Tivoli® NetView® for z/OS™

Resource Object Data Manager and GMFHS Programmer’s Guide

Version 5  Release 1
Resource Object Data Manager and GMFHS Programmer’s Guide

Version 5 Release 1
Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmers Guide

Copyright Notice

Copyright IBM Corporation 1997, 2002. All rights reserved. May only be used pursuant to a Tivoli Systems Software License Agreement or Addendum for Tivoli Products to IBM Customer or License Agreement. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without prior written permission of Tivoli Systems. Tivoli Systems grants you limited permission to make hardcopy or other reproductions of any machine-readable documentation for your own use, provided that each such reproduction shall carry the Tivoli Systems copyright notice. No other rights under copyright are granted without prior written permission of Tivoli Systems. The document is not intended for production and is furnished "as is" without warranty of any kind. All warranties on this document are hereby disclaimed including the warranties of merchantability and fitness for a particular purpose.

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication, or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Trademarks

The following product names are trademarks of IBM Corporation: AIX, APPN, BookManager, DB2, IBM, MVS, NetView, OS/2, OS/390, RACF, RS/6000, RISC System/6000, SecureWay, SystemView, Tivoli, Tivoli Enterprise, TME 10, VTAM, and z/OS.

Notice

Microsoft, Windows, Windows NT, and the Windows logo are trademarks or registered trademarks of Microsoft Corporation.

Other company, product, and service names mentioned in this document may be trademarks or service marks of others.

References in this publication to Tivoli Systems or IBM products, programs, or services do not imply that they will be available in all countries in which Tivoli Systems or IBM operates. Any reference to these products, programs, or services is not intended to imply that only Tivoli Systems or IBM products, programs, or services can be used. Subject to Tivoli Systems's or IBM's valid intellectual property or other legally protectable right, any functionally equivalent product, program or service can be used instead of the referenced product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by Tivoli Systems or IBM, are the responsibility of the user.

Tivoli Systems or 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 USA.

Programming Interfaces

This publication documents intended Programming Interfaces that allow the customer to write programs to obtain services of Tivoli NetView for z/OS.
Contents

Preface ................................................................. xi
Who Should Read This Document ................................ xi
What This Document Contains .................................... xi
Publications ........................................................... xii
  Prerequisite and Related Documents ......................... xii
  Accessing Publications Online ................................ xii
  Ordering Publications .......................................... xii
  Providing Feedback about Publications ....................... xiii
Contacting Customer Support ..................................... xiii
Accessibility Information .......................................... xiii
  Keyboard Access .................................................. xiii
Conventions Used in This Document .............................. xiv
Platform-specific Information .................................... xiv
Terminology ............................................................ xiv
Reading Syntax Diagrams .......................................... xv
  Required Syntax .................................................. xv
  Optional Keywords and Variables ............................. xvi
  Default Values .................................................... xvi
  Long Syntax Diagrams ......................................... xvi
  Syntax Fragments ................................................. xvi
  Commas and Parentheses .................................. xvii
  Highlighting, Brackets, and Braces ......................... xviii
Abbreviations ......................................................... xix

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 ............................................ 5
  Automation Example ............................................. 6
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 ................... 19
  Non-SNA Components of the Sample Network ............ 19
Identifying Which Network Elements to Define ............... 22
Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer’s Guide
### Chapter 11. Writing Applications that Use RODM ................................. 303

- Tasks Best Performed with User Applications ................................................... 303
- Using the User Application Program Interface
  - Register Conventions ................................................................................. 304
  - Usage Notes ................................................................................................. 304
  - Compiling and Link-Editing ................................................................. 305
- Using Control Blocks
  - Access Block ............................................................................................. 306
  - Transaction Information Block .................................................................. 307
  - Function Block ......................................................................................... 309
  - Entity Access Information Block ............................................................ 311
  - Field Access Information Block ............................................................... 314
  - Response Block ......................................................................................... 316
- Error Conditions in Transactions ................................................................. 318
- RODM Notification Process
  - Setup ............................................................................................................. 321
  - Wait .............................................................................................................. 323
  - Notification ................................................................................................. 326
  - Clean Up .................................................................................................... 327
- Asynchronous Error Notification ..................................................................... 328
- Object Deletion Notification
  - Setup for Object-Deletion Notification ....................................................... 329
  - Wait for Object-Deletion Notification ....................................................... 329
  - Notification for Object-Deletion Notification .......................................... 329
  - Cleanup for Object-Deletion Notification ................................................ 329
- Connecting to RODM ...................................................................................... 330
- Disconnecting from RODM ............................................................................ 330

### Chapter 12. Topology Object Correlation. .................................................... 333

- Enabling the Correlation Function ................................................................... 333
- Enabling MultiSystem Manager Object Correlation ...................................... 333
- Enabling SNA Topology Manager Object Correlation ................................... 333
- Enabling GMFHS Object Correlation ............................................................ 334
- Correlation Concepts ...................................................................................... 334
- Correlation Methods ....................................................................................... 334
- Objects Enabled for Correlation .................................................................... 335
- Types of Correlation ....................................................................................... 335
- Correlated Aggregate Object Classes and Names ......................................... 337
- Correlated Object Relationships .................................................................. 337
- Correlated Aggregate Object Display Labels .............................................. 337
- Correlated Aggregate Object Field Values ................................................. 338
- Using Correlation for Objects You Create ..................................................... 339
- Extending Correlation of Objects Created by MultiSystem Manager and SNA Topology Manager ........................................................................................................ 339
  - How to Determine Object Names ................................................................ 340
- Customizing the Correlation Function ........................................................... 341
  - Changing the Display Name Priority ............................................................ 341
  - Disabling Correlation for Specific Resources ............................................. 342

### Chapter 13. Writing RODM Methods ............................................................ 343

- Tasks Best Performed with Methods ............................................................. 343
- Types of Methods
  - Object-Independent Methods ................................................................... 344
Method Actions Function ........................................ 542
RODM Unload Function ........................................ 544
  Starting the RODM Unload Function ......................... 545
  Customizing the RODM Unload Function ..................... 545
  Running the RODM Unload Function .......................... 547
FLCARODM ................................................. 548
  Overview .............................................. 549
  Stem Building Subroutines .................................. 549
  About the Examples ...................................... 554
  FLCARODM Command ...................................... 555
  FLCARODM Functions .................................... 559
  Putting It All Together .................................. 571
  Result Stem ............................................ 578
  Return Codes .......................................... 586
  Object Data Stream Detail ................................ 588
BLDVIEWS ................................................. 592
  Before You Begin ....................................... 593
  BLDVIEWS Processing .................................... 593
  BLDVIEWS Control Cards ................................ 594
  Running BLDVIEWS ..................................... 675
  BLDVIEWS Control Card Examples ......................... 677
  Deleting Views ......................................... 683

**Appendix B. View Layout Facility** ................................ 685
  View Layout Examples .................................... 685
  Choosing a View Layout Type ............................... 690
  GMFHS Fields Used By the View Layout Facility .......... 691
  Layout Type Descriptions ................................ 691
    Radial Layout View by Link Type ......................... 691
    Radial Layout View by Cluster ID ....................... 692
    Local Area Network Layout View ......................... 693
    Token-Ring Network Layout View Interface ............... 693
    Bus Network Layout View Interface ...................... 694
    Hierarchical Graph Layout View ......................... 694
    Elliptical Layout View ................................ 695
    Connectivity Tree Layout View ......................... 696
    Grid Layout .......................................... 696
    Grid Layout Notes .................................... 698

**Index** .................................................................. 699
Preface

This document describes the Tivoli® NetView® for z/OS™ 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 (NMC). This book also describes how to implement network automation using RODM. Finally, this book describes the use of RODM for application programming.

Who Should Read This Document

This book is intended for network managers and system programmers who need to define their non-SNA networks to RODM. It is intended for application programmers and system programmers who need to create or modify RODM applications, methods, and data models. It is also intended for network planners who need to plan how to automate their networks using RODM.

What This Document Contains

This book contains the following sections:

- “Chapter 1. Overview” on page 3 provides an overview of RODM and GMFHS.
- “Chapter 2. Defining Your Network to GMFHS” on page 17 explains how to manually define your network configuration to NetView based on the GMFHS data model.
- “Chapter 3. Loading the GMFHS Data Model” on page 57 describes how to load the GMFHS and SNA topology manager data models, your network definition, and methods into RODM. It also explains how to make additions, changes, or deletions to objects when GMFHS is active.
- “Chapter 4. Communicating with Network Management Gateways” on page 61 describes how GMFHS communicates with network management gateways (NMGs).
- “Chapter 5. How GMFHS Uses RODM” on page 89 provides an overview of the interaction between GMFHS and RODM.
- “Chapter 7. Writing Automation Code” on page 183 describes how you can write automation applications and methods to interface with the NetView-supplied data models, including the GMFHS data model and the SNA topology manager data model.
- “Chapter 8. Using the RODM Automation Platform” on page 191 provides an overview of the RODM automation platform.
- “Chapter 9. Understanding RODM Concepts” on page 197 describes the structure of the RODM data cache, methods, and applications.
- “Chapter 10. Using the RODM Load Function” on page 241 describes how to create your own data model and load object definitions using the RODM load function.
- “Chapter 11. Writing Applications that Use RODM” on page 303 describes how to use the RODM application programming interface.
- “Chapter 12. Topology Object Correlation” on page 333 describes the object correlation function.
Chapter 13. Writing RODM Methods on page 343 describes RODM methods which enable you to maintain data in RODM and to automate functions related to the resources represented by objects in RODM.

Chapter 14. Application Programming Reference on page 371 describes function blocks, which define the transactions against RODM data.

Appendix A. RODM Tools on page 507 describes how to use RODM tools.

Appendix B. View Layout Facility on page 685 describes the view layout facility, which provides services that the NetView management console uses when laying out views.

Publications

This section lists prerequisite and related documents. It also describes how to access Tivoli publications online, how to order Tivoli publications, and how to make comments on Tivoli publications.

Prerequisite and Related Documents

To read about the new functions offered in this release, refer to the Tivoli NetView for z/OS Installation: Migration Guide.

You can find additional product information on these Internet sites:

<table>
<thead>
<tr>
<th>Table 1. Resource Web sites</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM®</td>
</tr>
<tr>
<td>Tivoli Systems</td>
</tr>
<tr>
<td>Tivoli NetView for z/OS</td>
</tr>
</tbody>
</table>

The Tivoli NetView for z/OS Web site offers demonstrations of the NetView product, related products, and several free NetView applications you can download. These applications can help you with tasks such as:

- Getting statistics for your automation table and merging the statistics with a listing of the automation table
- Displaying the status of a JES job or cancelling 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 TSO commands and receiving responses

Accessing Publications Online

You can access many Tivoli publications online using the Tivoli Information Center, which is available on the Tivoli Customer Support Web site:

http://www.tivoli.com/support/documents/

These publications are available in PDF format. Translated documents are also available for some products.

Ordering Publications

You can order many Tivoli publications online at the following Web site:

http://www.ibm.com/shop/publications/order
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, for a list of telephone numbers, see the following Web site: [http://www.tivoli.com/inside/store/lit_order.html](http://www.tivoli.com/inside/store/lit_order.html)

**Providing Feedback about Publications**

We are very interested in hearing about your experience with Tivoli products and documentation, and we welcome your suggestions for improvements. If you have comments or suggestions about our products and documentation, contact us in one of the following ways:

- Send an e-mail to pubs@tivoli.com.
- Complete our customer feedback survey at the following Web site: [http://www.tivoli.com/support/survey/](http://www.tivoli.com/support/survey/)

**Contacting Customer Support**

If you have a problem with any Tivoli product, you can contact Tivoli Customer Support. See the *Tivoli Customer Support Handbook* at the following Web site: [http://www.tivoli.com/support/handbook/](http://www.tivoli.com/support/handbook/)

The handbook provides information about how to contact Tivoli Customer Support, depending on the severity of your problem, and the following information:

- Registration and eligibility
- Telephone numbers and e-mail addresses, depending on the country you are in
- What information you should gather before contacting support

**Note:** Additional support for Tivoli NetView for z/OS is available at the NetView for z/OS Web site: [http://www.tivoli.com/nv390](http://www.tivoli.com/nv390)

Under Related Documents, select Other Online Sources.

The page displayed contains a list of newsgroups, forums, and bulletin boards.

**Accessibility Information**

Refer to *Tivoli NetView for z/OS User’s Guide* for information about accessibility.

**Keyboard Access**

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.

Refer to *Tivoli NetView for z/OS User’s Guide* for more information about keyboard access.
Conventions Used in This Document

The document uses several typeface conventions for special terms and actions. These conventions have the following meaning:

**Bold** Commands, keywords, flags, and other information that you must use literally appear like this, in bold.

**Italics** Variables and new terms appear like this, in italics. Words and phrases that are emphasized also appear like this, in italics.

**Monospace** Code examples, output, and system messages appear like this, in a monospace font.

**ALL CAPS** Tivoli NetView for z/OS commands are in ALL CAPITAL letters.

Platform-specific Information

For more information about the hardware and software requirements for NetView components, refer to the *Tivoli NetView for z/OS Licensed Program Specification*.

Terminology

For a list of Tivoli NetView for z/OS terms and definitions, refer to [http://www.networking.ibm.com/nsg/nsgmain.htm](http://www.networking.ibm.com/nsg/nsgmain.htm).

For brevity and readability, the following terms are used in this document:

**NetView**
- Tivoli NetView for z/OS Version 5 Release 1
- Tivoli NetView for OS/390® Version 1 Release 4
- Tivoli NetView for OS/390 Version 1 Release 3
- TME 10™ NetView for OS/390 Version 1 Release 2
- TME 10 NetView for OS/390 Version 1 Release 1
- IBM NetView for MVS Version 3
- IBM NetView for MVS Version 2 Release 4
- IBM NetView Version 2 Release 3

**MVS** OS/390 or z/OS operating systems.

**RACF®**
RACF is a component of the SecureWay® Security Server for z/OS and OS/390, providing the functions of authentication and access control for OS/390 and z/OS resources and data, including the ability to control access to DB2® objects using RACF profiles. Refer to: [http://www-1.ibm.com/servers/eserver/zseries/zos/security/racfss.html](http://www-1.ibm.com/servers/eserver/zseries/zos/security/racfss.html)

**Tivoli Enterprise™ software**
Tivoli software that manages large business networks.

**Tivoli environment**
The Tivoli applications, based upon the Tivoli Management Framework, that are installed at a specific customer location and that address network computing management issues across many platforms. In a Tivoli environment, a system administrator can distribute software, manage user configurations, change access privileges, automate operations, monitor resources, and schedule jobs. You may have used TME 10 environment in the past.
TME 10
In most product names, TME 10 has been changed to Tivoli.

V and R
Specifies the version and release.

VTAM® and TCP/IP
VTAM and TCP/IP are included in the IBM Communications Server element of the OS/390 and z/OS operating systems. Refer to http://www.ibm.com/software/network/commserver/about/.

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.

Reading Syntax Diagrams
Syntax diagrams start with double arrowheads on the left (▶▶) and move along the main line until they end with two arrowheads facing each other (◀◀).

As shown in the following table, syntax diagrams use position to indicate the required, optional, and default values for keywords, variables, and operands.

<table>
<thead>
<tr>
<th>Element Position</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>On the command line</td>
<td>Required</td>
</tr>
<tr>
<td>Above the command line</td>
<td>Default</td>
</tr>
<tr>
<td>Below the command line</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Required Syntax
The command name, required keywords, variables, and operands are always on the main syntax line. Figure 1 specifies that the resname variable must be used for the CCPLOADF command.

CCPLOADF

▶▶CCPLOADF resname◀◀

Figure 1. Required Syntax Elements

Keywords and operands are written in uppercase letters. Lowercase letters indicate variables such as values or names that you supply. In Figure 2, MEMBER is an operand and membername is a variable that defines the name of the data set member for that operand.

TRANSMSG

▶▶TRANSMSG MEMBER=membername◀◀

Figure 2. Syntax for Variables
Optional Keywords and Variables

Optional keywords, variables, and operands are below the main syntax line. Figure 3 specifies that the ID operand can be used for the DISPREG command, but is not required.

```
DISPREG
  ID=resname
```

*Figure 3. Optional Syntax Elements*

Default Values

Default values are above the main syntax line. If the default is a keyword, it appears only above the main line. You can specify this keyword or allow it to default.

If an operand has a default value, the operand appears both above and below the main line. A value below the main line indicates that if you choose to specify the operand, you must also specify either the default value or another value shown. If you do not specify an operand, the default value above the main line is used.

*Figure 4* shows the default keyword STEP above the main line and the rest of the optional keywords below the main line. It also shows the default values for operands MODNAME=* and OPTION=* above and below the main line.

```
RID
  TASK=opid
  STEP
  CONTINUE
  END
  RUN
  MODNAME=*
  MODNAME=*
  OPTION=*name

.option*  
  HAPIENTR
  HAPIEXIT
```

*Figure 4. Sample of Defaults Syntax*

Long Syntax Diagrams

When more than one line is needed for a syntax diagram, the continued lines end with a single arrowhead (►). The following lines begin with a single arrowhead (►), as shown in *Figure 4*.

Syntax Fragments

Commands that contain lengthy groups or a section that is used more than once in a command are shown as separate fragments following the main diagram. The
fragment name is shown in mixed case. See Figure 5 for a syntax with the fragments ReMote and FromTo.

**BROWSE**

```
+---+-------------------+---+-------------------+---+
|   |                  |   |                  |   |
| BROWSE | ReMote | NETLOGA | FromTo | NETLOGI | NETLOGP | NETLOGS |
|        |        | MemBer  | Dataset Name |
```

**ReMote:**

```
LU=luname  OPERID=**  NETID=**
OPERID=*  net_id  NETID=*  net_id
```

**FromTo:**

```
FROM today  first_record  TO today
FROM date1  time1  TO date2
FROM last_record  time2
```

**MemBer:**

```
XINCL NOKK SUBSYM
INCL  KK  NOSUBS
NOINCL
```

**Dataset Name:**

```
'fully qualified dataset name'
```

*Figure 5. Sample Syntax Diagram with Fragments*

**Commas and Parentheses**

Required commas and parentheses are included in the syntax diagram. When an operand has more than one value, the values are typically enclosed in parentheses and separated by commas. In Figure 6 on page xviii, the OP operand, for example, contains commas to indicate that you can specify multiple values for the testop variable.
If a command requires positional commas to separate keywords and variables, the commas are shown before the keyword or variable, as in Figure 4 on page xvii.

For example, to specify the BOSESS command with the *sessid* variable, enter:

```
NCCF BOSESS applid,,sessid
```

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

**Highlighting, Brackets, and Braces**

Syntax diagrams do not rely on highlighting, underscoring, brackets, or braces; variables are shown italicized in hardcopy or in a differentiating color for NetView help and BookManager® online books.

In parameter descriptions, the appearance of syntax elements in a diagram immediately tells you the type of element. See Table 3 for the appearance of syntax elements.

### Table 3. Syntax Elements Examples

<table>
<thead>
<tr>
<th>This element...</th>
<th>Looks like this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyword</td>
<td>CCPLOADF</td>
</tr>
<tr>
<td>Variable</td>
<td>resname</td>
</tr>
<tr>
<td>Operand</td>
<td>MEMBER=membername</td>
</tr>
<tr>
<td>Default</td>
<td>today or INCL</td>
</tr>
</tbody>
</table>
Abbreviations

Command and keyword abbreviations are described in synonym tables after each command description.
# Part 1. Learning About RODM

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

This book describes Tivoli NetView for z/OS V5R1 Resource Object Data Manager (RODM), which runs under the z/OS operating system. This book explains how to:

- Manually define your network resources to RODM so that you can manage these resources using NetView management console (NMC).
- 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.

GMFHS is the host program that works with RODM and the NetView program running on the host processor, and NMC to manage resources.

GMFHS works with the SNA topology manager and NMC to manage SNA resources. For more information, refer to the Tivoli NetView for z/OS SNA Topology Manager Implementation Guide, SC31-8868.

GMFHS also works with MultiSystem Manager and NMC to manage non-SNA resources. For more information, refer to the Tivoli NetView for z/OS MultiSystem Manager User’s Guide.

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 NMC. 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, refer to the 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 NMC. 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 book explains the RODM definitions that you need to create and how you can create them.
Defining Non-SNA Resources to NetView

NMC communicates with GMFHS. Figure 7 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.

![Diagram of MVS Host, NetView Address Space, GMFHS Address Space, RODM Address Space, and NMC Server/Workstation connections]

**Figure 7. Using RODM to Support the NMC**

**Resource Definition Task**

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

- Management objects
- Managed objects
- View objects

*Management objects* represent the programs that control parts of the network and that connect to the NetView program. LAN Manager and NetView/PC are examples of management objects. 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 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 NetView and receives commands for the resource.

*View objects* represent graphic views that can be displayed on NMC. Most graphic views are created automatically based on the configuration information contained in RODM. You may 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 169. It also explains 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 may 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, refer to the 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 for you, you can write automation routines using the data in RODM. The following RODM concepts are important to the discussion of 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. We refer to running or executing a method 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 could create a named method that contains the commands to activate the object the method is associated with.
- 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 could 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 NetView-supplied method 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 could 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 NetView-supplied method EKGSPPI, 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 book contains two chapters specifically about automation. Read "Chapter 7. Writing Automation Code" on page 183 for more information about automation with the GMFHS data model. Read "Chapter 8. Using the RODM Automation Platform" on page 191 for more information about the RODM automation platform services.

RODM Programming Tasks

While this overview has focused on using RODM to support NMC 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 executed. It contains the particular parameters that RODM needs to execute 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 371. 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 NetView-supplied methods 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. Refer to the 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.
This book 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 books 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 book:

**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. Topics include:  
- 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 DSQIQTLS Task

**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

**Tivoli NetView for z/OS Security Reference**  
This book contains information for defining RODM security.

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

**Tivoli NetView for z/OS Diagnosis Guide**  
This book contains information about diagnostics and troubleshooting, including:  
- Debugging methods  
- The RODM log  
- The RODM dump utility  
- The RODM load utility error listing  
- Using RODM API statistics to improve RODM performance

**Tivoli NetView for z/OS Messages and Codes**  
Describes the messages that are returned by RODM. RODM messages are prefixed with EKG.

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

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

**Tivoli NetView for z/OS Tuning Guide**  
This book provides information for tuning RODM and GMFHS.

**Tivoli NetView for z/OS User’s Guide**  
This book provides information for operators and system programmers on how to use NetView, including RODM and GMFHS.
Tools for RODM

NetView provides the following tools for use with RODM:

- RODMView
- RODM unload function
- FLCARODM (RODM Access Facility)
- BLDVIEWS
- Visual BLDVIEWS (VBV)

For more information about these tools, see “Appendix A. RODM Tools” on page 507.

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.V5R1M0.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.

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 371.

Some of these macros and parts of the sample code are described in this book. The names of the specific macros or functions are listed in the sections in which they are described.
RODM Samples and Macros
Part 2. Defining Resources to NetView

Chapter 2. Defining Your Network to GMFHS  
Manual Network Definition Overview  
Sample Network  
SNA Components of the Sample Network  
Non-SNA Components of the Sample Network  
Service Points  
DEC Network  
Ethernet Network  
Token-Ring Local Area Network  
NV6000 Network  
Identifying Which Network Elements to Define  
Identifying Management Objects  
SNA Domains  
Network Management Gateways  
Non-SNA Domains  
Identifying Managed Objects  
GMFHS_Shadow_Objects_Class Objects  
GMFHS_Managed_Real_Objects_Class Objects  
GMFHS_Aggregate_Objects_Class Objects  
Identifying Connectivity Relationships  
ComposedOfLogical and IsPartOf  
ComposedOfPhysical and IsPartOf  
AggregationParent and AggregationChild  
ParentAccess and ChildAccess  
PhysicalConnPP  
LogicalConnPP  
PhysicalConnUpstream and 
PhysicalConnDownstream  
LogicalConnUpstream and 
LogicalConnDownstream  
BackboneConnPP  
Identifying Views  
Exception Views  
Network Views  
Configuration Views  
More Detail Views  
Defining Your Configuration to RODM  
Defining Management Objects  
Defining SNA Domains  
Defining Network Management Gateways  
Defining Non-SNA Domains  
Defining Managed Objects  
Defining SNA Resources  
Defining Non-SNA Real Resources  
Defining GMFHS Aggregate Objects  
Defining Connectivity Relationships Between Objects  
Defining Logical Connectivity  
Defining Physical Connectivity  
Defining Parent-Child Relationships  
Defining Views  
Defining Exception Views  
Defining Network Views  
Defining Configuration Views  
Defining More Detail Views  
Defining Layout Parameters  
Defining Layout Parameters for Exception Views  
Defining Layout Parameters for Network, Configuration, and More Detail Views  
Defining Layout Parameters for Dynamically Built More Detail Views  
Putting It All Together  
Chapter 3. Loading the GMFHS Data Model  
Loading the Data Models and Network Definitions  
Changing Network Definitions When GMFHS Is Running  
Selecting the Required GMFHS CONFIG Command  
Non_SNA_Domain_Class Changes  
SNA_Domain_Class Changes  
NMG_Class Changes  
GMFHS_Managed_Real_Objects_Class Changes  
Adding NMGs and Domains When GMFHS Is Active  
Chapter 4. Communicating with Network Management Gateways  
Defining Non-SNA Presentation Protocol  
DOMP010 Presentation Protocol  
DOMP020 Presentation Protocol  
PASSTHRU Presentation Protocol  
NONE Presentation Protocol  
Output Formatting For All Presentation Protocols  
DOMP020 and PASSTHRU Output Formatting  
DOMP010 Output Formatting  
DOMP010 Formatting Rules  
General Packet Format  
Keyword and Value Definitions.  
Command Execution—CE  
Command—CM  
Component ID—CP  
Domain—DM  
Protocol—PT  
Reason—RN  
Response—RP  
Command Sender ID—SN  
Message Sequence Number—SQ  
Status—ST  
Time Stamp—TM  
Text—TX  
Command Formatting and Protocol Examples  
Single-Response Protocol  
Multiple-Response Protocol  
Timing Considerations.  
Alerts  
Command Responses  
Defining Non-SNA Session Protocols  
DOMS010  
PASSTHRU
Chapter 6. Customizing GMFHS to Process and Receive Alerts and Resolutions .......................... 169
Receiving and Monitoring Alerts or Resolutions ................................................................. 169
What GMFHS Receives from the Hardware
Monitor ......................................................................................................................... 169
Objects in RODM Representing SNA Resources .......................................................... 170
Objects in RODM Representing NMGs ......................................................................... 171
Objects in RODM Representing Non-SNA Domains ...................................................... 171
First Method .................................................................................................................. 171
Second Method ............................................................................................................. 172
Objects in RODM Representing Non-SNA Resources .................................................... 173
Single Non-SNA Resource .............................................................................................. 173
Multiple Non-SNA Resources ....................................................................................... 173
DUIFEDDEF Alert Processing ......................................................................................... 174
Parameters ...................................................................................................................... 174
Pointer to a reentrant work area .................................................................................... 174
Pointer to a second reentrant work area ......................................................................... 175
Value of the EMDomain field ......................................................................................... 175
Value of the DomainCharacteristics field ...................................................................... 175
Pointer to an array of structures .................................................................................... 175
Pointer to hardware monitor resource hierarchy .......................................................... 175
Pointer to the length of the hardware monitor resource hierarchy ............................... 176
Register 15 Conventions ................................................................................................. 176
Default DUIFEDDEF Actions ......................................................................................... 176
Alert Translation Tables ................................................................................................. 178
Chapter 2. Defining Your Network to GMFHS

This chapter explains how to manually define your network configuration to NetView based on the GMFHS data model. This chapter first describes a sample network, and then shows the steps in manually defining a network.

Notes:
1. You can use the SNA topology manager to define your SNA network to RODM. Refer to the 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 NetView. 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 241.
- 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 would put the data into RODM by issuing calls to the RODM user API. See “Chapter 11. Writing Applications that Use RODM” on page 307 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 8) to describe how to define your network to RODM. This network contains both SNA and non-SNA components.

---

Figure 8. 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 TRLAN network
- NMG A0488P31, which manages the NV6000
- TRLAN network
- NV6000 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 DEC network
- Ethernet network
- DEC 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, 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

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

Service Points

There are four service points, defined as network management gateways, in the sample network:
- NMG B3088P1 runs transaction program SYNOPTAP, which manages the Ethernet network.
- NMG B3088P2 runs transaction program NAP, which manages the DEC network.
- NMG A0488P21 runs in the token-ring LAN and runs transaction program LANMGR, which manages the TRLAN network.
- NMG A0488P31 runs transaction program A94306F8, which manages the NV6000 network.

DEC Network

Figure 9 on page 21 shows more detail of the DEC network shown in the sample network. The DEC network consists of:
- DEC host RALV4, which is attached to service point B3088P2
- Link TX-0-2, which attaches RALV4 to minicomputer RALXT1
Sample Network

- Link TX-1-2, which attaches RALV4 to minicomputer RALXT2

Transaction program NAP runs in service point B3088P2 and converts the events related to these resources into alerts, which are then sent to the NetView management console focal point host A01MPU. This transaction program also accepts commands for these resources.

Figure 9. DEC Network

Ethernet Network

Figure 10 shows more detail of the Ethernet network shown in the sample network. An adapter on service point B3088P1 connects the service point to synoptic concentrator CNTR3000. The concentrator is connected to the hosts and workstations through three connectors:

- Connector OEMLAB, which has non-SNA Hosts VAX6210 and 9370 associated with it
- Connector NSL_ENET, which is associated with DOS workstation DOSTCPIP and the RISC System/6000® workstation RS6000
- Connector NSL_B202, which is associated with host AS400.

Figure 10. Ethernet Network
**Token-Ring Local Area Network**

Figure 11 shows token-ring network TRLAN. It consists of the following:

- 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.

**NV6000 Network**

Figure 12 on page 22 shows more detail of the NV6000 network that was shown in the sample network. The NV6000 network consists of:

- RS/6000® host running Tivoli NetView for AIX®, AIX NetView Service Point, and AIX SNA Server/6000
- Programmable workstations T46A, T47A, T47B, T48A, and T48B

AIX SNA Server/6000 is configured as PU name A0488P31, and the Tivoli NetView for AIX SPAPPLD application is configured as A94306F8. Workstations T46A, T47A, T47B, T48A, and T48B are connected to the TCP/IP network in which Tivoli NetView for AIX resides. Tivoli NetView for AIX converts selected traps related to...
these resources into alerts, which are then sent to the focal point host A01MPU. The A94306F8 transaction program also accepts commands for these resources.

**Identifying Which 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 that you should 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, which is the SNA network management system, and the network management function of one or more non-SNA networks. The AIX and NetView/PC service points running one or more transaction programs 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.

The transaction program (TP) manages the non-SNA network from within the SNA network. The element management system (EMS) manages the non-SNA network from the other, or native, side of the network. The transaction program interacts with the element management system in managing the network.

Depending on the transaction program used, the transaction program and element management system might or might not identify themselves in alerts coming to NetView 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, a transaction program named SYNOPTAP, and no element management system.
- The DEC network, which has a service point named B3088P2 and a transaction program named NAP.
- The TRLAN network, which has a service point named A0488P21 and a transaction program named LANMGR.
- The NV6000 network, which has a service point named A0488P31 and a transaction program named A94306F8.

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:
Defining Network Elements

- SNA topology manager class objects. The SNA topology manager objects are not included in the sample network DUIFSNET. For more information, refer to the 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 there are 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 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 DEC network illustrated in Figure 9, a GMFHS_Managed_Real_Objects_Class object is defined for the following resources:
- The DEC host RALV4
- The minicomputers 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 133.

In the sample network, an aggregate object has been defined for each of the non-SNA networks: Ethernet, DEC, NV6000, 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 aggregate objects ETHERNET and DEC. The status of WESTCTR is determined by the status of the real resources in the Ethernet and DEC networks.
- Aggregate object EASTCTR is composed of aggregate objects NV6000 and TRLAN. The status of EASTCTR is determined by the status of the real resources in the NV6000 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 would 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 could 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 DEC, representing an entire non-SNA network, has a ComposedOfPhysical relationship with objects in RODM representing the host and two minicomputers, as shown in Figure 9 on page 20. The GMFHS_Managed_Real_Objects_Class objects in RODM representing these resources, in turn, have an IsPartOf relationship with aggregate object DEC.

If an operator selects the DEC object in a view and asks for more detail, GMFHS follows the ComposedOfPhysical relationship for the DEC 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 DEC 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 will link the AggregationParent field of the real object DECNET.RALV4.RALXT2 to the AggregationChild field of the aggregate object DEC:

```
OP DUIFCUAP INVOKED_WITH (SELFDEFINING)
  ((CHARVAR)'LINK'
   (CHARVAR)'GMFHS_Managed_Real_Objects_Class.DECNET.RALV4.RALXT2'
   (CHARVAR)'GMFHS_Aggregate_Objects_Class.DEC');
```

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. You should 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.
Defining Network Elements

In the sample network, the Host in the DEC 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. There is no connectivity relationship 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 would change to unsatisfactory. Depending on your exception view definition, the PU would be 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 would start an automation routine to change from the regular hours definition to the off hours definition, and the PUs would then be excluded from the exception view. For more information, see “Defining Exception View Objects and Criteria” on page 102.

Figure 13 shows an example of an exception view.

Figure 13. 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 14 on page 30 shows the high-level view named BIGPIC. In this view, WESTCTR is an aggregate object composed of aggregate objects ETHERNET and DEC. EASTCTR is an aggregate object composed of aggregate objects TRLAN and NV6000. Aggregate objects ETHERNET, DEC, TRLAN, and NV6000 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, DEC, TRLAN, and NV6000, 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 aggregate objects ETHERNET and DEC 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 aggregate objects ETHERNET and DEC that are in an exception state. This type of view can be valuable in a network that contains many real and aggregate objects.

![Figure 14. High-Level View BIGPIC](image)

Figure 15 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, DEC, TRLAN, and NV6000. 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.
Configuration Views
The following configuration views are predefined views. They are used to show objects in relationship to other objects.

<table>
<thead>
<tr>
<th>View Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peer</td>
<td>Displays objects that have a peer relationship.</td>
</tr>
<tr>
<td>Physical</td>
<td>Displays objects in a network based on a physical relationship between objects.</td>
</tr>
<tr>
<td>Logical</td>
<td>Displays objects in a network based on a logical relationship between objects.</td>
</tr>
<tr>
<td>Backbone</td>
<td>Displays objects that comprise a subarea backbone.</td>
</tr>
</tbody>
</table>

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 96. 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, you should 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 16 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 icon DUIU5N00 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.

More Detail Views

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

<table>
<thead>
<tr>
<th>View Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical</td>
<td>Displays the next lower layer of objects in a network based on a logical relationship between objects.</td>
</tr>
<tr>
<td>Physical</td>
<td>Displays the next lower layer of objects in a network based on a physical relationship between objects.</td>
</tr>
</tbody>
</table>

More detail views can also be dynamically built. For more information, see “More Detail Views” on page 99.
Defining Your Network 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.

In this section, we describe how to define each of the objects you identified in the previous section to RODM. For each type of object discussed above, we describe how that type of object is defined, identify the fields that must be defined for that type of object, and define a sample object using the RODM load function statements. For more information about the RODM load function statements, see Chapter 10. "Using the RODM Load Function" on page 241.

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
Defining Your Network Configuration to RODM

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, network management gateway B3088P2 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'.BO1NV.'ContainsResource'),
CommandRouteLUName ::= (CHARVAR) 'BO1NV',
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 BO1NV, 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
Defining Your Network Configuration to RODM

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 ReceiveAlerts and Resolutions" on page 169.

