PUBLIC_INDEXED high-level syntax keyword 277
  reference 258
  SET high-level statement 280
  statements 240
  structure load 247
  SUBFIELD_HAS_VALUE primitive 290
  SUBFIELD_INHERITS primitive 291
  submitting jobs, invoking load functions 250
  syntax rules
    high level statements 273
    primitive statements 282
  syntax, primitive statements 282
  using CLASSID data type 259
  using OBJECTID data type 259
  verify operation 258
LOAD parameter 270
loading class structure and method names 250
loading data models, RODM 244
loading definitions and method names 251
loading modules 251
loading object definitions 250
loading RODM data cache 241
loading the data models, RODM
  sample CNMSJH12 57
Local_copy_map function parameter 447
Local_copy_map parameter, EKG_QueryFieldStructure function 415
Local_inherited_flag function parameter 447
locate failing resource views, customizing 95
locate resource function 113
Log_message function parameter 447
Log_message parameter, EKG_OutputToLog function 408
logging
  controlling, EKG_LogLevel field 203
  controlling, EKG_MLogLevel field 203
logical 31
logical connectivity, defining 40
LogicalConnDownstream connectivity relationship 28
LogicalConnPP connectivity relationship 27
LogicalConnUpstream connectivity relationship 28
Long_lived_parm function parameter 447
Long_lived_parm parameter
  EKG_AddNotifySubscription function 373
  EKG_AddObjDelSubs function 375
  EKG_DeleteNotifySubscription function 393
  EKG_DelObjDelSubs function 397
long-lived parameters 355
LookAt message retrieval tool xviii

M

MACRO preprocessor option, PL/I
  applications 303
  macros 360

major vector
  execute 84
  reply, execute command 84
  text data parameter 84
MANAGED OBJECT CLASS high-level statement 276
managed objects
  defining 36
  definition 4
  identifying 23
managed real objects, definition 24
management objects
  defining 33
  definition 4
  identifying 22
manuals
  see publications xv
maximizing RODM performance
  customizing parameters and system fields 480
  data model structure and size 480
  method design 480
  user application design 480
  using indexed fields 480
message retrieval tool, LookAt xviii
Message_CCSID function parameter 447
Message_CCSID parameter, EKG_OutputToLog function 408
method API
  asynchronous error notification 325
  call statement format 354
  coding installation-written methods 358
  compiling programs 359
  control block relationships 305
  deciding method type 352
  description 339
  designing, performance 480
  function reference 371
  general restrictions 362
  grouping, API services 353
  languages 221
  linking programs 360
  long-lived parameters 355
  method API services 353, 370
  method parameters 355
  programming language specific preprocessor statements 359
  programming reference 367
  query field control block sample 355
  query field control block, sample 305
  restrictions 360, 362
  RODM system (z/OS), illustration 221
  services available, initialization methods 364
  services available, object-specific methods 364
  services available, RODM methods 363, 364
  short-lived parameters 356
  tasks best performed 339
  writing installation-written methods 358
  writing RODM methods 339
method name table
  bypassing load 261
  description 261
Method_name function parameter 448
Method_name parameter
  EKG_QueryNotifyQueue function 421
  EKG_TriggerOIMethod function 440
Method_output_message function parameter 448
Method_output_message parameter, EKG_SendNotification function 431
Method_parms function parameter 448
Method_parms parameter
  EKG_ChangeField function 376
  EKG_ChangeMultipleFields function 377
  EKG_CreateClass function 386
  EKG_CreateObject function 388
  EKG_DeleteClass function 391
  EKG_DeleteObject function 395
  EKG_LinkTrigger function 402
  EKG_QueryField function 411
  EKG_SwapField function 435
  EKG_TriggerNamedMethod function 438
  EKG_TriggerOIMethod function 440
  EKG_UnlinkTrigger function 441
method_spec, common syntactic element 295
method, class 206
METHODNAME
  load function data type 299
  null value 260
METHODPARAMETERLIST
  load function data type 299
  null value 260
methods
  change method 342
  deciding method type 352
  definition 6
  description 339
  DUIFCAAP method 488
  DUIFCADT method 488
  DUIFCAPC method 488
  DUIFCASB method 488
  DUIFCATC method 488
  DUIFCCAN method 489
  DUIFCCAP method 488
  DUIFCDTC method 488
  DUIFCDUC method 488
  DUIFCGR2 method 488
  DUIFCGR3 method 488
  DUIFCGRA method 488
  DUIFCGRT method 488
  DUIFCLRT method 489
  DUIFCLS2 method 488
  DUIFCLS3 method 488
  DUIFCLSR method 488
  DUIFCMUU method 488
  DUIFCRDC method 488
  DUIFCRTP method 488
  DUIFCRTU method 488
  DUIFCRUC method 488
  DUIFCSRT method 488
  DUIFCUAP method 491
  DUIFCURA method 488
  DUIFCUTC method 488
  DUIFCUUS method 492
  DUIFECDS method 494
  DUIFEGSN method 488
  DUIFFAWS method 495
  DUIFFIRS method 496
  DUIFFRAS method 497
  DUIFFSUS method 497
  DUIFITKN method 488
  DUIFRAIP method 488
  DUIFRFDS method 498
  DUIFRRTC method 488
  DUIFVCFT method 498
  DUIFVCVT method 488
  DUIFVDRT method 488
  DUIFVEFC method 488