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

```plaintext
-- Create Non_SNA Domain Object for DECNET --
CREATE INVOKER ::= 0000003;
OBJCLASS ::= Non_SNA_Domain_Class;
OBJINST ::= MyName = (CHARVAR) 'B3088P2.NAP.DECNET';

ATTRLIST
  EMDomain ::= (CHARVAR) 'DECNET',
  DomainCharacteristics ::= (ANONYMOUSVAR) x'3672',
  InitialResourceStatus ::= (INTEGER) 129,
  PresentationProtocolName ::= (CHARVAR) 'DOMP020',
  SessionProtocolName ::= (CHARVAR) 'PASSTHRU',
  TransactionProgram ::= (CHARVAR) 'NAP',
  ReportsToAgent ::= (OBJECTLINK)
    ('NMG_Class'.'B3088P2'.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 (B3088P2)
  - The name of the transaction program (NAP)
  - The name of the element management subsystem (DECNET)

  The element management subsystem contains only the element management domain name; DECNET 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.
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 169.

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 DUIFCR 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 have 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 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 discussed in "Defining Layout Parameters for Network, Configuration, and More Detail Views" on page 46.

In the sample network, minicomputer RALXT1 is a non-SNA real resource residing in the DEC network. RALXT1 is defined to RODM as a GMFHS_Managed_Real_Objects_Class object by the following RODM load function statement:

```plaintext
-- Create a GMFHS Managed Real Object for RALXT1 --
CREATE INVOKER ::= 0000003;
OBJCLASS ::= GMFHS_Managed_Real_Objects_Class;
OBJINST ::= MyName = (CHARVAR) 'DECNET.RALV4.RALXT1';
ATTRLIST
  DisplayResourceName ::= (CHARVAR) 'RALXT1',
  Domain ::= (OBJECTLINK)
    ('Non_SNA_Domain_Class'.'B3088P2.NAP.DECNET'.'ContainsResource'),
  DisplayStatusCommandText ::= (CHARVAR)
    'DECCMD/00,SHOW NODE RALXT1 SUMMARY';
END;
OP DUIFCLRT INVOKED_WITH (SELFDEFINING)
  ((CHARVAR)'LINK'
    (CHARVAR)'GMFHS_Managed_Real_Objects_Class.DECNET.RALV4.RALXT1'
    (CHARVAR)'Display_Resource_Type_Class.DUIXC_RTN_MINI');
```

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, minicomputer RALXT1 is associated with Non_SNA_Domain_Class object B3088P2.NAP.DECDNET, and is given a DisplayResourceType of DUIXC_RTN_MINI. Because the DisplayResourceName field is specified, the name that appears 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 244. The DUIFCLRT method is described in “DUIFCLRT - Link Resource Type Method” on page 491.

**Defining GMFHS Aggregate Objects**

Aggregate objects can be used to group resources into a higher-level resource 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 102.

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.

In the sample network, a DEC network is managed through service point B3088P2. The DEC network is composed of a host, two minicomputers, and two links, as illustrated in Figure 9 on page 20. An aggregate object, named DEC, is defined to represent the DEC network. The DEC 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 DEC is defined by the following RODM load function statements:

```寻常
-- Create a GMFHS Aggregate Object for DEC --
CREATE INVOKER ::= 0000004;
OBJCLASS ::= GMFHS_Aggregate_Objects_Class;
OBJINST ::= MyName = (CHARVAR) 'DEC';
ATTRLIST
ThresholdDegraded ::= (INTEGER) 1,
ThresholdSeverelyDegraded ::= (INTEGER) 2,
ThresholdUnsatisfactory ::= (INTEGER) 3,
ComposedOfPhysical ::= (OBJECTLINKLIST)
('GMFHS_Managed_Real_Objects_Class'.'DECNET.RALV4'.'IsPartOf')
('GMFHS_Managed_Real_Objects_Class'.'DECNET.RALV4.RALXT1'.'IsPartOf')
('GMFHS_Managed_Real_Objects_Class'.'DECNET.RALV4.RALXT2'.'IsPartOf')
('GMFHS_Managed_Real_Objects_Class'.'DECNET.RALV4.TX02'.'IsPartOf')
('GMFHS_Managed_Real_Objects_Class'.'DECNET.RALV4.TX12'.'IsPartOf');
END;
OP DUIFCLRT INVOKED_WITH (SELFDEFINING)
((CHARVAR)'LINK'
(CHARVAR)'GMFHS_Aggregate_Objects_Class.DEC'
(CHARVAR)'Display_Resource_Type_Class.DUIXC_RTN_HOST_AGG');
OP DUIFCUAP INVOKED_WITH (SELFDEFINING)
((CHARVAR)'LINK'
(CHARVAR)'GMFHS_Managed_Real_Objects_Class.DECNET.RALV4'
```

Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer’s Guide
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 DEC aggregate object is linked to the IsPartOf fields of the following GMFHS_Managed_Real_Objects_Class objects:

- DECNET.RALV4
- DECNET.RALV4.RALXT1
- DECNET.RALV4.RALXT2
- DECNET.RALV4.TX02
- DECNET.RALV4.TX12

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 DEC object in a view and asks for more detail.

The DEC aggregate object is assigned a DisplayResourceType of DUIXC_RTN_HOST_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 491.

The DEC object is an aggregate host that represents the underlying real resources in the DEC 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 493.
Defining Your Network Configuration to RODM

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 could 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, discussed in "Identifying Connectivity Relationships" on page 26, 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. In this section we illustrate 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 15 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 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 TX-0-2 by using the following RODM load function statements:

```
-- Link RALV4 to TX-0-2 --
OP 'GMFHS_Managed_Real_Objects_Class'.'DECNET.RALV4'.'PhysicalConnPP'
IS_LINKED_TO
'GMFHS_Managed_Real_Objects_Class'.'DECNET.RALV4.TX02'.'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, minicomputer RALXT1 is linked to the DEC Host RALV4 in the configuration illustrated in Figure 9 on page 20 by using the following RODM load function statement:

```
-- Link RALV4 to RALXT1 --
OP 'GMFHS_Managed_Real_Objects_Class'.'DECNET.RALV4'.'ChildAccess'
IS_LINKED_TO
'GMFHS_Managed_Real_Objects_Class'.'DECNET.RALV4.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 may attempt to access information about the view object before all of the information is defined, and this could 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 241

The views that are constructed in RODM are displayed by the NetView management console. The following sections discuss parameters and layout algorithms that are used by the graphic facility. See "Appendix B. View Layout Facility" on page 685 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 17 on page 42 shows an exception view of all objects in the GMFHS_Displayable_Objects_Parent_Class that have DisplayStatus of either severely degraded or unsatisfactory.
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 101.

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 18 on page 43 shows a network view of the DEC network component of the sample network. The icon and symbol displayed for each object are determined by the DisplayResourceType object it is linked to. For example, the resource DECENT.RALV4.RALXT1 is linked to DUIXC_RTN_MINI. The icon DUIU2N00 and the square-shaped host symbol are specified by DUIXC_RTN_MINI. The name
RALXT1 shown in the view is specified by the DisplayResourceName field of object DECNET.RALV4.RALXT1.

![Network View of DEC Network](image)

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

```plaintext
-- Create Network View for DECNET --
CREATE INVOKER ::= 0000004;
  OBJCLASS ::= Network_View_Class;
  OBJINST ::= MyName = (CHARVAR) 'DECNET';
  ATTRLIST
   Annotation ::= (CHARVAR) 'DEC NETWORK',
   LayoutType ::= (INTEGER) 8,
   ConnType ::= (ANONYMOUSVAR) x'80',
   ContainsObjects ::= (OBJECTLINKLIST)
   ('GMFHS_Managed_Real_Objects_Class'.'DECNET.RALV4.RALXT1'.'ContainedInView')
   ('GMFHS_Managed_Real_Objects_Class'.'DECNET.RALV4.RALXT2'.'ContainedInView')
   ('GMFHS_Managed_Real_Objects_Class'.'DECNET.RALV4.TX02'.'ContainedInView')
   ('GMFHS_Managed_Real_Objects_Class'.'DECNET.RALV4.TX12'.'ContainedInView')
   ('GMFHS_Shadow_Objects_Class'.'NETB.B3088P2'.'ContainedInView')
END;
```

In this example, a Network_View_Class object named DECNET is defined to represent the network view of the DEC network. The Annotation field of the object is assigned the value DEC NETWORK, which is displayed at the workstation with the view. The LayoutType field is assigned the value 8, which specifies that the view is to be displayed in connectivity tree layout. The ConnType field is assigned the value 80, which specifies that node to node connections are valid for this type of view, as well as node to link connections. The ContainsObjects field of the DECNET object is linked to the ContainedInView fields of the managed real objects that represent the real resources that make up the DEC network.

**Defining Configuration Views**

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

<table>
<thead>
<tr>
<th>View Type</th>
<th>Class Defined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null connector</td>
<td>B3088P2 Service point node</td>
</tr>
<tr>
<td>Service point node</td>
<td>RALV4 Host</td>
</tr>
<tr>
<td>Host</td>
<td>Local link Local link TX-0-2 RALXT1 Minicomputer</td>
</tr>
<tr>
<td>Minicomputer</td>
<td>RALXT2 Minicomputer</td>
</tr>
</tbody>
</table>

Figure 18. Network View of DEC Network

Defining Your Network Configuration to RODM

Chapter 2. Defining Your Network to GMFHS
Defining Your Network Configuration to RODM

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 96.

Defining Peer Views: Figure 19 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 19 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.
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.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 19, 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.
Defining Your Network Configuration to RODM

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:

<table>
<thead>
<tr>
<th>View Type</th>
<th>Class Defined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical</td>
<td>More Detail_Physical_View_Class</td>
</tr>
<tr>
<td>Logical</td>
<td>More Detail_Logical_View_Class</td>
</tr>
</tbody>
</table>

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 99.

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 683.

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 683.
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 4 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 4 indicates the optional (O) and required (R) fields. N/A indicates that the parameter is not applicable for that type of layout algorithm.

**Table 4. Layout Algorithms and View Parameters**

<table>
<thead>
<tr>
<th>Layout Algorithm</th>
<th>Link Cross Option Value</th>
<th>Bin Packing Flag</th>
<th>Bus Node</th>
<th>First Node</th>
<th>Second Node</th>
<th>Layout Orientation</th>
<th>Default Row Spacing</th>
<th>Ellipse Aspect Ratio Width/Height</th>
<th>Layout Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radial by cluster ID</td>
<td>O</td>
<td>O</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Radial by link type</td>
<td>O</td>
<td>O</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Local area net</td>
<td>O</td>
<td>O</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Token-ring net</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>R</td>
<td>R</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>LAN with central bus</td>
<td>N/A</td>
<td>N/A</td>
<td>R</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Hierarchical with proximity</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>O</td>
<td>O</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Single ellipse</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>O</td>
<td>N/A</td>
</tr>
<tr>
<td>Connectivity tree</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>O</td>
<td>O</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Grid</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>O</td>
</tr>
</tbody>
</table>
For information about the layout parameters and about the layout algorithms, see "Appendix B. View Layout Facility" on page 683.

**Layout Parameters:** Table 5 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 683.

**Table 5. Layout Algorithms and Layout Parameters**

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

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 15 on page 31. The following is the RODM load function statement that defines the LPTRLAN object:

```
-- Create Layout Parameters for Object TRLAN --
CREATE INVOKER ::= 00000004;
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 DECNET, as illustrated in Figure 18 on page 43. The following is the RODM load function statement that defines the LPB3088P2P layout parameters object:

```
-- Create Layout Parameters for Object B3088P2 --
CREATE INVOKER ::= 00000004;
OBJCLASS ::= Layout_Parameters_For_Object_Class;
OBJINST ::= MyName = (CHARVAR) 'LPB3088P2P';
```
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 NMC 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 Tivoli NetView for z/OS NetView Management Console User’s Guide 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 20 on page 53] 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.
Defining Your Network Configuration to RODM

---

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 ([1]) for this example. The sample member DUIFSNET contains the complete statements.

```plaintext
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 on page 49 to the DetailViewLayoutForSelectedResource field of the object you selected in Step 1 on page 42.

The following is part of the RODM load function statement that creates this link, shown as [1] in [Figure 20 on page 51]:

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

5. Specify which more detail view type this Layout_Parameters_For_View_Class object ([3]) represents. You specify the view type by linking the ViewClass field of this object to the DetailViewLayout field of an object ([4]) 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 20 on page 51]:

```plaintext
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 ([3]) specify the layout algorithm and other view parameters. [Table 4 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 4 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:

```plaintext
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 5 on page 48] lists required parameters.
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 21 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 49 to appear in the more detail view.

   For this example, define layout parameters for the object (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 ( ), shown in Figure 21 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 on page 51 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 21 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 Step 49).

   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 21 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 21 on page 54:

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

5. Specify the layout parameters for the object. Table 5 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 5 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:

```plaintext
OP 'Layout_Parameters_For_Object_Class'.
  'Detail_Layout_LANMGR.10005A89A267'.SelectedResource
IS_LINKED_TO 'GMFHS_Aggregate_Objects_Class'.TRLAN.
  'DetailLayoutParmListForSelectedResource';
```
Defining Your Network Configuration to RODM

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 241.
Chapter 3. Loading the GMFHS Data Model

This chapter explains how to load the GMFHS and SNA topology manager data models, your network definition, and methods into RODM. This chapter also explains 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, FLBTRDMx, which is also shipped with the NetView program. For more information about the FLBTRDMx load function input files, refer to Tivoli NetView for z/OS Installation: Configuring Graphical Components.

DUIFSTRC and all of the FLBTRDMx input files are loaded using sample CNMSJH12. Both the DUIFSTRC and all of the FLBTRDMx input files must be loaded for GMFHS operation. Note that input file DUIFSTRC must be loaded before any FLBTRDMx 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.V5R1M0.MYDEFS(OBJECTS), the last line of CNMSJH12 should be:

```
// DD DSN=NETVIEW.V5R1M0.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 should be issued 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 would require that the GMFHS CONFIG NETWORK command be used. Refer to 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 invoked 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.

Refer to 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, the RODM CHKPT command should be issued following the completion of the RODM load function job. This will cause 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 6 shows which GMFHS CONFIG command to use when changing a field of an object in the Non_SNA_Domain_Class.

<table>
<thead>
<tr>
<th>Field</th>
<th>GMFHS CONFIG Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlertProc</td>
<td>NETWORK</td>
</tr>
<tr>
<td>CommandTimeoutInterval</td>
<td>NETWORK</td>
</tr>
<tr>
<td>ContainsResource</td>
<td>NETWORK, DOMAIN (see note)</td>
</tr>
<tr>
<td>DomainCharacteristics</td>
<td>NETWORK</td>
</tr>
<tr>
<td>DomainCharacteristics2</td>
<td>NETWORK</td>
</tr>
<tr>
<td>EMDomain</td>
<td>NETWORK</td>
</tr>
<tr>
<td>InitialResourceStatus</td>
<td>NETWORK</td>
</tr>
<tr>
<td>PresentationProtocolName</td>
<td>NETWORK</td>
</tr>
<tr>
<td>ReportsToAgent</td>
<td>NETWORK</td>
</tr>
<tr>
<td>SessionProtocolName</td>
<td>NETWORK</td>
</tr>
<tr>
<td>TransactionProgram</td>
<td>NETWORK</td>
</tr>
<tr>
<td>WindowSize</td>
<td>NETWORK</td>
</tr>
</tbody>
</table>

*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. Refer to NetView online help for a complete description of the CONFIG DOMAIN command before deciding if it should be used.

**SNA_Domain_Class Changes**

Table 7 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>
<thead>
<tr>
<th>Field</th>
<th>GMFHS CONFIG Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContainsResource</td>
<td>NETWORK</td>
</tr>
<tr>
<td>SNANet</td>
<td>NETWORK</td>
</tr>
</tbody>
</table>
**NMG Class Changes**

Table 8 shows which GMFHS CONFIG command to use when changing a field of an object in the NMG Class.

**Table 8. GMFHS CONFIG Command for NMG Class Objects**

<table>
<thead>
<tr>
<th>Field</th>
<th>GMFHS CONFIG Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentStatusEffect</td>
<td>NETWORK</td>
</tr>
<tr>
<td>CommandRouteLUName</td>
<td>NETWORK</td>
</tr>
<tr>
<td>Domain</td>
<td>NETWORK</td>
</tr>
<tr>
<td>NMGCharacteristics</td>
<td>NETWORK</td>
</tr>
<tr>
<td>ReportsOnDomain</td>
<td>NETWORK</td>
</tr>
<tr>
<td>TransportProtocolName</td>
<td>NETWORK</td>
</tr>
<tr>
<td>WindowSize</td>
<td>NETWORK</td>
</tr>
</tbody>
</table>

**GMFHS_Managed_Real_Objects_Class Changes**

Table 9 shows which GMFHS CONFIG command to use when changing a field of an object in the GMFHS_Managed_Real_Objects_CLASS.

**Table 9. GMFHS CONFIG Command for GMFHS_Managed_Real_Objects_Class Objects**

<table>
<thead>
<tr>
<th>Field</th>
<th>GMFHS CONFIG Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>DOMAIN (see note)</td>
</tr>
</tbody>
</table>

**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. Refer to NetView online help for a complete description of the CONFIG DOMAIN command before deciding if it should be used.

**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 explains 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. If the NMG is running on the OS/2® operating system with Remote Operations Service installed, issue a RUNCMD from NetView to set the workstation clock using the ROP services.

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.

Finally, this chapter explains the differences between NETCENTER protocols and GMFHS protocols. If you are migrating from NETCENTER, use this chapter to understand how to use your existing NMGs with GMFHS.

Table 10 shows the values for the three GMFHS protocol fields for typical NMGs.

<table>
<thead>
<tr>
<th>NMG name</th>
<th>PresentationProtocolName</th>
<th>SessionProtocolName</th>
<th>TransportProtocolName</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAN Network Manager</td>
<td>DOMP020</td>
<td>PASSTHRU</td>
<td>COS</td>
</tr>
<tr>
<td>NAP</td>
<td>DOMP010</td>
<td>DOMS010</td>
<td>COS</td>
</tr>
<tr>
<td>NetView OST¹</td>
<td>DOMP020</td>
<td>PASSTHRU</td>
<td>OST</td>
</tr>
<tr>
<td>NetView OST</td>
<td>PASSTHRU</td>
<td>PASSTHRU</td>
<td>OST</td>
</tr>
<tr>
<td>NetView/PC</td>
<td>DOMP010</td>
<td>DOMS010</td>
<td>COS</td>
</tr>
</tbody>
</table>
Table 10. GMFHS Protocol Values for Typical NMGs (continued)

<table>
<thead>
<tr>
<th>NMG name</th>
<th>Presentation ProtocolName</th>
<th>Session ProtocolName</th>
<th>Transport ProtocolName</th>
</tr>
</thead>
<tbody>
<tr>
<td>NetView PPI</td>
<td>NONE</td>
<td>NONE</td>
<td>PPI</td>
</tr>
<tr>
<td>NetView/6000 V1</td>
<td>DOMP020</td>
<td>PASSTHRU</td>
<td>COS</td>
</tr>
<tr>
<td>NetView/6000 V2</td>
<td>DOMP010</td>
<td>DOMS010²</td>
<td>COS</td>
</tr>
<tr>
<td>NetView for AIX V3</td>
<td>DOMP010</td>
<td>DOMS010²</td>
<td>COS</td>
</tr>
<tr>
<td>NetView for AIX V4</td>
<td>DOMP010</td>
<td>DOMS010²</td>
<td>COS</td>
</tr>
<tr>
<td>Open Topology Interface Agent³</td>
<td>DOMP010</td>
<td>NONE</td>
<td>COS</td>
</tr>
<tr>
<td>PPI</td>
<td>DOMP020</td>
<td>PASSTHRU</td>
<td>PPI</td>
</tr>
</tbody>
</table>

¹ Use the DOMP020 presentation protocol if you want to use parameter substitution.
² See "Session Establishment for NetView/6000 V2, NetView for AIX V3, NetView for AIX V4, and DOMS010" on page 79 for more information.
³ 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 63.

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. GMFHS 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 protocol is similar to the NETCENTER NSI1 presentation protocol, but the DOMP010 protocol provides some enhancements. If you do not want to use these enhancements, set bit 13 on in the DomainCharacteristics field of the Non_SNA_Domain_Class object. GMFHS does not support the NETCENTER generic Enable and Disable commands. For a more complete description of the differences between GMFHS and NETCENTER, see "Migrating from NETCENTER Protocols to GMFHS Protocols" on page 87.

The DOMP010 presentation protocol is only applicable 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:

<table>
<thead>
<tr>
<th>Generic Command</th>
<th>GMFHS Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>ActivateCommandText</td>
</tr>
<tr>
<td>Deactivate</td>
<td>DeactivateCommandText</td>
</tr>
<tr>
<td>Display Abnormal Status</td>
<td>DisplayAbnormalStatusCommandText</td>
</tr>
<tr>
<td>Display Status</td>
<td>DisplayStatusCommandText</td>
</tr>
<tr>
<td>Reconfigure</td>
<td>ReconfigureCommandText</td>
</tr>
<tr>
<td>Recycle</td>
<td>RecycleCommandText</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Token</th>
<th>Action taken by GMFHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>%APPL%</td>
<td>Replace with the value of the TransactionProgram field of the Non_SNA_Domain_Class object.</td>
</tr>
<tr>
<td>%DOMAIN%</td>
<td>Replace with the value of the EMDomain field of the Non_SNA_Domain_Class object.</td>
</tr>
<tr>
<td>%RESOURCE%</td>
<td>Replace with the value of the MyName field of the resource.</td>
</tr>
<tr>
<td>%SPNAME%</td>
<td>Replace with the value of the MyName field of the NMG_Class object.</td>
</tr>
<tr>
<td>%TYPE%</td>
<td>Replace with the value of the TypeName field of the Display_Resource_Type_Class object associated with the resource.</td>
</tr>
</tbody>
</table>

**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 PASSTRU 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's 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 73 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)
```

Chapter 4. Communicating with Network Management Gateways 65
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:

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

The following sections describe each keyword and its values.

**Command Execution—CE**
The command execution status keyword (CE) indicates a failure to successfully execute 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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Display abnormal status</td>
</tr>
<tr>
<td>C</td>
<td>Reconfigure domain</td>
</tr>
<tr>
<td>D</td>
<td>Display status for a named resource or resources</td>
</tr>
<tr>
<td>I</td>
<td>Inactivate resource</td>
</tr>
<tr>
<td>N</td>
<td>Native command</td>
</tr>
<tr>
<td>P</td>
<td>Protocol message</td>
</tr>
<tr>
<td>R</td>
<td>Recycle resource</td>
</tr>
<tr>
<td>V</td>
<td>Activate resource</td>
</tr>
<tr>
<td>X</td>
<td>Negative response</td>
</tr>
</tbody>
</table>

- The second byte is the continuation.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>This is either an initial request or the last response to an initial request.</td>
</tr>
<tr>
<td>M</td>
<td>This is either a continuation request or not the last response when multiple responses are required to service an initial request.</td>
</tr>
</tbody>
</table>

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

**Table 11. Protocol Command Values**

<table>
<thead>
<tr>
<th>Protocol Command</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SESSION_REQUEST</td>
<td>Sent by GMFHS to the element manager to request that a session be established.</td>
</tr>
<tr>
<td>Protocol Command</td>
<td>Meaning</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SESSION_REQUEST_ACCEPT</td>
<td>A response acknowledging a SESSION_REQUEST protocol command. This command does not indicate that a session is established.</td>
</tr>
<tr>
<td>INIT_ACCEPT</td>
<td>Returned by GMFHS to acknowledge receipt of the INIT alert.</td>
</tr>
<tr>
<td>INIT_ACCEPT_ACCEPT</td>
<td>A response acknowledging the INIT_ACCEPT protocol command.</td>
</tr>
<tr>
<td>SET_CLOCK</td>
<td>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.</td>
</tr>
<tr>
<td>SET_CLOCK_ACCEPT</td>
<td>Returned by the native element manager to acknowledge the SET_CLOCK protocol command.</td>
</tr>
</tbody>
</table>

**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 78 for more information about these protocols.

**Reason—RN**
The reason keyword (RN) indicates why a request could not be 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 12 lists the supported text values.

Table 12. Reason Values

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

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 67. 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 could not 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 resource’s GMFHS external status or the NETCENTER internal status.

- A 1-byte value is used to describe the GMFHS external status of a resource.
- A value list is used to describe the NETCENTER internal status of a resource.

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.

Table 13 on page 72 shows the single-byte external statuses and the NETCENTER equivalents.

**Note:** You can define your non-SNA domain to recognize either type of status. If bit 13 of the DomainCharacteristics field is turned on in the object of the Non_SNA_Domain_Class, GMFHS translates the NETCENTER status keywords to the GMFHS equivalent.
Table 13. NETCENTER to NetView Status Keyword Conversions and Description

<table>
<thead>
<tr>
<th>GMFHS Status</th>
<th>NETCENTER Status</th>
<th>NETCENTER Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>U (unsatisfactory)</td>
<td>A</td>
<td>Abnormal—the device is running but there is an abnormal condition.</td>
</tr>
<tr>
<td>U (unsatisfactory)</td>
<td>B</td>
<td>Disabled—intentional deactivation by the operator or system</td>
</tr>
<tr>
<td>? (unknown)</td>
<td>C</td>
<td>Not configured—the device is not part of the network definition.</td>
</tr>
<tr>
<td>U (unsatisfactory)</td>
<td>D</td>
<td>Down</td>
</tr>
<tr>
<td>S (satisfactory)</td>
<td>N</td>
<td>Normal</td>
</tr>
<tr>
<td>I (intermediate)</td>
<td>P</td>
<td>Performance—performance problem</td>
</tr>
<tr>
<td>I (intermediate)</td>
<td>T</td>
<td>Transient—device is currently changing status.</td>
</tr>
<tr>
<td>? (unknown)</td>
<td>U</td>
<td>Unavailable—no status is available.</td>
</tr>
<tr>
<td>No change</td>
<td>X</td>
<td>Request for status could not be executed—any status previously reported is to be regarded as still in effect.</td>
</tr>
</tbody>
</table>

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

$T=U$

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:

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

GMFHS supports all NETCENTER status values for migration purposes. It automatically converts the NETCENTER internal resource status values to GMFHS status values.

The three NETCENTER categories of internal status (configuration, operation, and utilization) are placed in a value list. For example:

$ST=(configuration,operation,utilization)$

Each position within the list defines the status for that category of the component and is 1 byte in length. The values used to describe the status of the different resources of each list element are described in Table 14.

Table 14. Resource Status Values

<table>
<thead>
<tr>
<th>Category</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration</td>
<td>C</td>
<td>Nonconfigured</td>
</tr>
<tr>
<td></td>
<td>U</td>
<td>Unavailable</td>
</tr>
<tr>
<td></td>
<td>V</td>
<td>Active</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>Inactive</td>
</tr>
<tr>
<td>Operation</td>
<td>A</td>
<td>Abnormal</td>
</tr>
<tr>
<td></td>
<td>L</td>
<td>Nonoperational</td>
</tr>
<tr>
<td></td>
<td>O</td>
<td>Operational</td>
</tr>
<tr>
<td></td>
<td>U</td>
<td>Unavailable</td>
</tr>
</tbody>
</table>
Table 14. Resource Status Values (continued)

<table>
<thead>
<tr>
<th>Category</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Utilization</td>
<td>N</td>
<td>Normal</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>Overload</td>
</tr>
<tr>
<td></td>
<td>U</td>
<td>Unavailable</td>
</tr>
</tbody>
</table>

When the GMFHS displays resource status in a view or generic command response, it consists of the three internal status values.

For example, if the three categories for a resource are configuration=unavailable, operation=operational, and utilization=normal, the ST keyword parameter format is:

ST=(U,O,N)

**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=yyymmddhhmmss

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 66 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 67 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 22 shows the
packets exchanged for a Display Status command and response.

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 should 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 23* shows the packets exchanged for a Display Status command and the response in the simplest multiple-response case.

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 terminates 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 78 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 24 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 24. 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 80.
- 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 NetView/6000 V2, NetView for AIX V3, NetView for AIX V4, and DOMS010

NetView/6000 Version 2 (V2), NetView for AIX Version 3 (V3), and NetView for AIX Version 4 (V4) provide direct support for NETCENTER only. Because GMFHS has different domain naming conventions than NETCENTER, NetView supplies the sample CNMS4406 to facilitate session establishment between GMFHS and NetView/6000 V2, NetView for AIX V3, and NetView for AIX V4.

This sample provides the INIT and DOWN alert portion of DOMS010 session establishment. The sample allows the user to specify:

- The three named elements of a non-SNA domain (see “Defining Non-SNA Domains” on page 33). In sample CNMS4406 the service point (SP) is sp_name, the transaction program (TP) is tp_name, and the element management subsystem (EMS) is domain_name.
- Whether to send an INIT or DOWN alert. This alert then matches a similarly named domain object in RODM with the NetView/6000 V2 service point.

Sample CNMS4406 is a NetView command processor coded in the C language. To use it, 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, refer to the Tivoli NetView for z/OS Customization: Using PL/I and C. For information about the LONGNAME compile option, refer to OS/390 C/C++ Programming Guide (SC09-2362).

You must also place the following CMDMDL statement in the DSIPARM (DSICMD) member (use included file DSICMDU for migration purposes):

```
CNMS4406 CMDMDL MOD=CNMS4406,RES=N
```

The following is a syntax diagram for the sample:

```
+<.-VVVV)+<4bVVVV
CNMS4406

DOWN

sp_name tp_name domain_name
```

For example, to invoke sample CNMS4406 for a NetView/6000 V2 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 NetView/6000 or NetView for AIX 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 11 or page 68 for the DOMP010 protocol. The exchange occurs as illustrated in Figure 25.

**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 discussed in Session Establishment for DOMS010** on page 78.

**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 15 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 15. Generic Alert Subvectors**

<table>
<thead>
<tr>
<th>Subvector</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic alert data subvector</td>
<td>Alert Type: X’12’ (unknown)</td>
</tr>
<tr>
<td></td>
<td>Alert description code: XFE00’ (undetermined error)</td>
</tr>
<tr>
<td>Probable cause subvector</td>
<td>Probable cause code point: X’1001’ (application program)</td>
</tr>
<tr>
<td>Cause undetermined subvector</td>
<td>Recommended action code point: X’0700’ (no action necessary)</td>
</tr>
<tr>
<td>Subvector</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>First product set ID subvector</td>
<td>Product classification: X‘xC’ (non-IBM software)</td>
</tr>
<tr>
<td></td>
<td>Software product common name: Identifier of the NMG application (in the non-SNA network) that communicates across the NMG API.</td>
</tr>
<tr>
<td></td>
<td>Software product common level: 000000</td>
</tr>
<tr>
<td></td>
<td>Software product program number: USER0</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The first product set ID subvector is included to comply with SNA but does not carry significant information.</td>
</tr>
<tr>
<td>Second product set ID subvector</td>
<td>Product classification: X‘xC’ (non-IBM software)</td>
</tr>
<tr>
<td></td>
<td>Software product common name: name of the native element manager that receives commands</td>
</tr>
<tr>
<td></td>
<td>Software product common level: 000000</td>
</tr>
<tr>
<td></td>
<td>Software product program number: USER0</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The second product set ID subvector is included to comply with SNA but does not carry significant information.</td>
</tr>
<tr>
<td>Date/Time subvector (optional)</td>
<td>An X‘01’ subvector containing date and time information.</td>
</tr>
<tr>
<td>Hierarchy resource list subvector</td>
<td>First resource name (mandatory): Name of the service point</td>
</tr>
<tr>
<td></td>
<td>First resource type identifier (mandatory): X‘81’ (service point)</td>
</tr>
<tr>
<td></td>
<td>Transaction program resource (optional):</td>
</tr>
<tr>
<td></td>
<td>Transaction program identifier (optional): X‘18’ (transaction program)</td>
</tr>
<tr>
<td></td>
<td>Additional resource name (optional): As required, to uniquely identify the domain</td>
</tr>
<tr>
<td></td>
<td>Additional resource type identifier (optional): Any</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> 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.</td>
</tr>
<tr>
<td>Subvector</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>Self-defining text message subvector</td>
<td>Text message: INIT,GMT=chhmm</td>
</tr>
</tbody>
</table>

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 26 shows the alert exchange during session termination.

![Figure 26. Session Termination](image)

**Note:** The session termination alert is identical to the alert described in "INIT Generic Alert for Session Establishment" on page 80 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 should use RUNCMDs to communicate with the service point.
- **PPI** indicates that the NMG uses a program-to-program interface (PPI) and that GMFHS should 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, executes 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 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 explains why the PPI send request failed. Refer to the 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 16 shows the subvectors and subfields that are included in the execute command major vector.

**Table 16. Subvectors and Subfields in the Execute Command Major Vector**

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

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 17 on page 87 shows the subvectors and subfields of the text data parameter major vector.
Table 17. Subvectors and Subfields in the Text Data Parameter Major Vector

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

Migrating from NETCENTER Protocols to GMFHS Protocols

The protocols used by GMFHS are similar to the protocols used by NETCENTER. Table 18 shows the values specified for NETCENTER protocols and the corresponding values you can specify for GMFHS protocols.

Table 18. Conversion of Definition Names from NETCENTER to GMFHS

<table>
<thead>
<tr>
<th>Field</th>
<th>NETCENTER</th>
<th>GMFHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SessionProtocolName</td>
<td>NSI1</td>
<td>DOMS010</td>
</tr>
<tr>
<td></td>
<td>PASSTHRU</td>
<td>PASSTHRU</td>
</tr>
<tr>
<td></td>
<td>NONE</td>
<td>NONE</td>
</tr>
<tr>
<td>PresentationProtocolName</td>
<td>NSI1</td>
<td>DOMP010</td>
</tr>
<tr>
<td></td>
<td>PASSTHRU</td>
<td>DOMP020</td>
</tr>
<tr>
<td></td>
<td>NONE</td>
<td>NONE</td>
</tr>
<tr>
<td>TransportProtocolName</td>
<td>CNMI</td>
<td>COS</td>
</tr>
<tr>
<td></td>
<td>No equivalent</td>
<td>PPI</td>
</tr>
<tr>
<td></td>
<td>No equivalent</td>
<td>OST</td>
</tr>
<tr>
<td></td>
<td>NONE</td>
<td>NONE</td>
</tr>
</tbody>
</table>

Table 19 shows the names of the NETCENTER attributes used to specify protocols, and the corresponding names of GMFHS fields you use to specify protocols.

Table 19. Conversion of NETCENTER Attribute Names to GMFHS Field Names

<table>
<thead>
<tr>
<th>Protocol</th>
<th>NETCENTER Attribute</th>
<th>GMFHS Field</th>
<th>GMFHS Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session</td>
<td>SESS</td>
<td>SessionProtocolName</td>
<td>Non-SNA domain</td>
</tr>
<tr>
<td>Presentation</td>
<td>FORMAT</td>
<td>PresentationProtocolName</td>
<td>Non-SNA domain</td>
</tr>
<tr>
<td>Transport</td>
<td>TRAN</td>
<td>TransportProtocolName</td>
<td>Network management gateway</td>
</tr>
</tbody>
</table>

The differences between the NETCENTER NSI1 protocol and the GMFHS DOMP010 protocol are:

- GMFHS conditionally sends a SET_CLOCK protocol command to the element manager every 24 hours (depending on the contents of the DomainCharacteristics field).
- GMFHS includes the sender ID keyword (SN=GMFHS) on all commands.
Migrating from NETCENTER

- GMFHS recognizes new status values that are mnemonically related to the GMFHS DisplayStatus values. GMFHS converts NETCENTER type status to one of these values if specified in the DomainCharacteristics field.

- GMFHS does not support the NETCENTER generic Enable and Disable commands.

- GMFHS allows resource names to be as many characters as possible (depending on the gateway used). NETCENTER limits the resource names to a maximum of 8 characters.

- GMFHS allows the use of a ) character (right parenthesis) in the TX text string. See "Text—TX" on page 73 for more information.
Chapter 5. How GMFHS Uses RODM

The Graphic Monitor Facility host subsystem (GMFHS) works with RODM and a
NetView management console (NMC) 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
explains how GMFHS uses RODM. Using this information, you can then modify
the contents of RODM to change how GMFHS and the NMC 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 invoked 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,
DUIFAW, is invoked to initialize the fields related to status aggregation in the
real and aggregate objects in the RODM data cache. See "DUIFAW - Aggregation
Warm Start Method" on page 498 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
terminated, 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 will be 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 will not be 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 NMC 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 that should be displayed 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 102 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 should be 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 should be 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 will 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 115 for more information. The views that use this process are identified in "Object Discovery Process Description for Specific Views" on page 96.

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 96 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 27 is used.

Prior to NetView Version 3, method DUIFCLRT was usually invoked 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 27. The disadvantage of this is that you have to invoke this method for each object.

You can now associate a Display_Resource_Type_Class object with an object class in RODM as shown in Figure 28 on page 93. 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.
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 should be connected. See "Object Connectivity Process" on page 102 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, if available:

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 29 on page 94.
2. Starting with NetView Version 3, the second technique is available. The objects and fields used by this technique are illustrated in Figure 30 on page 95.
Figure 29. View_Information_Object_Class Object Determination Technique One
A displayable object can specify a View _Information_Object_Class object using both the DisplayResourceType field (as shown in Figure 29 on page 94) and the ViewInfoRefObjVIO field (as shown in Figure 30). 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 should 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 could not find a View_Information_Object_Class object for a resource object, it could not 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 NMC server establishes a session with GMFHS, and will be 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 102 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 invoked 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 102 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, NMC 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 102 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 102 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 102 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 102 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 102 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 102 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 102 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 should be in the view, GMFHS performs the following:

- Query the ContainsLogical field of the selected object to find the name of the field that should be 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 comprise the next lower layer of the selected object.
- Pass the list of objects to the GMFHS connectivity process. See [Object Connectivity Process](#) on page 102 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 should be in the view, GMFHS performs the following:

- Query the ContainsPhysical field of the selected object to find the name of the field that should be 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 comprise the next lower layer of the selected object.
- Pass the list of objects to the GMFHS connectivity process. See [Object Connectivity Process](#) on page 102 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 102 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 will be 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
**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 102 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 will be 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 NMC server establishes a session with GMFHS, and will be 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 should be 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 should be interconnected in the view.
GMFHS does this by performing the following process, sequentially, for each object
listed. For each object, GMFHS performs the following:

1. Find the View_Information_Object_Class object.
2. Query the RelFieldNames field. This field specifies which fields should be
   queried on the object.
3. Query those fields on the object.
4. 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.
5. Pass the view to the NMC.

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 should be 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 should be
   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 NMC.

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 should be 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 should be 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.
You can customize how the DisplayStatus is interpreted by modifying the DUIFSMT table. See "Customizing the DisplayStatus Mapping Table for Exception Views" on page 106 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 112 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 96 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** 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 will remain 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, refer to the [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 will be 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 should be 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 will not be 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 NMC to determine which objects have been suspended from aggregation.

Table 20 contains examples of alternate values for the ExceptionViewFilter field and the resultant exception view:

<table>
<thead>
<tr>
<th>Value</th>
<th>Objects in View</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘0000 0000 0000 0000’ (X’0000’)</td>
<td>All objects defined to the view regardless of the DisplayStatus or UserStatus.</td>
</tr>
<tr>
<td>‘0101 0000 0000 0000’ (X’5000’)</td>
<td>All objects in an exception state defined to the view that are <em>not</em> marked. All marked objects are filtered out of the view.</td>
</tr>
<tr>
<td>‘0110 0000 0000 0000’ (X’6000’)</td>
<td>All objects in an exception state defined to the view that are marked. All objects that are <em>not</em> marked are filtered out of the view.</td>
</tr>
</tbody>
</table>

**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.

It is recommended that you 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:

```plaintext
OP DUIFRFDS INVOKED_WITH;
```

Figure 32 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 32.

**DUIFSMTE**

```
DUIFSMTE

Class:

CLASS=class_name

,XCPT=(value)

,MYNAME=resource_name

,RESOURCE=resource_name

,STGRPn=value

,USRXMEM=method_name

```

*Figure 32. 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 21 lists the aliases you can enter and their corresponding actual class names as known to RODM; both are accepted by the DUIFSMTE macro.

Table 21. Aliases for RODM Class Names

<table>
<thead>
<tr>
<th>Alias for Class</th>
<th>MyName Value for Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPNEN</td>
<td>1.3.18.0.0.1821</td>
</tr>
<tr>
<td>APPNNN</td>
<td>1.3.18.0.0.1822</td>
</tr>
<tr>
<td>APPNTRANSMISSIONGROUP</td>
<td>1.3.18.0.0.1823</td>
</tr>
<tr>
<td>APPNTRANSMISSIONGROUPCIRCUIT</td>
<td>1.3.18.0.0.2058</td>
</tr>
<tr>
<td>CROSSDOMAINRESOURCE</td>
<td>1.3.18.0.0.2281</td>
</tr>
<tr>
<td>CROSSDOMAINRESOURCERMANAGER</td>
<td>1.3.18.0.0.2278</td>
</tr>
<tr>
<td>DEFINITIONGROUP</td>
<td>1.3.18.0.0.2267</td>
</tr>
<tr>
<td>INTERCHANGENODE</td>
<td>1.3.18.0.0.1826</td>
</tr>
<tr>
<td>LENNODE</td>
<td>1.3.18.0.0.1827</td>
</tr>
<tr>
<td>LOGICALLINK</td>
<td>1.3.18.0.0.2085</td>
</tr>
<tr>
<td>LOGICALUNIT</td>
<td>1.3.18.0.0.1829</td>
</tr>
<tr>
<td>MIGRATIONDATABASE</td>
<td>1.3.18.0.0.2155</td>
</tr>
<tr>
<td>PORT</td>
<td>1.3.18.0.0.2089</td>
</tr>
<tr>
<td>T2-1NODE</td>
<td>1.3.18.0.0.1843</td>
</tr>
<tr>
<td>T4NODE</td>
<td>1.3.18.0.0.1844</td>
</tr>
<tr>
<td>T5NODE</td>
<td>1.3.18.0.0.1845</td>
</tr>
<tr>
<td>VIRTUALROUTINGNODE</td>
<td>1.3.18.0.0.1845</td>
</tr>
</tbody>
</table>

See "Implementing Exception View Processing for MultiSystem Manager" on page 114 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 will not be included in an exception view that is defined to only include exception objects. CrossDomainResourceManager in Figure 36 on page 112 will not be displayed in an exception view that has an ExceptionViewFilter of X'4000'.

The following are possible **XCPT** values:

**DEGRD**

Specifies objects with a DisplayStatus value of 133 (degraded).

**INTER**

Specifies objects with a DisplayStatus value of 132 (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 131 (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 144). 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 will be 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” 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 112 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 DUIFSMTE 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 would match in its class. (See Usage Note 3 on page 111.) For example, assume that you have a resource in the GMFHS_Managed_Real_Objects_Class whose DisplayResourceName is RALV4 and MyName is DECNET.RALV4. If you want resource DECNET.RALV4 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 33.

If, in Figure 33, the second DUIFSMTE statement had been coded before the first DUIFSMTE statement, resource DECNET.RALV4 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. Refer to the Tivoli NetView for z/OS SNA Topology Manager Implementation Guide for more information.

Figure 33 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 DECNET.RALV4 and DisplayResourceName is RALV4. If you coded DUIFSMTE entries as shown in Figure 34, the resource would match against all 3 of the DUIFSMTE entries. However, because the order in which the statements are coded is important, the first DUIFSMTE entry is the one that would match the exception state. This object will be an exception only if its DisplayStatus is intermediate.

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 35 on page 113), 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 21 on page 108 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 21 on page 108 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 35 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 35. DisplayStatus Mapping Table Coding Example 1

For the second example (shown in Figure 36), assume the following conditions:
• You have created a RODM method named CUSTMTH1 to decide whether objects of the t2-1Node should 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 36 shows the coding for the DisplayStatus mapping table.

DUIFSMTE CLASS=T2-1NODE,USRXMETH=CUSTMTH1
DUIFSMTE CLASS=CROSSDOMAINRESOURCEMANAGER
DUIFSMTE CLASS=APPNEN, RESOURCE=USIBMNT.NCPPU1, XCPT=(DEGRD,INTER,SATIS,SDGRD,UNKWN,UNSAT,MEDSA,MEDUN,LOC WSA,LOWUN)
DUIFSMTE CLASS=APPNEN, RESOURCE=USIBMNT.*, XCPT=(DEGRD,INTER,SDGRD,UNKWN,UNSAT,MEDSA,MEDUN,LOWSA,LOC WUN)

Figure 36. 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 DUIFSMTE table. A sample user method, DUIFCUXM, is provided for this purpose. Refer to 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 343.

The method is triggered asynchronously from the DUIFCRDC method and is passed the object ID for which a DisplayStatus change has occurred. The following are the input parameters for this method:

```c
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 "Chapter 11: Writing Applications that Use RODM" on page 303.

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 will be 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, should be 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 will change 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 will change 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 object’s DisplayStatus or UserStatus fields. 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. Execute 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 will be 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 sample FLCSEXV 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. Sample FLCSEXV resides in the CNMSAMP data set.
   FLCSEXV contains sample exception view statements for all of the MultiSystem Manager real object classes. There is a section for each of the MultiSystem Manager features. You can add exception views for aggregate objects. You can also create an object in the Exception_View_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. Refer to 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 CNMSTYLE.

5. The MultiSystem Manager data model is loaded using NetView sample CNMSJH12. The prologue of each of these samples contains a short description of the data model members shipped with MultiSystem Manager.

Each of the sections in FLCSEXV correlate to a data model sample.

In JCL sample CNMSJH12, uncomment the statement for the appropriate function data model sample:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Data Model Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP</td>
<td>FLCSDM6I</td>
</tr>
<tr>
<td>LAN Network Manager</td>
<td>FLCSDM6L</td>
</tr>
<tr>
<td>NetFinity</td>
<td>FLCSDM6H</td>
</tr>
<tr>
<td>Open</td>
<td>FLCSDM6O</td>
</tr>
<tr>
<td>TMR</td>
<td>FLCSDM6T</td>
</tr>
</tbody>
</table>

If you want information about... Refer to...

<table>
<thead>
<tr>
<th>Exception view processing</th>
<th>Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer’s Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>DUIFSMT</td>
<td>Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer’s Guide</td>
</tr>
</tbody>
</table>

**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 once 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. Objects in either list will have the requested views built. 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 should 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 will only query 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 invokes 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 invoked for each new class or connectivity field that is added to the data model. See "DUIFVINS - Install View Granularity Method (DUIFVNOT)" on page 501 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. Refer to the 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 should be done for this operator.

For more information about defining resources and views to spans in the NetView span table, refer to the 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 91, 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 will be 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 would match the *MY_NETWORK_VIEW* view name. Following are some 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:

<table>
<thead>
<tr>
<th>View Type</th>
<th>Suffix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Backbone</td>
<td>-BAK</td>
</tr>
<tr>
<td>Configuration Child</td>
<td>-CHD</td>
</tr>
<tr>
<td>Configuration Child II (More Detail LU)</td>
<td>-MLU</td>
</tr>
<tr>
<td>Configuration Child III (More Detail Definition Group)</td>
<td>-MDF</td>
</tr>
<tr>
<td>Configuration Logical</td>
<td>-LOG</td>
</tr>
<tr>
<td>Configuration Logical/Physical</td>
<td>-LP</td>
</tr>
<tr>
<td>Configuration Parent</td>
<td>-PAR</td>
</tr>
<tr>
<td>Configuration Physical</td>
<td>-PHY</td>
</tr>
</tbody>
</table>

Chapter 5. How GMFHS Uses RODM  117
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 will verify 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, refer to the 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 would match 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 will be unable to 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 could be 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:

\[
\text{SPANDEF SPAN=SPAN1,VIEW=(*-FP<NETA*-FP>)};
\]

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:

\[
\text{SPANDEF SPAN=SPAN1,VIEW=*<*-M*>};
\]

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 should 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:

\[
\text{SPANDEF SPAN=SPAN1,VIEW=*SHIFT1*};
\]

View name NETWORKA.OPCENTER22.OPERATOR-PAR will not match this SPANDEF statement and the operator will be unable to 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 NMC 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, refer to the Tivoli NetView for z/OS MultiSystem Manager User’s Guide.

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 are provided to help you understand how to restrict resources within views. The examples assume the following:

- 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 DECNET.RALV4. The DisplayResourceName and UserSpanName are null. The following statement is defined in the NetView span table:

```plaintext
SPANDEF SPAN=SPAN1,RESOURCE=DECNET.RALV4;
```

Thus, an operator with span SPAN1 started can display resource DECNET.RALV4 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 DECNET.RALV4 and a DisplayResourceName value of RALV4. The UserSpanName is null. The following statement is defined in the NetView span table:

```plaintext
SPANDEF SPAN=SPAN1,RESOURCE=RALV4;
```

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 RALV4. If the DisplayResourceName is null, the MyName value of DECNET.RALV4 is used to determine span authorization. For example:

```plaintext
SPANDEF SPAN=SPAN1,RESOURCE=RALV4;
```

**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:
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 DECNET.RALV4 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=DECNET.RALV4;
- SPANDEF SPAN=SPAN1,RESOURCE=RALV4;

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 only occur when one of the following 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 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 assigned in the NMC operator’s profile remains in effect for the duration of that NMC operator’s session. A changed NGMFVSPN attribute is retrieved only if the NetView operator signs off and signs back on with the new NGMFVSPN attribute and the NMC 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 NMC views. All NMC 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 119.
- For more information about using spans to protect resources, refer to the Tivoli NetView for z/OS Security Reference.

Applying Policy to Views
Using NMCSTATUS policy definitions, you can define time schedules for resources in NMC 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 NMC 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 via 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 126 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 is a list of 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 NMC 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=(RALV4)
  - TIME=(06.00.00,18.00.00)
  - STOPUPDATE=YES
  - SUSPENDAGG=YES

RODM field values:
- MyName='NMC1POLICY2'
- CollectionSpec1='|GMFHS_Managed_Real_Objects_Class|DisplayResourceName|RALV4|.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=(DEC*)
  - TIME=(06.00.00,18.00.00)
  - SUSPENDAGG=YES

RODM field values:
- MyName='NMC1POLICY3'
- CollectionSpec1='|GMFHS_Managed_Real_Objects_Class|MyName|DEC*|.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=XA

Policy definition:
- NMCSTATUS POLICY4
  - BLDVIEWSSPEC=(QSAMDSN,USER.INIT(FILE1))
  - TIME=(06.00.00,18.00.00)
  - STOPUPDATE=YES

RODM field values:
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 decremented 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 should not be sent to resource ABC on Saturdays. POLICY2 specifies status updates should 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 incremented 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 incremented 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 decremented 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 decremented 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 incremented 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 incremented 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 decremented 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 decremented by one. PolicyCtrSA=0 for resource ABC. The resource is no longer suspended from aggregation.

Example 3: An NMC operator can resume aggregation for a resource that is currently suspended from aggregation by a policy. Setting or clearing the suspend flag from NMC overrides any policy that is active. However, the PolicyCtrSA field
is incremented and decremented 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 will continue to update the suspend flag when policies are activated and deactivated.

Policy definitions:

1. Saturday at 12:00 a.m., a timer pops and POLICY1 is activated. The PolicyCtrSA field of resource PC1 is incremented 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 incremented 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 decremented 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., an NMC operator clears the suspend flag for resource PC1. PolicyCtrSA remains unchanged (it is still equal to one) 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 decremented 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.

Even though an NMC operator can change the value of the suspend flag of a resource, policy will continue 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:
1. Saturday at 12:00 a.m., a timer pops and POLICY1 is activated. The PolicyCtrSU field of resource PC1 is incremented 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 incremented 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 incremented 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 decremented 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 decremented 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 decremented 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 occurs in GMFHS:

- The resource’s suspend flag is set.
- The resource’s suspend flag note is set to Scheduled.
- One is added to the resource’s PolicyCtrSA.

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 occurs in GMFHS:

- The resource’s suspend flag is cleared.
- The resource’s suspend flag note is cleared.
- One is subtracted from the resource’s PolicyCtrSA.

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.

An NMC operator can override the setting of the suspend flag. Refer to “Resources Belonging to Multiple Policies” on page 126 for more information.

Suspending Aggregation Via 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 occurs in GMFHS:

- The real resource’s suspend flag is set.
- The real resource’s suspend flag note to Scheduled.
- One is added to the real resource’s PolicyCtrSA.
- The aggregate’s child suspended flag is set.
- The aggregate’s child suspended flag note 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 occurs in GMFHS:

- The real resource’s suspend flag is cleared.
- The real resource’s suspend flag note is cleared.
- One is subtracted from the real resource’s PolicyCtrSA.
- The aggregate’s child suspended flag is cleared.
- The aggregate’s child suspended flag note 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 incremented 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 decremented 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 AGGPOICY. 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 appears in an open NMC 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 occurs 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 appears in an open NMC view.
- One is added to the resource’s PolicyCtrSU field.

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 NMC.

When system status updates are resumed, the resource is removed from the collection representing the policy. The following occurs in GMFHS:
- One is subtracted from the resource’s PolicyCtrSU field.
- If the resource’s PolicyCtrSU field=0, then the DisplayStatus field is set to the current value of the PolicyDisplayStatus field. This drives an NMC update to change the resource from Scheduled status to its current system status.
- If the resource’s 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**

Refer to the [Tivoli NetView for z/OS Administration Reference](#) for information about creating and loading a policy file containing NMCSTATUS policy definitions.

Refer to the [Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer’s Guide](#) for information about the RODM Collection Manager.

Refer to [Tivoli NetView for z/OS Installation: Configuring Graphical Components](#) for information about the tasks necessary to process NMCSTATUS policy definitions.

Refer to the [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 NMC 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 96.
- 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 37 on page 133 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 37 on page 133, 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 37 on page 133, 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 37, 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 37, 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 37. 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 37 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 37, 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 terminates one or more aggregation paths. For example, in Figure 37, 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 37, 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 139].

• 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 could not be a child of A33. This would result in the path A12→A33→A22→A12→A33→A22..., which would loop 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 37 on page 133, 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 could be composed of objects that form a hierarchy similar to that shown in Figure 37 on page 133, 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, refer to the [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 38 on page 133, R2’s AggregationParent field contains links to two objects, A11 and A12. A22’s AggregationParent field contains links to one object, A12. A22’s AggregationChild field contains links to three objects, A33, A34, and A35.
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 493.

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 37 on page 133, A34 can be unlinked from A22 and linked to A31. This procedure has no affect 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 could be different. GMFHS dynamically handles these hierarchy changes when a link or unlink is done using method DUIFCUAP.

**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 37 on page 133, 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 141 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 140 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 a real object’s contribution to an aggregate object’s status. 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 106.

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 102.

**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 37 on page 133, 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 39. 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 should 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 39 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 NMC, 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, refer to the NMC online help.
• By setting the UserStatus field directly in RODM, using RODMView. For more information, refer to the 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 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 (value + 100)\(\times\) (Total number of reals reporting to the aggregate)\(\times\) 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 aggregate parent’s XCPT group, 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 [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 aggregate object’s status.

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 138, 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 aggregate object’s highest matching status group. 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 SeverelyDegraded 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 should 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 138

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 is a description of 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 object’s contribution 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 could 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:

- Using the NetView management console workstation. For more information, refer to the Tivoli NetView for z/OS NetView Management Console User’s Guide.
- Using the NMC. For more information, refer to the NMC online help.
- By setting the AggregationPriorityValue field directly in RODM (using RODMView for example). For more information, refer to the 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, ThresholdSeverely Degraded, and ThresholdUnsatisfactory fields:

- Using the NMC. For more information, refer to the NMC online help.
- By setting the fields directly in RODM (using RODMView for example). For more information, refer to the 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, refer to the Tivoli NetView for z/OS Data Model Reference.

Note: You cannot use the NMC 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:

- Using the NMC. For more information, refer to the NMC online help.
- By setting the UserStatus field directly in RODM (using RODMView for example). For more information, refer to [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 should 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 should 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:

- Using the NMC. For more information, refer to the NMC online help.
- By setting the field directly in RODM (using RODMView for example). For more information, refer to the [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 490 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 aggregate object’s status.

See “GMFHS Methods” on page 490 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 103 is used for this purpose. The STGRPn 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 106.

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 STGRPn 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 Display Status

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 40 shows the coding of the DisplayStatus mapping table. The fourth statement sets the defaults.

```plaintext
DUIFSMTE CLASS=T4NODE,XCPT=(UNSAT,UNKWN),STGRP1=(UNSAT)
DUIFSMTE CLASS=1.3.18.0.0.1821,XCPT=(UNSAT,INTER,UNKWN),
    STGRP2=(INTER,UNKWN)
DUIFSMTE CLASS=GMFHS_Aggregate_Objects_Class,XCPT=(SDGRD),
    STGRP1=(DS136),STGRP2=(DS137)
DUIFSMTE CLASS=ALL,XCPT=(UNSAT,MEDUN),STGRP3=(UNSAT)
```

Figure 40. Example of Customizing Aggregate Display Status

Using the Collection Definition Objects

This section describes how to use 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 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 which holds information that is interpreted by the NMC console.
- A logic tree of rules an object must pass to be included in the Network_View_Class or GMFHS_Aggregate_Objects_Class collection creation object.

**Collection Definition Object Fields**

Refer to the *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 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 fields of the collection definition object. These fields are concatenated together in ascending numerical order of the numeric portion of the field to create the full collection specification. The first CollectionSpec 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 would affect 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:

```plaintext
{Class/Field} operation (Value) => list_of_objects
```
For each object in the given Class, take the value of the object’s Field 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 discussed later.

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 would be:
|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 collection creation object’s object list field.

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, (list_of_objects) (leaf_specification) (leaf_specification) AND 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
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 should 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 shows the stack operations that occur while evaluating the example on page 149:

(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 \textit{a\_b\_objects} and added to the stack. The result is:

<table>
<thead>
<tr>
<th>Stack contains:</th>
<th>a_b_objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remaining spec.</td>
<td>\textit{c_d_objects} (e) (f) EQ (g) (h) EQ AND OR</td>
</tr>
</tbody>
</table>

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 \textit{c\_d\_objects} and added to the stack. The result is:

<table>
<thead>
<tr>
<th>Stack contains:</th>
<th>c_d_objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack contains:</td>
<td>a_b_objects</td>
</tr>
<tr>
<td>Remaining spec.</td>
<td>\textit{AND (e) (f) EQ (g) (h) EQ AND OR}</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Stack contains:</th>
<th>a_b_AND_c_d_objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remaining spec.</td>
<td>(e) (f) EQ (g) (h) EQ AND OR</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Stack contains:</th>
<th>e_f_objects</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a_b_AND_c_d_objects</td>
</tr>
<tr>
<td>Remaining specifications:</td>
<td>(g) (h) EQ AND OR</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Stack contains:</th>
<th>g_h_objects</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>e_f_objects</td>
</tr>
<tr>
<td></td>
<td>a_b_AND_c_d_objects</td>
</tr>
<tr>
<td>Remaining specifications:</td>
<td>AND OR</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Stack contains:</th>
<th>e_f_AND_g_h_objects</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a_b_objects AND c_d_objects</td>
</tr>
<tr>
<td>Remaining specifications:</td>
<td>OR</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Stack contains:</th>
<th>a_b_AND c_d_objects_ORS_f_AND_g_h_objects)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remaining specifications:</td>
<td></td>
</tr>
</tbody>
</table>

At this point, there should 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:

```plaintext
<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>
<conjunction> -or- <node_specification><separator><node_specification><separator><conjunction>

<leaf_specification> ::= <class_name><separator><field_name><separator><value_list>
<separator><operator>
```

Chapter 5. How GMFHS Uses RODM  151
<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 could
possibly appear in a <class_name>, <field_name>, or <value>. The NetView
Management Console GUI uses the vertical bar (|) 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 wildcarding. 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 would be 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 would be interpreted as a DOS wildcarded value of +value. The plus sign as an escape character is only effective 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* would match any string that had an re that was 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...xYyZz]. 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, would fail to match any that contains even one a, b, or c.

**expression[m] or expression[m,u] 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: 99ae9, aaaaae, a999e99.

$ (dollar symbol)

The dollar symbol matches the end of the string.

\ (backslash)

The backslash character turns off the special meaning of any character following the backslash, thereby forcing the character to be interpreted 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.

Groups a subexpression allowing an operator, such as *, +, or [ ], to work on the subexpression enclosed in parentheses. For example, (a*(cb+)*) would match 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 would match:
- BISTRO
- MISERLY
- MISER

but not:
- MISTER
- DISRUPT
The same pattern expressed as a regular expression would be \.*IS.R.*

The pattern \RE[AGLRU]+.E[\^A-0]+.*ON is a regular expression that would match:
- REGULAR EXPRESSION
- REGAL-EXPATRIATION

but not:
- 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.

<table>
<thead>
<tr>
<th>RODM Data Type</th>
<th>Allowed in Leaf Specification</th>
<th>Allows Wildcards /Regular Expressions</th>
<th>Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANONYMOUS</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ANONYMOUSVAR</td>
<td>Yes</td>
<td>No</td>
<td>[Value] contains only the characters ‘0’ or ‘1’, which are converted to an actual bitstring before the comparison.</td>
</tr>
<tr>
<td>APPLICATIONID</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>BERVAR</td>
<td>Yes</td>
<td>No</td>
<td>[Value] contains only the characters ‘0’ or ‘1’, which are converted to an actual bitstring before the comparison.</td>
</tr>
<tr>
<td>CHARVAR</td>
<td>Yes</td>
<td>Yes</td>
<td>None (treated as a character string)</td>
</tr>
<tr>
<td>CHARAVARADDR</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>CLASSID</td>
<td>No</td>
<td>No</td>
<td>None (treated as a character string)</td>
</tr>
<tr>
<td>CLASSIDLIST</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>CLASSLINKLIST</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ECBADDRESS</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>RODM Data Type</td>
<td>Allowed in Leaf Specification</td>
<td>Allows Wildcards /Regular Expressions</td>
<td>Conversion</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------</td>
<td>----------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FIELDID</td>
<td>Yes</td>
<td>No</td>
<td>{Value} is converted to an integer. It is an error if {Value} contains characters that cannot be converted to a floating point variable.</td>
</tr>
<tr>
<td>FLOATING</td>
<td>Yes</td>
<td>No</td>
<td>{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.</td>
</tr>
<tr>
<td>GRAPHICVAR</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>INTEGER</td>
<td>Yes</td>
<td>No</td>
<td>{Value} is converted to an integer. It is an error if {Value} contains characters that cannot be converted to an integer.</td>
</tr>
<tr>
<td>INDEXLIST</td>
<td>Yes</td>
<td>Yes</td>
<td>None (Each value in the IndexList is treated as a CharVar, regardless of it’s actual type. At least one value must compare successfully for the IndexList to compare successfully.</td>
</tr>
<tr>
<td>METHODNAME</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>METHODPARAMETERLIST</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>METHODSPEC</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>OBJECTID</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>OBJECTIDLIST</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>OBJECTLINK</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>OBJECTLINKLIST</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>OBJECTNAME</td>
<td>Yes</td>
<td>Yes</td>
<td>None (treated as a character string)</td>
</tr>
<tr>
<td>RECIPIENTSPEC</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
### 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 (|) will serve as the separator character on the CollectionSpec1 field.

The CDO object that describes this collection could be specified as follows in a RODM loader file:

```plaintext
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

<table>
<thead>
<tr>
<th>RODM Data Type</th>
<th>Allowed in Leaf Specification</th>
<th>Allows Wildcards /Regular Expressions</th>
<th>Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELFDEFINING</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>SHORTNAME</td>
<td>No</td>
<td>No</td>
<td>None (treated as a character string)</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>Yes</td>
<td>No</td>
<td>[Value] is converted to a short integer. It is an error if [Value] contains characters that cannot be converted to a short integer,</td>
</tr>
<tr>
<td>SUBSCRIBEID</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>SUBSCRIPTSPEC</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>SUBSCRIPTSPEC_LIST</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>TRANSID</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
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 could 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 fields. It could have been 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 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 specification would, by definition, produce object lists that containe 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 could be specified as follows in a RODM loader file:

```plaintext
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.$|CONTAINS.'
  'GMFHS_Managed_Real_Objects_Class|MyName|TESTB*|NCONTAINS.|AND.'
  'GMFHS_Aggregate_Objects_Class|DisplayStatus|130|LE.|AND.'
  'GMFHS_Aggregate_Objects_Class|MyName|TEST[A-C].*$|CONTAINS.'
  'GMFHS_Aggregate_Objects_Class|MyName|TESTB*|NCONTAINS.|AND.|OR.'; END;
```

Assume the following objects exist in the GMFHS_Managed_Real_Objects_Class:

<table>
<thead>
<tr>
<th>Object ID</th>
<th>MyName</th>
<th>DisplayStatus</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TESTACPU10</td>
<td>131</td>
</tr>
<tr>
<td>2</td>
<td>TESTACPU11</td>
<td>129</td>
</tr>
<tr>
<td>3</td>
<td>TESTBCPU10</td>
<td>130</td>
</tr>
<tr>
<td>4</td>
<td>TESTBCPU11</td>
<td>132</td>
</tr>
<tr>
<td>5</td>
<td>TESTCCPU10</td>
<td>129</td>
</tr>
<tr>
<td>6</td>
<td>TESTCCPU11</td>
<td>132</td>
</tr>
<tr>
<td>7</td>
<td>TESTCCPU12</td>
<td>129</td>
</tr>
<tr>
<td>8</td>
<td>TESTCCPU12X</td>
<td>130</td>
</tr>
<tr>
<td>9</td>
<td>TESTDCPU10</td>
<td>129</td>
</tr>
</tbody>
</table>
Assume the following objects exist in the GMFHS_Aggregate_Objects_Class:

<table>
<thead>
<tr>
<th>Object ID</th>
<th>MyName</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>TESTAAGGs</td>
</tr>
<tr>
<td>11</td>
<td>TESTBAGGS</td>
</tr>
<tr>
<td>12</td>
<td>TESTCAGGS</td>
</tr>
<tr>
<td>13</td>
<td>TESTDAGGS</td>
</tr>
</tbody>
</table>

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 will 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 (NRM) views and how they can be customized. NetView Resource Manager enables you to graphically monitor and manage NetView tasks for resource utilization and status via NMC. You can monitor all NetViews in your enterprise via one NMC. For more information about setting up and using NetView Resource manager see:

- The *Tivoli NetView for z/OS Installation: Configuring Graphical Components*
- *Tivoli NetView for z/OS User’s Guide*

**NetView Resource Manager Views**

When NRM is active, **NetViewTasks** appears in the NetView management console view tree. This opens a view of the NRM 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, open the **Resource Properties** notebook.
The monitor value is in the **Data 1** field.
This field does not automatically update dynamically. If you would like for this field to update dynamically, see "Using DUIFVINS with NetView Resource Manager" on page 166.

The default status values for NRM real objects are:

- 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 NRM object representing 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. NRM real objects are in the GMFH5_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 NRM objects, see "Modifying DUIFSMT for NetView Resource Manager" on page 166.
NetView Resource Manager Object Information
NetView Resource Manager aggregate objects are in the GMFHS_Aggregate_NRM_Objects_Class class. NRM real objects are in the GMFHS_Managed_Real_NRM_Objects_Class class. All NRM objects have an "NRM" prefix in the MyName field.

NMC 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 22. The available commands can be selected by clicking the right mouse button on the selected object. Command results will be displayed on console log of the NetView management console.

Table 22. NMC Commands Supported

<table>
<thead>
<tr>
<th>Task</th>
<th>Available commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSRBS</td>
<td>• DST</td>
</tr>
<tr>
<td>LIST SAFOP=opid</td>
<td>• OST</td>
</tr>
<tr>
<td></td>
<td>• NNT</td>
</tr>
<tr>
<td></td>
<td>• AOST (Autotask)</td>
</tr>
<tr>
<td>LIST taskname</td>
<td>• PPT</td>
</tr>
<tr>
<td></td>
<td>• Automatic Tasks</td>
</tr>
<tr>
<td></td>
<td>• DST</td>
</tr>
<tr>
<td></td>
<td>• OPT</td>
</tr>
<tr>
<td></td>
<td>• OST</td>
</tr>
<tr>
<td></td>
<td>• NNT</td>
</tr>
<tr>
<td></td>
<td>• AOST (Autotask)</td>
</tr>
<tr>
<td></td>
<td>• HCT</td>
</tr>
<tr>
<td>LIST STATUS=TASKS</td>
<td>• NetView Aggregate</td>
</tr>
<tr>
<td>LIST STATUS=VOST</td>
<td>• VOST (Virtual OST)</td>
</tr>
<tr>
<td>Message</td>
<td>• OST</td>
</tr>
<tr>
<td></td>
<td>• NNT</td>
</tr>
<tr>
<td></td>
<td>• AOST (Autotask)</td>
</tr>
<tr>
<td></td>
<td>• VOST (Virtual OST)</td>
</tr>
<tr>
<td>Query/Set Thresholds(^1)</td>
<td>• NetView Aggregate</td>
</tr>
<tr>
<td></td>
<td>• MAINTASK</td>
</tr>
<tr>
<td></td>
<td>• PPT</td>
</tr>
<tr>
<td></td>
<td>• Automatic Tasks</td>
</tr>
<tr>
<td></td>
<td>• DST</td>
</tr>
<tr>
<td></td>
<td>• OPT</td>
</tr>
<tr>
<td></td>
<td>• OST</td>
</tr>
<tr>
<td></td>
<td>• NNT</td>
</tr>
<tr>
<td></td>
<td>• AOST (Autotask)</td>
</tr>
<tr>
<td></td>
<td>• VOST (Virtual OST)</td>
</tr>
<tr>
<td></td>
<td>• HCT</td>
</tr>
<tr>
<td>RECYCLET</td>
<td>• DST</td>
</tr>
<tr>
<td></td>
<td>• OPT</td>
</tr>
<tr>
<td>RESOURCE</td>
<td>• NetView Aggregate</td>
</tr>
</tbody>
</table>
Table 22. NMC Commands Supported (continued)

<table>
<thead>
<tr>
<th>Task</th>
<th>Available commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>START HCL=hclname¹</td>
<td>• HCT</td>
</tr>
</tbody>
</table>
| START TASK=taskname¹ | • DST  
| | • OPT |
| STOP FORCE=taskname¹ | • DST  
| | • OPT  
| | • OST  
| | • NNT  
| | • AOST (Autotask)  
| | • VOST (Virtual OST)  
| | • HCT |
| STOP TASK=taskname¹ | • DST  
| | • OPT  
| | • OST  
| | • NNT  
| | • AOST (Autotask)  
| | • VOST (Virtual OST)  
| | • HCT |
| TASKMON | • NetView Aggregate  
| | • MAINTASK  
| | • PPT  
| | • Automatic Tasks  
| | • DST  
| | • OPT  
| | • OST  
| | • NNT  
| | • AOST (Autotask)  
| | • VOST (Virtual OST)  
| | • HCT |
| TASKUTIL | • NetView Aggregate  
| | • MAINTASK  
| | • PPT  
| | • Automatic Tasks  
| | • 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:

- CPU
- STG
- IO
- MQIN
- MQOUT

TASKMON *taskname* is issued for the STATUS and MSGCT objects.

**Note:** For more information about Automatic Tasks, see the tasks listed in the Automatic Tasks section of the *Tivoli NetView for z/OS User's Guide*. With the exception of DSIWEB and FLBTOPO, all of the tasks listed are valid for NRM.

The Query/Set Threshold command, which is presented as a dialog, enables you to examine/change the effective NRM thresholds. This dialog is available for all objects except the STATUS object. The thresholds can also be set with the DEFAULTS/OVERRIDE commands. Message command, which is also presented as a dialog, enables you to send a message to the selected operator task.

### 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, use DUIFSMT.

**Example:**

```plaintext
DUIFSMTE CLASS=GMFHS_Managed_Real_NRM_Objects_Class, C
  XCPT=(UNSAT,DS152,DS153,DS154,DS155,DS156,DS157,DS158,DSC159,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

If you want the NRM monitor values to update dynamically, code the following RODM loader statement:

```plaintext
OP DUIFVINS INVOKE 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 501 for more information.

---

1. These commands are protected by the default security for NetView (CNMSCAT2/CNMSAF2).
**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.V5R1M0.CNMSAMP(DUIFNRM1),DISP=SHR <-NRM RCM SAMPLE
// DD DSN=NETVIEW.V5R1M0.CNMSAMP(DUIFNRM2),DISP=SHR <-NRM RCM SAMPLE
```

Sample DUIFNRM1 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 NetView, 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 NetViews 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".

Sample DUIFNRM2 is an example of selecting objects from a single NetView. It contains the following view:

- **View - NRM_DSI_TASKS** - A01NV tasks starting with DSI

**Customizing Sample Loader Files**

After you have loaded the sample RODM loader files, you can modify the collections using the NMC console. As an administrator, click on Tasks->RODM Collection Manager, to open the RODM Collection Manager GUI. From there, you can browse and modify the collections. In order 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 cold start RODM, load the data sets containing your modified collections, and they will be available to NMC 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 invoked. 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" on page 171, "Objects in RODM Representing Non-SNA Domains" on page 171, and "Objects in RODM Representing Non-SNA Resources" on page 173. 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 178 is used.

Note: Command processor DUIFECMV must run under the autotask DUIFEAUT. Refer to the NetView online help or the Tivoli NetView for z/OS Command Reference 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 171 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 172.

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 173. 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:
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 172.

**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 170 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMGPU5</td>
<td>PU</td>
</tr>
<tr>
<td>SP010</td>
<td>SP</td>
</tr>
<tr>
<td>RING010</td>
<td>RING</td>
</tr>
<tr>
<td>PRINTER1</td>
<td>PRTR</td>
</tr>
</tbody>
</table>

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 may 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMGPU5</td>
<td>PU</td>
</tr>
<tr>
<td>PRINTER2</td>
<td>DEV</td>
</tr>
</tbody>
</table>

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:
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 171, it tries to identify the non-SNA resource. GMFHS does this by calling the load module named in the AlertProc field of the Non_SNA_Domain_Class object. Refer to the 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 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.
Alerts and Resolutions Reference

- 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 calls 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 may not 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 should 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. The status mapping table should be used. |
    | Non-0 | The DisplayStatus value that should 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. You should 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 *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 *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 17)* GMFHS supplies a list containing one name and type pair:
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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>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.</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>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.</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>The AlertProc module detected a calling parameter error. GMFHS logs the error in a system error synopsis and issues console message DUI3913E.</td>
</tr>
</tbody>
</table>

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 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 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.
Alerts and Resolutions Reference