methods *(continued)*
  DUIFVEVF method 489
  DUIFVEXV method 489
  DUIFVFPV method 489
  DUIFVGET method 489
  DUIFVIEW method 489
  DUIFVINS method 499
  DUIFVLST method 489
  DUIFVLTT method 489
  DUIFVMDR method 489
  DUIFVNGI method 489
  DUIFVNGN method 489
  DUIFVNOI method 489
  DUIFVNOT method 489
  DUIFVPFR method 489
  DUIFVSUB method 489
  DUIFVTKN method 489
  DUIFVUNS method 489
  DUIFVUPD method 489
  DUIFVVLC method 489
  EKGCPPI method 485
  EKGCTIM method 484, 485
  EKGLIILM 249
  EKGLISLM 249
  EKGMIMV method 485
  EKGNEQL notification method 482
  EKGNLST notification method 483
  EKGNOTF notification method 482
  EKGNTHD notification method 483
  EKGOPPI method 485
  EKGSPPI notification method 485
  general restrictions 362
  identifying installation methods 245
  inheritance 350
  initialization method 341
  installation-written methods 358
  installing and freeing methods 356
  long-lived parameters 355
  method library 365
  method parameters 355
  name table 245
  named methods 349
  NetView 358
  NetView methods 481
  not for customer use, list of 488
  notification methods 346
  null method 352
  object-independent methods 340
  object-specific methods 342
  obtaining storage 357
  query methods 344
  reason codes 479
  restrictions 360, 363
  return and reason codes 452
  services available, initialization methods 364
  services available, object-specific methods 364
  services available, RODM methods 363, 364
  short-lived parameters 356
  types 340
  using object-independent methods 352
  using object-specific methods 352
  writing RODM methods 339
METHODSPEC load function data type 299
MODE high-level syntax keyword 280
MODLIST high-level syntax keyword 280
monitoring alerts 167
monitoring non-network devices 83
more detail view
  description 33
  types of 33
more detail views
  defining
    logical 46
    physical 46
more detail views, defining layout parameters 49
multiple policies, resources belonging 124
multiple-response presentation protocol 73
MultiSystem Manager
  exception views 112
multivalued fields
  description 216
  example 217
MyClassChildren field 212
MyID field 212
MyName field 212
MyObjectChildren field 212
MyPrimaryParentID field 211
MyPrimaryParentName field 212

N

NAME parameter 271
named method
  description 349
  parameters 349
  procedure interface 350
  restrictions 362
Naming_count parameter, entity access information block 310
Naming_count parameter, field access information block 313
navigating using menus 503
NetView interface, RODM 189
NetView methods
  change methods 484
  descriptions 358
  EKGCTIM method 484, 485
  EKGMIMV method 485
  EKGNEQL notification method 482
  EKGNLST notification method 483
  EKGNOTF notification method 482
  EKGNTHD notification method 483
  EKGSPPI notification method 485
  GMFHS methods 488
  named methods 485
  notification methods 481
  object-independent methods 485
  reason codes 479
NetView Resource Manager 159
network command manager 83
network configuration
  defining to RODM 33
  definition 5
network management gateway
  defining 34
  definition 22
network view
  defining 42
  description 29
  illustration 30, 31
New_char_data_length function parameter 448
New_char_data_length parameter
  EKG_ChangeField function 376
  EKG_ChangeMultipleFields function 377
  EKG_ChangeSubfield function 379
  EKG_SwapField function 435

New_char_data_length parameter *(continued)*
  EKG_SwapSubfield function 437
New_data_ptr function parameter 448
New_data_ptr parameter
  EKG_ChangeField function 376
  EKG_ChangeMultipleFields function 377
  EKG_ChangeSubfield function 379
  EKG_SwapField function 435
  EKG_SwapSubfield function 437
NMG
  communicating 61
  COS 83
  defining 34
  definition 22
  OST 83
  PPI 83
  types 83
non-network devices, monitoring 83
non-SNA domain
  defining 35
  definition 23
non-SNA real resources, defining 37
NOT_A_CLASS load function primitive 290
notation
  environment variables xxi
  path names xxi
  typeface xxi
notification block 421
notification method
  description 346
  example 481
  parameters 346
  procedure interface 348
  restrictions 362
notification process
  C coding example 323
  clean up 325
  definition 9
  EKGWAIT 321
  notification 324
  PL/I coding example 322
  setup 319
  wait 321
notification queue
  creating 320
  definition 9
  deleting 325
  example 481
notification queue class 204
notification subscription, definition 9
Notification_queue function parameter 448
Notification_queue parameter
  EKG_AddNotifySubscription function 373
  EKG_AddObjDelSubs function 375
  EKG_DelObjDelSubs function 397
  EKG_QueryNotifyQueue function 421
  EKG_SendNotification function 431
Notification_queue_count function parameter 448
Notification_queue_count parameter, EKG_QueryNotifyQueue function 421
notify subfield, definition 214
Notify_method function parameter 448
Notify_method parameter
  EKG_AddNotifySubscription function 373
  EKG_DeleteNotifySubscription function 393
null method 352
null pointer 302
null values, data type 222
NullMeth 352
Number_of_fields function parameter 448
Number_of_fields parameter, EKG_ChangeMultipleFields function 377
Number_of_functions function parameter 448
Number_of_Functions parameter, EKG_ExecuteFunctionList function 400
Number_of_subfields function parameter 448
Number_of_subfields parameter, EKG_QueryMultipleSubfields function 419
numeric_literal data type 296

O

OBJCLASS high-level syntax keyword 278
object
  deletion notification 326
  identifiers 210
  locking 220
  names 208
  RODM 208
object correlation
  See corsee
object definitions 245
object deletion, notification 326
object linking 216
object load, load function 248
Object_array function parameter 448
Object_array parameter, EKG_LockObjectList function 405
Object_ID function parameter 448
Object_ID parameter
  EKG_LockObjectList function 405
  EKG_QueryNotifyQueue function 421
  EKG_QueryObjectName function 423
  EKG_WhereAmI function 444
Object_ID parameter, entity access information block 310
Object_list_length function parameter 448
Object_list_length parameter, EKG_LockObjectList function 405
Object_name function parameter 448
Object_name parameter, EKG_QueryObjectName function 423
Object_name_length parameter, entity access information block 310
Object_name_ptr parameter, entity access information block 310
object-independent methods
  definition 6
  description 340
  initialization method 341
  installing and freeing methods 356
  parameters 341
  procedure interface 341
  restrictions 362
  services available, object-independent methods 363, 364
  using 352
object-specific methods
  change method 342, 350
  definition 6
  description 340
  installing and freeing methods 356
  named method 350
  named methods 349
  notification methods 346
  parameters 342
  query method 344, 350