- 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
```

```
+<.-VVVV)+<.-VVVV
+<.-VVVV
+<.-VVVV
```

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)
Alerts and Resolutions Reference

- 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.

**Note:** NETCENTER service points use alert type X’12'(unknown) for session protocol alerts and to simulate resolutions. To maintain compatibility with NETCENTER service points, the DUIFEIBM translation table does not provide a code point translation for alert type X’12’. You can add a code point translation for alert type X’12’ to the DUIFEUSR translation table. If you are using NETCENTER and you add a code point translation for alert type X’12’, it should translate these alerts to SATISFACTORY; all NETCENTER resolutions will be translated to the status you specify in this code point translation.

**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, you should 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 23 contains the default alert translations that exist in DUIFEIBM.

Table 23. Default Alert Translations in DUIFEIBM

<table>
<thead>
<tr>
<th>Alert Type</th>
<th>Display Status Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>UNSATISFACTORY</td>
</tr>
<tr>
<td>02</td>
<td>UNSATISFACTORY</td>
</tr>
<tr>
<td>03</td>
<td>UNSATISFACTORY</td>
</tr>
<tr>
<td>04</td>
<td>INTERMEDIATE</td>
</tr>
<tr>
<td>0A</td>
<td>INTERMEDIATE</td>
</tr>
<tr>
<td>0F</td>
<td>SATISFACTORY</td>
</tr>
<tr>
<td>10</td>
<td>UNSATISFACTORY</td>
</tr>
<tr>
<td>11</td>
<td>INTERMEDIATE</td>
</tr>
<tr>
<td>12</td>
<td>RESERVED</td>
</tr>
<tr>
<td>14</td>
<td>INTERMEDIATE</td>
</tr>
<tr>
<td>15</td>
<td>INTERMEDIATE</td>
</tr>
</tbody>
</table>
Part 3. Using RODM for Network Automation

Chapter 7. Writing Automation Code ........ 183
Advantages of Using the NetView-Supplied Data Models for Automation .......... 183
Notifying Your Application about Changes in GMFHS Fields .......... 183
Accessing and Changing GMFHS-Defined Fields 184
Using GMFHS Methods .......... 185
  DUIFCCAN - Clear All Notes .......... 185
  DUIFCATC - Aggregation Threshold Change 185
  DUIFCLRT - Link Resource Type .......... 185
  DUIFCUAP - Update Aggregation Path .......... 185
  DUIFCUUS - Update User Status .......... 186
  DUIFECDS - Change Display Status .......... 186
  DUIFFAWS - Aggregation Warm Start .......... 186
  DUIFFIRS - Set Initial Resource Status .......... 186
  DUIFFRAS - Recalculate Aggregate Status .......... 186
  DUIFFSUS - Set Unknown Status .......... 186
  DUIFRFDS - Refresh DisplayStatus Change Method DUIFCRDC .......... 187
  DUIFVCFT - Change Exception State .......... 187
  DUIFVINS - Install View Notification Granularity Method .......... 187
GMFHS Methods That Cannot Be Used .......... 187
GMFHS Automation Example .......... 187
Sample Automation Application and Method 188

Chapter 8. Using the RODM Automation Platform .......... 191
RODM Automation Platform Services .......... 191
Sample Automation Code .......... 192
Chapter 7. Writing Automation Code

This chapter explains how you can write automation applications and methods to interface with the NetView-supplied data models, including the GMFHS data model and the SNA topology manager data model. It also discusses the rules and considerations involved in extending a NetView-supplied data model to meet your automation needs. When you design automation applications involving RODM, you can either design your own data model or use the NetView-supplied data models.

Advantages of Using the NetView-Supplied Data Models for Automation

Though you can create your own data model instead of using the NetView-supplied data models, you should consider the following advantages of designing your automation routines around the NetView-supplied data models:

- The NetView-supplied data models are available. The NetView-supplied data models are designed to model networks, and if you use it, you avoid the extra step of having to design and implement your own data model, which can be time consuming and costly.
- The NetView-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 NetView-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 NetView-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 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 321 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.
Notifying Your Application about Changes in Fields

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 [NetView-Supplied Methods on page 483] for a description of these methods. You can also write your own notification methods if the NetView-supplied methods 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 183] 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 in order 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.
- Do not change the value of the fields of any object that is a descendant of one of 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 which 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 will not be able to 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 NetView-supplied methods 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 IBM-created fields 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 490 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 invoke this method. However, when you design your application, you should consider that if more than one threshold value is being changed for an object, all but the last change should use the non method triggering (subfield) form of the change request. 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 INVOKED_WITH RODM load function primitive statement when you create your network definition statements for GMFHS. Any application that links or unlinks objects of the Display_Resource_Type_Class with objects of the GMFHS_Managed_Real_Objects_Class, or its child classes, or with objects of the GMFHS_Aggregate_Objects_Class should also use this method. 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 491 for a description of the parameters for this method.

DUIFCUAP - Update Aggregation Path

This object-independent method is intended to be invoked using the INVOKED WITH primitive of the RODM load function but should be used by 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...
Using GMFHS Methods

the change. Note that running the DUIFFAWS method (aggregation warm start) after such a change will accomplish 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. 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. 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 message DUI4020A 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. An application that is initializing and maintaining its own real resource DisplayStatus (in place of GMFHS) might find this method useful.

**DUIFFFRAS - Recalculate Aggregate Status**

This object-independent method can be invoked 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 DUIFFFRAS 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 DUIFFFRAS 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. An application that is initializing and maintaining its own real resource DisplayStatus (in place of GMFHS) might find this method useful.

**DUIFRFDS - Refresh DisplayStatus Change Method DUIFCRDC**

This object-independent method can be invoked 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 invoked 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 106.

**DUIFVCFT - Change Exception State**

This object-independent method can be invoked 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 invoke 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 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 501 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 which cannot be used by your programs. See “GMFHS Methods” on page 490 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 explain 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, 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 could 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 would occur 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 9 on page 20. When the application determines that the status of one of these resources has changed to unsatisfactory, it invokes 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 invoking 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 invoking 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 invoking application assumes that the alert represents a new problem. In this case, the application might open a problem report for the new problem via 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; it should not be viewed 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 the DEC network.
3. Issues EKGWAIT and waits for the DisplayStatus fields of the DEC 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 could 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.

GMFHS Automation Example
GMFHS Automation Example
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 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

There are six services that make up the RODM automation platform:

- DSIQTSK task
- ORCONV command
- EKGSPPPI 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 EKGSPPPI 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 EKGSPPPI NetView-supplied method sends commands from RODM to the DSIQTSK task in the NetView product using the program-to-program interface. See "EKGSPPPI - Send a command to NetView" on page 487 for a description of the EKGSPPPI method.

The CNMQAPI service routine is an enhanced API which 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 Tivoli NetView for z/OS Customization: Using 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 Tivoli NetView for z/OS Customization: Using 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.
RODM Automation Platform Services

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.V5R1M0.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

RODM Classes

Class Names

Class Name Characteristics with CHARACTER_VALIDATION(YES)
Class Name Characteristics with CHARACTER_VALIDATION(NO)
System-Defined Classes

UniversalClass
EKG_SystemDataParent Class
EKG_System Class
EKG_User Class
EKG_NotificationQueue Class
EKG_Method Class

RODM Objects

Object Names

Object Name Characteristics with CHARACTER_VALIDATION(YES)
Object Name Characteristics with CHARACTER_VALIDATION(NO)
Object Identifiers

RODM Fields

Field Names

Field Name Characteristics with CHARACTER_VALIDATION(YES)
Field Name Characteristics with CHARACTER_VALIDATION(NO)
Field Identifiers
System-Defined Fields

RODM Subfields

Data Types for Subfields
Multivalued Fields and Links between Objects
Link and Unlink Action Functions
Subfields Associated with Fields

Indexed Fields

Object and Class Locking in RODM

Using the Application Program Interfaces

User Application Program Interface (API)
Method Application Program Interface (API)

RODM Abstract Data Types

Null Values of Data Type
Data Type Identifiers
Types of Data in Fields

Abstract Data Type Reference

Anonymous(N) (Reserved)
AnonymousVar
ApplicationID (Reserved)
BERVar
CharVar
CharVarAddr (Reserved)
ClassID (Reserved)
ClassIDList (Reserved)
ClassLinkList (Reserved)
ECBAddress (Reserved)
FieldID
Floating

GraphicVar
Integer
IndexList
MethodName (Reserved)
method_parameter_list (Reserved)
MethodSpec
ObjectID (Reserved)
ObjectIDList (Reserved)
ObjectLink
ObjectLinkList
ObjectName (Reserved)
RecipientSpec (Reserved)
SelfDefining
ShortName (Reserved)
Smallint
SubscribeID (Reserved)
SubscribeSpec (Reserved)
SubscribeSpecList (Reserved)
TimeStamp
TransID (Reserved)

Chapter 10. Using the RODM Load Function

Considerations When Designing a Data Model
Introduction to the RODM Load Function
Load Function Statements
Load Function Operations
Loading the RODM Data Cache
Using Load Function Statements
High-Level Load Function Statements
Load Function Primitive Statements
When to Use High-Level or Primitive Load Function Statements
Process for Loading the RODM Data Cache
Identifying the Methods to Install
Creating the Class Structure and Object Definitions
Data Definition Statement Labels
Concatenation of Data Sets
Definition Examples
Deciding on the Type of Load
Initialization Load
Structure Load Only
Object Load Only
Running the RODM Load Function
The Load Function as an Initialization Method
Invoking the Load Function As a Batch Job
Calling the Load Function from a Module
Considerations When Running the RODM Load Function
Checking the Output Listings
RODM Load Function Output Listing
RODM Load Function Output Format
Load Function Reference
Understanding the Verify Operation
Using CLASSID and OBJECTID Data Types
### Chapter 11. Writing Applications that Use RODM

- Tasks Best Performed with User Applications ........................................... 303
- Using the User Application Program Interface ......................................... 304
- Register Conventions .................................................................................. 304
- Usage Notes ................................................................................................. 304
- Compiling and Link-Editing ........................................................................ 305
  - Compiling C Modules that Call EKGUAPI .............................................. 305
  - Compiling PL/I Modules that Call EKGUAPI ........................................ 305
  - Linking Modules that Call EKGUAPI ....................................................... 305
    - Directly .................................................................................................... 306
    - Linking Modules that Load and then Call EKGUAPI ............................. 306
- Using Control Blocks .................................................................................... 306
  - Access Block ............................................................................................ 307
  - Transaction Information Block ............................................................... 309
  - Function Block ......................................................................................... 311
  - Entity Access Information Block ............................................................ 311
  - Field Access Information Block .............................................................. 314
- Coding RODM High-Level Load Function Statements .................................. 275
  - Syntax Rules for High-Level Load Function Statements ....................... 275
  - Syntax for High-Level Load Function Statements .................................... 277
- Coding RODM Load Function Primitive Statements ................................... 284
  - Global Character ..................................................................................... 284
  - Syntax Rules for Load Function Primitives .............................................. 284
  - Syntax and Processing Logic for Load Function Primitives ..................... 284
  - Common Syntactic Elements .................................................................. 294
  - Syntax for Common Syntactic Elements .................................................. 294

### Chapter 12. Topology Object Correlation

- Enabling the Correlation Function ................................................................ 333
  - Enabling MultiSystem Manager Object Correlation ................................ 333
  - Enabling SNA Topology Manager Object Correlation ............................ 333
  - Enabling GMFHS Object Correlation ....................................................... 334
- Correlation Concepts .................................................................................... 334
  - Correlation Methods ................................................................................ 334
    - Method FLCMCONI ............................................................................. 334
    - Method FLCMCON ............................................................................. 334
    - Method FLCMCO .............................................................................. 335
- Objects Enabled for Correlation ................................................................. 335
  - Types of Correlation ............................................................................... 335
    - Network Address Correlation ............................................................... 335
    - Free-Form Correlation ......................................................................... 335
  - Correlated Aggregate Object Classes and Names .................................... 337
  - Correlated Object Relationships ............................................................ 337
  - Correlated Aggregate Object Display Labels ......................................... 337
  - Correlated Aggregate Object Field Values ............................................. 338
- Using Correlation for Objects You Create .................................................. 339
  - Extending Correlation of Objects Created by MultiSystem Manager and SNA Topology Manager 339
  - How to Determine Object Names .............................................................. 340
  - Correlating MultiSystem Manager Objects .......................................... 340
  - Correlating SNA Topology Manager Objects ......................................... 340
- Customizing the Correlation Function ....................................................... 341
  - Changing the Display Name Priority ....................................................... 341
  - Disabling Correlation for Specific Resources ........................................ 342

### Chapter 13. Writing RODM Methods

- Tasks Best Performed with Methods ............................................................ 343
  - Types of Methods ..................................................................................... 344
    - Object-Independent Methods ............................................................... 344
    - Initialization Method ............................................................................ 345
    - Object-Specific Methods ..................................................................... 346
      - Change Methods ................................................................................ 346
      - Query Methods ................................................................................. 349
      - Notify Methods ................................................................................. 350
      - Named Methods ................................................................................. 353
      - Inheritance in Object-Specific Methods ............................................ 354
Null Method .......................... 356
Deciding Which Method Type to Use ....... 356
When to Use an Object-Independent Method ... 356
When to Use an Object-Specific Method ....... 356
Query Method .......................... 375
Change Method .......................... 375
Notify Method .......................... 375
Named Method .......................... 375
Using the Method API .................... 375
Method Anchor Service ................... 362
Coding Your RODM Method ............... 362
Installation Written Methods ............. 362
NetView-Supplied Methods ............... 362
Programming Language Specific Preprocessor
Statements ................................ 363
Compiling IBM C Methods .................. 363
Compiling IBM PL/I Methods .............. 364
Directly .................................. 364
Restrictions on Methods ................... 364
PL/I Language Restrictions ............... 364
C Language Restrictions ................. 366
Restrictions in General ................... 366
RODM Method Services ................... 367
Services Available to both Object-Specific and
Object-Independent Methods ............. 367
Other Services Available to Object-Independent
Methods .................................. 368
Other Services Available to Object-Specific
Methods .................................. 368
Services Available to the Initialization Method 368
RODM Method Library ..................... 369

Chapter 14. Application Programming
Reference ................................ 371
Summarizing RODM Functions .......... 371
Access Functions ........................ 371
Control Functions ........................ 371
Administrative Functions ................. 371
Action Functions ........................ 372
Query Functions ........................ 373
RODM User API Services ................. 374
RODM Method API Services ............... 374
Function Reference ....................... 375
Function Reference Format ............... 375
Purpose .................................. 375
Examples ................................. 375
Summary .................................. 376
Usage Notes .............................. 376
EKG_AddNotifySubscription — Add
Notification Subscription ............... 377
EKG_AddObjDelSubs — Add Object Deletion
Subscription ............................. 378
EKG_ChangeField — Change a Field ....... 380
EKG_ChangeMultipleFields — Change Multiple
Fields .................................... 381
EKG_ChangeSubfield — Change a Subfield .... 382
EKG_Checkpoint — Checkpoint RODM to
DASD .................................... 384
EKG_Connect — Connect to RODM ........ 387
EKG_CreateClass — Create a Class ......... 388
EKG_CreateField — Create a Field ......... 389
EKG_CreateObject — Create an Object ....... 391
EKG_CreateSubfield — Create a Subfield .... 392
EKG_DeleteClass — Delete a Class ........ 393
EKG_DeleteField — Delete a Field ......... 394
EKG_DeleteNotifySubscription — Delete
Notification Subscription ............... 395
EKG_DeleteObject — Delete an Object ....... 397
EKG_DeleteSubfield — Delete a Subfield .... 398
EKG_DelObjDelSubs — Delete Object Deletion
Subscription ............................. 399
EKG_Disconnect — Disconnect from RODM 401
EKG_ExecuteFunctionList — Execute a List of
Functions ............................... 402
EKG_LinkNoTrigger, EKG_LinkTrigger — Link
Two Objects .............................. 404
EKG_Locate—Locate Objects Using Public
Indexed Field ............................. 406
EKG_LockObjectList — Lock List of Objects 407
EKG_MessageTriggeredAction — Trigger an
Action by a Message ..................... 409
EKG_OutputToLog — Output to Log ......... 410
EKG_QueryEntityStructure — Query Structure
of an Entity .............................. 411
EKG_QueryField — Query a Field .......... 413
EKG_QueryFieldID — Query Field Identifier 414
EKG_QueryFieldName — Query a Field Name.... 416
EKG_QueryFieldStructure — Query Structure of
a Field ................................. 417
EKG_QueryFunctionBlockContents — Query
Function Block Contents .................. 419
EKG_QueryMultipleSubfields — Query Multiple
Value Subfields .......................... 421
EKG_QueryNotifyQueue — Query Notification
Queue ..................................... 423
EKG_QueryObjectName — Query Object Name .... 425
EKG_QueryResponseBlockOverflow — Query
for Response Block Overflow ............ 426
EKG_QuerySubfield — Query a Subfield ....... 428
EKG_ResponseBlock — Output to Response
Block ................................. 430
EKG_RevertToInherited — Revert to Inherited
Value .................................. 431
EKG_SendNotification — Send a Notification 433
EKG_SetReturnCode — Set Return and Reason
Codes ................................. 434
EKG_Stop — Stop RODM .................. 436
EKG_SwapField — Swap a Field .......... 437
EKG_SwapSubfield — Swap a Subfield ....... 439
EKG_TriggerNamedMethod — Trigger a Named
Method ............................... 440
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 43 on page 198 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 211), field names (see “Field Names” on page 212), 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, may be alphabetic, numeric, the break character (_), the commercial “at” sign (@), the number sign (#), or the period (.).
RODM Classes

- 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 43 shows the RODM system-defined classes and their hierarchy.

![Figure 43. RODM System-Defined Classes](image)

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
RODM 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 24 describes the fields of UniversalClass, the access for each field, the data type of the field, and the subfields defined on each field.

**Table 24. UniversalClass Fields**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Access</th>
<th>Data Type</th>
<th>Query</th>
<th>Change</th>
<th>Notify</th>
<th>Time stamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyName</td>
<td>Read</td>
<td>ObjectName or ShortName</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MyID</td>
<td>Read</td>
<td>ObjectId or ClassID</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MyPrimaryParentName</td>
<td>Read</td>
<td>ShortName</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MyPrimaryParentID</td>
<td>Read</td>
<td>ClassID</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WhatIAm</td>
<td>Read</td>
<td>Enumerated Integer</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MyClassChildren</td>
<td>Read</td>
<td>ClassIDList</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>MyObjectChildren</td>
<td>Read</td>
<td>ObjectIdList</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
RODM Classes

The UniversalClass fields are:

**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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Object</td>
</tr>
<tr>
<td>2</td>
<td>Class with no children</td>
</tr>
<tr>
<td>3</td>
<td>Class with object children</td>
</tr>
<tr>
<td>4</td>
<td>Class with class children</td>
</tr>
<tr>
<td>5</td>
<td>Class with class and object children</td>
</tr>
</tbody>
</table>

**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 *Tivoli NetView for z/OS Administration Reference* for information about the RODM customization file.

Table 25 describes the fields of the EKG_System class, the access for each field, the data type, and the subfields for each field.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Access</th>
<th>Data Type</th>
<th>Query</th>
<th>Change</th>
<th>Notify</th>
<th>Time stamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>EKG_Name</td>
<td>Read</td>
<td>CharVar</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG_APIVersion</td>
<td>Read</td>
<td>Integer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG_ReleaseID</td>
<td>Read</td>
<td>CharVar</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG_ExternalLogState</td>
<td>Write</td>
<td>Enumerated Integer</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG_LastCheckpointID</td>
<td>Read</td>
<td>TransID</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG_LastCheckpointResult</td>
<td>Read</td>
<td>SelfDefining</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG_LastAsyncError</td>
<td>Read</td>
<td>AnonymousVar</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG.AsyncTasks</td>
<td>Read</td>
<td>Integer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG_ConcurrentUsers</td>
<td>Read</td>
<td>Integer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG_PLL_ISA</td>
<td>Read</td>
<td>Integer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG_SSBChain</td>
<td>Read</td>
<td>Integer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG_TransSegment</td>
<td>Read</td>
<td>Integer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG_WindowSize</td>
<td>Read</td>
<td>Integer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The field definitions are:

**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 RODMN510. The value RODMN510 indicates Tivoli NetView for z/OS V5R1.

**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. Valid values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Log</td>
</tr>
<tr>
<td>2</td>
<td>No log</td>
</tr>
</tbody>
</table>
RODM Classes

This logging applies only to the external file dataset. 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 24 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>
<thead>
<tr>
<th>Table 26. EKG_LastCheckpointResult System Field</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Offset</strong></td>
</tr>
<tr>
<td>000</td>
</tr>
<tr>
<td>002</td>
</tr>
<tr>
<td>004</td>
</tr>
<tr>
<td>008</td>
</tr>
<tr>
<td>010</td>
</tr>
<tr>
<td>014</td>
</tr>
<tr>
<td>016</td>
</tr>
</tbody>
</table>

**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 may or may 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 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 27 describes the fields of EKG_User class, the access for each field, the data type, and the subfields defined for each field.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Access</th>
<th>Data Type</th>
<th>Query</th>
<th>Change</th>
<th>Notify</th>
<th>Time stamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>EKG_Status</td>
<td>Read</td>
<td>Enumerated</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>EKG_StopMode</td>
<td>Write</td>
<td>Enumerated</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>EKG_LastAsyncError</td>
<td>Read</td>
<td>AnonymousVar</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>EKG_Uses_Q</td>
<td>Read</td>
<td>ObjectLinkList</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG_RBOverflowAction</td>
<td>Write</td>
<td>Enumerated</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG_LogLevel</td>
<td>Write</td>
<td>Integer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG_MLogLevel</td>
<td>Write</td>
<td>Integer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG_MTraceType</td>
<td>Write</td>
<td>4-Byte Integer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The field definitions are:

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Connected</td>
</tr>
<tr>
<td>2</td>
<td>Disconnected</td>
</tr>
<tr>
<td>3</td>
<td>Unknown</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Purge notification queues and subscriptions</td>
</tr>
<tr>
<td>2</td>
<td>Purge notification queue elements only</td>
</tr>
<tr>
<td>3</td>
<td>Do not purge notification queues or subscriptions</td>
</tr>
</tbody>
</table>

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_RBOOverflowAction**
Response block overflow action control. Valid values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Save</td>
</tr>
<tr>
<td>2</td>
<td>Discard</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning if bit is set</th>
</tr>
</thead>
<tbody>
<tr>
<td>1...</td>
<td>Trace object deletion methods</td>
</tr>
<tr>
<td>.1..</td>
<td>Trace object independent methods</td>
</tr>
<tr>
<td>..1.</td>
<td>Trace named methods</td>
</tr>
<tr>
<td>...1</td>
<td>Trace notify methods</td>
</tr>
<tr>
<td>....</td>
<td>Trace change methods</td>
</tr>
<tr>
<td>....</td>
<td>Trace query methods</td>
</tr>
<tr>
<td>....</td>
<td>Trace method exit and storage</td>
</tr>
<tr>
<td>....</td>
<td>Trace method entry and storage</td>
</tr>
</tbody>
</table>

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 321 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 28 describes the fields of the EKG_NotificationQueue class, the access for
each field, the data type, and the subfields defined for each field.

Table 28. EKG_NotificationQueue Fields

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Access</th>
<th>Data Type</th>
<th>Query</th>
<th>Change</th>
<th>Notify</th>
<th>Time stamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>EKG_Status</td>
<td>Write</td>
<td>Enumerated</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG_ECBAddress</td>
<td>Write</td>
<td>ECBAddress</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG_ECBPostedStatus</td>
<td>Read</td>
<td>Enumerated</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG_UsedBy</td>
<td>Read</td>
<td>ObjectLink</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG_SubscribedFromClass</td>
<td>Read</td>
<td>ClassLinkList</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG_SubscribedFromObject</td>
<td>Read</td>
<td>ObjectLinkList</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG_Maximum_Q_Entries</td>
<td>Write</td>
<td>Integer</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG_MessagesOnQueue</td>
<td>Read</td>
<td>Integer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG_SubscribedForDelete</td>
<td>Read</td>
<td>ObjectIDList</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The field definitions are:

**EKG_Status**

The status of the notification queue. Valid values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Inactive</td>
</tr>
<tr>
<td>1</td>
<td>Active</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>False</td>
</tr>
<tr>
<td>1</td>
<td>True</td>
</tr>
</tbody>
</table>
RODM Classes

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 225.

EKG_SubscribedFromClassObject
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 225.

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 225.

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

Tivoli NetView for z/OS Resource Object Data Manager and GMFHs Programmer’s Guide
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 360.

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 NetView-supplied null method, 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 29 describes the fields of EKG_Method class, the access for each field, the data type, and the applicable operations.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Access</th>
<th>Data Type</th>
<th>Query</th>
<th>Change</th>
<th>Notify</th>
<th>Time stamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>EKG_InstallerID</td>
<td>Read</td>
<td>CharVar</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG_UsageCount</td>
<td>Read</td>
<td>Integer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG_Refresh</td>
<td>Read</td>
<td>MethodSpec</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG_MTraceFlag</td>
<td>Write</td>
<td>Integer</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

The field definitions are:

**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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Defers the trace decision to EKG_MTraceType.</td>
</tr>
<tr>
<td>1</td>
<td>Ensures tracing.</td>
</tr>
</tbody>
</table>

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 may 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 EKGdxxxxxx, where xxxxxx ranges from 000000 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 197), field names (see “Field Names” on page 212), 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.
- EKGxxxxxxx (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 512K (524288) 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.
RODM Objects

- 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.
- EKGxxxxxxx (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 512K (524288) 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 213 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 211), class names (see "Class Names" on page 197), 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 38 on page 315 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.
RODM Fields

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>An object</td>
</tr>
<tr>
<td>2</td>
<td>A class with no children</td>
</tr>
<tr>
<td>3</td>
<td>A class with object children</td>
</tr>
<tr>
<td>4</td>
<td>A class with class children</td>
</tr>
<tr>
<td>5</td>
<td>A class with both class children and object children</td>
</tr>
</tbody>
</table>

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 225, 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.

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 225 and is identical to the data type of the field. The value subfield is
RODM Subfields

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.

<table>
<thead>
<tr>
<th>Subfield</th>
<th>Valid Abstract Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>• AnonymousVar</td>
</tr>
<tr>
<td></td>
<td>• BERVar</td>
</tr>
<tr>
<td></td>
<td>• CharVar</td>
</tr>
<tr>
<td></td>
<td>• FieldID</td>
</tr>
</tbody>
</table>

Chapter 9. Understanding RODM Concepts 217
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 discussed in this section are RODM-defined relational links. These links are defined between two objects in the RODM data cache and should 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 44 shows single-value links using fields of data type ObjectLink and a multivalue link using a field of data type ObjectLinkList.
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 404 and "EKG_UnlinkNoTrigger, EKG_UnlinkTrigger — Unlink Two Objects" on page 443 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.
Links between Objects

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 should 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 may 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 482 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 45 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 discussed in "Chapter 11. Writing Applications that Use RODM" on page 303.

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 45 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 discussed in "Chapter 13. Writing RODM Methods" on page 343.

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 225 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 225.

**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:

```pli
%include EKGLIB(ekg1iadt); /* Abstract data declaration */
DCL ThisMethodName MethodName; /* 8-byte char */
```

To declare the same variable in C, use the statements:

```c
#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:

```pli
% Anonymous = 'CHAR';
```

C/370 Declaration:

```c
typedef char Anonymous;
```

AnonymousVar

Data Type Identifier: 30
**RODM Abstract Data Types**

**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/370 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/370 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 46 on page 227 shows the 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 47. Identifier Byte in Short Form](image1)

  - 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 48. Identifier Byte in Long Form](image2)

- **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 49 shows the short form of a length byte with the value of 86 bytes.

<table>
<thead>
<tr>
<th>Bits</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

*Figure 49. Length Byte in Short Form*

- If bit 8 equals 1, the length byte is long. For this form, bits 7 to 1 represents the number of subsequent bytes that comprise the length bytes and is 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 50 shows the long form of a length byte with the value of 357 bytes. Two length bytes are needed to represent 357.

<table>
<thead>
<tr>
<th>Bits</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

*Figure 50. Length Byte in Long Form*

**Null Value:** Length field is zero.

**PL/I Declaration:**

```pli
% BERVar = 'CHAR(32767) VARYING';
```

**C/370 Declaration:**

```c
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 233.

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 231.

Null Value: Length field is zero.

PL/I Declaration:

% CharVar = 'CHAR(32767) VARYING';

C/370 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/370 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)';
RODM Abstract Data Types

C/370 Declaration:

```c
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:**

```pli
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/370 Declaration:

```c
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:**

```pli
DCL 1 ClassLinkList EKG_BOUNDARY,
     3 Len   Integer,
     3 List(1),
     5 ClassIdentifier ClassID,
     5 FieldIdentifier FieldID;
```

C/370 Declaration:

```c
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/370 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/370 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/370 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
RODM Abstract Data Types

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 GraphicVar EKG_BOUNDARY,
  3 Len Smallint,
  3 Text CHAR(1);
```

C/370 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/370 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 51 on page 233 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
Null Value: Length field is zero.

PL/I Declaration:

```pli
% IndexList = 'CHAR(32767) VARYING';
```

C/370 Declaration:

```c
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:

```pli
% MethodName = 'CHAR(8)';
```

C/370 Declaration:

```c
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:

```pli
% method_parameter_list = 'SelfDefining';
```
**C/370 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/370 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/370 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/370 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/370 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/370 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 211 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/370 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/370 Declaration:

typedef _Packed struct {
    ApplicationID User_appl_ID;
    SubscribeID Notification_queue;
    Anonymous User_Word[8];
} RecipientSpec;

SelfDefining

Data Type Identifier: 19
**Description:** A SelfDefining data string of no more than 32767 bytes. The string is a concatenation of tagged data items, where each tagged data item comprises 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 52 shows the format of SelfDefining data.

**Self Defining**

![Diagram of SelfDefining Data Type Syntax]

**Figure 52. 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 53 shows an example SelfDefining string that contains a Smallint with the decimal value 1992, a CharVar with the value RODM, and an ApplicationID with the value NETV23.

![Example SelfDefining Field]

**Figure 53. Example SelfDefining Field**

**Null Value:** Length field is zero.

**PL/I Declaration:**

```
% SelfDefining = 'CHAR(32767) VARYING';
```
RODM Abstract Data Types

C/370 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 212.

Null Value: Length field is zero; in PL/I, set with string = '.'.

PL/I Declaration:

% ShortName = 'CHAR(64) VARYING';

C/370 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/370 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:
C/370 Declaration:

```c
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:**

```pli
DCL
  1 SubscriptSpecList EKG_BOUNDARY,
  3 Len   Integer,
  3 Text  CHAR(1);
```

C/370 Declaration:

```c
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
RODM Abstract Data Types

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. This recommendation is from the *IBM S/370 Principles of Operation*.

**Null Value:** All bits are zero.

**PL/I Declaration:**

% TimeStamp = 'FLOAT BINARY(53)';

**C/370 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/370 Declaration:**

typedef _Packed struct {
    char Content[8];
} TransID;
Chapter 10. Using the RODM Load Function

This chapter explains 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 an IBM-supplied data model. 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 executes 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 54.

Figure 54. 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 may 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 260 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 invoke the RODM load function either as:

- An initialization method executed 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 246.

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 284.
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:

- **HAS_VALUE**
  Defines a value for a field within a class.

---

Chapter 10. Using the RODM Load Function
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.

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:

```plaintext
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:

```plaintext
OP SNA_Domain_Class.CNM01.SNANet HAS_VALUE (CHARVAR) 'NETC';
```

### Process for Loading the RODM Data Cache

This section explains the process used to load the RODM data cache using the RODM load function. The process steps are first listed in order and explained in the same order.

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 263 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 243 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 248 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 248.
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.

<table>
<thead>
<tr>
<th>Member</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>EKGIN1</td>
<td>An example of load function statements designed to:</td>
</tr>
<tr>
<td></td>
<td>• Create a class under the UniversalClass</td>
</tr>
<tr>
<td></td>
<td>• Create fields for all data types supported</td>
</tr>
<tr>
<td></td>
<td>• Set initial values for the fields</td>
</tr>
<tr>
<td>EKGIN3</td>
<td>An example of load function statements designed to:</td>
</tr>
<tr>
<td></td>
<td>• Create 3 objects</td>
</tr>
<tr>
<td></td>
<td>• Set initial values</td>
</tr>
</tbody>
</table>

Deciding on the Type of Load
The steps in the loading process differ, depending on how you intend to invoke the RODM load function and on what type of load you are performing. You can invoke the RODM load function as an initialization method during a cold start of RODM or during a warm start of RODM. You can invoke the RODM load function by means of a JCL job. You can invoke 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 invokes 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 invoked, 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 264 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 66 on page 266.

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 263.

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 55 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 55. 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 56 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 56. 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
- Calling the load function from a module
- Considerations when running the load function

You can run the RODM load function by invoking 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 NetView 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 NetView-supplied initialization 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 execute 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 invoking 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 invoke 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 invoking 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.

Invoke the procedure by entering:

```
$ 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 254 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.V5R1M0 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 EKGLLOADP 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 57 shows the lines in EKGLLOAD updated with example values.

```
//STEP01 EXEC EKGLOADP,
//   RODMNAME=EKGXRODM,
//   EKGIN3=NETVIEW.V5R1M0.CNMSAMP(EKGIN3)
```

**Figure 57. 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 EKGLLOADP procedure if you do not use NETVIEW.V5R1M0
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.V5R1M0.CNMSAMP. You do not need to specify this data set name. Figure 58 shows the lines in EKGLLOAD updated with example values.

```
//STEP01 EXEC EKGLOADP,
   // RODMNAME=EKGXRODM,
   // EKGIN1=NETVIEW.V5R1M0.CNMSAMP(EKGIN1),
   // LOAD=STRUCTURE
```

Figure 58. 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 253 and then load the objects, following the steps in "Loading Object Definitions Only" on page 253.
- 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 253.

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 invoke 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 253 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 268. The RODM load function executes 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 executed. Use the same linking conventions as you would for EKGLJOB. See “z/OS Linkage Conventions” on page 268.

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 run-time libraries are installed or available prior to submitting or invoking 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 274.

Ensure that the name you use to invoke 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 273.

Authorization and Authorization Levels: The TSO ID and TSO password that you use to invoke the RODM load function and user application programs that invoke 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 invokes the load function must have an authorization level of at least 3 or 5, depending on the load function statements used. Table 30 shows the load function statement, the statement type, the minimum authorization level, and a reference to additional information about the statement.

Table 30. Load Function Statements and Minimum Authorization Levels

<table>
<thead>
<tr>
<th>Statement</th>
<th>Statement Type</th>
<th>Minimum Authorization Level</th>
<th>See Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>High-level</td>
<td>3</td>
<td>280</td>
</tr>
<tr>
<td>DELETE</td>
<td>High-level</td>
<td>3</td>
<td>281</td>
</tr>
<tr>
<td>FORCE_HAS_NO_INSTANCE</td>
<td>Primitive</td>
<td>3</td>
<td>285</td>
</tr>
<tr>
<td>FORCE_NOT_A_CLASS</td>
<td>Primitive</td>
<td>5</td>
<td>285</td>
</tr>
<tr>
<td>HAS_FIELD</td>
<td>Primitive</td>
<td>5</td>
<td>286</td>
</tr>
</tbody>
</table>
Table 30. Load Function Statements and Minimum Authorization Levels (continued)

<table>
<thead>
<tr>
<th>Statement</th>
<th>Statement Type</th>
<th>Minimum Authorization Level</th>
<th>See Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAS_INSTANCE</td>
<td>Primitive</td>
<td>3</td>
<td>286</td>
</tr>
<tr>
<td>HAS_NO_FIELD</td>
<td>Primitive</td>
<td>5</td>
<td>287</td>
</tr>
<tr>
<td>HAS_NO_INSTANCE</td>
<td>Primitive</td>
<td>3</td>
<td>287</td>
</tr>
<tr>
<td>HAS_NO_SUBFIELD</td>
<td>Primitive</td>
<td>5</td>
<td>288</td>
</tr>
<tr>
<td>HAS_PARENT</td>
<td>Primitive</td>
<td>5</td>
<td>288</td>
</tr>
<tr>
<td>HAS_PRV_FIELD</td>
<td>Primitive</td>
<td>5</td>
<td>288</td>
</tr>
<tr>
<td>HAS_SUBFIELD</td>
<td>Primitive</td>
<td>5</td>
<td>288</td>
</tr>
<tr>
<td>HAS_VALUE</td>
<td>Primitive</td>
<td>3</td>
<td>288</td>
</tr>
<tr>
<td>INHERITS</td>
<td>Primitive</td>
<td>3</td>
<td>290</td>
</tr>
<tr>
<td>INVOKED_WITH</td>
<td>Primitive</td>
<td>3</td>
<td>290</td>
</tr>
<tr>
<td>IS_LINKED_TO</td>
<td>Primitive</td>
<td>3</td>
<td>290</td>
</tr>
<tr>
<td>IS_NOT_LINKED_TO</td>
<td>Primitive</td>
<td>3</td>
<td>292</td>
</tr>
<tr>
<td>MANAGED OBJECT CLASS</td>
<td>High-level</td>
<td>5</td>
<td>278</td>
</tr>
<tr>
<td>NOT_A_CLASS</td>
<td>Primitive</td>
<td>5</td>
<td>292</td>
</tr>
<tr>
<td>SET</td>
<td>High-level</td>
<td>3</td>
<td>282</td>
</tr>
<tr>
<td>SUBFIELD_HAS_VALUE</td>
<td>Primitive</td>
<td>3</td>
<td>292</td>
</tr>
<tr>
<td>SUBFIELD_INHERITS</td>
<td>Primitive</td>
<td>3</td>
<td>293</td>
</tr>
</tbody>
</table>

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 invoked, load function input, actions taken by the function, echoed syntax when an error occurs, and messages including an END OF JOB message. See Figure 61 on page 259 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 would hinder 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 272 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 272.

Compare the following figures for format differences:

- **Figure 59 on page 257**, a PARSE operation output example
- **Figure 60 on page 258**, a structure load output example
- **Figure 61 on page 259**, an object load output example
**SUPERCLASS**

**MANAGED OBJECT CLASS**;

**PARENT IS**

**UNIVERSALCLASS**;

**ATTRLIST**

FIELD ANONYMOUSVAR ANONYMOUSVAR INITIAL (X'4040'),
FIELD_BERVAR BERVAR INIT(X'81049FF8FF'),
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 59. Example of PARSE Operation Output to EKGPRINT*
Figure 60. Example of Structure Load Output to EKGPRINT
Figure 61. 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 invoke 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 explained first.

Each load function primitive statement description in “Syntax and Processing Logic for Load Function Primitives” on page 284 includes an explanation of the verify operation logic for that statement. The verify operation logic explains 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 could 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.

```plaintext
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;
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 284.

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 62 shows an example control table. The column scale is inserted for explanation purposes and is not part of the control table.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+</td>
<td>0</td>
<td>+</td>
<td>0</td>
</tr>
</tbody>
</table>

**Figure 62. 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 63 on page 263 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.V5R1M0.CNMSAMP containing the members EKGCTABL and EKGPTENU. The DD statement labeled EKGIN2 identifies the partitioned data set NETVIEW.V5R1M0.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.

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.V5R1M0.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 IBM-supplied control table the member name specified for the method name table is EKGINMTB. For more information about control table EKGCTABL, see "Control Table—EKGCTABL" on page 262.
Figure 64 shows a method name table (EKGINMTB) that declares two user-written methods and seven NetView-supplied methods. The column scale is inserted for explanation purposes and is not part of the method name table.

```
1...+...8...+...0...+...0...+...0...+...0...+...0...
1...+...0...+...0...+...0...+...0...+...1...+...0...
```

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EKGNOTF</td>
<td>NOTIFICATION</td>
</tr>
<tr>
<td>EKGNLST</td>
<td>Notify</td>
</tr>
<tr>
<td>EKGNQNL</td>
<td>Notify</td>
</tr>
<tr>
<td>EKGNTHD</td>
<td>Change method to trigger an OI method</td>
</tr>
<tr>
<td>EKGMIMV</td>
<td>Named method to increment a value</td>
</tr>
<tr>
<td>EKGSPPII</td>
<td>Object-Independent method</td>
</tr>
<tr>
<td>SOFTMTHD</td>
<td>Change Method - (user written)</td>
</tr>
<tr>
<td>OSSOMTHD</td>
<td>Change Method - (user written)</td>
</tr>
</tbody>
</table>

**Figure 64. 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 65. The column scale is inserted for explanation purposes and is not part of the method name table.

```
1...+...0...+...0...+...0...+...0...+...0...+...0...
```

**Figure 65. 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 63 on page 263 for a pictorial of this relationship.

**Parameter Mapping Table**

When you invoke 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 63 on page 263 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 272 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 66 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.
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 invoke the load function declare the data sets. Ensure that the data sets appropriate to the type of load you are invoking 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 invoked. The DD list structure is described in "z/OS Linkage Conventions" on page 268.

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

<table>
<thead>
<tr>
<th>Column Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>1...+0...+0...1...+0...+0...</td>
</tr>
</tbody>
</table>

```
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 ERRORSYNTAX
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
```

Figure 66. Sample Parameter Table EKGPTENU with Column Scale
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 262.

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 IBM-supplied method name table 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 explained 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 invoking 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 invoking 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 invoking 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 67 on page 269 shows the z/OS linkage requirements for invoking 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.
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 invoking the load function require that the character field 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):

001CD5C1D4C57EC5D2C7E7D9D6C4D46B0D3D6C1C47EE2E3D9E4C3E3E4D9C5
NAME=EKGXRODM, LOAD=STRUCTURE

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 266 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):

```
0020C5D2C7C9D5F1404E2E3D9E4C3E34040C5D2C7C9D5F34040D6C2D1C5C3E34040
EKGIN1  STRUC T  EKGIN3  OBJE 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 67 on page 269. 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 68 on page 277 and locate the statement:

```
DCL EKGLJOB OPTIONS(ASM INTER) ENTRY EXTERNAL;
```

Figure 68 on page 277 is an example of how to call the RODM load function from a PL/I program.
DECLARE PL1_OR_C FIXED; /* Flag indicates whether this */
/* module is IBM PL/1 or C*/
%PL1_OR_C = 1; /* Input parm */
DCL MODULETYPE FIXED INIT(1); /* Input parm */

DECLARE the parms to pass to RODM LOAD function */

DCL PARM_STRING CHAR(100) VARYING ALIGNED;
/* Keyword parms for load */
DCL DD_STRING CHAR(160) VARYING ALIGNED;
/* Load DD name mapping */

Figure 68. Calling the RODM Load Function from a PL/I Program (Part 1 of 4)

DCL EKGLJOB OPTIONS(ASM INTER) ENTRY EXTERNAL;
/* This entry is used when */
/* calling C or IBM PLI */
/* modules */

DCL EKGLTOLM OPTIONS(ASM INTER) ENTRY EXTERNAL;
/* This entry is used */
/* otherwise */

Figure 68. Calling the RODM Load Function from a PL/I Program (Part 2 of 4)

ASSIGN the value for the parms */

PARM_STRING = 'OPERATION=LOAD,LOAD=INSTANCE,NAME=EKGXRODM';
/* Load function input parms */
/* 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. */

Figure 68. Calling the RODM Load Function from a PL/I Program (Part 3 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
```

**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
```
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

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 you would 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 invoked 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=PARSE)

(OPERATION=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 260.

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.

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 terminates 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

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 explains 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 31 describes valid syntax delimiters for RODM high-level load function statements.

Table 31. Syntax Delimiters for RODM High-Level Load Function Statements

<table>
<thead>
<tr>
<th>Delimiter</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>' '</td>
<td>Used to enclose a character string.</td>
</tr>
<tr>
<td>X'0E' (Shift-out)</td>
<td>Marks the start of a DBCS mixed string data type.</td>
</tr>
<tr>
<td>X'0F' (Shift-in)</td>
<td>Marks the end of a DBCS mixed string data type.</td>
</tr>
<tr>
<td>-- (two hyphens)</td>
<td>Marks the beginning or end of a comment.</td>
</tr>
</tbody>
</table>

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 would result in a single quoted string:

```
INIT(' This is the first line of two lines ' ' that results in one quoted string ' );
```

A quote contained within quotes is represented by two single quotes, for example:

```
INIT('This is ' ' a quote ' ' within a quote. ' );
```

Quotes 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 231.

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 225. 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 302.

Comments: Comments are delimited by two hyphens (--) at the beginning and at the end. An example is:
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 its name, purpose, external syntax, 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 69. These statements create and use the hierarchical pseudo-structure shown in Figure 70 on page 278. This structure and the associated fields are an example for explanation purposes only, they are not part of RODM.

![Diagram of hierarchical pseudo-structure](Figure 69. 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-BB2',
    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 70. 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

field type PUBLIC|PRIVATE|PUBLIC_INDEXED

INIT (init_value)—

;— END;—
```

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 301.

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 222.

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:
CREATE:

Purpose: Use the CREATE high-level load function statement to create an object of a specific class in the RODM data cache.

Syntax:

```
+<.-VVVV)+<.-VVVV
CREATE
INVOKER ::= invoke_value;
OBJCLASS ::= class;
OBJINST ::= MyName = (CHARVAR) 'object';
ATTRLIST
field ::= typed_value;
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 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 "typing_values" on page 307.

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:

```plaintext
CREATE
  INVOKER ::= 00000003;
  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 71. 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:

```plaintext
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:

```plaintext
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.

```plaintext
DELETE
  INVOKER ::= invoke_value;
  OBJCLASS ::= class;
  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 72 shows a DELETE statement that deletes an object from the data model.

```plaintext
DELETE INVKER ::= 0000005;
OBJCLASS ::= SystemSoftware;
OBJINST ::= MyName = (CHARVAR) 'SDSF';
END;
```

*Figure 72. 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 INVKER ::= invoke_value;       --- MODE ::= mode_value---;

OBJCLASS ::= class;              --- OBJINST ::= MyName = (CHARVAR) 'object'--;

MODLIST field ::= typed_value --- modifier; --- 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.

**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 302.

**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 73.

```
SET 
INVOKER ::= 0000004;
MODE ::= non-confirmed;
OBJCLASS ::= SystemSoftware;
OBJINST ::= MyName = (CHARVAR) 'SDSF';
MODLIST
ProductName ::= (CHARVAR) 'SDSF V2', REPLACE,
ProgramNumber ::= (CHARVAR) '5697-B82',
LatestPTFNnumber ::= (CHARVAR), SET TO DEFAULT,
CorrespondingAPARNumber ::= (CHARVAR) ' ',
DateApplied ::= (CHARVAR) '03/01/97',
UseInHost ::= (OBJECTLINKLIST)
('Host_Class'.HostA.'UseSystemSoftware'),REMOVE VALUE;
END;
```

Figure 73. 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 explains how to code RODM load function primitive statements. It provides the syntax and processing logic along with the associated syntax rules. It also explains 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 short-hand 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 275.

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
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
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:

```plaintext
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:

```plaintext
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:
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:


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 285.

Syntax:


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:

```
+<.-VVVV)+<.-VVVV
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
```

```
+<.-VVVV)+<.-VVVV
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<object> INHERITS field;
```

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 invokes a named object-specific method or an object-independent method.

A maximum of 8 parameters can be specified with sd_parm.

Syntax:

```
Invoked_With
```

Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer’s Guide
An object-specific method is invoked with `sd_parm` parameters.

An object-independent method is invoked 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.object.field.subfield SUBFIELD_HAS_VALUE typed_value;
```

subfield of field of object 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.object.field SUBFIELD_INHERITS subfield;
```

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

```
  printable_character
   +<.-VVVV)
```

Usage Notes: A double-byte character string must be preceded by a shift-out character and terminated with a shift-in character.

char_literal:

Purpose: A character string within single quotes.

Format:

Char_Literal

```
  'chars'
```

Usage Notes: To indicate a single-quote character (’) within a char_literal, use two immediately adjacent single-quotes with no spaces or new lines between the two single-quotes. 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-quotes.

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 quotes.

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

dbcx_literal:

Purpose: A concatenation of a shift-out character, one or more valid double-byte characters, and a shift-in character.

Format:

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 231].

**digits:**

*Purpose:* The concatenation of any of the decimal digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9.

*Format:*

```
Digits
```

![Digits Diagram]

**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 quotes.

**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

\[
\text{numeric_literal} \pm \text{digits} \pm \text{exp_digits}
\]

hex_chars:

Purpose: The concatenation of hexadecimal character pairs, where each pair represents one byte.

Format:

Hex_CHARS

\[
\text{Hex_Literal} = X'\text{hex_chars}'
\]

hex_literal:

Purpose: One or more pairs of hexadecimal characters, within the hex delimiters.

Format:

Hex_Literal

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 quotes.

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:

\texttt{objectid\_list}


\texttt{class.object}

\texttt{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:

\texttt{objectlink\_list}

\texttt{(class.object.field)}

\texttt{recipient\_spec:}

Purpose: A recipient specification is a concatenation of two character literals and a hexadecimal literal, all of which must be exactly eight bytes in length.

Format:

\texttt{recipient\_spec}

\texttt{\textquotesingle appl\_id\textquotesingle,\textquotesingle subscribe\_id\textquotesingle,X\textquotesingle hex\_chars\textquotesingle}

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.

\texttt{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 sub_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  |
2. For limitations in CLASSID and OBJECTID, see "Using CLASSID and OBJECTID Data Types" on page 261.

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

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 262.
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 304.

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 343.

The NetView program supplies a set of general-purpose methods. For a description of these methods, see “NetView-Supplied Methods” on page 483.

Tasks Best Performed with User Applications

This section explains 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 74 shows typical user API invocations, first in C and then in PL/I.

```c
#include <EKGCEEP.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 */
```

```pli
%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 74. 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 75 on page 307.

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 V2R3 or later, this value is provided by the built-in function SYSNULL. 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 executed. 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 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 EKGUAPI, EKGMAPI, and EKGWAIT.

If no RODM control blocks are referenced in the modules, but the modules call EKGUAPI or EKGWAIT, include file EKG3CEEP.H in your source file.

Example:

```c
#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 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 EKGUAPI, EKGMAPI, and EKGWAIT.

If no RODM control blocks are referenced in the modules but the modules call EKGUAPI or EKGWAIT, include file EKG1IEEP 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);
```
Using the User Application Program Interface

/* 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 ENTRY PLISTART statement should be specified for PL/I.

The ENTRY CEESTART statement should be specified for C.

Example for PL/I:

```
<module code>
INCLUDE SYSLIB(EKGUAPI)
ENTRY PLISTART
NAME module_name(R)
```

Example for C:

```
<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 75 on page 307. 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:
Using the User Application Program Interface

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 75, 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.

Access Block

Description
The access block contains user information that RODM needs to process user API requests.

Figure 75. API Query Function Control Block Example

Call EKGUAPI( access_block,
transaction_info_block,
query_function_specific_data,
response_block
);

Figure 11. Writing Applications that Use RODM 307
Table 32. RODM Access Block

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>8</td>
<td>character(8)</td>
<td>In</td>
<td>RODM_name</td>
</tr>
<tr>
<td>008</td>
<td>16</td>
<td>Anonymous(16)</td>
<td>In/Out</td>
<td>Sign_on_token</td>
</tr>
<tr>
<td>024</td>
<td>8</td>
<td>ApplicationID</td>
<td>In</td>
<td>User_appl_ID</td>
</tr>
</tbody>
</table>

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 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. You should include these control blocks in your programs.

Table 33. Sample Names for Access Block

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I access block</td>
<td>EKG1ACCB</td>
</tr>
</tbody>
</table>
Table 33. Sample Names for Access Block (continued)

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>C access block</td>
<td>EKG3ACCB</td>
</tr>
</tbody>
</table>

**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 user API fills in a value for the sign_on_token field. After the access block is established by a connect request, the application should not modify the information in that block.

More details about connection to RODM are provided in "Connecting to RODM" on page 330.

**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 34 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>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>API_version</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>EPL_blk_len</td>
</tr>
<tr>
<td>008</td>
<td>8</td>
<td>TransID</td>
<td>Out</td>
<td>Transaction_ID</td>
</tr>
<tr>
<td>016</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Return_code</td>
</tr>
</tbody>
</table>

Chapter 11. Writing Applications that Use RODM 309
Using the User Application Program Interface

Table 34. RODM Transaction Information Block (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>020</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Reason_code</td>
</tr>
<tr>
<td>024</td>
<td>0</td>
<td>Structure</td>
<td></td>
<td>EPL_info</td>
</tr>
<tr>
<td>024</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Lock_level</td>
</tr>
</tbody>
</table>

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 explained in detail in the section of this document on Registering for Checkpoint Notification. See “Coding Checkpoint Control” on page 386.

**Return_code**
Return code from RODM. See “RODM Return and Reason Codes” on page 454 for a list of return codes.

**Reason_code**
Reason code from RODM. See “RODM Return and Reason Codes” on page 454 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. You should include these control blocks in your programs.

Table 35. Sample Names for Transaction Information Block

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I transaction information block</td>
<td>EKG1TRAB</td>
</tr>
<tr>
<td>C transaction information block</td>
<td>EKG3TRAB</td>
</tr>
</tbody>
</table>

**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 373.

Function Block Field Descriptions
A description of each parameter used in the function blocks is listed in "Function Parameter Descriptions" on page 447.