object-specific methods *(continued)*
  services available, object-specific methods 363, 364
  types 342
  using 352
object, common syntactic element 296
OBJECTID load function data type 299
objectid_list, common syntactic element 296
OBJECTIDLIST load function data type 299
OBJECTLINK load function data type 299
objectlink_list, common syntactic element 296
OBJECTLINKLIST load function data type 299
OBJECTNAME
  load function data type 299
  null value 260
objects, collection definition 143, 144
OBJINST high-level syntax keyword 278
offset, GMT 74, 80
Old_char_data_length function parameter 449
Old_char_data_length parameter
  EKG_SwapField function 435
  EKG_SwapSubfield function 437
Old_data_ptr function parameter 449
Old_data_ptr parameter
  EKG_SwapField function 435
  EKG_SwapSubfield function 437
online publications
  accessing xix
OPERATION parameter 271
operator station task (OST) NMGs 83
ORCNTL command, description 189
ORCONV command, description 189
OST NMGs 83
OST transport protocol, definition 82
overflow, response block 315

P

parameter mapping table
  modifying 262
  sample 264
parameter substitution using DOMP010 protocol 62
parameters
  customizing, performance 480
  long-lived 355
  method 355
  short-lived 356
PARENT IS high-level syntax keyword 277
Parent_access_info_ptr function parameter 449
Parent_access_info_ptr parameter, EKG_CreateClass function 386
parent-child relationships, defining 41
ParentAccess connectivity relationship 27
PASSTHRU presentation protocol, definition 64
PASSTHRU session protocol, definition 76
path names, notation xxi
path, resource owner 27
peer 31, 44
performance 480
physical 31
physical connectivity, defining 41
PhysicalConnDownstream connectivity relationship 28
PhysicalConnPP connectivity relationship 27
PhysicalConnUpstream connectivity relationship 28
PL/I, definition 6
pointer, null 302
policies, resources belonging to multiple 124
policy, resources suspended from aggregation due to 127
policy, system status updates no longer sent to resources due to 129
postfix notation in conditional statements 146
PPI command transport envelope 84
PPI NMGs 83
PPI transport protocol, definition 82
presentation protocol
  defining 62
  multiple-response 73
  single-response 72
PresentationProtocolName
  defining 62
  DOMP010 62
  DOMP020 63
  PASSTHRU 64
  typical values 61
prev_val subfield, definition 215
primitive statements
  description 242
  FORCE_HAS_NO_INSTANCE 283
  FORCE_NOT_A_CLASS 283
  global character 282
  HAS_FIELD 284
  HAS_INDEXED_FIELD 284
  HAS_INSTANCE 284
  HAS_NO_FIELD 285
  HAS_NO_INSTANCE 285
  HAS_NO_SUBFIELD 286
  HAS_PARENT 286
  HAS_PRV_FIELD 286
  HAS_SUBFIELD 287
  HAS_VALUE 287
  INHERITS 288
  INVOKED_WITH 288
  IS_LINKED_TO 289
  IS_NOT_LINKED_TO 289
  NOT_A_CLASS 290
  processing logic 282
  SUBFIELD_HAS_VALUE 290
  SUBFIELD_INHERITS 291
  syntax rules 282
PRIVATE high-level syntax keyword 277
Private_public_flag function parameter 449
probable cause subvector, INIT alert 78
process, loading data cache 244
processing logic, primitive statements 282
program calls, RODM 301
program-to-program interface (PPI) NMGs 83
programming languages
  C 6, 9
  PL/I 6, 9
protocol
  multiple-response 73
  single-response 72
protocol command
  INIT_ACCEPT 68
  INIT_ACCEPT_ACCEPT 68
  SESSION_REQUEST 68
  SESSION_REQUEST_ACCEPT 68
  SET_CLOCK 68
  SET_CLOCK_ACCEPT 68
PT keyword, DOMP010 protocol 68
PUBLIC high-level syntax keyword 277
PUBLIC_INDEXED high-level syntax keyword 277
publications
  accessing online xix
  NetView for z/OS xv

publications *(continued)*
  ordering xix

Q

query field control block sample, method API 355
query functions 369
query method
  description 344
  parameters 345
  procedure interface 345
query subfield, definition 213