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 36 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>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Anonymous(4)</td>
<td>—</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
Using the User Application Program Interface

Table 36. RODM Entity Access Information Block (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>004</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Naming_count</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>ClassID</td>
<td>In/Out</td>
<td>Class_ID</td>
</tr>
<tr>
<td>012</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Class_name_length</td>
</tr>
<tr>
<td>016</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Class_name_ptr</td>
</tr>
<tr>
<td>020</td>
<td>8</td>
<td>ObjectID</td>
<td>In/Out</td>
<td>Object_ID</td>
</tr>
<tr>
<td>028</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Object_name_length</td>
</tr>
<tr>
<td>032</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Object_name_ptr</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,2</td>
<td>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.</td>
</tr>
<tr>
<td>1</td>
<td>Specifies that the target of the function is a class and that only the class access information is valid.</td>
</tr>
</tbody>
</table>

Interpretation of all this information is subject to the rules in "Usage" on page 313.

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 V2R3 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 V2R3 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 37. Sample Names for Entity Access Information Block

<table>
<thead>
<tr>
<th>Sample</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I entity access information block</td>
<td>EKG1ENTB</td>
</tr>
<tr>
<td>C entity access information block</td>
<td>EKG3ENTB</td>
</tr>
</tbody>
</table>

**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, executes 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 38 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>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Anonymous(4)</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Naming_count</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>FieldID</td>
<td>In/Out</td>
<td>Field_ID</td>
</tr>
<tr>
<td>012</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Field_name_length</td>
</tr>
<tr>
<td>016</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Field_name_ptr</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The information is valid</td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Naming_count should always be set 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 39. Sample Names for Field Access Information Block

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I field access information block</td>
<td>EKG1FLDB</td>
</tr>
<tr>
<td>C field access information block</td>
<td>EKG3FLDB</td>
</tr>
</tbody>
</table>
Using the User Application Program Interface

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, executes 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 40 contains a page reference to each response block format by function.

<table>
<thead>
<tr>
<th>Function with Response Block</th>
<th>See Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EKG_Locate</td>
<td>406</td>
</tr>
<tr>
<td>EKG_QueryEntityStructure</td>
<td>411</td>
</tr>
<tr>
<td>EKG_QueryField</td>
<td>413</td>
</tr>
<tr>
<td>EKG_QueryFieldID</td>
<td>415</td>
</tr>
<tr>
<td>EKG_QueryFieldName</td>
<td>416</td>
</tr>
<tr>
<td>EKG_QueryFieldStructure</td>
<td>417</td>
</tr>
<tr>
<td>EKG_QueryMultipleSubfields</td>
<td>421</td>
</tr>
<tr>
<td>EKG_QueryFunctionBlockContents</td>
<td>419</td>
</tr>
<tr>
<td>EKG_QueryNotifyQueue</td>
<td>423</td>
</tr>
<tr>
<td>EKG_QueryObjectName</td>
<td>425</td>
</tr>
<tr>
<td>EKG_QueryResponseBlockOverflow</td>
<td>427</td>
</tr>
<tr>
<td>EKG_QuerySubfield</td>
<td>428</td>
</tr>
<tr>
<td>EKG_TriggerNamedMethod</td>
<td>441</td>
</tr>
<tr>
<td>EKG_TriggerOIMethod</td>
<td>442</td>
</tr>
<tr>
<td>EKG_WhereAmI</td>
<td>446</td>
</tr>
</tbody>
</table>

**Function Block Field Descriptions**
A description of each parameter used in the response blocks is listed in "Function Parameter Descriptions" on page 447.

**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 terminated with an error return code.
- If the supplied response block has 8 or more bytes, the transaction is executed 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 426."

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 executed from a list of transactions, remaining transactions in the list are executed 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 invoking 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. You should 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 EKGU_API. 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 434. 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

Chapter 11. Writing Applications that Use RODM
Error Conditions in Transactions

When 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 203.
  - If the method is asynchronous, the return code must be greater than the LOG_LEVEL parameter in the RODM customization file. Refer to the 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 would have returned 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 should 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 328 describes notification for errors and checkpointing. User applications should set up any required notifications as soon as possible after connecting to RODM.
- Deleted objects. "Object Deletion Notification" on page 328 describes notification for deleted objects. Instead of installing your own notification methods, your applications use the EKG_AddObjDelSubs function (described on page 378) 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
RODM Notification Process

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 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 360 explains how to install a method.

In this example, we are using one of the notification methods supplied with RODM. 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 483. If the NetView-supplied methods do not meet your needs, you could 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 322. 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 322. 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 483.

Repeat Step 5 on page 322 once for each field you subscribe to. The setup for the notification process is complete when the EKG_AddNotifySubscription function has executed successfully for each object.

Although this example discusses 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 could 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 would need 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 should suspend 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.
RODM Notification Process

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 will contain the index into ECB_Array for which the ECB was posted.

PL/I Coding Example

Figure 76 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 76. PL/I Coding Example (Part 1 of 4)
Figure 76. PL/I Coding Example (Part 2 of 4)

```pli
Dcl
    ECB_Array(n) Pointer,
    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 76. PL/I Coding Example (Part 3 of 4)

```pli
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 76. PL/I Coding Example (Part 4 of 4)

**C Coding Example**

Figure 77 is an example for calling EKGWAIT from a C user application:

```c
#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 */
```
**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.
RODM Notification Process

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 terminates 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 203 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 NetView-supplied method EKGNOTF can be used for these notification subscriptions. See “RODM Notification Methods” on page 483 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 423.
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 321.

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 322.

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 322. 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 once for each object you subscribe to. The setup for the deletion-notification process is complete when the EKG_AddObjDelSubs function has executed successfully for each object.

**Wait for Object-Deletion Notification**

This step is the same as "Wait" on page 323.

**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 395.
Connecting to RODM

Before you can execute any user API functions, you must connect to RODM using the EKG_Connect 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, 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 abnormally terminated 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 387.

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 execute another API function request, unless the API function request is an EKG_Connect 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 terminate either normally or abnormally while they are connected to RODM, are disconnected.
For more information about disconnecting from RODM, see "EKG Disconnect — Disconnect from RODM" on page 401.
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 (NMC) operator can:
- 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, refer to the *Tivoli NetView for z/OS MultiSystem Manager User’s Guide.*

Enabling the Correlation Function

Object correlation is enabled by loading input file FLCSDM8 into RODM. To load FLCSDM8, remove the asterisk (*) from the following line in job CNMSJH12:

```plaintext
//* DD DSN=NETVIEW.V5R1M0.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 file FLCSDM8. MultiSystem Manager and SNA topology manager automatically sets the value of these fields, which causes correlation to occur and the views are displayed on a NMC.

Enabling MultiSystem Manager Object Correlation

To optimize navigation and storage for resources managed by the NetFinity and TMR agents, issue the GETTOPO commands in the order listed:
1. GETTOPO NFRES
2. GETTOPO TMERES
3. Any other GETTOPO commands

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:

```plaintext
WRITE_CORRELATABLE_FIELDS=YES
```

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 OS/2 workstations. If SNA topology manager does not monitor any OS/2 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 339.

**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

RODM load input file FLCSDM8 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 method FLCMCON is installed changes. Correlation automatically associates resources managed by different agents. The correlation function executes 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 LAN 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:

- The correlation function is enabled.
- You have a workstation that contains an internet host and a NetWare server.
- The resources are represented by objects in RODM, and on 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.

**Method FLCMCONI**

Method FLCMCONI is an initialization method that loads method FLCMCON on classes that support correlation. Method FLCMCONI is used instead of RODM load input file DUIFSTRC because method FLCMCONI passes parameters to method FLCMCON.

**Method FLCMCON**

Method FLCMCON 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 method FLCMCON is loaded, browse RODM load file FLCSDM8. FLCMCON invokes FLCMCOR.

**Method FLCMCOR**
Method FLCMCOR is an object-independent method that creates and updates correlated aggregate objects.

The load and customization of these methods is accomplished using RODM load file FLCSDM8. For more information, refer to "Enabling the Correlation Function" on page 333 and "Customizing the Correlation Function" on page 341.

**Objects Enabled for Correlation**
Loading sample FLCSDM8 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 RODM load file FLCSDM8. All classes on which method FLCMCON is loaded are automatically enabled.

For example, the following code enables correlation by IP address on objects of the internetRouter class, which are created by the MultiSystem Manager IP feature:

```op
OP FLCMCONI INVOKED_WITH (SELFDEFINING)
  {
    (OBJECTID) EKG_Method.FLCMCON
    (CLASSID) '1.3.18.0.0.3330' -- internetRouter
    (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 (for example, 9.37.263.47).

**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:
You can also enter a multipart string value in the Correlator field. Entering a multipart string enables you to link the correlated object to a hierarchy of correlated aggregate objects as shown in Figure 78.

To enable correlation to create the objects in Figure 78, 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 will be located or created. If parent relationships do not already exist between the different correlated aggregate objects identified in the multipart string, they will be 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 will locate or create two objects – Jane and Doe.

All of the characters supported by the RODM CharVar data type are supported. This enables you to use an underscore character (_) between string values that you want to be treated as one correlated aggregate object (for example, Margaret_Thatcher).
For Tivoli NetView for z/OS V5R1, free-form correlation was changed to create 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, it is now easier to 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:

- 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 *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:

![Aggregate Resource Symbol](image)

*Figure 79. Aggregate Resource Symbol*

Correlated aggregate object labels are determined by the first value for which a correlation was found:
Table 41. Correlated Aggregate Object Labels

<table>
<thead>
<tr>
<th>First Correlation Value</th>
<th>Resource Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC address</td>
<td>LAN workstation aggregate</td>
</tr>
<tr>
<td>IP address</td>
<td>IP system aggregate</td>
</tr>
<tr>
<td>Correlator field value</td>
<td>Open system aggregate</td>
</tr>
</tbody>
</table>

Correlated Aggregate Object Field Values

The correlation function is triggered when the value of a field on which method FLCMCON is installed changes. Method FLCMCON triggers method FLCMCOR. Method FLCMCOR 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. Refer to the Tivoli NetView for z/OS Data Model Reference for more information about these fields.

Use the NMC to display data contained in the DisplayResourceOtherData field. This information is displayed in the NMC Data1 field.

The value in the DisplayResourceOtherData field is not always provided by the correlation function. The NetFinity and TMR agents also create aggregateSystem class objects and set a value in the DisplayResourceOtherData field. Information 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 could 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, refer to the Tivoli NetView for z/OS MultiSystem Manager User’s Guide.

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 340.

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 file FLCSDM8. Enabling objects of the open data model requires less setup, and sample file FLCSOX01 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. Sample file FLCSOX01 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 340 and “Correlating SNA Topology Manager Objects” on page 341 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 may 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 Tivoli NetView for z/OS Data Model Reference.

Correlating MultiSystem Manager Objects

If method FLMCON is loaded directly on the field of an object you want to correlate, set a value on the field. To determine which fields have method FLMCON loaded, browse RODM load file FLCSDM8. 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 method FLMCON 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 IP or NetFinity agents also discover 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 an OS/2 workstation that has a SNA PU and a LAN adapter with IP support. SNA topology manager discovers MAC addresses only on OS/2 workstations.

To enable IP address correlation for SNA resources, manually set the address on the iPAddress field on an object enabled in file FLCSDM8. 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 RODM load file FLCSDM8. After customization, RODM load file FLCSDM8 must be loaded into RODM. If RODM load file 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 sample file EKGLLOAD to load file FLCSDM8. Ensure that you specify the dataset 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 correlator, 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 80 on page 342 are used to determine the correlated aggregate object display name. To determine which correlated aggregate object field will be used to label an object, the correlate function uses a prioritized list of those fields in file FLCSDM8. 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 42 lists the default priority used and the agents the priorities are used for.

<table>
<thead>
<tr>
<th>Priority</th>
<th>Name Type</th>
<th>Discovered By</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Computer name</td>
<td>NetFinity and TMR</td>
</tr>
<tr>
<td>2</td>
<td>IP host name</td>
<td>Internet, TMR, and NetFinity</td>
</tr>
<tr>
<td>3</td>
<td>TCP/IP address</td>
<td>Internet and TMR</td>
</tr>
<tr>
<td>5</td>
<td>SNA node name</td>
<td>SNATM</td>
</tr>
<tr>
<td>6</td>
<td>LAN MAC address</td>
<td>LNM, SNATM, Internet, and NetFinity</td>
</tr>
</tbody>
</table>

You can determine which label will be displayed by customizing the order in which the fields are listed.

For example, using the default priority shown in Figure 80 on page 342, a workstation that contains an IP 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.
Now, assume that you have customized file FLCSDM8 to put TCP/IP address (priority 3) before IP host name (priority 2) as shown in Figure 81. In this case, the TCP/IP address is used to label the workstation object because the IP agent provides both an IP host name and an IP address, and the IP address name is listed first.

Disabling Correlation for Specific Resources

Correlation is enabled for objects of the classes on which method FLCMCON is explicitly loaded in file FLCSDM8. If you do not want topology correlation to execute for a class of managed resource objects, comment out the method load statement that loads file FLCMCON on the class.

The method load statements are grouped in file FLCSDM8 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 Tivoli NetView for z/OS Data Model Reference and match the label with the DisplayResourceType values listed.

Note: Using file FLCSDM8 as shipped, method FLCMCON 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 explains 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 invoked by user applications, by changes to fields in RODM, by other methods, and at RODM initialization. Methods are classified by the way they are invoked.

The NetView program supplies several general-purpose methods that may meet some or all of your needs. Before you spend time writing your own methods, review the NetView-supplied methods as described in “NetView-Supplied Methods” on page 483 for applicability.

You must install each method, including NetView-supplied methods, 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 explains 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 344 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 345 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 346 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 349 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 350 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 353 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 invoked under certain circumstances. Methods are classified according to the circumstances under which they are invoked. 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 executed only in the context of a particular object. For example, they are invoked by transactions that refer to a specific object. When an object-specific method is executing, 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 invoked 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 invoked from the user API or the method API.
Types of Methods

- They can be invoked 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 invoked. RODM does not restrict the contents of that string. You must coordinate the parameter passed when the method is invoked with the parsing and meaning that the message attaches to the string of bytes that is passed.

Figure 82 shows how an object-independent method is defined in PL/I. Figure 83 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 82. Object-Independent Method Procedure Interface for PL/I*

```
VOID ObjIndpMeth(SelfDefiningDataPtr **in_ChStrParm);
    . . .
    /* code */
    . . .
```

*Figure 83. Object-Independent Method Procedure Interface for C*

**Initialization Method**

The initialization method is a special kind of object-independent method. It is invoked by RODM at initialization time. When RODM is started with the initialization method, RODM installs, invokes, and then frees the method automatically. The main purpose of the initialization method is to set up the initial
Types of Methods

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:

- Invoked implicitly as the side effect of a transaction
  - Query method (when querying data)
  - Change method (when changing data)
  - Notify method (after changing data)
- Invoked 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.
- Executes 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_livedParms
  A SelfDefining string containing application-defined parameters. These parameters are provided to the change method when it is installed.

short_livedParms
  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 invoked every time a transaction is executed (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 invoked only when the field on which they are defined is directly changed. A change method is not invoked when the same field on the parent class is changed and the changed value is inherited. A change method is not invoked by changes in a child object or class. A change method is not invoked 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 invokes 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 should send any request to the resource asynchronously and set the appropriate flags to indicate that the request has been sent. The change method should 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 executed. 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 invoked. It optionally contains long-lived parameters that are passed to the method when it is invoked. 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 invoked. 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
Types of Methods

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 84 shows example change method parameters for PL/I. Figure 85 shows example change method parameters 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:

- Terminate 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.

Figure 84. Change Method Procedure Interface for PL/I

Figure 85. Change Method Procedure Interface for C
Query Methods
A query method is invoked by RODM when a transaction queries the value of a field; but not invoked 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
- Executes 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 invoked 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 could 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 timestamps 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 should return immediately to the caller with a reason code indicating that a request for new data has been submitted. No method should enter 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 executed.

A query subfield has data type MethodSpec. A query subfield can preserve the name of a query method to be invoked 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 86 on page 351 shows an example of query method parameters for PL/I. Figure 87 on page 351 shows an example of query method parameters for C.
Notify Methods

Notification methods are invoked by RODM after certain functions are made. To determine which functions invoke notification methods, see the description for the function in "Chapter 14. Application Programming Reference" on page 371.

A notification method:
- Generates notifications to subscribed users
- Is inherited only from class to object
- Executes 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_livedParms**
SelfDefining string containing application-defined parameters. These parameters are provided to the notification method when it is installed.

**short_livedParms**
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.
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 executed. 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 executed. 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 executed.

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 executed. When a change is made to an object field, the notification subscriptions assigned to the field in that object are executed. Any notification subscriptions assigned to the same field in the primary parent are also executed, 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 483. Figure 88 shows an example of notification parameters for PL/I. Figure 89 on page 353 shows an example of notification parameters for C.

```
NotifMeth: Procedure ( FieldID, LLPrams, SLPrams, Change_status,
                      User_Appl_ID, Notif_queue_ID, User_word );

Declare
    FieldID      Field-identifier, /* Field-identifier of named field */
    LLPrams     SelfDefiningDataPtr, /* Pointer to Long-lived field parameters */
    SLPrams     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 88. Notification Method Procedure Interface for PL/I
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 timestamp subfields

The named method parameters are:

field_id
  FieldID of the field being invoked.

long_livedParms
  SelfDefining string containing application-defined parameters. These parameters are provided to the named method when it is installed.

short_livedParms
  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 invoked. 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 NetView-supplied 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
Types of Methods

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 executed. 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 487.

Figure 90 shows an example of named method parameters for PL/I. Figure 91 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 90. Named Method Procedure Interface for PL/I

```
VOID NamedMeth(FieldID *in_FieldID,
SelfDefiningDataPtr **in_LLParms,
SelfDefiningDataPtr **in_SLParms);
....
/* code */
....
```

Figure 91. 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 would be used on objects. Complex changes to a class can be executed 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 executed 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 executes 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 execute on an object and at other times can execute 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.
Types of Methods

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 executed 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 should be 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 invoke an action against another object or class, an object-specific method can use the
Deciding Method Type

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 368 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 43 lists where you can find more information about these blocks.

Table 43. Additional Information About Blocks

<table>
<thead>
<tr>
<th>If You Want More Information on</th>
<th>See Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction_info_block</td>
<td>310</td>
</tr>
<tr>
<td>Response_block</td>
<td>311</td>
</tr>
<tr>
<td>Function_block</td>
<td>311</td>
</tr>
</tbody>
</table>

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 92 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 92. 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 run-time 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 371.

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 93. 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
Using the Method API

- 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 executed (by the appropriate triggering mechanism), and they are made available to that method when the method executes. 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 invoked 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 executed, 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 executed 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 NetView-supplied methods 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 invoked.

The EKGMANC service routine should be invoked using the following code for PL/I:

```pli
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:

```c
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 explain 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 V2 R3 or in C V2R1. C V2R1 is compatible with PL/I V2R3 because routines in each language can call routines written in the other. These methods can use the national language 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 execute the method using test data and debugging aids to find any syntax or logic errors. Refer to the Tivoli NetView for z/OS Customization: Using 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-Supplied 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 NetView-supplied methods with your methods. All NetView-supplied methods reside in the CNMLINK target library for the NetView program product.

Currently, the methods supplied with RODM consist of the following:
Coding a RODM Method

EKGNOTF  
Notify for any change

EKGNLST  
Notify if changed value is equal to one value in a list of values

EKGNBQVL  
Notify if changed value is equal to a specific value

EKGNTHD  
Notify if changed value is within a specified threshold

EKGCIM  
Change method to trigger an Object-independent method to complete an action asynchronously

EKGMIMV  
Named method to increment a value

EKGSPP  
Object-independent method used by the RODM automation platform

All notification methods return, in the notification block, the current value, previous value, and timestamp (if these subfields are defined) from the field causing the notification message.

The NetView-supplied methods for RODM are described below on a functional basis. All parameters 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 OS/390 storage key 8, which is the default. Do not change the storage key of 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 should be freed. In addition, PL/I programs that execute 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 should attempt 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 should 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 should not normally be used. Instead, the EKG_OutputToLog API function should be used.

- **Use of PL/I multitasking**
  The PL/I multitasking facilities should not be 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 should 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 invoked to execute 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 invoked to execute 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 213 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 execute another method must not cause recursive execution of any previously invoked 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 list 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 413)
  - EKG_QueryMultipleSubfields (See “EKG_QueryMultipleSubfields — Query Multiple Value Subfields” on page 421)
  - EKG_QuerySubfield (See “EKG_QuerySubfield — Query a Subfield” on page 428)
  - EKG_QueryEntityStructure (See “EKG_QueryEntityStructure — Query Structure of an Entity” on page 411)
  - EKG_QueryFieldStructure (See “EKG_QueryFieldStructure — Query Structure of a Field” on page 417)
  - EKG_QueryFieldID (See “EKG_QueryFieldID — Query Field Identifier” on page 414)
  - EKG_QueryFieldName (See “EKG_QueryFieldName — Query a Field Name” on page 416)

- Actions against RODM Data
  - EKG_ChangeField (See “EKG_ChangeField — Change a Field” on page 380)
  - EKG_ChangeMultipleFields (See “EKG_ChangeMultipleFields — Change Multiple Fields” on page 381)
  - EKG_ChangeSubfield (See “EKG_ChangeSubfield — Change a Subfield” on page 385)
  - EKG_RevertToInherited (See “EKG_RevertToInherited — Revert to Inherited Value” on page 431)
  - EKG_AddNotifySubscription (See “EKG_AddNotifySubscription — Add Notification Subscription” on page 372)
  - EKG_DeleteNotifySubscription (See “EKG_DeleteNotifySubscription — Delete Notification Subscription” on page 395)
RODM Method Services

- EKG_TriggerNamedMethod (See "EKG_TriggerNamedMethod — Trigger a Named Method” on page 440)

• 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 303 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 404)
• EKG_UnlinkNoTrigger, EKG_UnlinkTrigger (See "EKG_UnlinkNoTrigger, EKG_UnlinkTrigger — Unlink Two Objects” on page 443)
• EKG_CreateObject (See "EKG_CreateObject — Create an Object” on page 391)
• EKG_DeleteObject (See "EKG_DeleteObject — Delete an Object” on page 397)
• EKG_TriggerOIMethod (See "EKG_TriggerOIMethod — Trigger an Object-Independent Method” on page 442)

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 invoke 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 388)
  - EKG_CreateField (See "EKG_CreateField — Create a Field” on page 389)
  - EKG_CreateSubfield (See "EKG_CreateSubfield — Create a Subfield” on page 392)

• Control functions
The access to the above mentioned functions is similar to the access available through the user API. These functions are invoked 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 303 for an explanation of how function blocks are used in application programs. See “Chapter 13. Writing RODM Methods” on page 343 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 - 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 terminate 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 terminate RODM, with or without checkpointing data.

EKG_Chekpoint - Checkpoint RODM
Checkpoint RODM data to DASD.

EKG_Stop - Stop RODM
Terminate 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.
Summarizing RODM Functions

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_DeleteObjDelSubs - Delete an Object Deletion Subscription
Delete a subscription to an object.
Summarizing RODM Functions

EKG_LinkNoTrigger - Link Two Objects
Link two objects; do not invoke notify methods.

EKG_LinkTrigger - Link Two Objects
Link two objects; invoke 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
Invoke a named method.

EKG_TriggerOIMethod - Trigger an Object-Independent Method
Invoke an object independent method.

EKG_UnlinkNoTrigger - Unlink Two Objects
Unlink two objects; do not invoke notify methods.

EKG_UnlinkTrigger - Unlink Two Objects
Unlink two objects; invoke 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 invoked 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.
Summarizing RODM Functions

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 447. 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.V5R1M0.SCNMMAC1`. The default data set name for usage coding examples is `NETVIEW.V5R1M0.CNMSAMP`. For example, the complete name of the function block example in PL/I for the `EKG_Connect` function is `NETVIEW.V5R1M0.SCNMMAC1(EKG11101)`. The complete name of the PL/I usage coding example for this function is `NETVIEW.V5R1M0.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 executed 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. Refer to 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 447. 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 454. 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 NetView-supplied methods. These include notification and change methods you can use with RODM. “NetView-Supplied Methods” on page 483 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>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Entity_access_info_ptr</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Field_access_info_ptr</td>
</tr>
<tr>
<td>012</td>
<td>8</td>
<td>ApplicationID</td>
<td>In</td>
<td>User_appl_ID</td>
</tr>
<tr>
<td>020</td>
<td>8</td>
<td>SubscribeID</td>
<td>In</td>
<td>Notification_queue</td>
</tr>
<tr>
<td>028</td>
<td>8</td>
<td>Anonymous(8)</td>
<td>In</td>
<td>User_word</td>
</tr>
<tr>
<td>036</td>
<td>8</td>
<td>ObjectID</td>
<td>In</td>
<td>Notify_method</td>
</tr>
<tr>
<td>044</td>
<td>4</td>
<td>SelfDefiningDataPtr</td>
<td>In</td>
<td>Long_lived_parm</td>
</tr>
</tbody>
</table>
```