R

RCVRID_CHARVAR data item 486
reason codes
  each function 470
  functions 472
  NetView methods 479
  return code 0 453
  return code 12 467
  return code 4 453
  return code 8 457
  RODM 452
Reason_code function parameter 449
Reason_code parameter
  EKG_ChangeMultipleFields function 377
  EKG_ExecuteFunctionList function 400
  EKG_LockObjectList function 405
  EKG_QueryMultipleSubfields function 419
Reason_code parameter, transaction information block 308
recipient_spec, common syntactic element 297
RECIPIENTSPEC load function data type 299
register conventions 302
reply, execute command major vector 84
representing policy definitions in RODM 122
Requested_data parameter, EKG_Locate function 404
Requested_info_array parameter, EKG_QueryMultipleSubfields function 419
Requesting_method_ID function parameter 449
Requesting_method_ID parameter, EKG_WhereAmI function 444
reserved data types 223
resource owner path 27
resources belonging to multiple policies 124
resources suspended from aggregation due to policy 127
resources, updates no longer sent 129
ResourceTraits field
  changing with a user method 112
  defining exception criteria 102
response block
  definition 8
  description 314
  overflow 315
response block, error message 315
Response_block_length function parameter 449
Response_block_length parameter
  EKG_ExecuteFunctionList function 401
  EKG_Locate function 404
  EKG_QueryEntityStructure function 409
  EKG_QueryField function 411
  EKG_QueryFieldID function 412
  EKG_QueryFieldName function 414
  EKG_QueryFieldStructure function 415
  EKG_QueryFunctionBlockContents function 417
Response_block_length parameter *(continued)*
  EKG_QueryMultipleSubfields function 419
  EKG_QueryNotifyQueue function 421
  EKG_QueryObjectName function 423
  EKG_QueryResponseBlockOverflow function 424
  EKG_QuerySubfield function 426
  EKG_TriggerNamedMethod function 439
  EKG_TriggerOIMethod function 440
  EKG_WhereAmI function 444
Response_block_reference function parameter 449
Response_block_reference parameter
  EKG_ExecuteFunctionList function 400
  EKG_QueryMultipleSubfields function 419
Response_block_type function parameter 449
Response_block_type, EKG_QueryNotifyQueue function 421
Response_block_used function parameter 449
Response_block_used parameter
  EKG_ExecuteFunctionList function 400, 401
  EKG_Locate function 404
  EKG_QueryEntityStructure function 409
  EKG_QueryField function 411
  EKG_QueryFieldID function 412
  EKG_QueryFieldName function 414
  EKG_QueryFieldStructure function 415
  EKG_QueryFunctionBlockContents function 417
  EKG_QueryMultipleSubfields function 419
  EKG_QueryNotifyQueue function 421
  EKG_QueryObjectName function 423
  EKG_QueryResponseBlockOverflow function 424
  EKG_QuerySubfield function 426
  EKG_TriggerNamedMethod function 439
  EKG_TriggerOIMethod function 440
  EKG_WhereAmI function 444
Response_data function parameter 450
restrictions 403, 442
  ESTAE routines in methods 363
  ESTAX routines in methods 363
  GMFHS methods 488
  input columns for load function 273
  link-edit, calling load function as module 251
  SPIE routines in methods 363
  STAE routines in methods 363
  using C 362
  using change methods 362
  using methods 360, 362
  using named methods 362
  using notification methods 362
  using object-independent methods 362
  using PL/I 360
return code 0 reason codes 453
return code 12 reason codes 467
return code 4 reason codes 453
return code 8 reason codes 457
return codes, RODM 452
Return_code function parameter 450
Return_code parameter
  EKG_ChangeMultipleFields function 377
  EKG_ExecuteFunctionList function 400
  EKG_QueryMultipleSubfields function 419
Return_code parameter, transaction information block 308
RN keyword, DOMP010 protocol 69
RODM (Resource Object Data Manager)
  abstract data types 221, 223
  adding NMGs and domains, GMFHS 60
  adding, changing, deleting objects, GMFHS 57
  asynchronous error notification 325
  automation platform 189

RODM (Resource Object Data Manager) *(continued)*
  automation platform, definition 6
  checkpoint process 380
  class locking 220
  class names 195
  class structure definitions 245
  classes 195
  concepts 195
  connecting 327
  creating class structure and object definitions 245
  creating data models 239
  data definition statements 264
  designing data models 239
  disconnecting 328
  error conditions, user API transactions 317
  field identifiers 211
  field names 210
  fields, classes and objects 210
  function summary 367
  interface, NetView 189
  languages, methods 221
  languages, RODM user applications 220
  load function introduction 240
  load function primitive statements, definition 10
  loading data cache 241, 244
  loading data models 244
  loading the data models 57
  maximizing performance 480
  method API services 370
  method library 365
  network configuration, defining 33
  notification process 318
  notification process, definition 9
  object definitions 245
  object deletion notification 326
  object identifiers 210
  object locking 220
  object names 208
  objects 208
  program calls 301
  reserved data types 223
  return and reason codes 452
  structure 195
  subfields 213
  system structure (z/OS), illustration 221
  system-defined classes 196
  system-defined fields 211
  user API services 370
  using load functions 239
  using user APIs 302
  writing RODM application programs 301
  writing RODM methods 339
RODM collection manager 585
RODM unload function
  customizing 538
  description 537
  running 540
  starting 538
RODM_name parameter, access block 306
RODM, representing policy definitions 122
RODMView
  change field function 527
  compound query function 514
  create actions function 531
  delete actions function 533
  link function 524
  locate objects function 521
RODMView *(continued)*
  method actions function 535
  navigating within RODMView 503
  restrictions 504
  signing on to 506
  simple query function 508
  starting 505
  subfields actions function 530
  unlink function 524
ROUTECODE parameter 272
RP keyword, DOMP010 protocol 70
running RODM load functions 248

S

sample network
  illustration 18
  loading 57
samples
  CNMS4406 76
  EKG5VDCL sample variable declarations 223
  EKG5WAIT sample PL/I call, EKGWAIT 321
  EKG6VDCL sample variable declarations 223
  EKG6WAIT sample C call, EKGWAIT 321
  EKGLLOAD sample job, load function 250
  EKGLOADP sample procedure, load function 250
  FLCSEXV 113
  FLCSSMT 112
sd_parm, common syntactic element 297
second product set ID subvector, INIT alert 79
self-defining text message subvector, INIT alert 80
SELFDEFINING
  load function data type 299
  null value 260
SENDER_CHARVAR data item 487
service point 5, 19
services, method API 353
session
  element management system 75
  establishing with DOMS010 protocol 76
  termination 80
session protocol 75
session termination alert 80
SESSION_REQUEST protocol command 68
SESSION_REQUEST_ACCEPT protocol command 68
SessionProtocolName
  defining 75
  DOMS010 75
  NONE 76
  PASSTHRU 76
  typical values 61
SET high-level statement 280
SET_CLOCK protocol command 68
SET_CLOCK_ACCEPT protocol command 68
SEVERITY parameter 273
shadow objects 36
  definition 23
  NetView management console support for 23
short-lived parameters 356
SHORTNAME
  load function data type 299
  null value 260
Sign_on_token parameter, access block 306
single-response presentation protocol 72
SMALLINT load function data type 299
SN keyword, DOMP010 protocol 70