See “Function Parameter Descriptions” on page 447 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 45. Example Names for the EKG_AddNotifySubscription Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11412</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51412</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31412</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61412</td>
</tr>
</tbody>
</table>
```
EKG_AddNotifySubscription

Summary

Table 46. Summary of the EKG_AddNotifySubscription Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>1412</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Action</td>
</tr>
<tr>
<td>User API</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>Yes</td>
</tr>
<tr>
<td>Initialization method</td>
<td>Yes</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>Notification method of MyObjectChildren field of the EKG_NotificationQueue class triggered if the notification queue object is created</td>
</tr>
<tr>
<td>Triggered by the EKG_MessageTriggeredAction function</td>
<td>No</td>
</tr>
<tr>
<td>Authorization</td>
<td>3</td>
</tr>
</tbody>
</table>

Usage

See [RODM Notification Process" on page 321] 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 invoked.

When a notification method is invoked, 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 executed 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 47. Function Block for the EKG_AddObjDelSubs Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
</tbody>
</table>
Table 47. Function Block for the EKG_AddObjDelSubs Function (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Entity_access_info_ptr</td>
</tr>
<tr>
<td>008</td>
<td>8</td>
<td>ApplicationID</td>
<td>In</td>
<td>User_appl_ID</td>
</tr>
<tr>
<td>016</td>
<td>8</td>
<td>SubscribeID</td>
<td>In</td>
<td>Notification_queue</td>
</tr>
<tr>
<td>024</td>
<td>8</td>
<td>Anonymous(8)</td>
<td>In</td>
<td>User_word</td>
</tr>
<tr>
<td>032</td>
<td>4</td>
<td>SelfDefiningDataPtr</td>
<td>In</td>
<td>Long_lived_parm</td>
</tr>
</tbody>
</table>

See “Function Parameter Descriptions” on page 447 for more information about the parameters listed. See “Abstract Data Type Reference” on page 225 for more information about the abstract data types listed.

Examples

Table 48. Example Names for the EKG_AddObjDelSubs Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11417</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51417</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31417</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61417</td>
</tr>
</tbody>
</table>

Summary

Table 49. Summary of the EKG_AddObjDelSubs Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>Type</th>
<th>User API</th>
<th>Object-specific method</th>
<th>Object-independent method</th>
<th>Initialization method</th>
<th>Methods triggered</th>
<th>Triggered by the EKG_MessageTriggeredAction function</th>
<th>Authorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1417</td>
<td>Action</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>3</td>
</tr>
</tbody>
</table>

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 423.
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 50. Function Block for the EKG_ChangeField Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Entity_access_info_ptr</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Field_access_info_ptr</td>
</tr>
<tr>
<td>012</td>
<td>2</td>
<td>Smallint</td>
<td>In</td>
<td>Subfield</td>
</tr>
<tr>
<td>014</td>
<td>2</td>
<td>Smallint</td>
<td>In</td>
<td>Data_type</td>
</tr>
<tr>
<td>016</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>New_char_data_length</td>
</tr>
<tr>
<td>020</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>New_data_ptr</td>
</tr>
<tr>
<td>024</td>
<td>4</td>
<td>SelfDefiningDataPtr</td>
<td>In</td>
<td>Method_parms</td>
</tr>
</tbody>
</table>

Note that the Subfield parameter at offset 012 is not currently used.

See “Function Parameter Descriptions” on page 447 for more information about the parameters listed. See “Abstract Data Type Reference” on page 225 for more information about the abstract data types listed.

Examples

Table 51. Example Names for the EKG_ChangeField Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11401</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51401</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31401</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61401</td>
</tr>
</tbody>
</table>

Summary

Table 52. Summary of the EKG_ChangeField Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>Type</th>
<th>User API</th>
<th>Object-specific method</th>
<th>Object-independent method</th>
<th>Initialization method</th>
<th>Methods triggered</th>
<th>Triggered by the</th>
</tr>
</thead>
<tbody>
<tr>
<td>1401</td>
<td>Action</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Change and notification methods triggered</td>
<td>EKG_MessageTriggeredAction function</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Entity_access_info_ptr</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Number_of_fields</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>First element, array of structure</td>
</tr>
<tr>
<td>012</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Field_access_info_ptr</td>
</tr>
<tr>
<td>016</td>
<td>2</td>
<td>Anonymous(2)</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>018</td>
<td>2</td>
<td>Smallint</td>
<td>In</td>
<td>Data_type</td>
</tr>
<tr>
<td>020</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>New_char_data_length</td>
</tr>
<tr>
<td>024</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>New_data_ptr</td>
</tr>
<tr>
<td>028</td>
<td>4</td>
<td>SelfDefiningDataPtr</td>
<td>In</td>
<td>Method_parms</td>
</tr>
<tr>
<td>032</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Return_code</td>
</tr>
<tr>
<td>036</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Reason_code</td>
</tr>
</tbody>
</table>

See “Function Parameter Descriptions” on page 447 for more information about the parameters listed. See “Abstract Data Type Reference” on page 225 for more information about the abstract data types listed.

**Examples**

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11419</td>
</tr>
</tbody>
</table>
**EKG_ChangeMultipleFields**

**Table 54. Example Names for the EKG_ChangeMultipleFields Function (continued)**

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51419</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31419</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61419</td>
</tr>
</tbody>
</table>

**Summary**

**Table 55. Summary of the EKG_ChangeMultipleFields Function**

<table>
<thead>
<tr>
<th>Function ID</th>
<th>1419</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Action</td>
</tr>
<tr>
<td>User API</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>Yes</td>
</tr>
<tr>
<td>Initialization method</td>
<td>Yes</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>Change and notification methods triggered</td>
</tr>
<tr>
<td>Triggered by the EKG_MessageTriggeredAction function</td>
<td>Yes</td>
</tr>
<tr>
<td>Authorization</td>
<td>3</td>
</tr>
</tbody>
</table>

**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 56. Function Block for the EKG_ChangeSubfield Function**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Entity_access_info_ptr</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Field_access_info_ptr</td>
</tr>
<tr>
<td>012</td>
<td>2</td>
<td>Smallint</td>
<td>In</td>
<td>Subfield</td>
</tr>
</tbody>
</table>
Table 56. Function Block for the EKG_ChangeSubfield Function (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>014</td>
<td>2</td>
<td>Smallint</td>
<td>In</td>
<td>Data_type</td>
</tr>
<tr>
<td>016</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>New_char_data_length</td>
</tr>
<tr>
<td>020</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>New_data_ptr</td>
</tr>
<tr>
<td>024</td>
<td>4</td>
<td>—</td>
<td>—</td>
<td>Not used</td>
</tr>
</tbody>
</table>

See “Function Parameter Descriptions” on page 447 for more information about the parameters listed. See “Abstract Data Type Reference” on page 225 for more information about the abstract data types listed.

Examples

Table 57. Example Names for the EKG_ChangeSubfield Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11403</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51403</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31403</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61403</td>
</tr>
</tbody>
</table>

Summary

Table 58. Summary of the EKG_ChangeSubfield Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>1403</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Action</td>
</tr>
<tr>
<td>User API</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>Yes</td>
</tr>
<tr>
<td>Initialization method</td>
<td>Yes</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>No</td>
</tr>
<tr>
<td>Triggered by the EKG_MessageTriggeredAction function</td>
<td>Yes</td>
</tr>
<tr>
<td>Authorization</td>
<td>3</td>
</tr>
</tbody>
</table>

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 invoke 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 59. Function Block for the EKG_Checkpoint Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
</tbody>
</table>

See "Function Parameter Descriptions" on page 447 for more information about the parameters listed. See "Abstract Data Type Reference" on page 225 for more information about the abstract data types listed.

Examples
Table 60. Example Names for the EKG_Checkpoint Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11201</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51201</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31201</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61201</td>
</tr>
</tbody>
</table>

Summary
Table 61. Summary of the EKG_Checkpoint Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>Type</th>
<th>User API</th>
<th>Object-specific method</th>
<th>Object-independent method</th>
<th>Initialization method</th>
<th>Methods triggered</th>
<th>Triggered by the EKG_MessageTriggeredAction function</th>
<th>Authorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1201</td>
<td>Control</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Notification</td>
<td>No</td>
<td>4</td>
</tr>
</tbody>
</table>

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. Refer to the Tivoli NetView for z/OS Administration Reference for more information.
The data that the EKG_Checkpoint function writes to the checkpoint datasets 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. The operator should choose this option if a checkpoint is really desired, but give RODM another quiesce period to successfully quiesce.
   2. Unconditionally, start first stage checkpoint. The operator should choose this option if a checkpoint is immediately necessary or after having tried option one.
   3. Stop the checkpoint request. The operator should choose this option if option one has been attempted or if critical RODM tasks should not be terminated.

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 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.

The user should 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 datasets 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 372. You can use the NetView-supplied notification method EKGNOTF for this subscription. See “RODM Notification Methods” on page 483 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 the application can issue to RODM.

Function Block Format

Table 62. Function Block for the EKG_Connect Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>8</td>
<td>Char(8)</td>
<td>In</td>
<td>User_password</td>
</tr>
<tr>
<td>012</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Stop_ECB</td>
</tr>
<tr>
<td>016</td>
<td>8</td>
<td>TransID</td>
<td>Out</td>
<td>Last_checkpoint_ID</td>
</tr>
<tr>
<td>024</td>
<td>4</td>
<td>Anonymous(4)</td>
<td>—</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

See “Function Parameter Descriptions” on page 447 for more information about the parameters listed. See “Abstract Data Type Reference” on page 225 for more information about the abstract data types listed.

Examples

Table 63. Example Names for the EKG_Connect Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11101</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51101</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31101</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61101</td>
</tr>
</tbody>
</table>

Summary

Table 64. Summary of the EKG_Connect Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>1101</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Access</td>
</tr>
<tr>
<td>User API</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>No</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>No</td>
</tr>
<tr>
<td>Initialization method</td>
<td>No</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>Notification</td>
</tr>
<tr>
<td>Triggered by the EKG_MessageTriggeredAction function</td>
<td>No</td>
</tr>
<tr>
<td>Authorization</td>
<td>1</td>
</tr>
</tbody>
</table>

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_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 307, must be passed. The user’s sign_on_token parameter in the access block is set by RODM. This parameter should 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_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 terminates 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 65. Function Block for the EKG_CreateClass Function*

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Class_access_info_ptr</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Parent_access_info_ptr</td>
</tr>
<tr>
<td>012</td>
<td>4</td>
<td>SelfDefiningDataPtr</td>
<td>In</td>
<td>Method_parms</td>
</tr>
</tbody>
</table>

See “Function Parameter Descriptions” on page 447 for more information about the parameters listed. See “Abstract Data Type Reference” on page 225 for more information about the abstract data types listed.

**Examples**

*Table 66. Example Names for the EKG_CreateClass Function*

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11302</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51302</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31302</td>
</tr>
</tbody>
</table>
Table 66. Example Names for the EKG_CreateClass Function (continued)

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61302</td>
</tr>
</tbody>
</table>

**Summary**

Table 67. Summary of the EKG_CreateClass Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>1302</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Administrative</td>
</tr>
<tr>
<td>User API</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>No</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>No</td>
</tr>
<tr>
<td>Initialization method</td>
<td>Yes</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>Notification methods on MyClassChildren and WhatIAm fields of parent class triggered</td>
</tr>
<tr>
<td>Triggered by the EKG_MessageTriggeredAction function</td>
<td>No</td>
</tr>
<tr>
<td>Authorization</td>
<td>5</td>
</tr>
</tbody>
</table>

**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 68. Function Block for the EKG_CreateField Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Class_access_info_ptr</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Field_access_info_ptr</td>
</tr>
<tr>
<td>012</td>
<td>2</td>
<td>Smallint</td>
<td>In</td>
<td>Field_type_flag</td>
</tr>
<tr>
<td>014</td>
<td>2</td>
<td>Smallint</td>
<td>In</td>
<td>Data_type</td>
</tr>
<tr>
<td>016</td>
<td>4</td>
<td>Bit(32)</td>
<td>In</td>
<td>Subfield_map</td>
</tr>
</tbody>
</table>

See "Function Parameter Descriptions" on page 447 for more information about the parameters listed. See "Abstract Data Type Reference" on page 225 for more information about the abstract data types listed.
**EKG_CreateField**

**Examples**

*Table 69. Example Names for the EKG_CreateField Function*

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11304</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51304</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31304</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61304</td>
</tr>
</tbody>
</table>

**Summary**

*Table 70. Summary of the EKG_CreateField Function*

<table>
<thead>
<tr>
<th>Function ID</th>
<th>1304</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Administrative</td>
</tr>
<tr>
<td>User API</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>No</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>No</td>
</tr>
<tr>
<td>Initialization method</td>
<td>Yes</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>No</td>
</tr>
<tr>
<td>Triggered by the EKG_MessageTriggeredAction function</td>
<td>No</td>
</tr>
<tr>
<td>Authorization</td>
<td>5</td>
</tr>
</tbody>
</table>

**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 71. Function Block for the EKG_CreateObject Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Entity_access_info_ptr</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>SelfDefiningDataPtr</td>
<td>In</td>
<td>Method_Parms</td>
</tr>
</tbody>
</table>

See “Function Parameter Descriptions” on page 447 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 72. Example Names for the EKG_CreateObject Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11409</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51409</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31409</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61409</td>
</tr>
</tbody>
</table>

Summary

Table 73. Summary of the EKG_CreateObject Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>1409</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Action</td>
</tr>
<tr>
<td>User API</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>No</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>Yes</td>
</tr>
<tr>
<td>Initialization method</td>
<td>Yes ¹</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>Notification methods on MyClassChildren and WhatIAm fields of parent class triggered</td>
</tr>
</tbody>
</table>

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.
**EKG_CreateObject**

**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 typeObjectName 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 EKGddddddd where dddddddd 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>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Class_access_info_ptr</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Field_access_info_ptr</td>
</tr>
<tr>
<td>012</td>
<td>4</td>
<td>Bit(32)</td>
<td>In</td>
<td>Subfield_map</td>
</tr>
</tbody>
</table>

See "Function Parameter Descriptions" on page 447 for more information about the parameters listed. See "Abstract Data Type Reference" on page 225 for more information about the abstract data types listed.
Examples

Table 75. Example Names for the EKG_CreateSubfield Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11306</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51306</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31306</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61306</td>
</tr>
</tbody>
</table>

Summary

Table 76. Summary of the EKG_CreateSubfield Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>Type</th>
<th>User API</th>
<th>Object-specific method</th>
<th>Object-independent method</th>
<th>Initialization method</th>
<th>Methods triggered</th>
<th>Triggered by the EKG_MessageTriggeredAction function</th>
<th>Authorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1306</td>
<td>Administrative</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>5</td>
</tr>
</tbody>
</table>

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 77. Function Block for the EKG_DeleteClass Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Class_access_info_ptr</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>SelfDefiningDataPtr</td>
<td>In</td>
<td>Method_parms</td>
</tr>
</tbody>
</table>
EKG_DeleteClass

See "Function Parameter Descriptions" on page 447 for more information about the parameters listed. See "Abstract Data Type Reference" on page 225 for more information about the abstract data types listed.

Examples

Table 78. Example Names for the EKG_DeleteClass Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11303</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51303</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31303</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61303</td>
</tr>
</tbody>
</table>

Summary

Table 79. Summary of the EKG_DeleteClass Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>Type</th>
<th>User API</th>
<th>Object-specific method</th>
<th>Object-independent method</th>
<th>Initialization method</th>
<th>Methods triggered</th>
</tr>
</thead>
<tbody>
<tr>
<td>1303</td>
<td>Administrative</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Notification methods on MyClassChildren and WhatIAm fields of parent class triggered</td>
</tr>
</tbody>
</table>

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 80. Function Block for the EKG_DeleteField Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Class_access_info_ptr</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Field_access_info_ptr</td>
</tr>
</tbody>
</table>
See “Function Parameter Descriptions” on page 447 for more information about the parameters listed. See “Abstract Data Type Reference” on page 225 for more information about the abstract data types listed.

**Examples**

*Table 81. Example Names for the EKG_DeleteField Function*

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11305</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51305</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31305</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61305</td>
</tr>
</tbody>
</table>

**Summary**

*Table 82. Summary of the EKG_DeleteField Function*

<table>
<thead>
<tr>
<th>Function ID</th>
<th>Type</th>
<th>User API</th>
<th>Object-specific method</th>
<th>Object-independent method</th>
<th>Initialization method</th>
<th>Methods triggered</th>
<th>Triggered by the EKG_MessageTriggeredAction function</th>
<th>Authorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1305</td>
<td>Administrative</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>5</td>
</tr>
</tbody>
</table>

**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.
EKG_DeleteNotifySubscription

Function Block Format

Table 83. Function Block for the EKG_DeleteNotifySubscription Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Entity_access_info_ptr</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Field_access_info_ptr</td>
</tr>
<tr>
<td>012</td>
<td>24</td>
<td>RecipientSpec</td>
<td>In</td>
<td>Subscription_info</td>
</tr>
<tr>
<td>036</td>
<td>8</td>
<td>ObjectID</td>
<td>In</td>
<td>Notify_method</td>
</tr>
<tr>
<td>044</td>
<td>4</td>
<td>SelfDefiningDataPtr</td>
<td>In</td>
<td>Long_lived_parm</td>
</tr>
</tbody>
</table>

See “Function Parameter Descriptions” on page 447 for more information about the parameters listed. See “Abstract Data Type Reference” on page 225 for more information about the abstract data types listed.

Examples

Table 84. Example Names for the EKG_DeleteNotifySubscription Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11413</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51413</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31413</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61413</td>
</tr>
</tbody>
</table>

Summary

Table 85. Summary of the EKG_DeleteNotifySubscription Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>Type</th>
<th>User API</th>
<th>Object-specific method</th>
<th>Object-independent method</th>
<th>Initialization method</th>
<th>Methods triggered</th>
<th>Triggered by the EKG_MessageTriggeredAction function</th>
<th>Authorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1413</td>
<td>Action</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>3</td>
</tr>
</tbody>
</table>

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 377).

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 86. Function Block for the EKG_DeleteObject Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Entity_access_info_ptr</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>SelfDefiningDataPtr</td>
<td>In</td>
<td>Method parms</td>
</tr>
</tbody>
</table>

See “Function Parameter Descriptions” on page 447 for more information about the
parameters listed. See “Abstract Data Type Reference” on page 225 for more
information about the abstract data types listed.

Examples

Table 87. Example Names for the EKG_DeleteObject Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11410</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51410</td>
</tr>
</tbody>
</table>
### EKG_DeleteObject

*Table 87. Example Names for the EKG_DeleteObject Function (continued)*

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>C function block</td>
<td>EKG31410</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61410</td>
</tr>
</tbody>
</table>

### Summary

*Table 88. Summary of the EKG_DeleteObject Function*

<table>
<thead>
<tr>
<th>Function ID</th>
<th>1410</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Action</td>
</tr>
<tr>
<td>User API</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>No</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>Yes</td>
</tr>
<tr>
<td>Initialization method</td>
<td>Yes</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>Notification methods on MyClassChildren and WhatIAm fields of object class triggered</td>
</tr>
<tr>
<td>Triggered by the</td>
<td>Yes</td>
</tr>
<tr>
<td>EKG_MessageTriggeredAction function</td>
<td></td>
</tr>
<tr>
<td>Authorization</td>
<td>5 (delete method object) 3 (delete other object)</td>
</tr>
</tbody>
</table>

### 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 89. Function Block for the EKG_DeleteSubfield Function*

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Class_access_info_ptr</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Field_access_info_ptr</td>
</tr>
<tr>
<td>012</td>
<td>4</td>
<td>Bit(32)</td>
<td>In</td>
<td>Subfield_map</td>
</tr>
</tbody>
</table>
See “Function Parameter Descriptions” on page 447 for more information about the parameters listed. See “Abstract Data Type Reference” on page 225 for more information about the abstract data types listed.

Examples

Table 90. Example Names for the EKG_DeleteSubfield Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11307</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51307</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31307</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61307</td>
</tr>
</tbody>
</table>

Summary

Table 91. Summary of the EKG_DeleteSubfield Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>1307</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Administrative</td>
</tr>
<tr>
<td>User API</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>No</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>No</td>
</tr>
<tr>
<td>Initialization method</td>
<td>No</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>No</td>
</tr>
<tr>
<td>Triggered by the EKG_MessageTriggeredAction function</td>
<td>No</td>
</tr>
<tr>
<td>Authorization</td>
<td>5</td>
</tr>
</tbody>
</table>

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 92. Function Block for the EKG_DelObjDelSubs Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Entity_access_info_ptr</td>
</tr>
<tr>
<td>008</td>
<td>8</td>
<td>ApplicationID</td>
<td>In</td>
<td>User_appl_ID</td>
</tr>
<tr>
<td>016</td>
<td>8</td>
<td>SubscribeID</td>
<td>In</td>
<td>Notification_queue</td>
</tr>
<tr>
<td>024</td>
<td>8</td>
<td>Anonymous(8)</td>
<td>In</td>
<td>User_word</td>
</tr>
<tr>
<td>032</td>
<td>4</td>
<td>SelfDefiningDataPtr</td>
<td>In</td>
<td>Long_lived_parm</td>
</tr>
</tbody>
</table>

See [Function Parameter Descriptions on page 447](#) for more information about the parameters listed. See [Abstract Data Type Reference on page 225](#) for more information about the abstract data types listed.

Examples

Table 93. Example Names for the EKG_DelObjDelSubs Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11418</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51418</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31418</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61418</td>
</tr>
</tbody>
</table>

Summary

Table 94. Summary of the EKG_DelObjDelSubs Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>1418</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Action</td>
</tr>
<tr>
<td>User API</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>Yes</td>
</tr>
<tr>
<td>Initialization method</td>
<td>No</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>No</td>
</tr>
<tr>
<td>Triggered by the EKG_MessageTriggeredAction function</td>
<td>No</td>
</tr>
<tr>
<td>Authorization</td>
<td>3</td>
</tr>
</tbody>
</table>

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 447).

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 95. Function Block for the EKG_Disconnect Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
</tbody>
</table>

See “Function Parameter Descriptions” on page 447 for more information about the parameters listed. See “Abstract Data Type Reference” on page 225 for more information about the abstract data types listed.

Examples

Table 96. Example Names for the EKG_Disconnect Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11102</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51102</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31102</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61102</td>
</tr>
</tbody>
</table>

Summary

Table 97. Summary for the EKG_Disconnect Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>1102</th>
</tr>
</thead>
</table>
Table 97. Summary for the EKG_Disconnect Function (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>User API</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>No</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>No</td>
</tr>
<tr>
<td>Initialization method</td>
<td>No</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>Notification</td>
</tr>
<tr>
<td>Triggered by the</td>
<td>No</td>
</tr>
<tr>
<td>EKG_MessageTriggeredAction</td>
<td></td>
</tr>
<tr>
<td>function</td>
<td></td>
</tr>
<tr>
<td>Authorization</td>
<td>1</td>
</tr>
</tbody>
</table>

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 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 203.

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 executes 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 98. Function Block for the EKG_ExecuteFunctionList Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Number_of_Functions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>008</td>
<td>0</td>
<td>Structure</td>
<td>—</td>
<td>Function_info_array</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Function_block_ptr</td>
</tr>
<tr>
<td>012</td>
<td>4</td>
<td>Pointer</td>
<td>Out</td>
<td>Response_block_reference</td>
</tr>
<tr>
<td>016</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Response_block_used</td>
</tr>
</tbody>
</table>
Table 98. Function Block for the EKG_ExecuteFunctionList Function (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>020</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Return_code</td>
</tr>
<tr>
<td>024</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Reason_code</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Second element, array of structure (if used)</td>
</tr>
<tr>
<td>028</td>
<td>0</td>
<td>Structure</td>
<td>—</td>
<td>Function_info_array</td>
</tr>
<tr>
<td>028</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Function_block_ptr</td>
</tr>
<tr>
<td>032</td>
<td>4</td>
<td>Pointer</td>
<td>Out</td>
<td>Response_block_reference</td>
</tr>
<tr>
<td>036</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Response_block_used</td>
</tr>
<tr>
<td>040</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Return_code</td>
</tr>
<tr>
<td>044</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Reason_code</td>
</tr>
</tbody>
</table>

Note: Function block contains Number_of_functions array elements

Table 99. Response Block for the EKG_FunctionList Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Response_block_length</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Response_block_used</td>
</tr>
<tr>
<td>008</td>
<td>—</td>
<td>—</td>
<td>Out</td>
<td>Response_data</td>
</tr>
</tbody>
</table>

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 447 for more information about the parameters listed. See “Abstract Data Type Reference” on page 225 for more information about the abstract data types listed.

Examples

Table 100. Example Names for the EKG_ExecuteFunctionList Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11600</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51600</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31600</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61600</td>
</tr>
</tbody>
</table>

Summary

Table 101. Summary of the EKG_ExecuteFunctionList Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>1600</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>User API Service</td>
</tr>
<tr>
<td>User API</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>No</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>No</td>
</tr>
</tbody>
</table>
### EKG_ExecuteFunctionList

**Table 101. Summary of the EKG_ExecuteFunctionList Function (continued)**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization method</td>
<td>No</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>Yes</td>
</tr>
<tr>
<td>Triggered by EKG_MessageTriggeredAction function</td>
<td>No</td>
</tr>
<tr>
<td>Authorization</td>
<td>2 (list of queries only) 3 (list includes actions)</td>
</tr>
</tbody>
</table>

**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 102. Function Block for EKG_LinkNoTrigger Function and the EKG_LinkTrigger Function**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Entity_access_info_ptr_1</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Field_access_info_ptr_1</td>
</tr>
<tr>
<td>012</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Entity_access_info_ptr_2</td>
</tr>
<tr>
<td>016</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Field_access_info_ptr_2</td>
</tr>
<tr>
<td>020</td>
<td>4</td>
<td>SelfDefiningDataPtr</td>
<td>In</td>
<td>Method_parms¹</td>
</tr>
</tbody>
</table>

:
See “Function Parameter Descriptions” on page 447 for more information about the parameters listed. See “Abstract Data Type Reference” on page 225 for more information about the abstract data types listed.

Examples

Table 103. Example Names for the EKG_LinkNoTrigger Function and the EKG_LinkTrigger Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block (EKG_LinkTrigger)</td>
<td>EKG11405</td>
</tr>
<tr>
<td>PL/I function block (EKG_LinkNoTrigger)</td>
<td>EKG11406</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding (EKG_LinkTrigger)</td>
<td>EKG51405</td>
</tr>
<tr>
<td>PL/I usage coding (EKG_LinkNoTrigger)</td>
<td>EKG51406</td>
</tr>
<tr>
<td>C function block (EKG_LinkTrigger)</td>
<td>EKG31405</td>
</tr>
<tr>
<td>C function block (EKG_LinkNoTrigger)</td>
<td>EKG31406</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding (EKG_LinkTrigger)</td>
<td>EKG61405</td>
</tr>
<tr>
<td>C usage coding (EKG_LinkNoTrigger)</td>
<td>EKG61406</td>
</tr>
</tbody>
</table>

Summary

Table 104. Summary of the EKG_LinkNoTrigger Function and the EKG_LinkTrigger Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>EKG_LinkNoTrigger</td>
<td>1406</td>
</tr>
<tr>
<td>EKG_LinkTrigger</td>
<td>1405</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>User API</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>No</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>Yes</td>
</tr>
<tr>
<td>Initialization method</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods triggered EKG_LinkTrigger</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>EKG_LinkNoTrigger</td>
<td>Change methods and notification methods</td>
</tr>
<tr>
<td>EKG_LinkTrigger</td>
<td>No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Triggered by the</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>EKG_MessageTriggeredAction function</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Authorization</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

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.
EKG_LinkNoTrigger, EKG_LinkTrigger

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 222 for a description of using public indexed fields and the EKG_Locate function.

Function Block Format

**Table 105. Function Block for the EKG_Locate Function**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Anonymous(4)</td>
<td>—</td>
<td>Reserved, must be X'00000000'</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Field_access_info_ptr</td>
</tr>
<tr>
<td>012</td>
<td>2</td>
<td>Smallint</td>
<td>In</td>
<td>Data_type, must be 4 or 32</td>
</tr>
<tr>
<td>014</td>
<td>2</td>
<td>Anonymous(2)</td>
<td>—</td>
<td>Reserved, must be X'0000'</td>
</tr>
<tr>
<td>016</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Indexed_data_length</td>
</tr>
<tr>
<td>020</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Indexed_data_ptr</td>
</tr>
</tbody>
</table>

**Table 106. Response Block for the EKG_Locate Function**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Response_block_length</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Response_block_used</td>
</tr>
<tr>
<td>008</td>
<td>—</td>
<td>ObjectIdList</td>
<td>Out</td>
<td>Requested_data</td>
</tr>
</tbody>
</table>
See "Function Parameter Descriptions" on page 447 for more information about the parameters listed. See "Abstract Data Type Reference" on page 225 for more information about the abstract data types listed.

**Examples**

*Table 107. Example Names for the EKG_Locate Function*

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11509</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>EKG21509</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51509</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31509</td>
</tr>
<tr>
<td>C response block</td>
<td>EKG41509</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61509</td>
</tr>
</tbody>
</table>

**Summary**

*Table 108. Summary of the EKG_Locate Function*

<table>
<thead>
<tr>
<th>Function ID</th>
<th>1509</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Query</td>
</tr>
<tr>
<td>User API</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>Yes</td>
</tr>
<tr>
<td>Initialization method</td>
<td>Yes</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>No</td>
</tr>
<tr>
<td>Triggered by EKG_MessageTriggeredAction function</td>
<td>No</td>
</tr>
<tr>
<td>Authorization</td>
<td>2</td>
</tr>
</tbody>
</table>

**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.
EKG_LockObjectList

Function Block Format

Table 109. Function Block for the EKG_LockObjectList Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Object_list_length</td>
</tr>
<tr>
<td>:</td>
<td></td>
<td></td>
<td></td>
<td>First element, array of structure</td>
</tr>
<tr>
<td>008</td>
<td>0</td>
<td>Structure</td>
<td>—</td>
<td>Object_array</td>
</tr>
<tr>
<td>008</td>
<td>8</td>
<td>ObjectID</td>
<td>In</td>
<td>Object_ID</td>
</tr>
<tr>
<td>016</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Reason_code¹</td>
</tr>
<tr>
<td>:</td>
<td></td>
<td></td>
<td></td>
<td>Second element, array of structure (if used)</td>
</tr>
<tr>
<td>020</td>
<td>0</td>
<td>Structure</td>
<td>—</td>
<td>Object_array</td>
</tr>
<tr>
<td>020</td>
<td>8</td>
<td>ObjectID</td>
<td>In</td>
<td>Object_ID</td>
</tr>
<tr>
<td>028</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Reason_code¹</td>
</tr>
</tbody>
</table>

Note: Function block contains Object_list_length array elements

See “Function Parameter Descriptions” on page 447 for more information about the parameters listed. See “Abstract Data Type Reference” on page 225 for more information about the abstract data types listed.

Examples

Table 110. Example Names for the EKG_LockObjectList Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG12002</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG52002</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG32002</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG62002</td>
</tr>
</tbody>
</table>

Summary

Table 111. Summary of the EKG_LockObjectList Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>2002</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Method API Service</td>
</tr>
<tr>
<td>User API</td>
<td>No</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>No</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>Yes</td>
</tr>
<tr>
<td>Initialization method</td>
<td>Yes</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>No</td>
</tr>
<tr>
<td>Triggered by EKG_MessageTriggeredAction function</td>
<td>No</td>
</tr>
<tr>
<td>Authorization</td>
<td>None</td>
</tr>
</tbody>
</table>

408  Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer's Guide
**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 executes a RODM function asynchronously. It enables an object-specific method to act on other objects in the data cache.

**Function Block Format**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Function_block_ptr</td>
</tr>
</tbody>
</table>

See "Function Parameter Descriptions" on page 447 for more information about the parameters listed. See "Abstract Data Type Reference" on page 225 for more information about the abstract data types listed.

**Examples**

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG12009</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG52009</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG32009</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG62009</td>
</tr>
</tbody>
</table>

**Summary**

<table>
<thead>
<tr>
<th>Function ID</th>
<th>2009</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Method API Service</td>
</tr>
<tr>
<td>User API</td>
<td>No</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>Yes</td>
</tr>
<tr>
<td>Initialization method</td>
<td>Yes</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>No</td>
</tr>
<tr>
<td>Triggered by EKG_MessageTriggeredAction function</td>
<td>No</td>
</tr>
<tr>
<td>Authorization</td>
<td>None</td>
</tr>
</tbody>
</table>
EKG_MessageTriggeredAction

Usage
Not all functions can be executed 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 executed 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 executed 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 328 for more information.

Functions executed 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 115. Function Block for the EKG_OutputToLog Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Log_message</td>
</tr>
<tr>
<td>008</td>
<td>2</td>
<td>Smallint</td>
<td>In</td>
<td>Message_CCSID</td>
</tr>
</tbody>
</table>

See "Function Parameter Descriptions" on page 447 for more information about the parameters listed. See "Abstract Data Type Reference" on page 225 for more information about the abstract data types listed.

Examples

Table 116. Example Names for the EKG_OutputToLog Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG12008</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG52008</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG32008</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG62008</td>
</tr>
</tbody>
</table>
Summary

Table 117. Summary of the EKG_OutputToLog Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>2008</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Method API Service</td>
</tr>
<tr>
<td>User API</td>
<td>No</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>Yes</td>
</tr>
<tr>
<td>Initialization method</td>
<td>Yes</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>No</td>
</tr>
<tr>
<td>Triggered by EKG_MessageTriggeredAction function</td>
<td>No</td>
</tr>
<tr>
<td>Authorization</td>
<td>None</td>
</tr>
</tbody>
</table>

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 118. Function Block for the EKG_QueryEntityStructure Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Entity_access_info_ptr</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>—</td>
<td>—</td>
<td>Not used</td>
</tr>
<tr>
<td>012</td>
<td>2</td>
<td>—</td>
<td>—</td>
<td>Not used</td>
</tr>
<tr>
<td>014</td>
<td>2</td>
<td>Anonymous(2)</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>016</td>
<td>4</td>
<td>—</td>
<td>—</td>
<td>Not used</td>
</tr>
</tbody>
</table>

Table 119. Response Block for the EKG_QueryEntityStructure Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Response_block_length</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Response_block_used</td>
</tr>
<tr>
<td>008</td>
<td>2</td>
<td>Smallint</td>
<td>Out</td>
<td>Field_info_element_size</td>
</tr>
<tr>
<td>010</td>
<td>2</td>
<td>Smallint</td>
<td>Out</td>
<td>Field_info_count</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Structure</td>
<td>—</td>
<td>Field_info_array</td>
</tr>
</tbody>
</table>
### EKG_QueryEntityStructure

**Table 119. Response Block for the EKG_QueryEntityStructure Function (continued)**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>012</td>
<td>4</td>
<td>FieldID</td>
<td>Out</td>
<td>Field_ID</td>
</tr>
<tr>
<td>016</td>
<td>2</td>
<td>Bit(16)</td>
<td>—</td>
<td>Bit_map</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Private_public_flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Local_inherited_flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Indexed_flag</td>
</tr>
<tr>
<td>018</td>
<td>2</td>
<td>Smallint</td>
<td>Out</td>
<td>Data_type</td>
</tr>
<tr>
<td>020</td>
<td>67</td>
<td>ShortName</td>
<td>Out</td>
<td>Field_name</td>
</tr>
<tr>
<td>087</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

: Second element, array of structure (if used)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>088</td>
<td>0</td>
<td>Structure</td>
<td>—</td>
<td>Field_info_array</td>
</tr>
<tr>
<td>088</td>
<td>4</td>
<td>FieldID</td>
<td>Out</td>
<td>Field_ID</td>
</tr>
<tr>
<td>092</td>
<td>2</td>
<td>Bit(16)</td>
<td>—</td>
<td>Bit_map</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Private_public_flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Local_inherited_flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Indexed_flag</td>
</tr>
<tr>
<td>094</td>
<td>2</td>
<td>Smallint</td>
<td>Out</td>
<td>Data_type</td>
</tr>
<tr>
<td>096</td>
<td>67</td>
<td>ShortName</td>
<td>Out</td>
<td>Field_name</td>
</tr>
<tr>
<td>161</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**Note:** Function block contains Field_info_count array elements

See [“Function Parameter Descriptions” on page 447](#) for more information about the parameters listed. See [“Abstract Data Type Reference” on page 225](#) for more information about the abstract data types listed.

### Examples

**Table 120. Example Names for the EKG_QueryEntityStructure Function**

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11503</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>EKG21503</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51503</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31503</td>
</tr>
<tr>
<td>C response block</td>
<td>EKG41503</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61503</td>
</tr>
</tbody>
</table>

### Summary

**Table 121. Summary of the EKG_QueryEntityStructure Function**

<table>
<thead>
<tr>
<th>Function ID</th>
<th>Type</th>
<th>User API</th>
<th>Object-specific method</th>
<th>Object-independent method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1503</td>
<td>Query</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Table 121. Summary of the EKG_QueryEntityStructure Function (continued)

<table>
<thead>
<tr>
<th>Initialization method</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Methods triggered</td>
<td>No</td>
</tr>
<tr>
<td>Triggered by EKG_MessageTriggeredAction function</td>
<td>No</td>
</tr>
<tr>
<td>Authorization</td>
<td>2</td>
</tr>
</tbody>
</table>

**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 122. Function Block for the EKG_QueryField Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Entity_access_info_ptr</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Field_access_info_ptr</td>
</tr>
<tr>
<td>012</td>
<td>2</td>
<td>—</td>
<td>—</td>
<td>Not used</td>
</tr>
<tr>
<td>014</td>
<td>2</td>
<td>Anonymous(2)</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>016</td>
<td>4</td>
<td>SelfDefiningDataPtr</td>
<td>In</td>
<td>Method_parms</td>
</tr>
</tbody>
</table>

Table 123. Response Block for the EKG_QueryField Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Response_block_length</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Response_block_used</td>
</tr>
<tr>
<td>008</td>
<td>2</td>
<td>Smallint</td>
<td>Out</td>
<td>Data_type</td>
</tr>
<tr>
<td>010</td>
<td>—</td>
<td>Anonymous</td>
<td>Out</td>
<td>Data</td>
</tr>
</tbody>
</table>

See [“Function Parameter Descriptions” on page 447](#) for more information about the parameters listed. See [“Abstract Data Type Reference” on page 225](#) for more information about the abstract data types listed.

**Examples**

Table 124. Example Names for the EKG_QueryField Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11501</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>EKG21501</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51501</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31501</td>
</tr>
<tr>
<td>C response block</td>
<td>EKG41501</td>
</tr>
</tbody>
</table>
Table 124. Example Names for the EKG_QueryField Function (continued)

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>C usage coding</td>
<td>EKG61501</td>
</tr>
</tbody>
</table>

Summary

Table 125. Summary of the EKG_QueryField Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>Type</th>
<th>User API</th>
<th>Object-specific method</th>
<th>Object-independent method</th>
<th>Initialization method</th>
<th>Methods triggered</th>
<th>Triggered by EKG_MessageTriggeredAction function</th>
<th>Authorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1501</td>
<td>Query</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Query method for the target field triggered</td>
<td>No</td>
<td>2</td>
</tr>
</tbody>
</table>

Usage

If there is a query method on the field, the Method_parm field is passed to that query method when the method is invoked. 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 126. Function Block for the EKG_QueryFieldID Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>—</td>
<td>—</td>
<td>Not used</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Field_access_info_ptr</td>
</tr>
<tr>
<td>012</td>
<td>2</td>
<td>—</td>
<td>—</td>
<td>Not used</td>
</tr>
<tr>
<td>014</td>
<td>2</td>
<td>Anonymous(2)</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>016</td>
<td>4</td>
<td>—</td>
<td>—</td>
<td>Not used</td>
</tr>
</tbody>
</table>
### Table 127. Response Block for the EKG_QueryFieldID Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Response_block_length</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Response_block_used</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>FieldID</td>
<td>Out</td>
<td>Field_ID</td>
</tr>
</tbody>
</table>

See “Function Parameter Descriptions” on page 447 for more information about the parameters listed. See “Abstract Data Type Reference” on page 225 for more information about the abstract data types listed.

### Examples

**Table 128. Example Names for the EKG_QueryFieldID Function**

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11505</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>EKG21505</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51505</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31505</td>
</tr>
<tr>
<td>C response block</td>
<td>EKG41505</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61505</td>
</tr>
</tbody>
</table>

### Summary

**Table 129. Summary of the EKG_QueryFieldID Function**

<table>
<thead>
<tr>
<th>Function ID</th>
<th>1505</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Query</td>
</tr>
<tr>
<td>User API</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>Yes</td>
</tr>
<tr>
<td>Initialization method</td>
<td>Yes</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>No</td>
</tr>
<tr>
<td>Triggered by EKG_MessageTriggeredAction function</td>
<td>No</td>
</tr>
<tr>
<td>Authorization</td>
<td>2</td>
</tr>
</tbody>
</table>

### 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

EKG_QueryFieldName — Query a Field Name

Purpose
This function returns a field name from a specified field ID.

Function Block Format

Table 130. Function Block for the EKG_QueryFieldName Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Entity_access_info_ptr</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Field_access_info_ptr</td>
</tr>
<tr>
<td>012</td>
<td>2</td>
<td>—</td>
<td>—</td>
<td>Not used</td>
</tr>
<tr>
<td>014</td>
<td>2</td>
<td>Anonymous(2)</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>016</td>