SNA domain
    defining 33
    definition 22
SNA resources, defining 36
SNA topology manager
    loading the data model, RODM 57
spans
    defining dynamically built views to 115
    defining predefined views to 115
    DisplayResourceName, use with spans 117
    examples of defining views to 116
    examples of restricting resources in views 119
    GMFHS processing 114
    MyName field, use with spans 117
    RACF 121
    resolving problems for views 120
    restricting resources in views 117
    set and clear operator status 121
    UserSpanName, use with spans 117
SPIE routines, method restrictions 363
ST keyword, DOMP010 protocol 70
stack model postfix processing 148
STAE routines, method restrictions 363
statements, complex conditional 147
statements, conditional 145
statements, postfix notation in conditional 146
status groups
    description 142
    using 142
    using to customize DisplayStatus 143
STEPLIB DD statement 264
Stop_ECB function parameter 450
Stop_ECB parameter, EKG_Connect function 383
Stop_ECB parameter, EKG_ConnectLong function 384
Stop_type function parameter 450
Stop_type parameter, EKG_Stop function 434
storage key 360
structure load, load function 247
Subfield function parameter 450
Subfield parameter
    EKG_ChangeField function 376
    EKG_ChangeSubfield function 378
    EKG_QueryNotifyQueue function 421
    EKG_QuerySubfield function 426
    EKG_RevertToInherited function 429
    EKG_SwapSubfield function 437
    EKG_WhereAmI function 444
SUBFIELD_HAS_VALUE load function primitive 290
SUBFIELD_INHERITS load function primitive 291
Subfield_map function parameter 450
Subfield_map parameter
    EKG_CreateField function 387
    EKG_CreateSubfield function 390
    EKG_DeleteSubfield function 396
    EKG_QueryFieldStructure function 415
subfield, common syntactic element 297
subfields
    associated fields 219
    change subfield 214
    data types 215
    notify subfield 214
    prev_val subfield 215
    query subfield 213
    RODM fields 213
    time-stamp subfield 215
    value subfield 213
submitting jobs, invoking load functions 250
subs_spec_list, common syntactic element 298
subs_spec, common syntactic element 298
SUBSCRIBEID
    load function data type 299
    null value 260
subscribing 318
Subscription_info function parameter 451
Subscription_info parameter, EKG_DeleteNotifySubscription function 393
subscription, definition 9
SUBSCRIPTSPEC load function data type 299
SUBSCRIPTSPECLIST load function data type 299
substitution, parameter, using DOMP010 protocol GMFHS replaces the 62
subvector
    cause undetermined 78
    Date/Time 79
    first product set ID 79
    generic alert data 78
    hierarchy resource list 79
    probable cause 78
    second product set ID 79
    self-defining text message 80
    supporting data correlation 84
supporting data correlation subvector 84
suspending aggregation using an aggregate 128
syntax
    common syntactic elements 291
    high-level load function statements 275
    primitive statements 282
syntax rules
    high-level statements 273
    primitive statements 282
syntax, collection specification 149
system class 198
system class definitions 196
system data parent class 198
system object class, defined fields 198
system status updates no longer sent to resources due to policy 129
system-defined classes in RODM 196
system-defined fields, RODM classes and objects 211

T

target, definition 7
TASKINFO_CHARVAR data item 486
TASKNAME_CHARVAR data item 487
tasks best performed with methods 339
text data parameter major vector 84
time-stamp keyword 71
TIMESTAMP load function data type 299
timestamp subfield, definition 215
timing
    alerts 74
    command responses 75
    considerations 74
    time stamp 74
Tivoli
    training, technical xix
    user groups xx
Tivoli Software Information Center xix
TM keyword, DOMP010 protocol 71
token-ring network layout 20
tracing
    controlling, EKG_MTraceFlag field 208
    controlling, EKG_MTraceType field 204

training, Tivoli technical xix
transaction
  definition 7
  handling error conditions 317
transaction information block
  API_version parameter 307
  definition 8
  description 307
  Reason_code parameter 308
  Return_code parameter 308
  Transaction_ID parameter 308
Transaction_ID parameter, transaction information block 308
TRANSID load function data type 299
TRANSPARENT_CHECKPOINT keyword 382
transport protocol, defining 81
TransportProtocolName
  COS 81
  defining 81
  OST 82
  PPI 82
  typical values 61
trigger, definition 6
TX keyword, DOMP010 protocol 71
type, common syntactic element 298
typed_value, common syntactic element 299
typeface conventions xxi
types, values and data 153

U

UNALIGNED attribute 221, 228
UNALIGNED BASED(*) 371
UniversalClass 197
unlink action functions 218
updates no longer sent to resources 129
user API
  asynchronous error notification 325
  calls, RODM 301
  compiling programs 303
  control block relationships 305
  designing, performance 480
  EKGUAPI module 302
  error conditions, API transactions 317
  function reference 371
  link-editing programs 303
  parameters, API calls 304
  programming reference 367
  query field control block, sample 305
  register conventions 302
  user API calls, RODM 304
  user API services 370
  using 302
  using control blocks 304
  writing RODM application programs 301
user application, definition 6
user class 201
user data, notification queue 481
user groups
  NetView, on Yahoo xxi
  Tivoli xx
User_appl_ID function parameter 451
User_appl_ID parameter
  EKG_AddNotifySubscription function 373
  EKG_AddObjDelSubs function 375
  EKG_DelObjDelSubs function 397
  EKG_QueryNotifyQueue function 421
  EKG_SendNotification function 431
User_appl_ID parameter, access block 306
User_area function parameter 451
User_area parameter, EKG_QueryNotifyQueue function 421
User_password function parameter 451
User_password parameter, EKG_Connect function 383
User_password_phrase parameter, EKG_ConnectLong function 384
User_word function parameter 451
User_word parameter
  EKG_AddNotifySubscription function 373
  EKG_AddObjDelSubs function 375
  EKG_DelObjDelSubs function 397
  EKG_QueryNotifyQueue function 421
  EKG_SendNotification function 431
UserStatus field
  defining exception criteria 101
  ExceptionViewFilter field 103
using collection specification 145
using control blocks 304
using data fields 222
using GMFHS methods 183
using OBJECTID data type 259
using RODM load functions 239
using RODM methods 339

V

valid characters 195
  class name 195
  field name 210
  object name 208
value subfield, definition 213
Value_for_reason_code function parameter 452
Value_for_reason_code parameter, EKG_SetReturnCode function 432
Value_for_return_code function parameter 452
Value_for_return_code parameter, EKG_SetReturnCode function 432
values and data types 153
values, collection specification 150
variables, notation for xxi
vector
  execute 84
  reply, execute command 84
  supporting data correlation 84
  text data parameter 84
verify operation 258
view layout facility 657
  list of GMFHS fields used by 661
view objects, definition 5
View_Information_Object_Class object 92, 93
views
  defining 41
    configuration backbone view 43
    configuration logical view 43
    configuration peer view 43
    configuration physical view 43
    exception 41
    more detail logical view 46
    more detail physical view 46
    network 42
    peer 44
  deleting 655
  identifying 28
  layout 657
  layout definition
    bus network layout 664

views *(continued)*
  - layout definition *(continued)*
    - connectivity tree layout 666
    - elliptical layout 665
    - grid layout 667
    - hierarchical graph layout 665
    - local area network layout 663
    - radial layout view by cluster ID 662
    - radial layout view by link type 662
    - token-ring network layout 664
  - layout type
    - examples of 657
  - layout types
    - choosing, advantages and disadvantages 660
    - description 661
  - span
    - *See* spans

views, applying policy 122

views, build process
  - *See* GMFHS, view building process

Visual BLDVIEWS 586

W

warm start, definition 5

WhatIAm field 212

workstations 21

writing installation-written methods 358

writing RODM methods
  - compiling programs 359
  - linking programs 360

Y

Yahoo user group, NetView xxi

Z

z/OS linkage conventions 266

Product Number: 5697-NV6

Printed in USA

SC27-2862-00