<td>4</td>
<td>—</td>
<td>—</td>
<td>Not used</td>
</tr>
</tbody>
</table>

Table 131. Response Block for the EKG_QueryFieldName Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Response_block_length</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Response_block_used</td>
</tr>
<tr>
<td>008</td>
<td>67</td>
<td>ShortName</td>
<td>Out</td>
<td>Field_name</td>
</tr>
</tbody>
</table>

See “Function Parameter Descriptions” on page 447 for more information about the parameters listed. See “Abstract Data Type Reference” on page 225 for more information about the abstract data types listed.

Examples

Table 132. Example Names for the EKG_QueryFieldName Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11506</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>EKG21506</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51506</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31506</td>
</tr>
<tr>
<td>C response block</td>
<td>EKG41506</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61506</td>
</tr>
</tbody>
</table>

Summary

Table 133. Summary of the EKG_QueryFieldName Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>Type</th>
<th>User API</th>
<th>Object-specific method</th>
<th>Object-independent method</th>
<th>Initialization method</th>
<th>Methods triggered</th>
</tr>
</thead>
<tbody>
<tr>
<td>1506</td>
<td>Query</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
Table 133. Summary of the EKG_QueryFieldName Function (continued)

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 134. Function Block for the EKG_QueryFieldStructure Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Entity_access_info_ptr</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Field_access_info_ptr</td>
</tr>
<tr>
<td>012</td>
<td>2</td>
<td>—</td>
<td>—</td>
<td>Not used</td>
</tr>
<tr>
<td>014</td>
<td>2</td>
<td>Anonymous(2)</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>016</td>
<td>4</td>
<td>—</td>
<td>—</td>
<td>Not used</td>
</tr>
</tbody>
</table>

Table 135. Response Block for the EKG_QueryFieldStructure Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Response_block_length</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Response_block_used</td>
</tr>
<tr>
<td>008</td>
<td>2</td>
<td>Smallint</td>
<td>Out</td>
<td>Data_type</td>
</tr>
<tr>
<td>010</td>
<td>2</td>
<td>Smallint</td>
<td>Out</td>
<td>Inheritance_state</td>
</tr>
<tr>
<td>012</td>
<td>4</td>
<td>Bit(32)</td>
<td>Out</td>
<td>Subfield_map</td>
</tr>
<tr>
<td>016</td>
<td>4</td>
<td>Bit(32)</td>
<td>Out</td>
<td>Local_copy_map</td>
</tr>
</tbody>
</table>
EKG_QueryFieldStructure

See “Function Parameter Descriptions” on page 447 for more information about the parameters listed. See “Abstract Data Type Reference” on page 225 for more information about the abstract data types listed.

Examples

Table 136. Example Names for the EKG_QueryFieldStructure Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11504</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>EKG21504</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51504</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31504</td>
</tr>
<tr>
<td>C response block</td>
<td>EKG41504</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61504</td>
</tr>
</tbody>
</table>

Summary

Table 137. Summary of the EKG_QueryFieldStructure Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>1504</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Query</td>
</tr>
<tr>
<td>User API</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>Yes</td>
</tr>
<tr>
<td>Initialization method</td>
<td>Yes</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>No</td>
</tr>
<tr>
<td>Triggered by EKG_MessageTriggeredAction function</td>
<td>No</td>
</tr>
<tr>
<td>Authorization</td>
<td>2</td>
</tr>
</tbody>
</table>

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 138. Function Block for the EKG_QueryFunctionBlockContents Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
</tbody>
</table>

Table 139. Response Block for the EKG_QueryFunctionBlockContents Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Response_block_length</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Response_block_used</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Function_block_origin</td>
</tr>
<tr>
<td>012</td>
<td></td>
<td></td>
<td></td>
<td>Function_block_copy</td>
</tr>
</tbody>
</table>

See “Function Parameter Descriptions” on page 447 for more information about the parameters listed. See “Abstract Data Type Reference” on page 225 for more information about the abstract data types listed.

Examples

Table 140. Example Names for the EKG_QueryFunctionBlockContents Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG12001</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>EKG22001</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG52001</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG32001</td>
</tr>
<tr>
<td>C response block</td>
<td>EKG42001</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG62001</td>
</tr>
</tbody>
</table>

Summary

Table 141. Summary of the EKG_QueryFunctionBlockContents Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>2001</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Method API Service</td>
</tr>
<tr>
<td>User API</td>
<td>No</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>Yes</td>
</tr>
<tr>
<td>Initialization method</td>
<td>No</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>No</td>
</tr>
<tr>
<td>Triggered by EKG_MessageTriggeredAction function</td>
<td>No</td>
</tr>
</tbody>
</table>
Table 141. Summary of the EKG_QueryFunctionBlockContents Function (continued)

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 142. Function Block for the EKG_QueryMultipleSubfields Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Entity_access_info_ptr</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Number_of_subfields</td>
</tr>
</tbody>
</table>

**Note:** First element, array of structure

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>012</td>
<td>0</td>
<td>Structure</td>
<td></td>
<td>Field_info_array</td>
</tr>
<tr>
<td>012</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Field_access_info_ptr</td>
</tr>
<tr>
<td>016</td>
<td>4</td>
<td>Anonymous(4)</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>020</td>
<td>4</td>
<td>Pointer</td>
<td>Out</td>
<td>Response_block_reference</td>
</tr>
<tr>
<td>024</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Response_block_used</td>
</tr>
<tr>
<td>028</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Return_code</td>
</tr>
<tr>
<td>032</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Reason_code</td>
</tr>
</tbody>
</table>

**Note:** Second element, array of structure (if used)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>036</td>
<td>0</td>
<td>Structure</td>
<td></td>
<td>Field_info_array</td>
</tr>
<tr>
<td>036</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Field_access_info_ptr</td>
</tr>
<tr>
<td>040</td>
<td>4</td>
<td>Anonymous(4)</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>044</td>
<td>4</td>
<td>Pointer</td>
<td>Out</td>
<td>Response_block_reference</td>
</tr>
<tr>
<td>048</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Response_block_used</td>
</tr>
<tr>
<td>052</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Return_code</td>
</tr>
<tr>
<td>056</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Reason_code</td>
</tr>
</tbody>
</table>

**Note:** Function block contains Number_of_subfields array elements

Table 143. Response Block for the EKG_QueryMultipleSubfields Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Response_block_length</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Response_block_used</td>
</tr>
<tr>
<td>008</td>
<td>0</td>
<td>Anonymous(1)</td>
<td>Out</td>
<td>Requested_info_array</td>
</tr>
</tbody>
</table>

**Note:** First and subsequent elements, array of requested information

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>008</td>
<td>2</td>
<td>Smallint</td>
<td></td>
<td>Data_type</td>
</tr>
<tr>
<td>010</td>
<td>—</td>
<td>Anonymous</td>
<td></td>
<td>Data_value</td>
</tr>
</tbody>
</table>
Table 143. Response Block for the EKG_QueryMultipleSubfields Function (continued)

Offset | Length | Type | Use | Parameter Name
--- | --- | --- | --- | ---
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 447 for more information about the parameters listed. See “Abstract Data Type Reference” on page 225 for more information about the abstract data types listed.

Examples

Table 144. Example Names for the EKG_QueryMultipleSubfields Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11508</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>EKG21508</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51508</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31508</td>
</tr>
<tr>
<td>C response block</td>
<td>EKG41508</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61508</td>
</tr>
</tbody>
</table>

Summary

Table 145. Summary of the EKG_QueryMultipleSubfields Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>1508</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Method API Service</td>
</tr>
<tr>
<td>User API</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>Yes</td>
</tr>
<tr>
<td>Initialization method</td>
<td>Yes</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>No</td>
</tr>
<tr>
<td>Triggered by EKG_MessageTriggeredAction function</td>
<td>No</td>
</tr>
<tr>
<td>Authorization</td>
<td>2</td>
</tr>
</tbody>
</table>

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).
EKG_QueryMultipleSubfields

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>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>8</td>
<td>SubscribeID</td>
<td>In</td>
<td>Notification_queue</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Response_block_length</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Response_block_used</td>
</tr>
<tr>
<td>008</td>
<td>2</td>
<td>Smallint</td>
<td>Out</td>
<td>Notification_queue_count</td>
</tr>
<tr>
<td>010</td>
<td>2</td>
<td>Smallint</td>
<td>Out</td>
<td>Response_block_type</td>
</tr>
<tr>
<td>012</td>
<td>4</td>
<td>ClassID</td>
<td>Out</td>
<td>Class_ID</td>
</tr>
<tr>
<td>016</td>
<td>8</td>
<td>ObjectID</td>
<td>Out</td>
<td>Object_ID</td>
</tr>
<tr>
<td>024</td>
<td>4</td>
<td>FieldID</td>
<td>Out</td>
<td>Field_ID</td>
</tr>
<tr>
<td>028</td>
<td>2</td>
<td>Smallint</td>
<td>Out</td>
<td>Subfield</td>
</tr>
<tr>
<td>030</td>
<td>8</td>
<td>ApplicationID</td>
<td>Out</td>
<td>User_appl_ID</td>
</tr>
<tr>
<td>038</td>
<td>8</td>
<td>SubscribeID</td>
<td>Out</td>
<td>Notification_queue</td>
</tr>
<tr>
<td>046</td>
<td>8</td>
<td>MethodName</td>
<td>Out</td>
<td>Method_name</td>
</tr>
<tr>
<td>054</td>
<td>8</td>
<td>Anonymous(8)</td>
<td>Out</td>
<td>User_word</td>
</tr>
<tr>
<td>062</td>
<td>—</td>
<td>SelfDefining</td>
<td>Out</td>
<td>User_area</td>
</tr>
</tbody>
</table>
EKG_QueryNotifyQueue

See "Function Parameter Descriptions" on page 447 for more information about the parameters listed. See "Abstract Data Type Reference" on page 225 for more information about the abstract data types listed.

Examples

Table 148. Example Names for the EKG_QueryNotifyQueue Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11507</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>EKG21507</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51507</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31507</td>
</tr>
<tr>
<td>C response block</td>
<td>EKG41507</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61507</td>
</tr>
</tbody>
</table>

Summary

Table 149. Summary of the EKG_QueryNotifyQueue Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>1507</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Query</td>
</tr>
<tr>
<td>User API</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>No</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>No</td>
</tr>
<tr>
<td>Initialization method</td>
<td>No</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>No</td>
</tr>
<tr>
<td>Triggered by EKG_MessageTriggeredAction function</td>
<td>No</td>
</tr>
<tr>
<td>Authorization</td>
<td>2</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>8</td>
<td>ObjectID</td>
<td>In</td>
<td>Object_ID</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Response_block_length</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Response_block_used</td>
</tr>
<tr>
<td>008</td>
<td>67</td>
<td>ShortName</td>
<td>Out</td>
<td>Class_name</td>
</tr>
<tr>
<td>075</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>076</td>
<td>—</td>
<td>ObjectName</td>
<td>Out</td>
<td>Object_name</td>
</tr>
</tbody>
</table>

See “Function Parameter Descriptions” on page 447 for more information about the parameters listed. See “Abstract Data Type Reference” on page 223 for more information about the abstract data types listed.
**EKG_QueryObjectName**

**Examples**

*Table 152. Example Names for the EKG_QueryObjectName Function*

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG12011</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>EKG22011</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG52011</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG32011</td>
</tr>
<tr>
<td>C response block</td>
<td>EKG42011</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG62011</td>
</tr>
</tbody>
</table>

**Summary**

*Table 153. Summary of the EKG_QueryObjectName Function*

<table>
<thead>
<tr>
<th>Function ID</th>
<th>2011</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Method API Service</td>
</tr>
<tr>
<td>User API</td>
<td>No</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>No</td>
</tr>
<tr>
<td>Initialization method</td>
<td>No</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>No</td>
</tr>
<tr>
<td>Triggered by EKG_MessageTriggeredAction function</td>
<td>No</td>
</tr>
<tr>
<td>Authorization</td>
<td>None</td>
</tr>
</tbody>
</table>

**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 154. Function Block for the EKG_QueryResponseBlockOverflow Function*

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Anonymous</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>008</td>
<td>8</td>
<td>TransID</td>
<td>In</td>
<td>Correlation_ID</td>
</tr>
</tbody>
</table>
Table 155. Response Block for the EKG_QueryResponseBlockOverflow Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Response_block_length</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Response_block_used</td>
</tr>
<tr>
<td>008</td>
<td>—</td>
<td>Anonymous</td>
<td>Out</td>
<td>Data</td>
</tr>
</tbody>
</table>

See "Function Parameter Descriptions" on page 447 for more information about the parameters listed. See "Abstract Data Type Reference" on page 225 for more information about the abstract data types listed.

Examples

Table 156. Example Names for the EKG_QueryResponseBlockOverflow Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11510</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>EKG21510</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51510</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31510</td>
</tr>
<tr>
<td>C response block</td>
<td>EKG41510</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61510</td>
</tr>
</tbody>
</table>

Summary

Table 157. Summary of the EKG_QueryResponseBlockOverflow Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>1510</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Query</td>
</tr>
<tr>
<td>User API</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>No</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>No</td>
</tr>
<tr>
<td>Initialization method</td>
<td>No</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>No</td>
</tr>
<tr>
<td>Triggered by EKG_MessageTriggeredAction function</td>
<td>No</td>
</tr>
<tr>
<td>Authorization</td>
<td>2</td>
</tr>
</tbody>
</table>

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.
EKG_QueryResponseBlockOverflow

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 316 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 158. Function Block for EKG_QuerySubfield Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Entity_access_info_ptr</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Field_access_info_ptr</td>
</tr>
<tr>
<td>012</td>
<td>2</td>
<td>Smallint</td>
<td>In</td>
<td>Subfield</td>
</tr>
<tr>
<td>014</td>
<td>2</td>
<td>Anonymous(2)</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>016</td>
<td>4</td>
<td>—</td>
<td>—</td>
<td>Not used</td>
</tr>
</tbody>
</table>

Table 159. Response Block for the EKG_QuerySubfield Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Response_block_length</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Response_block_used</td>
</tr>
<tr>
<td>008</td>
<td>2</td>
<td>Smallint</td>
<td>Out</td>
<td>Data_type</td>
</tr>
<tr>
<td>010</td>
<td>—</td>
<td>Anonymous</td>
<td>Out</td>
<td>Data</td>
</tr>
</tbody>
</table>
See “Function Parameter Descriptions” on page 447 for more information about the parameters listed. See “Abstract Data Type Reference” on page 225 for more information about the abstract data types listed.

Examples

Table 160. Example Names for the EKG_QuerySubfield Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11502</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>EKG21502</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51502</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31502</td>
</tr>
<tr>
<td>C response block</td>
<td>EKG41502</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61502</td>
</tr>
</tbody>
</table>

Summary

Table 161. Summary of the EKG_QuerySubfield Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>1502</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Query</td>
</tr>
<tr>
<td>User API</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>Yes</td>
</tr>
<tr>
<td>Initialization method</td>
<td>Yes</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>No</td>
</tr>
<tr>
<td>Triggered by EKG_MessageTriggeredAction function</td>
<td>No</td>
</tr>
<tr>
<td>Authorization</td>
<td>2</td>
</tr>
</tbody>
</table>

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'.
EKG_QuerySubfield

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 162. Function Block for the EKG_ResponseBlock Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>SelfDefiningDataPtr</td>
<td>In</td>
<td>Data_to_be_returned</td>
</tr>
</tbody>
</table>

See "Function Parameter Descriptions" on page 447 for more information about the parameters listed. See "Abstract Data Type Reference" on page 225 for more information about the abstract data types listed.

Examples

Table 163. Example Names for the EKG_ResponseBlock Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG12004</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG52004</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG32004</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG62004</td>
</tr>
</tbody>
</table>

Summary

Table 164. Summary of the EKG_ResponseBlock Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>Method API Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>2004</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Query and named only</td>
</tr>
<tr>
<td>User API</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>No</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>No</td>
</tr>
<tr>
<td>Initialization method</td>
<td>No</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>No</td>
</tr>
<tr>
<td>Triggered by EKG_MessageTriggeredAction function</td>
<td>No</td>
</tr>
</tbody>
</table>
Usage
Each time an object-independent or named method invokes this function, a new SelfDefining data string is appended to the current response block. Each time a query method invokes 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 invoked.

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>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Entity_access_info_ptr</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Field_access_info_ptr</td>
</tr>
<tr>
<td>012</td>
<td>2</td>
<td>Smallint</td>
<td>In</td>
<td>Subfield</td>
</tr>
</tbody>
</table>

See "Function Parameter Descriptions" on page 447 for more information about the parameters listed. See "Abstract Data Type Reference" on page 225 for more information about the abstract data types listed.
EKG_RevertToInherited

Examples

Table 166. Example Names for the EKG_RevertToInherited Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11411</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51411</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31411</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61411</td>
</tr>
</tbody>
</table>

Summary

Table 167. Summary of the EKG_RevertToInherited Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>1411</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Action</td>
</tr>
<tr>
<td>User API</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>Yes</td>
</tr>
<tr>
<td>Initialization method</td>
<td>Yes</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>No</td>
</tr>
<tr>
<td>Triggered by the</td>
<td>EKG_MessageTriggeredAction function</td>
</tr>
<tr>
<td>Authorization</td>
<td>3</td>
</tr>
</tbody>
</table>

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 could be inherited from the immediate parent class, and the value of the query subfield could 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>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>8</td>
<td>ApplicationID</td>
<td>In</td>
<td>User_appl_ID</td>
</tr>
<tr>
<td>012</td>
<td>8</td>
<td>SubscribeID</td>
<td>In</td>
<td>Notification_queue</td>
</tr>
<tr>
<td>020</td>
<td>8</td>
<td>Anonymous(8)</td>
<td>In</td>
<td>User_word</td>
</tr>
<tr>
<td>028</td>
<td>4</td>
<td>SelfDefiningDataPtr</td>
<td>In</td>
<td>Method_output_message</td>
</tr>
</tbody>
</table>

See “Function Parameter Descriptions” on page 447 for more information about the parameters listed. See “Abstract Data Type Reference” on page 225 for more information about the abstract data types listed.

**Examples**

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG12005</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG52005</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG32005</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG62005</td>
</tr>
</tbody>
</table>

**Summary**

<table>
<thead>
<tr>
<th>Function ID</th>
<th>2005</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Method API Service</td>
</tr>
<tr>
<td>User API</td>
<td>No</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>Yes</td>
</tr>
<tr>
<td>Initialization method</td>
<td>No</td>
</tr>
</tbody>
</table>
Table 170. Summary of the EKG_SendNotification Function (continued)

<table>
<thead>
<tr>
<th>Methods triggered</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triggered by EKG_MessageTriggeredAction function</td>
<td>No</td>
</tr>
<tr>
<td>Authorization</td>
<td>None</td>
</tr>
</tbody>
</table>

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 377, “EKG_DeleteNotifySubscription — Delete Notification Subscription” on page 395, and “EKG_QueryNotifyQueue — Query Notification Queue” on page 423. 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 203 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 171. Function Block for the EKG_SetReturnCode Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Value_for_return_code</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Value_for_reason_code</td>
</tr>
</tbody>
</table>

See “Function Parameter Descriptions” on page 447 for more information about the parameters listed. See “Abstract Data Type Reference” on page 225 for more information about the abstract data types listed.

Examples

Table 172. Example Names for the EKG_SetReturnCode Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG12006</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG52006</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG32006</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG62006</td>
</tr>
</tbody>
</table>
Summary

Table 173. Summary of the EKG_SetReturnCode Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>Type</th>
<th>User API</th>
<th>Object-specific method</th>
<th>Object-independent method</th>
<th>Initialization method</th>
<th>Methods triggered</th>
<th>Triggered by EKG_MessageTriggeredAction function</th>
<th>Authorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>2006</td>
<td>Method API Service</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>None</td>
</tr>
</tbody>
</table>

Usage

The EKG_SetReturnCode function changes the return code of the caller to the value of the Value_for_return_code parameter if the 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

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>12</td>
</tr>
</tbody>
</table>

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 319.

Method writers should be aware of the implications of issuing return and reason codes from methods. See “Error Conditions in Transactions” on page 319 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>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>2</td>
<td>Smallint</td>
<td>In</td>
<td>Stop_type</td>
</tr>
</tbody>
</table>

See “Function Parameter Descriptions” on page 447 for more information about the parameters listed. See “Abstract Data Type Reference” on page 225 for more information about the abstract data types listed.

**Examples**

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11202</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51202</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31202</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61202</td>
</tr>
</tbody>
</table>

**Summary**

<table>
<thead>
<tr>
<th>Function ID</th>
<th>1202</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Control</td>
</tr>
<tr>
<td>User API</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>No</td>
</tr>
</tbody>
</table>
Table 176. Summary of the EKG_Stop Function (continued)

<table>
<thead>
<tr>
<th>Method</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object-independent method</td>
<td></td>
</tr>
<tr>
<td>Initialization method</td>
<td></td>
</tr>
<tr>
<td>Methods triggered</td>
<td></td>
</tr>
<tr>
<td>Notification methods installed on the EKG_LastCheckpointID field are trigged 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.</td>
<td></td>
</tr>
<tr>
<td>Triggered by the EKG_MessageTriggeredAction function</td>
<td>No</td>
</tr>
<tr>
<td>Authorization</td>
<td>6</td>
</tr>
</tbody>
</table>

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 177. Function Block for the EKG_SwapField Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Entity_access_info_ptr</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Field_access_info_ptr</td>
</tr>
<tr>
<td>012</td>
<td>2</td>
<td>—</td>
<td>—</td>
<td>Not used</td>
</tr>
<tr>
<td>014</td>
<td>2</td>
<td>Smallint</td>
<td>In</td>
<td>Data_type</td>
</tr>
<tr>
<td>016</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>New_char_data_length</td>
</tr>
<tr>
<td>020</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>New_data_ptr</td>
</tr>
<tr>
<td>024</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Old_char_data_length</td>
</tr>
<tr>
<td>028</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Old_data_ptr</td>
</tr>
<tr>
<td>032</td>
<td>4</td>
<td>SelfDefiningDataPtr</td>
<td>In</td>
<td>Method_parms</td>
</tr>
</tbody>
</table>

See “Function Parameter Descriptions” on page 447 for more information about the parameters listed. See “Abstract Data Type Reference” on page 225 for more information about the abstract data types listed.

Examples

Table 178. Example Names for the EKG_SwapField Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11402</td>
</tr>
</tbody>
</table>
**EKG_SwapField**

_**Table 178. Example Names for the EKG_SwapField Function (continued)**_

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51402</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31402</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61402</td>
</tr>
</tbody>
</table>

**Summary**

_**Table 179. Summary of the EKG_SwapField Function**_

<table>
<thead>
<tr>
<th>Function ID</th>
<th>1402</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Action</td>
</tr>
<tr>
<td>User API</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>No</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>No</td>
</tr>
<tr>
<td>Initialization method</td>
<td>No</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>Notification and Change methods triggered</td>
</tr>
<tr>
<td>Triggered by the EKG_MessageTriggeredAction function</td>
<td>Yes</td>
</tr>
<tr>
<td>Authorization</td>
<td>3</td>
</tr>
</tbody>
</table>

**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 could 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 180. Function Block for the EKG_SwapSubfield Function*

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Entity_access_info_ptr</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Field_access_info_ptr</td>
</tr>
<tr>
<td>012</td>
<td>2</td>
<td>Smallint</td>
<td>In</td>
<td>Subfield</td>
</tr>
<tr>
<td>014</td>
<td>2</td>
<td>Smallint</td>
<td>In</td>
<td>Data_type</td>
</tr>
<tr>
<td>016</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>New_char_data_length</td>
</tr>
<tr>
<td>020</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>New_data_ptr</td>
</tr>
<tr>
<td>024</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Old_char_data_length</td>
</tr>
<tr>
<td>028</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Old_data_ptr</td>
</tr>
<tr>
<td>032</td>
<td>4</td>
<td>—</td>
<td>—</td>
<td>Not used</td>
</tr>
</tbody>
</table>

See "Function Parameter Descriptions" on page 447 for more information about the parameters listed. See "Abstract Data Type Reference" on page 225 for more information about the abstract data types listed.

**Examples**

*Table 181. Example Names for the EKG_SwapSubfield Function*

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11404</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51404</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31404</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61404</td>
</tr>
</tbody>
</table>
**EKG_SwapSubfield**

**Summary**

<table>
<thead>
<tr>
<th>Function ID</th>
<th>1404</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Action</td>
</tr>
<tr>
<td>User API</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>No</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>No</td>
</tr>
<tr>
<td>Initialization method</td>
<td>No</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>No</td>
</tr>
<tr>
<td>Triggered by the EKG_MessageTriggeredAction function</td>
<td>Yes</td>
</tr>
<tr>
<td>Authorization</td>
<td>3</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Reason code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A local value existed and was changed.</td>
</tr>
<tr>
<td>26</td>
<td>The existing value is the same as the new value.</td>
</tr>
<tr>
<td>142</td>
<td>An inherited value existed and was replaced by a local value.</td>
</tr>
</tbody>
</table>

If both 0 (zero) and 26 or both 26 and 142 could 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 183. Function Block for the EKG_TriggerNamedMethod Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Entity_access_info_ptr</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Field_access_info_ptr</td>
</tr>
<tr>
<td>012</td>
<td>4</td>
<td>SelfDefiningDataPtr</td>
<td>In</td>
<td>Methodparms</td>
</tr>
</tbody>
</table>

Table 184. Response Block for the EKG_TriggerNamedMethod Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Response_block_length</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Response_block_used</td>
</tr>
<tr>
<td>008</td>
<td>—</td>
<td>Anonymous</td>
<td>Out</td>
<td>Concat_of_strings</td>
</tr>
</tbody>
</table>

See “Function Parameter Descriptions” on page 447 for more information about the parameters listed. See “Abstract Data Type Reference” on page 225 for more information about the abstract data types listed.

Examples

Table 185. Example Names for the EKG_TriggerNamedMethod Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG11415</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>EKG21415</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG51415</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31415</td>
</tr>
<tr>
<td>C response block</td>
<td>EKG41415</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61415</td>
</tr>
</tbody>
</table>

Summary

Table 186. Summary of the EKG_TriggerNamedMethod Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>Type</th>
<th>User API</th>
<th>Object-specific method</th>
<th>Object-independent method</th>
<th>Initialization method</th>
<th>Methods triggered</th>
<th>Triggered by the EKG_MessageTriggeredAction function</th>
<th>Authorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1415</td>
<td>Action</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>5 (trigger EKG_Refresh named method) 3 (trigger other named method)</td>
</tr>
</tbody>
</table>
**EKG_TriggerNamedMethod**

**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. You should 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 187. Function Block for the EKG_TriggerOIMethod Function*

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>8</td>
<td>MethodName</td>
<td>In</td>
<td>Method_name</td>
</tr>
<tr>
<td>012</td>
<td>4</td>
<td>SelfDefiningDataPtr</td>
<td>In</td>
<td>Methodparms</td>
</tr>
</tbody>
</table>

*Table 188. Response Block for the EKG_TriggerOIMethod Function*

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Response_block_length</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Response_block_used</td>
</tr>
<tr>
<td>008</td>
<td>—</td>
<td>Anonymous</td>
<td>Out</td>
<td>Concat_of_strings</td>
</tr>
</tbody>
</table>

See [“Function Parameter Descriptions” on page 447](#) for more information about the parameters listed. See [“Abstract Data Type Reference” on page 225](#) for more information about the abstract data types listed.

**Examples**

*Table 189. Example Names for the EKG_TriggerOIMethod Function*

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL'/I function block</td>
<td>EKG11416</td>
</tr>
<tr>
<td>PL'/I response block</td>
<td>EKG21416</td>
</tr>
<tr>
<td>PL'/I usage coding</td>
<td>EKG51416</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG31416</td>
</tr>
</tbody>
</table>
Table 189. Example Names for the EKG_TriggerOIMethod Function (continued)

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>C response block</td>
<td>EKG41416</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG61416</td>
</tr>
</tbody>
</table>

Summary

Table 190. Summary of the EKG_TriggerOIMethod Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>Type</th>
<th>User API</th>
<th>Object-specific method</th>
<th>Object-independent method</th>
<th>Initialization method</th>
<th>Methods triggered</th>
<th>Triggered by the EKG_MessageTriggeredAction function</th>
<th>Authorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1416</td>
<td>Action</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>3</td>
</tr>
</tbody>
</table>

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. You should 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 191. Function Block for the EKG_UnlinkNoTrigger Function and the EKG_UnlinkTrigger Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Entity_access_info_ptr_1</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Field_access_info_ptr_1</td>
</tr>
</tbody>
</table>
Table 191. Function Block for the EKG_UnlinkNoTrigger Function and the EKG_UnlinkTrigger Function (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>012</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Entity_access_info_ptr_2</td>
</tr>
<tr>
<td>016</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Field_access_info_ptr_2</td>
</tr>
<tr>
<td>000</td>
<td>4</td>
<td>Pointer</td>
<td>In</td>
<td>Method_parms¹</td>
</tr>
</tbody>
</table>

See “Function Parameter Descriptions” on page 447 for more information about the parameters listed. See “Abstract Data Type Reference” on page 225 for more information about the abstract data types listed.

Examples

Table 192. Example Names for the EKG_UnlinkNoTrigger Function and the EKG_UnlinkTrigger Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block (EKG_UnlinkTrigger)</td>
<td>EKG11407</td>
</tr>
<tr>
<td>PL/I function block (EKG_UnlinkNoTrigger)</td>
<td>EKG11408</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding (EKG_UnlinkTrigger)</td>
<td>EKG51407</td>
</tr>
<tr>
<td>PL/I usage coding (EKG_UnlinkNoTrigger)</td>
<td>EKG51407</td>
</tr>
<tr>
<td>PL/I usage coding (EKG_UnlinkNoTrigger)</td>
<td>EKG51408</td>
</tr>
<tr>
<td>C function block (EKG_UnlinkTrigger)</td>
<td>EKG31407</td>
</tr>
<tr>
<td>C function block (EKG_UnlinkNoTrigger)</td>
<td>EKG31408</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding (EKG_UnlinkTrigger)</td>
<td>EKG61407</td>
</tr>
<tr>
<td>C usage coding (EKG_UnlinkNoTrigger)</td>
<td>EKG61408</td>
</tr>
</tbody>
</table>

Summary

Table 193. Summary of the EKG_UnlinkNoTrigger Function and the EKG_UnlinkTrigger Function

<table>
<thead>
<tr>
<th>Function ID EKG_UnlinkNoTrigger</th>
<th>1408 1407</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Action</td>
</tr>
<tr>
<td>User API</td>
<td>Yes</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>No</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>Yes</td>
</tr>
<tr>
<td>Initialization method</td>
<td>Yes</td>
</tr>
<tr>
<td>Methods triggered EKG_UnlinkTrigger</td>
<td>Change methods and notification methods No</td>
</tr>
<tr>
<td>Triggered by the EKG_MessageTriggeredAction function</td>
<td>Yes</td>
</tr>
<tr>
<td>Authorization</td>
<td>3</td>
</tr>
</tbody>
</table>
**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 194. Function Block for the EKG_UnlockAll Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
</tbody>
</table>

See "Function Parameter Descriptions" on page 447 for more information about the parameters listed. See "Abstract Data Type Reference" on page 225 for more information about the abstract data types listed.

**Examples**

Table 195. Example Names for the EKG_UnlockAll Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG12003</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>None</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG52003</td>
</tr>
</tbody>
</table>
EKG_UnlockAll

Table 195. Example Names for the EKG_UnlockAll Function (continued)

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>C function block</td>
<td>EKG32003</td>
</tr>
<tr>
<td>C response block</td>
<td>None</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG62003</td>
</tr>
</tbody>
</table>

Summary

Table 196. Summary of the EKG_UnlockAll Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>2003</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Method API Service</td>
</tr>
<tr>
<td>User API</td>
<td>No</td>
</tr>
<tr>
<td>Object-specific method</td>
<td>No</td>
</tr>
<tr>
<td>Object-independent method</td>
<td>Yes</td>
</tr>
<tr>
<td>Initialization method</td>
<td>Yes</td>
</tr>
<tr>
<td>Methods triggered</td>
<td>No</td>
</tr>
<tr>
<td>Triggered by EKG_MessageTriggeredAction function</td>
<td>No</td>
</tr>
<tr>
<td>Authorization</td>
<td>None</td>
</tr>
</tbody>
</table>

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 executed.

Function Block Format

Table 197. Function Block for the EKG_WhereAmI Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Function_ID</td>
</tr>
</tbody>
</table>

Table 198. Response Block for the EKG_WhereAmI Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Type</th>
<th>Use</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4</td>
<td>Integer</td>
<td>In</td>
<td>Response_block_length</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Integer</td>
<td>Out</td>
<td>Response_block_used</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>ClassID</td>
<td>Out</td>
<td>Class_ID</td>
</tr>
<tr>
<td>012</td>
<td>8</td>
<td>ObjectID</td>
<td>Out</td>
<td>Object_ID</td>
</tr>
<tr>
<td>020</td>
<td>4</td>
<td>FieldID</td>
<td>Out</td>
<td>Field_ID</td>
</tr>
<tr>
<td>024</td>
<td>2</td>
<td>Smallint</td>
<td>Out</td>
<td>Subfield</td>
</tr>
<tr>
<td>026</td>
<td>2</td>
<td>Anonymous(2)</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>028</td>
<td>8</td>
<td>ObjectId</td>
<td>Out</td>
<td>Requesting_method_ID</td>
</tr>
</tbody>
</table>
See “Function Parameter Descriptions” for more information about the parameters listed. See “Abstract Data Type Reference” on page 225 for more information about the abstract data types listed.

Examples

Table 199. Example Names for the EKG_WhereAml Function

<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I function block</td>
<td>EKG12007</td>
</tr>
<tr>
<td>PL/I response block</td>
<td>EKG22007</td>
</tr>
<tr>
<td>PL/I usage coding</td>
<td>EKG52007</td>
</tr>
<tr>
<td>C function block</td>
<td>EKG32007</td>
</tr>
<tr>
<td>C response block</td>
<td>EKG42007</td>
</tr>
<tr>
<td>C usage coding</td>
<td>EKG62007</td>
</tr>
</tbody>
</table>

Summary

Table 200. Summary of the EKG_WhereAml Function

<table>
<thead>
<tr>
<th>Function ID</th>
<th>Type</th>
<th>User API</th>
<th>Object-specific method</th>
<th>Object-independent method</th>
<th>Initialization method</th>
<th>Methods triggered</th>
<th>Triggered by EKG_MessageTriggeredAction function</th>
<th>Authorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td>Method API Service</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>None</td>
</tr>
</tbody>
</table>

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 should be set to null values if the naming_count information is set to zero.
Function Parameter Descriptions

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Public</td>
</tr>
<tr>
<td>2</td>
<td>Private</td>
</tr>
<tr>
<td>3</td>
<td>Public-indexed</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>User application</td>
</tr>
<tr>
<td>2</td>
<td>Method</td>
</tr>
</tbody>
</table>

Function_block_ptr
The pointer to the function block for a function to be executed. 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 executed.

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.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Subfield</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Value</td>
</tr>
<tr>
<td>2</td>
<td>Query</td>
</tr>
<tr>
<td>3</td>
<td>Change</td>
</tr>
<tr>
<td>4</td>
<td>Notify</td>
</tr>
<tr>
<td>5</td>
<td>Prev_val</td>
</tr>
</tbody>
</table>
Function Parameter Descriptions

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Timestamp</td>
</tr>
<tr>
<td>7-32</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**Local_inherited_flag**
A flag that specifies whether a field is locally defined or is inherited from a parent class. Valid values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Locally defined</td>
</tr>
<tr>
<td>1</td>
<td>Inherited</td>
</tr>
</tbody>
</table>

**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 321.

**Notification_queue_count**
The number of notification blocks on the notification_queue before this function acts on the queue.
Function Parameter Descriptions

**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 executed. 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Public</td>
</tr>
<tr>
<td>1</td>
<td>Private</td>
</tr>
</tbody>
</table>

**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 should 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Generated by a notification method</td>
</tr>
<tr>
<td>2</td>
<td>Generated when an object was deleted and an object-deletion subscription existed for that object</td>
</tr>
</tbody>
</table>

**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 should have been 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 should 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Subfield</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>All subfields except Notify (valid only for EKG_RevertToInherited function)</td>
</tr>
<tr>
<td>1</td>
<td>Value</td>
</tr>
<tr>
<td>2</td>
<td>Query</td>
</tr>
<tr>
<td>3</td>
<td>Change</td>
</tr>
<tr>
<td>4</td>
<td>Notify</td>
</tr>
</tbody>
</table>
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 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 472 and "List of Functions for Each Reason Code" on page 474 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 NetView-Supplied Methods" on page 481 lists the reason codes returned by the NetView-supplied methods.

Reason codes can fall into one of three ranges based on which program or method issued the reason code:

<table>
<thead>
<tr>
<th>Range</th>
<th>Issued By</th>
</tr>
</thead>
<tbody>
<tr>
<td>0–32767</td>
<td>RODM application programming interfaces</td>
</tr>
<tr>
<td>32768–49151</td>
<td>NetView-supplied methods</td>
</tr>
<tr>
<td>49152–65535</td>
<td>Customer-written methods</td>
</tr>
</tbody>
</table>

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 NetView-supplied methods EKGCTIM, EKGMIMV, EKGEQNL, EKGNLST, EKGNOTF, EKGNTHD, and FLBTRNMM. 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 CNMNETV. The reason code issued is 32809 plus the return code from CNMNETV. Subtract 32809 from the reason code issued by the EKGSPPI method. The result is the return code from CNMNETV. Refer to the Tivoli NetView for z/OS Application Programmer’s Guide for the meaning of this return code.

Writers of methods should be aware of the implications of issuing return and reason codes from methods. See "Error Conditions in Transactions" on page 313 for information about how an application might interpret reason and return codes that are returned by methods.

The Tivoli NetView for z/OS Diagnosis Guide contains additional information about troubleshooting RODM problems, especially abend problems.

### Reason Codes for Return Code 0

Table 201 describes the reason codes that are returned with return code 0.

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Description</th>
<th>Corrective Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The system successfully performs the requested function.</td>
<td>None</td>
</tr>
<tr>
<td>26</td>
<td>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.</td>
<td>None</td>
</tr>
<tr>
<td>48</td>
<td>Not used.</td>
<td>None</td>
</tr>
<tr>
<td>142</td>
<td>The system performs the request successfully and a local copy is created.</td>
<td>None</td>
</tr>
<tr>
<td>143</td>
<td>The system performs the request successfully and the returned value is an inherited value.</td>
<td>None</td>
</tr>
<tr>
<td>167</td>
<td>Not used.</td>
<td>None</td>
</tr>
<tr>
<td>180</td>
<td>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.</td>
<td>None</td>
</tr>
<tr>
<td>185</td>
<td>The Disconnect is successful. The user object is not deleted from RODM because links to Notification Queue objects still exist.</td>
<td>Try to connect and disconnect again.</td>
</tr>
<tr>
<td>32769</td>
<td>Compared data values do not match.</td>
<td>Specify the value subfield for the data to be compared.</td>
</tr>
</tbody>
</table>
Table 202 describes the reason codes that are returned with return code 4.

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Description</th>
<th>Corrective Action</th>
</tr>
</thead>
</table>
| 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 datasets.  
RODM rejects all new user API requests and returns this reason code. | Retry the request after the checkpoint process is completed. |
<p>| 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. |
| 5           | The system rejects the request because RODM has been 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 function, retry the request after restarting the specified RODM. If this reason code was not result of a EKG_Connect 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 has been 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 function, retry the request after restarting the specified RODM. If this reason code was not result of a EKG_Connect 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 |</p>
<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Description</th>
<th>Corrective Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>The specified queue object is created but the link with the user object cannot be created. The required storage might not be available.</td>
<td>None</td>
</tr>
<tr>
<td>38</td>
<td>The operator terminated 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.</td>
<td>Contact operator.</td>
</tr>
<tr>
<td>40</td>
<td>The system does not change the field value because the field already contains the primary inheritance value.</td>
<td>None</td>
</tr>
<tr>
<td>41</td>
<td>The system rejects the request because the field is locally created.</td>
<td>None</td>
</tr>
<tr>
<td>42</td>
<td>The specified method is a null module because it has been deleted by an unsuccessful module refresh. The transaction failed.</td>
<td>Refresh the method and retry the request. If not successful, delete the method and reinstall it.</td>
</tr>
<tr>
<td>44</td>
<td>There is no message in the specified notification queue for the user.</td>
<td>None</td>
</tr>
<tr>
<td>46</td>
<td>The overflow block is cleaned without retrieving because the response block provided by the user is null.</td>
<td>None</td>
</tr>
<tr>
<td>47</td>
<td>Some of the overflow data is discarded because the response block provided by the user is not large enough.</td>
<td>None</td>
</tr>
<tr>
<td>48</td>
<td>Not used.</td>
<td>None</td>
</tr>
<tr>
<td>49</td>
<td>Not used.</td>
<td>None</td>
</tr>
<tr>
<td>50</td>
<td>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.</td>
<td>Correct the class or parent class. Retry the request.</td>
</tr>
<tr>
<td>52</td>
<td>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.</td>
<td>Correct the object data. Retry the request.</td>
</tr>
<tr>
<td>54</td>
<td>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.</td>
<td>Correct the field data. Retry the request.</td>
</tr>
<tr>
<td>56</td>
<td>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.</td>
<td>Correct the field data. Retry the request.</td>
</tr>
<tr>
<td>57</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>Reason Code</td>
<td>Description</td>
<td>Corrective Action</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>62</td>
<td>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.</td>
<td>Correct the subfield data. Retry the request.</td>
</tr>
<tr>
<td>72</td>
<td>The target fields have already been linked. The system has taken no action.</td>
<td>Correct the entity and field information. Retry the request.</td>
</tr>
<tr>
<td>75</td>
<td>The target fields are not linked. The system has taken no action.</td>
<td>Correct the field information. Retry the request.</td>
</tr>
<tr>
<td>81</td>
<td>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.</td>
<td>Install the specified method. Retry the request.</td>
</tr>
<tr>
<td>92</td>
<td>The system rejects the request because the field to be created already exists under the specified class.</td>
<td>Correct the field data. Retry the request.</td>
</tr>
<tr>
<td>97</td>
<td>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.</td>
<td>None</td>
</tr>
<tr>
<td>100</td>
<td>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.</td>
<td>Correct the subfield map. Retry the request.</td>
</tr>
<tr>
<td>104</td>
<td>One or more specified subfields already exist.</td>
<td>Correct the subfield map. Retry the request.</td>
</tr>
<tr>
<td>110</td>
<td>The system rejects the request because the specified object name is used by another object under the specified parent class.</td>
<td>Correct the object name. Retry the request.</td>
</tr>
<tr>
<td>112</td>
<td>The system rejects the request because the specified field already has a notification subscription with the same parameters.</td>
<td>Correct the request data. Retry the request.</td>
</tr>
<tr>
<td>133</td>
<td>The system cannot update the value of the timestamp subfield. There might not be enough storage.</td>
<td>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.</td>
</tr>
<tr>
<td>146</td>
<td>One or more specified subfields do not exist in the specified field.</td>
<td>Correct the subfield information. Retry the request.</td>
</tr>
<tr>
<td>158</td>
<td>The notification cannot be placed in the notification queue because the queue has reached its maximum limit.</td>
<td>Query the notification queue content or enlarge the value of EKG_Maximum_Q_Entries.</td>
</tr>
<tr>
<td>173</td>
<td>The system performs the request successfully and one notification queue is created by RODM.</td>
<td>Change the EKG_ECBAddress of this notification queue object to a valid value.</td>
</tr>
<tr>
<td>174</td>
<td>The notification information block has been put into the notification queue. The system cannot post the specified user because the ECB address is null.</td>
<td>None</td>
</tr>
<tr>
<td>175</td>
<td>Part of the user message is truncated because it is longer than 32767 bytes.</td>
<td>None</td>
</tr>
</tbody>
</table>
### Reason Codes for Return Code 4

**Table 202. Reason Codes for Return Code 4 (continued)**

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Description</th>
<th>Corrective Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>181</td>
<td>The notification cannot be attached to the specified queue because the queue is not active.</td>
<td>Change the EKG_Status value of the specified queue object.</td>
</tr>
<tr>
<td>182</td>
<td>The notification has been put in the notification queue. The system cannot post the specified user.</td>
<td>None</td>
</tr>
<tr>
<td>183</td>
<td>The information from the notification block has been put in the response block. The system cannot release the storage used by the notification block.</td>
<td>None</td>
</tr>
<tr>
<td>191</td>
<td>The system rejects the request because the specified method object is the NullMeth object.</td>
<td>Correct the method object information. Retry the request.</td>
</tr>
<tr>
<td>204</td>
<td>The original data in the response block is overwritten.</td>
<td>None</td>
</tr>
<tr>
<td>205</td>
<td>Not used.</td>
<td>None</td>
</tr>
<tr>
<td>206</td>
<td>Not used.</td>
<td>None</td>
</tr>
<tr>
<td>208</td>
<td>The response block overflow data is discarded because the user has specified to not save overflow data.</td>
<td>If the response block overflow data is needed, change the value of the EKG_RBOverflowAction field to save. Retry the request.</td>
</tr>
<tr>
<td>209</td>
<td>The user request to wait on a list of ECBs cannot be completed because an ECB address of 0 is found.</td>
<td>Correct the ECB address.</td>
</tr>
<tr>
<td>221</td>
<td>Not used.</td>
<td>None</td>
</tr>
<tr>
<td>604</td>
<td>A correlated aggregate object was not created because the agent provided an invalid correlation value (network address).</td>
<td>Modify the agent (distributed manager) to provide a valid network address.</td>
</tr>
<tr>
<td>605</td>
<td>A correlated aggregate object was not created because a correlated aggregate object already exists.</td>
<td>None</td>
</tr>
<tr>
<td>32770</td>
<td>Part of the method output message from the NetView-supplied notification method is discarded because the length exceeds 32767 bytes. The request completed successfully.</td>
<td>Correct the method output message.</td>
</tr>
<tr>
<td>45081</td>
<td>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.</td>
<td>The condition that caused this error should 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.</td>
</tr>
</tbody>
</table>

**Reason Codes for Return Code 8**

**Table 203 on page 460** describes the reason codes that are returned with return code 8.
<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Description</th>
<th>Corrective Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>The system rejects the request because the API version is not valid.</td>
<td>Correct the API version information in the transaction information block. Retry the request.</td>
</tr>
<tr>
<td>9</td>
<td>The system rejects the request because the caller is not authorized to use the requested function.</td>
<td>Make sure that the User_appl_ID is correct or contact the system administrator to change the authority level.</td>
</tr>
<tr>
<td>10</td>
<td>The system rejects the request because the function ID is not valid.</td>
<td>Correct the function ID. Retry the request.</td>
</tr>
<tr>
<td>11</td>
<td>The requested function is not complete because the system does not have enough storage to copy the short-lived parameters into RODM.</td>
<td>Remove unused entities and fields or contact the system administrator. Retry the request.</td>
</tr>
<tr>
<td>13</td>
<td>The system rejects the request because the specified RODM is not found.</td>
<td>Start the RODM with the specified name or correct the RODM_name field in the RODM access block. Retry the request.</td>
</tr>
<tr>
<td>14</td>
<td>The system rejects the request because an incorrect Sign_on_token is detected. The user application has not connected to the specified RODM using the EKG_Connect function, or the Sign_on_token has been changed.</td>
<td>Make sure the user application does not modify the Sign_on_token. Connect to the specified RODM using the EKG_Connect function to get a valid Sign_on_token. Retry the request with the new Sign_on_token.</td>
</tr>
<tr>
<td>15</td>
<td>The system rejects the request because the number of concurrently executing API function calls has reached the limit specified in the customization file.</td>
<td>Retry the request later or increase the CONCURRENT_USERS value in the RODM customization file. Warm start RODM.</td>
</tr>
<tr>
<td>16</td>
<td>The system rejects the request because no RODM currently exists in the system.</td>
<td>Start the RODM with the specified name. Retry the request.</td>
</tr>
<tr>
<td>17</td>
<td>The system rejects the request because the RODM service module in CSA is not found.</td>
<td>Contact the system administrator.</td>
</tr>
<tr>
<td>18</td>
<td>The system rejects the request because the specified function is not allowed for this method.</td>
<td>Correct the function ID in the function block. Retry the request.</td>
</tr>
<tr>
<td>21</td>
<td>The system cannot perform the requested list of functions because the number of list requests provided by the user is zero or negative.</td>
<td>Correct the Number_of_functions field. Retry the request.</td>
</tr>
<tr>
<td>22</td>
<td>The system rejects the request because the notification queue name is null.</td>
<td>Correct the notification queue name. Retry the request.</td>
</tr>
<tr>
<td>23</td>
<td>The system rejects the request because the data (types CharVar, GraphicVar, MethodSpec, SelfDefining, or BERVar) passed to RODM is not valid.</td>
<td>Correct the data. Retry the request.</td>
</tr>
<tr>
<td>33</td>
<td>The system rejects the request because no storage is available for storing the log record information.</td>
<td>Delete unused entities. Retry the request.</td>
</tr>
<tr>
<td>Reason Code</td>
<td>Description</td>
<td>Corrective Action</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>35</td>
<td>Checkpoint master window error. The VSAM dataset 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.</td>
<td>Contact the system administrator.</td>
</tr>
<tr>
<td>36</td>
<td>Checkpoint translation window error. The VSAM dataset 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.</td>
<td>Contact the system administrator.</td>
</tr>
<tr>
<td>37</td>
<td>Checkpoint data window error. One or more of the VSAM datasets 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.</td>
<td>Contact the system administrator.</td>
</tr>
<tr>
<td>39</td>
<td>The system rejects the request because the data pointed to by Old_data_ptr is not equal to the target field.</td>
<td>Correct Old_data_ptr. Retry the request.</td>
</tr>
<tr>
<td>52</td>
<td>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.</td>
<td>Correct the class or parent class. Retry the request.</td>
</tr>
<tr>
<td>54</td>
<td>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.</td>
<td>Correct the object data. Retry the request.</td>
</tr>
<tr>
<td>56</td>
<td>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.</td>
<td>Correct the field data. Retry the request.</td>
</tr>
<tr>
<td>57</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>60</td>
<td>The system rejects the request because the field type is public and there are still objects existing under the class or descendent classes.</td>
<td>Delete the objects under the class before deleting the public field or its subfields.</td>
</tr>
<tr>
<td>61</td>
<td>The system rejects the request because the subfield number is not valid.</td>
<td>Correct the subfield number. Retry the request.</td>
</tr>
<tr>
<td>62</td>
<td>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.</td>
<td>Correct the subfield data. Retry the request.</td>
</tr>
<tr>
<td>Reason Code</td>
<td>Description</td>
<td>Corrective Action</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>65</td>
<td>The system rejects the request because this function does not apply to fields with data type ObjectLink or ObjectLinkList.</td>
<td>Correct the function ID or field identifier. Retry the request.</td>
</tr>
<tr>
<td>66</td>
<td>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.</td>
<td>Correct the data type or field. Retry the request.</td>
</tr>
<tr>
<td>67</td>
<td>The system rejects the request because this function does not apply to a system-defined field.</td>
<td>Correct the function ID or the field. Retry the request.</td>
</tr>
<tr>
<td>70</td>
<td>The system rejects the request because this function does not apply to a notify subfield.</td>
<td>Correct the subfield or function ID. Retry the request.</td>
</tr>
<tr>
<td>71</td>
<td>The system rejects the request because this function does not apply to a prev_val or timestamp subfield.</td>
<td>Correct the subfield or function ID. Retry the request.</td>
</tr>
<tr>
<td>73</td>
<td>The system rejects the request because the two target objects are identical.</td>
<td>Correct the entity information. Retry the request.</td>
</tr>
<tr>
<td>74</td>
<td>The system rejects the request because the field data type is not allowed for a link or unlink function.</td>
<td>Correct the field information. Retry the request.</td>
</tr>
<tr>
<td>76</td>
<td>The system rejects the request because the notify subfield does not exist.</td>
<td>Create a notify subfield for the specified field.</td>
</tr>
<tr>
<td>77</td>
<td>The system rejects the request because this function does not apply to some of the system-defined fields.</td>
<td>Correct the fields. Retry the request.</td>
</tr>
<tr>
<td>79</td>
<td>The system rejects the request because the specified function block pointer in the list is null.</td>
<td>Correct the function block pointer. Retry the request.</td>
</tr>
<tr>
<td>80</td>
<td>The system rejects the request because this module recursively calls itself.</td>
<td>Update the related methods to remove the recursive call. Retry the request.</td>
</tr>
<tr>
<td>81</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>83</td>
<td>The system rejects the request because the response block length is less than eight bytes.</td>
<td>Correct the response block length. Retry the request.</td>
</tr>
<tr>
<td>84</td>
<td>The user has already connected to RODM.</td>
<td>None</td>
</tr>
<tr>
<td>85</td>
<td>The system rejects the request because the specified Stop_type is not valid.</td>
<td>Correct the Stop_type. You can specify the value of Stop_type as 1 or 2. Retry the request.</td>
</tr>
<tr>
<td>86</td>
<td>The system rejects the request because the specified class name is not valid or is a RODM reserved class name.</td>
<td>Correct the class name. Retry the request.</td>
</tr>
<tr>
<td>87</td>
<td>The system rejects the request because the specified class name has been used by another class.</td>
<td>Correct the class name. Retry the request.</td>
</tr>
<tr>
<td>89</td>
<td>The system rejects the request because the universal class or a system-created class cannot be deleted.</td>
<td>Correct the class information. Retry the request.</td>
</tr>
<tr>
<td>90</td>
<td>The system rejects the request because some entities exist under the specified class.</td>
<td>Delete all entities under the specified class. Retry the request.</td>
</tr>
<tr>
<td>Reason Code</td>
<td>Description</td>
<td>Corrective Action</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>------------------</td>
</tr>
<tr>
<td>91</td>
<td>The system rejects the request because the specified field name is not valid or is a reserved RODM field name.</td>
<td>Correct the field name. Retry the request.</td>
</tr>
<tr>
<td>93</td>
<td>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.</td>
<td>Correct the field data. Retry the request.</td>
</tr>
<tr>
<td>94</td>
<td>The system rejects the request because the field to be created already exists in a child class with a different field type.</td>
<td>Correct the field data. Retry the request.</td>
</tr>
<tr>
<td>95</td>
<td>The system rejects the request because the field type flag is not valid.</td>
<td>Correct the field type flag. You can specify the value of the field type flag as 1, 2, or 3. Retry the request.</td>
</tr>
<tr>
<td>96</td>
<td>The system rejects the request because the data type is not valid or is a reserved data type.</td>
<td>Correct the data type. You cannot create fields using reserved data types. Retry the request.</td>
</tr>
<tr>
<td>98</td>
<td>The system rejects the request because a user application is not allowed to delete system-defined fields.</td>
<td>Correct the field information. Retry the request.</td>
</tr>
<tr>
<td>100</td>
<td>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.</td>
<td>Correct the subfield map. Retry the request.</td>
</tr>
<tr>
<td>103</td>
<td>The system rejects the request because the field or subfield does not exist under the specified class.</td>
<td>Correct the class information to specify the class where the field or subfield exists. Retry the request.</td>
</tr>
<tr>
<td>106</td>
<td>The system rejects the request because the value subfield cannot be deleted.</td>
<td>Correct the subfield name. Retry the request.</td>
</tr>
<tr>
<td>107</td>
<td>The system rejects the request because the method object name is not valid.</td>
<td>Correct the method name. Retry the request.</td>
</tr>
<tr>
<td>108</td>
<td>The system rejects the request to delete the method because the method is in use.</td>
<td>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.</td>
</tr>
<tr>
<td>109</td>
<td>The system rejects the request because the user-provided object name is not valid or is a RODM reserved object name.</td>
<td>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.</td>
</tr>
<tr>
<td>111</td>
<td>The system rejects the request because the specified object is linked to other objects.</td>
<td>Unlink all other objects from the specified object. Retry the request.</td>
</tr>
<tr>
<td>113</td>
<td>The system rejects the request because the specified subscription does not exist.</td>
<td>Correct the request data. Retry the request.</td>
</tr>
<tr>
<td>115</td>
<td>The system rejects the request because the data type for this field is not valid for this function.</td>
<td>Correct the field data type. Retry the request.</td>
</tr>
<tr>
<td>117</td>
<td>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.</td>
<td>Correct the function ID in the function information array.</td>
</tr>
<tr>
<td>120</td>
<td>The system rejects the request because an overflow block with the specified correlation ID does not exist.</td>
<td>Correct the correlation ID. Retry the request.</td>
</tr>
</tbody>
</table>
### Reason Codes for Return Code 8

**Table 203. Reason Codes for Return Code 8 (continued)**

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Description</th>
<th>Corrective Action</th>
</tr>
</thead>
</table>
| 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 EKCCUST.) 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 RACE, 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 a 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 is expired.  
- The password is not authorized.  
- The user ID has been revoked in the SAF product. | If the password is expired or not authorized, correct the problem and retry the request. If the password is not the problem, have the security administrator check the status of the user ID in the SAF product. |
<p>| 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. |</p>
<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Description</th>
<th>Corrective Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>134</td>
<td>The system cannot update the value of the prev_val subfield. There might not be enough storage.</td>
<td>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.</td>
</tr>
<tr>
<td>135</td>
<td>The system rejects the request because the length of the long-lived parameters is not valid.</td>
<td>Correct the parameter lengths. Retry the request.</td>
</tr>
<tr>
<td>136</td>
<td>The system rejects the request because the length of the Method_parms is not valid.</td>
<td>Correct the parameter length. The maximum length is 254 bytes. Retry the request.</td>
</tr>
<tr>
<td>138</td>
<td>The system rejects the request because the Old_data_ptr is null.</td>
<td>Correct the Old_data_ptr. Retry the request.</td>
</tr>
<tr>
<td>139</td>
<td>The system rejects the request because the field ID is not specified and the field name pointer or field name length is not valid.</td>
<td>Specify either a valid field ID or a valid field name pointer and field name length. Retry the request.</td>
</tr>
<tr>
<td>140</td>
<td>The system rejects the request because the class ID is not specified and the class name is not valid.</td>
<td>Specify a valid class ID or a valid class name. Retry the request.</td>
</tr>
<tr>
<td>141</td>
<td>The system rejects the request because the specified field of data type ObjectLink is already linked to another field.</td>
<td>Correct the field information. Retry the request.</td>
</tr>
<tr>
<td>144</td>
<td>The system rejects the request because a system-created field or subfield cannot be deleted by a user application.</td>
<td>Correct the field or subfield information. Retry the request.</td>
</tr>
<tr>
<td>145</td>
<td>The system rejects the request because the specified field or subfield is read only.</td>
<td>Correct the field or subfield information. Retry the request.</td>
</tr>
<tr>
<td>147</td>
<td>The system rejects the request because the length of the new data is not valid.</td>
<td>Correct the data length. Retry the request.</td>
</tr>
<tr>
<td>148</td>
<td>The system rejects the request because the create subfield function is not valid for a system-defined field.</td>
<td>Correct the field information. Retry the request.</td>
</tr>
<tr>
<td>150</td>
<td>The system rejects the request because the object ID is not specified and the object name information is not valid.</td>
<td>Specify a valid object ID or a valid object name. Retry the request.</td>
</tr>
<tr>
<td>159</td>
<td>The system rejects the request because the object directory or the field name table has reached its maximum size limit.</td>
<td>Select another object or field. Retry the request.</td>
</tr>
<tr>
<td>160</td>
<td>The system rejects the request because the field name is not specified.</td>
<td>Specify the field name. Retry the request.</td>
</tr>
<tr>
<td>163</td>
<td>The system rejects the request because the pointer to the entity access information block is not valid.</td>
<td>Correct the entity access information block pointer. Retry the request.</td>
</tr>
<tr>
<td>164</td>
<td>The system rejects the request because the pointer to the field access information block is not valid.</td>
<td>Correct the field access information block pointer. Retry the request.</td>
</tr>
<tr>
<td>165</td>
<td>The system rejects the request because the Naming_count of the entity access information block is not valid.</td>
<td>Correct the Naming_count value. Valid Naming_count values are 0, 1, and 2. Retry the request.</td>
</tr>
<tr>
<td>166</td>
<td>The system rejects the request because the Naming_count of the field access information block is not valid.</td>
<td>Correct the Naming_count value. Valid Naming_count values are 0 and 1. Retry the request.</td>
</tr>
</tbody>
</table>
### Reason Codes for Return Code 8

**Table 203. Reason Codes for Return Code 8 (continued)**

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Description</th>
<th>Corrective Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>169</td>
<td>The system rejects the request because the object ID is not specified.</td>
<td>Specify the object ID. Retry the request.</td>
</tr>
<tr>
<td>170</td>
<td>The system rejects the request because a user application cannot create or delete a system object.</td>
<td>Correct the parent class information. Retry the request.</td>
</tr>
<tr>
<td>176</td>
<td>The system rejects the request because the new data is not valid.</td>
<td>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.</td>
</tr>
<tr>
<td>186</td>
<td>The system rejects the request because the user application cannot create classes under the system classes.</td>
<td>Correct the parent class information. Retry the request.</td>
</tr>
<tr>
<td>187</td>
<td>The system rejects the request because the specified subfield map is null.</td>
<td>Correct the subfield map. Retry the request.</td>
</tr>
<tr>
<td>192</td>
<td>The system rejects the request because the specified function ID is not valid asynchronous execution.</td>
<td>Correct the function ID. Retry the request.</td>
</tr>
<tr>
<td>193</td>
<td>The return or reason code set by the method is not valid.</td>
<td>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.</td>
</tr>
<tr>
<td>195</td>
<td>The system rejects the request because the system field cannot be changed at the class level.</td>
<td>Correct the data. Retry the request.</td>
</tr>
<tr>
<td>201</td>
<td>The system rejects the request because the data to be return is a null string.</td>
<td>Correct the data. Retry the request.</td>
</tr>
<tr>
<td>202</td>
<td>The system rejects the request because the checkpoint function is disabled.</td>
<td>Make sure that all checkpoint datasets are defined when RODM is started.</td>
</tr>
<tr>
<td>203</td>
<td>The system rejects the request because there is no response block.</td>
<td>Specify a response block. Retry the request.</td>
</tr>
<tr>
<td>207</td>
<td>The EKG_Connect function cannot be completed. Possible causes are that RACF is active but the class specified in the customization file is not active in RACF or the class is not defined in RACF.</td>
<td>Contact the system administrator.</td>
</tr>
<tr>
<td>210</td>
<td>The user request to wait on a list of ECBs cannot be completed because the number of ECBs is zero.</td>
<td>Correct the number of ECBs in the list.</td>
</tr>
<tr>
<td>214</td>
<td>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.</td>
<td>Correct the Naming_count. Valid values are 0 and 2. Retry the request.</td>
</tr>
<tr>
<td>215</td>
<td>The system rejects the request because the user is not allowed to update EKG_MTraceFlag of NullMeth.</td>
<td>Correct the method object information.</td>
</tr>
<tr>
<td>220</td>
<td>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.</td>
<td>Examine the change method to see what criteria it uses to allow links or unlinks and make sure you meet those criteria.</td>
</tr>
</tbody>
</table>
## Table 203. Reason Codes for Return Code 8 (continued)

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Description</th>
<th>Corrective Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>223</td>
<td>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.</td>
<td>Specify a correct value for the Number_of_subfields field. Specify a correct value for the Number_of_Fields field.</td>
</tr>
<tr>
<td>224</td>
<td>The system rejects the request because the input data type is not allowed for an indexed field.</td>
<td>Correct the input data type or the Field_type_flag. Retry the request.</td>
</tr>
<tr>
<td>225</td>
<td>The system rejects the request because the field has not been created with the corresponding Field_type_flag.</td>
<td>Correct the field name, or the field ID and field type information. Retry the request.</td>
</tr>
<tr>
<td>226</td>
<td>You tried to connect a program that is not APF authorized with a blank password specified.</td>
<td>Either specify the correct password in the User_password field of the function block, or make the program APF authorized.</td>
</tr>
<tr>
<td>227</td>
<td>The system rejects the request because a reserved field in the function block is not zero.</td>
<td>Ensure that all of the reserved fields in the function block are set to zero. Retry the request.</td>
</tr>
<tr>
<td>228</td>
<td>The system rejects the request because the indexed data length field for the locate function is not valid.</td>
<td>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.</td>
</tr>
<tr>
<td>229</td>
<td>The system rejects the request because the index data value pointer for the locate function is not valid.</td>
<td>Correct the indexed data value pointer. Retry the request.</td>
</tr>
<tr>
<td>230</td>
<td>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.</td>
<td>Ensure that the length is correct. Retry the request.</td>
</tr>
<tr>
<td>231</td>
<td>The system rejects the request because the IndexList field contains a duplicate value.</td>
<td>Ensure that each value is unique within the field. Retry the request.</td>
</tr>
<tr>
<td>232</td>
<td>The system rejects the request because a length found in a value of the IndexList field is invalid.</td>
<td>Ensure that the length of each value is between 0 and 254 bytes. Retry the request.</td>
</tr>
<tr>
<td>32768</td>
<td>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.</td>
<td>Correct the Long_lived_parm.</td>
</tr>
<tr>
<td>32771</td>
<td>The system rejects the request because the data type of the value subfield queried is not valid.</td>
<td>Verify the correct data type for the method being used. See <a href="#">&quot;NetView-Supplied Methods&quot;</a> on page 483 for a description of the NetView-supplied methods. Correct the parameter list passed to the method.</td>
</tr>
<tr>
<td>32772</td>
<td>The system rejects the NetView-supplied notification method because the data type of the value in the specified field is not valid. The request failed.</td>
<td>Correct the field data type of the specified field. Valid field data types are Smallint, Integer, Floating, TimeStamp or CharVar.</td>
</tr>
<tr>
<td>32790</td>
<td>The short-lived parameter passed to the method is a null pointer.</td>
<td>Correct the pointer.</td>
</tr>
</tbody>
</table>
## Reason Codes for Return Code 8

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Description</th>
<th>Corrective Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>32791</td>
<td>One or more data items in the short-lived parameter is not of data type CharVar.</td>
<td>Correct the short-lived parameter.</td>
</tr>
<tr>
<td>32792</td>
<td>One or more data items in the short-lived parameter is too long.</td>
<td>Correct the short-lived parameter.</td>
</tr>
<tr>
<td>32793</td>
<td>Incorrect number of data items in the short-lived parameter.</td>
<td>Correct the short-lived parameter.</td>
</tr>
<tr>
<td>32794</td>
<td>The RCVRID_CHARVAR value in the short-lived parameter passed to the EKGSPPI method is blank or null.</td>
<td>Specify a valid value for RCVRID_CHARVAR.</td>
</tr>
<tr>
<td>32795</td>
<td>The ASSIST_CHARVAR value in the short-lived parameter passed to the EKGSPPI method is not valid.</td>
<td>Specify a valid value for ASSIST_CHARVAR.</td>
</tr>
<tr>
<td>32796</td>
<td>The TASKINFO_CHARVAR value in the short-lived parameter passed to the EKGSPPI method is not valid.</td>
<td>Specify a valid value for TASKINFO_CHARVAR.</td>
</tr>
<tr>
<td>32797</td>
<td>The TASKNAME_CHARVAR value in the short-lived parameter passed to the EKGSPPI method is blank or null.</td>
<td>Specify a valid value for TASKNAME_CHARVAR.</td>
</tr>
<tr>
<td>32798</td>
<td>The CMD_CHARVAR value in the short-lived parameter passed to the EKGSPPI method is blank or null.</td>
<td>Specify a valid value for CMD_CHARVAR.</td>
</tr>
<tr>
<td>45057</td>
<td>The DUIFCUAP method parameters specify deleting the AggregationParent to AggregationChild link between two objects. However, the specified objects do not have this link.</td>
<td>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.</td>
</tr>
<tr>
<td>45058</td>
<td>The DUIFCUAP method parameters specify creating the AggregationParent to AggregationChild link between two objects. However, the specified objects already have this link.</td>
<td>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.</td>
</tr>
<tr>
<td>45061</td>
<td>Not used.</td>
<td>None</td>
</tr>
<tr>
<td>45066</td>
<td>The DUIFCUAP does not create the requested link. Creating the requested link would create 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.</td>
<td>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.</td>
</tr>
<tr>
<td>45070</td>
<td>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.</td>
<td>Check the RODM log and verify that the parameters sent to the method have the correct format and value for the method.</td>
</tr>
</tbody>
</table>
Table 203. Reason Codes for Return Code 8 (continued)

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Description</th>
<th>Corrective Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>45071</td>
<td>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.</td>
<td>Create the missing object or correct the input parameters for the method and retry the request.</td>
</tr>
<tr>
<td>45077</td>
<td>An error occurred for a method that was triggered for this transaction. Diagnostic information is written to the RODM log.</td>
<td>Check the RODM log for information on the specific error that occurred.</td>
</tr>
<tr>
<td>45078</td>
<td>An error occurred while processing a transaction. The RODM data cache might contain inconsistent field values.</td>
<td>Check the RODM log for information on the specific error that occurred. Correct the specific error. Repeat the transaction or run the DUIFFAWS method.</td>
</tr>
<tr>
<td>45079</td>
<td>An error occurred while processing a transaction. Some part of the change required for the transaction was completed, but not all of it.</td>
<td>Check the RODM log for information on the specific error that occurred. Correct the specific error. Repeat the transaction.</td>
</tr>
<tr>
<td>45080</td>
<td>The value or data type of the data specified by the New_data_ptr parameter for an EKG_ChangeField function request is not valid.</td>
<td>Check the RODM log for information on the specific field where the error occurred. Correct the error. Repeat the transaction.</td>
</tr>
<tr>
<td>45082</td>
<td>An error occurred while processing a transaction. The value of the DisplayStatus field of one or more aggregate objects might be incorrect.</td>
<td>Check the RODM log for information on the specific error that occurred. Correct the specific error. Repeat the transaction or run the DUIFFRAS method.</td>
</tr>
<tr>
<td>45083</td>
<td>An object passed to the method in the self-defining method parameters is not in the expected class.</td>
<td>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.</td>
</tr>
<tr>
<td>45092</td>
<td>An attempt to connect the GMFHS application to RODM failed. Another GMFHS application is already connected to RODM.</td>
<td>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.</td>
</tr>
<tr>
<td>45093</td>
<td>The version of GMFHS methods installed in RODM is incompatible with the version of the GMFHS application that attempted a connection with RODM.</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Reason Codes for Return Code 12**

Table 204 describes the reason codes that are returned with return code 12.

Table 204. Reason Codes for Return Code 12

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Description</th>
<th>Corrective Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Not used.</td>
<td>None</td>
</tr>
</tbody>
</table>
### Reason Codes for Return Code 12

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Description</th>
<th>Corrective Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>All or some of the response block overflow data is discarded because the overflow block does not have enough storage.</td>
<td>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.</td>
</tr>
<tr>
<td>20</td>
<td>The requested function might not complete because an abend occurred during the transaction.</td>
<td>Verify that the control blocks passed to RODM for the transaction are valid. Refer to the <em>Tivoli NetView for z/OS Diagnosis Guide</em> for information on diagnosing abend problems.</td>
</tr>
<tr>
<td>25</td>
<td>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.</td>
<td>Retry the request later.</td>
</tr>
<tr>
<td>63</td>
<td>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.</td>
<td>Verify that the method exists in the method library.</td>
</tr>
<tr>
<td>68</td>
<td>Not used.</td>
<td>None</td>
</tr>
<tr>
<td>82</td>
<td>The module of the specified method has been deleted by an unsuccessful module refresh.</td>
<td>Refresh the module of the method again.</td>
</tr>
<tr>
<td>118</td>
<td>Not used.</td>
<td>None</td>
</tr>
</tbody>
</table>
| 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 datasets  
• In the translation window | The most likely reason for this return and reason code is that the VSAM dataset 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 dataset. The checkpoint dataset 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 datasets  
• In the RODM dataspace | The most likely reason for this return and reason code is that the VSAM checkpoint datasets 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 dataset or add another data window checkpoint dataset. The checkpoint data sets are specified by DDname EKGDnnn in the RODM startup JCL. |
<p>| 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. |</p>
<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Description</th>
<th>Corrective Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>179</td>
<td>The system rejects the request because the system cannot create the user object. The possible cause is that not enough storage is available.</td>
<td>Retry the request later.</td>
</tr>
<tr>
<td>188</td>
<td>Not used.</td>
<td>None</td>
</tr>
<tr>
<td>194</td>
<td>The system cannot complete the request because the method has an execution error.</td>
<td>Check the RODM log record for further information.</td>
</tr>
<tr>
<td>198</td>
<td>The system rejects the request because the system cannot change the fields of the user object. There might not be enough storage available.</td>
<td>Free some storage and retry the request.</td>
</tr>
<tr>
<td>199</td>
<td>An operator canceled the user transaction.</td>
<td>Check with the operator.</td>
</tr>
<tr>
<td>200</td>
<td>The system cancels the user transaction because of RODM is quiescing.</td>
<td>Retry the request or method later.</td>
</tr>
<tr>
<td>211</td>
<td>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.</td>
<td>Contact the system administrator to restart RODM.</td>
</tr>
<tr>
<td>212</td>
<td>The system cannot complete the transaction because an unrecoverable error occurred. RODM will write a type-2 log record to the RODM log.</td>
<td>Check the content of the log record for information about the transaction that caused the abend. Refer to the Tivoli NetView for z/OS Diagnosis Guide for information on diagnosing abend problems.</td>
</tr>
<tr>
<td>213</td>
<td>The requested function did not complete because an abend occurred when RODM accessed the interface blocks of the application or method.</td>
<td>Check the interface blocks for errors that would cause address exceptions. Refer to the Tivoli NetView for z/OS Diagnosis Guide for information on diagnosing abend problems.</td>
</tr>
<tr>
<td>216</td>
<td>Not used.</td>
<td>None</td>
</tr>
<tr>
<td>240</td>
<td>The RODM transaction did not complete normally. An ABEND might have occurred.</td>
<td>Check the RODM log for information on the specific error that occurred. After correcting the error, repeat the transaction.</td>
</tr>
<tr>
<td>600</td>
<td>An EKG_QueryMultipleSubfields request issued by the correlation function failed for one real object.</td>
<td>Ignore this error if the correlation function performed correctly. If the correlation function did not perform correctly, contact Tivoli customer support.</td>
</tr>
<tr>
<td>601</td>
<td>An EKG_QueryMultipleSubfields request issued by the correlation function failed for one aggregate object.</td>
<td>Ignore this error if the correlation function performed correctly. If the correlation function did not perform correctly, contact Tivoli customer support.</td>
</tr>
<tr>
<td>602</td>
<td>An EKG_ChangeMultipleFields request issued by the correlation function failed for one aggregate object.</td>
<td>Ignore this error if the correlation function performed correctly. If the correlation function did not perform correctly, contact Tivoli customer support.</td>
</tr>
<tr>
<td>603</td>
<td>An EKG_Locate request issued by the correlation function failed for one real object.</td>
<td>Ignore this error if the correlation function performed correctly. If the correlation function did not perform correctly, contact Tivoli customer support.</td>
</tr>
<tr>
<td>603</td>
<td>An EKG_Locate request issued by the correlation function failed for one real object.</td>
<td>Ignore this error if the correlation function performed correctly. If the correlation function did not perform correctly, contact Tivoli customer support.</td>
</tr>
</tbody>
</table>
### Reason Codes for Return Code 12

Table 204. Reason Codes for Return Code 12 (continued)

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Description</th>
<th>Corrective Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>604</td>
<td>An aggregateSystem object was not created by the correlate function because the correlatable value was less than 2 characters in length.</td>
<td>Ignore this error if the correlation function performed correctly. If the correlation function did not perform correctly, contact Tivoli customer support.</td>
</tr>
<tr>
<td>605</td>
<td>An EKG_CreateObject request issued by the correlate function failed for one aggregate object.</td>
<td>Ignore this error if the correlation function performed correctly. If the correlation function did not perform correctly, contact Tivoli customer support.</td>
</tr>
<tr>
<td>606</td>
<td>An EKG_TriggerOIMethod request issued by the correlate function failed to link to a DisplayResourceType for one aggregate object.</td>
<td>Ignore this error if the correlation function performed correctly. If the correlation function did not perform correctly, contact Tivoli customer support.</td>
</tr>
<tr>
<td>45085</td>
<td>Not used.</td>
<td>None</td>
</tr>
<tr>
<td>45086</td>
<td>An error occurred when the objects in a view changed.</td>
<td>Check the RODM log for information on the specific error that occurred. After correcting the error, repeat the transaction.</td>
</tr>
</tbody>
</table>

---

### List of Reason Codes for Each Function

Table 205 lists the function IDs of the RODM API functions and the reason codes returned by each function.

Table 205. Reason Codes for API Functions

<table>
<thead>
<tr>
<th>Function ID</th>
<th>Reason Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common reason codes for user API</td>
<td>0 1 2 3 5 6 8 9 10 13 14 15 16 17 20 23 25 131 199 200 201 212 123 240</td>
</tr>
<tr>
<td>Common reason codes for method API</td>
<td>0 10 18 20 192</td>
</tr>
<tr>
<td>1101</td>
<td>30 84 109 127 128 130 179 198 207</td>
</tr>
<tr>
<td>1102</td>
<td>180 198</td>
</tr>
<tr>
<td>1201</td>
<td>35 36 37 38 202</td>
</tr>
<tr>
<td>1202</td>
<td>85 202</td>
</tr>
<tr>
<td>1302</td>
<td>24 52 86 87 121 122 124 136 140 163 165 186 32768 32769 32770 32772</td>
</tr>
<tr>
<td>1303</td>
<td>24 52 89 90 136 140 163 165 32768 32769 32770 32772</td>
</tr>
<tr>
<td>1304</td>
<td>52 91 92 93 94 95 96 97 100 121 122 126 139 140 159 160 163 164 165 166</td>
</tr>
<tr>
<td>1305</td>
<td>52 60 98 103 139 140 144 163 164 165 166</td>
</tr>
<tr>
<td>1306</td>
<td>52 100 103 104 122 139 140 148 163 164 165 166 187</td>
</tr>
<tr>
<td>1307</td>
<td>52 60 98 103 106 139 140 144 146 163 164 165 166 187</td>
</tr>
<tr>
<td>1401</td>
<td>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</td>
</tr>
</tbody>
</table>
### Table 205. Reason Codes for API Functions (continued)

<table>
<thead>
<tr>
<th>Function ID</th>
<th>Reason Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1402</td>
<td>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</td>
</tr>
<tr>
<td>1403</td>
<td>26 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</td>
</tr>
<tr>
<td>1404</td>
<td>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</td>
</tr>
<tr>
<td>1405</td>
<td>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</td>
</tr>
<tr>
<td>1406</td>
<td>52 54 56 57 72 73 74 122 133 139 140 141 145 150 163 164 166 214</td>
</tr>
<tr>
<td>1407</td>
<td>24 52 54 56 57 73 74 75 133 136 139 140 145 150 163 164 166 214 220 32768 32769 32770 32772</td>
</tr>
<tr>
<td>1408</td>
<td>52 54 56 57 73 74 75 133 139 140 145 150 163 164 166 214</td>
</tr>
<tr>
<td>1409</td>
<td>22 24 34 52 63 107 109 110 121 122 136 140 150 156 159 163 165 170 177 214 32768 32769 32770 32772</td>
</tr>
<tr>
<td>1410</td>
<td>22 24 52 54 57 81 107 108 111 113 136 140 150 163 170 191 214 32768 32769 32770 32772</td>
</tr>
<tr>
<td>1411</td>
<td>40 41 52 54 56 57 61 62 65 67 70 71 122 139 140 145 150 163 164 165 166</td>
</tr>
<tr>
<td>1412</td>
<td>22 52 54 56 57 61 62 65 67 70 71 81 112 122 135 139 140 150 156 163 164 165 166 173</td>
</tr>
<tr>
<td>1413</td>
<td>22 52 54 56 57 76 77 81 112 122 135 139 140 150 163 164 165 166</td>
</tr>
<tr>
<td>1414</td>
<td>42 52 54 56 57 80 81 82 115 136 139 140 150 163 164 165 166 191 194 214 32768 32769 32770 32771</td>
</tr>
<tr>
<td>1415</td>
<td>11 42 80 81 191 194</td>
</tr>
<tr>
<td>1416</td>
<td>22 52 54 57 77 109 112 122 135 140 150 156 163 173 214</td>
</tr>
<tr>
<td>1417</td>
<td>22 52 54 57 77 109 113 135 140 150 156 163 173 214</td>
</tr>
<tr>
<td>1418</td>
<td>22 52 54 57 77 109 113 135 140 150 156 163 173 214</td>
</tr>
<tr>
<td>1419</td>
<td>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</td>
</tr>
<tr>
<td>1501</td>
<td>19 27 42 52 54 56 57 80 83 136 139 140 143 150 163 164 165 166 194 208</td>
</tr>
<tr>
<td>1502</td>
<td>19 27 52 54 56 57 61 62 83 136 139 140 143 150 163 164 165 166 208</td>
</tr>
<tr>
<td>1503</td>
<td>19 27 52 54 57 83 139 140 150 163 164 165 166 208</td>
</tr>
<tr>
<td>1504</td>
<td>19 27 52 54 56 57 83 139 140 150 163 164 165 166 208</td>
</tr>
</tbody>
</table>
Table 205. Reason Codes for API Functions  (continued)

<table>
<thead>
<tr>
<th>Function ID</th>
<th>Reason Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1505</td>
<td>19 27 56 139 164 166 208</td>
</tr>
<tr>
<td>1506</td>
<td>19 27 52 54 56 57 83 139 140 150 163 164 165 166 208</td>
</tr>
<tr>
<td>1507</td>
<td>19 22 27 44 54 57 83 183 208</td>
</tr>
<tr>
<td>1508</td>
<td>27 52 56 83 139 140 150 163 164 165 166 223 600 601</td>
</tr>
<tr>
<td>1509</td>
<td>139 164 166 224 225 227 228 229 603</td>
</tr>
<tr>
<td>1510</td>
<td>46 47 120</td>
</tr>
<tr>
<td>1600</td>
<td>21 79 83 117</td>
</tr>
<tr>
<td>2001</td>
<td>27 83</td>
</tr>
<tr>
<td>2002</td>
<td>21 52 54 118</td>
</tr>
<tr>
<td>2004</td>
<td>27 83 201 203 204</td>
</tr>
<tr>
<td>2005</td>
<td>22 50 157 158 174 175 181 182</td>
</tr>
<tr>
<td>2006</td>
<td>28 29 33 193 45086</td>
</tr>
<tr>
<td>2008</td>
<td>28 29 33</td>
</tr>
<tr>
<td>2011</td>
<td>19 27 54 169 208</td>
</tr>
<tr>
<td>EKGWAIT</td>
<td>209 210</td>
</tr>
</tbody>
</table>

List of Functions for Each Reason Code

Table 206 lists the RODM reason codes and the function IDs of the RODM API functions that return each reason code. See Table 207 on page 480 to resolve a function ID to its function name.

Table 206. Function IDs for Each Reason Code

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Function ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Common reason code for user API and method API.</td>
</tr>
<tr>
<td>1</td>
<td>Common reason code for user API.</td>
</tr>
<tr>
<td>2</td>
<td>Common reason code for user API.</td>
</tr>
<tr>
<td>3</td>
<td>Common reason code for user API.</td>
</tr>
<tr>
<td>5</td>
<td>Common reason code for user API.</td>
</tr>
<tr>
<td>6</td>
<td>Common reason code for user API.</td>
</tr>
<tr>
<td>8</td>
<td>Common reason code for user API.</td>
</tr>
<tr>
<td>9</td>
<td>Common reason code for user API.</td>
</tr>
<tr>
<td>10</td>
<td>Common reason code for user API and method API.</td>
</tr>
<tr>
<td>11</td>
<td>1416</td>
</tr>
<tr>
<td>13</td>
<td>Common reason code for user API.</td>
</tr>
<tr>
<td>14</td>
<td>Common reason code for user API.</td>
</tr>
<tr>
<td>15</td>
<td>Common reason code for user API.</td>
</tr>
<tr>
<td>16</td>
<td>Common reason code for user API.</td>
</tr>
<tr>
<td>17</td>
<td>Common reason code for user API.</td>
</tr>
</tbody>
</table>
Table 206. Function IDs for Each Reason Code (continued)

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Function ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>Common reason code for method API.</td>
</tr>
<tr>
<td>19</td>
<td>1501 1502 1503 1504 1505 1506 1507 2011</td>
</tr>
<tr>
<td>20</td>
<td>Common reason code for user API and method API.</td>
</tr>
<tr>
<td>21</td>
<td>1600 2002</td>
</tr>
<tr>
<td>22</td>
<td>1409 1410 1412 1413 1417 1418 1507 2005</td>
</tr>
<tr>
<td>23</td>
<td>Common reason code for user API.</td>
</tr>
<tr>
<td>24</td>
<td>1302 1303 1401 1402 1405 1407 1409 1410 1419</td>
</tr>
<tr>
<td>25</td>
<td>Common reason code for user API.</td>
</tr>
<tr>
<td>26</td>
<td>1401 1402 1403 1404 1419</td>
</tr>
<tr>
<td>27</td>
<td>1501 1502 1503 1504 1505 1506 1507 1508 2001 2004 2011</td>
</tr>
<tr>
<td>28</td>
<td>2006 2008</td>
</tr>
<tr>
<td>29</td>
<td>2006 2008</td>
</tr>
<tr>
<td>30</td>
<td>1101</td>
</tr>
<tr>
<td>33</td>
<td>2006 2008</td>
</tr>
<tr>
<td>34</td>
<td>1409</td>
</tr>
<tr>
<td>35</td>
<td>1201</td>
</tr>
<tr>
<td>36</td>
<td>1201</td>
</tr>
<tr>
<td>37</td>
<td>1202</td>
</tr>
<tr>
<td>38</td>
<td>1201</td>
</tr>
<tr>
<td>39</td>
<td>1402 1404</td>
</tr>
<tr>
<td>40</td>
<td>1411</td>
</tr>
<tr>
<td>41</td>
<td>1411</td>
</tr>
<tr>
<td>42</td>
<td>1401 1402 1415 1416 1419 1501</td>
</tr>
<tr>
<td>44</td>
<td>1507</td>
</tr>
<tr>
<td>46</td>
<td>1510</td>
</tr>
<tr>
<td>47</td>
<td>1510</td>
</tr>
<tr>
<td>50</td>
<td>2005</td>
</tr>
<tr>
<td>52</td>
<td>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</td>
</tr>
<tr>
<td>54</td>
<td>1401 1402 1403 1404 1405 1406 1407 1408 1410 1411 1412 1413 1415 1417 1418 1419 1501 1502 1503 1504 1506 1507 2002 2011</td>
</tr>
<tr>
<td>56</td>
<td>1401 1402 1403 1404 1405 1406 1407 1408 1411 1412 1413 1415 1419 1501 1502 1504 1505 1506 1508</td>
</tr>
<tr>
<td>57</td>
<td>1401 1402 1403 1404 1405 1406 1407 1408 1410 1411 1412 1413 1415 1417 1418 1419 1501 1502 1503 1504 1506 1507</td>
</tr>
<tr>
<td>60</td>
<td>1305 1307</td>
</tr>
<tr>
<td>Reason Code</td>
<td>Function ID</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>61</td>
<td>1403 1404 1411 1502</td>
</tr>
<tr>
<td>62</td>
<td>1403 1404 1411 1502</td>
</tr>
<tr>
<td>63</td>
<td>1409</td>
</tr>
<tr>
<td>65</td>
<td>1401 1402 1403 1404 1411 1419</td>
</tr>
<tr>
<td>66</td>
<td>1401 1402 1403 1404 1419</td>
</tr>
<tr>
<td>67</td>
<td>1401 1402 1403 1404 1411 1419</td>
</tr>
<tr>
<td>70</td>
<td>1403 1404 1411</td>
</tr>
<tr>
<td>71</td>
<td>1403 1404 1411</td>
</tr>
<tr>
<td>72</td>
<td>1405 1406</td>
</tr>
<tr>
<td>73</td>
<td>1405 1406 1407 1408</td>
</tr>
<tr>
<td>74</td>
<td>1405 1406 1407 1408</td>
</tr>
<tr>
<td>75</td>
<td>1407 1408</td>
</tr>
<tr>
<td>76</td>
<td>1412 1413</td>
</tr>
<tr>
<td>77</td>
<td>1412 1413 1417 1418</td>
</tr>
<tr>
<td>79</td>
<td>1600</td>
</tr>
<tr>
<td>80</td>
<td>1401 1402 1415 1416 1419 1501</td>
</tr>
<tr>
<td>81</td>
<td>1401 1402 1403 1404 1410 1412 1415 1416 1419</td>
</tr>
<tr>
<td>82</td>
<td>1415</td>
</tr>
<tr>
<td>83</td>
<td>1501 1502 1503 1504 1506 1507 1508 1600 2001 2004</td>
</tr>
<tr>
<td>84</td>
<td>1101</td>
</tr>
<tr>
<td>85</td>
<td>1202</td>
</tr>
<tr>
<td>86</td>
<td>1302</td>
</tr>
<tr>
<td>87</td>
<td>1302</td>
</tr>
<tr>
<td>89</td>
<td>1303</td>
</tr>
<tr>
<td>90</td>
<td>1303</td>
</tr>
<tr>
<td>91</td>
<td>1304</td>
</tr>
<tr>
<td>92</td>
<td>1304</td>
</tr>
<tr>
<td>93</td>
<td>1304</td>
</tr>
<tr>
<td>94</td>
<td>1304</td>
</tr>
<tr>
<td>95</td>
<td>1304</td>
</tr>
<tr>
<td>96</td>
<td>1304</td>
</tr>
<tr>
<td>97</td>
<td>1304</td>
</tr>
<tr>
<td>98</td>
<td>1305 1307</td>
</tr>
<tr>
<td>100</td>
<td>1304 1306</td>
</tr>
<tr>
<td>103</td>
<td>1305 1306 1307</td>
</tr>
<tr>
<td>104</td>
<td>1306</td>
</tr>
<tr>
<td>106</td>
<td>1307</td>
</tr>
<tr>
<td>107</td>
<td>1409 1410</td>
</tr>
</tbody>
</table>
# Functions for Each Reason Code

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Function ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>108</td>
<td>1410</td>
</tr>
<tr>
<td>109</td>
<td>1101 1409 1417 1418</td>
</tr>
<tr>
<td>110</td>
<td>1409</td>
</tr>
<tr>
<td>111</td>
<td>1410</td>
</tr>
<tr>
<td>112</td>
<td>1412 1417</td>
</tr>
<tr>
<td>113</td>
<td>1410 1413 1418</td>
</tr>
<tr>
<td>115</td>
<td>1415</td>
</tr>
<tr>
<td>117</td>
<td>1600</td>
</tr>
<tr>
<td>118</td>
<td>2002</td>
</tr>
<tr>
<td>120</td>
<td>1510</td>
</tr>
<tr>
<td>121</td>
<td>1302 1304 1409</td>
</tr>
<tr>
<td>122</td>
<td>1302 1304 1306 1401 1402 1403 1404 1405 1406 1409 1411 1412 1417 1419</td>
</tr>
<tr>
<td>124</td>
<td>1302</td>
</tr>
<tr>
<td>126</td>
<td>1304</td>
</tr>
<tr>
<td>127</td>
<td>1101</td>
</tr>
<tr>
<td>128</td>
<td>1101</td>
</tr>
<tr>
<td>130</td>
<td>1101</td>
</tr>
<tr>
<td>131</td>
<td>Common reason code for user API.</td>
</tr>
<tr>
<td>133</td>
<td>1401 1402 1405 1406 1407 1408 1419</td>
</tr>
<tr>
<td>134</td>
<td>1401 1402 1419</td>
</tr>
<tr>
<td>135</td>
<td>1401 1402 1403 1404 1412 1413 1417 1418 1419</td>
</tr>
<tr>
<td>136</td>
<td>1302 1303 1401 1402 1405 1407 1409 1410 1415 1419 1501 1502</td>
</tr>
<tr>
<td>138</td>
<td>1402 1404</td>
</tr>
<tr>
<td>139</td>
<td>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</td>
</tr>
<tr>
<td>140</td>
<td>1302 1303 1304 1305 1306 1307 1401 1402 1403 1404 1405 1406 1409 1410 1411 1412 1413 1415 1417 1418 1419 1501 1502 1503 1504 1506 1508</td>
</tr>
<tr>
<td>141</td>
<td>1405 1406</td>
</tr>
<tr>
<td>142</td>
<td>1401 1402 1403 1404 1419</td>
</tr>
<tr>
<td>143</td>
<td>1501 1502</td>
</tr>
<tr>
<td>144</td>
<td>1305 1307</td>
</tr>
<tr>
<td>145</td>
<td>1401 1402 1403 1404 1405 1406 1407 1408 1411 1419</td>
</tr>
<tr>
<td>146</td>
<td>1307</td>
</tr>
<tr>
<td>147</td>
<td>1401 1402 1403 1404 1419</td>
</tr>
<tr>
<td>148</td>
<td>1306</td>
</tr>
</tbody>
</table>
## Functions for Each Reason Code

Table 206. Function IDs for Each Reason Code (continued)

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Function ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>150</td>
<td>1401 1402 1403 1404 1405 1406 1407 1408 1409 1410 1411 1412 1413 1415 1417 1418 1419 1501 1502 1503 1504 1506 1508</td>
</tr>
<tr>
<td>156</td>
<td>1409 1412 1417</td>
</tr>
<tr>
<td>157</td>
<td>2005</td>
</tr>
<tr>
<td>158</td>
<td>2005</td>
</tr>
<tr>
<td>159</td>
<td>1304 1409</td>
</tr>
<tr>
<td>160</td>
<td>1304</td>
</tr>
<tr>
<td>163</td>
<td>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</td>
</tr>
<tr>
<td>164</td>
<td>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</td>
</tr>
<tr>
<td>165</td>
<td>1302 1303 1304 1305 1306 1307 1401 1402 1403 1404 1405 1406 1407 1408 1411 1412 1413 1415 1419 1501 1502 1503 1504 1506 1508</td>
</tr>
<tr>
<td>166</td>
<td>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</td>
</tr>
<tr>
<td>169</td>
<td>2011</td>
</tr>
<tr>
<td>170</td>
<td>1409 1410</td>
</tr>
<tr>
<td>173</td>
<td>1412 1417</td>
</tr>
<tr>
<td>174</td>
<td>2005</td>
</tr>
<tr>
<td>175</td>
<td>2005</td>
</tr>
<tr>
<td>176</td>
<td>1401 1402 1403 1404 1419</td>
</tr>
<tr>
<td>177</td>
<td>1409</td>
</tr>
<tr>
<td>179</td>
<td>1101</td>
</tr>
<tr>
<td>180</td>
<td>1102</td>
</tr>
<tr>
<td>181</td>
<td>2005</td>
</tr>
<tr>
<td>182</td>
<td>2005</td>
</tr>
<tr>
<td>183</td>
<td>1507</td>
</tr>
<tr>
<td>186</td>
<td>1302</td>
</tr>
<tr>
<td>187</td>
<td>1306 1307</td>
</tr>
<tr>
<td>191</td>
<td>1410 1415 1416</td>
</tr>
<tr>
<td>192</td>
<td>Common reason code for method API.</td>
</tr>
<tr>
<td>193</td>
<td>2006</td>
</tr>
<tr>
<td>194</td>
<td>1401 1402 1415 1416 1419 1501</td>
</tr>
<tr>
<td>195</td>
<td>1401 1402 1403 1404</td>
</tr>
<tr>
<td>198</td>
<td>1101 1102</td>
</tr>
<tr>
<td>199</td>
<td>Common reason code for user API.</td>
</tr>
<tr>
<td>200</td>
<td>Common reason code for user API.</td>
</tr>
<tr>
<td>Reason Code</td>
<td>Function ID</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>201</td>
<td>2004</td>
</tr>
<tr>
<td>202</td>
<td>1201 1202</td>
</tr>
<tr>
<td>203</td>
<td>2004</td>
</tr>
<tr>
<td>204</td>
<td>2004</td>
</tr>
<tr>
<td>207</td>
<td>1101</td>
</tr>
<tr>
<td>208</td>
<td>1501 1502 1503 1504 1505 1506 1507 2011</td>
</tr>
<tr>
<td>209</td>
<td>EKGWAIT</td>
</tr>
<tr>
<td>210</td>
<td>EKGWAIT</td>
</tr>
<tr>
<td>211</td>
<td>Common reason code for user API.</td>
</tr>
<tr>
<td>212</td>
<td>Common reason code for user API.</td>
</tr>
<tr>
<td>213</td>
<td>Common reason code for user API.</td>
</tr>
<tr>
<td>214</td>
<td>1405 1406 1407 1408 1409 1410 1415 1417 1418</td>
</tr>
<tr>
<td>215</td>
<td>1401 1402 1403 1404 1419</td>
</tr>
<tr>
<td>220</td>
<td>1405 1407</td>
</tr>
<tr>
<td>223</td>
<td>1419 1508</td>
</tr>
<tr>
<td>224</td>
<td>1509</td>
</tr>
<tr>
<td>225</td>
<td>1509</td>
</tr>
<tr>
<td>226</td>
<td>1101</td>
</tr>
<tr>
<td>227</td>
<td>1419 1509</td>
</tr>
<tr>
<td>228</td>
<td>1509</td>
</tr>
<tr>
<td>229</td>
<td>1509</td>
</tr>
<tr>
<td>230</td>
<td>1401 1402 1403 1404 1419</td>
</tr>
<tr>
<td>231</td>
<td>1401 1402 1403 1404 1419</td>
</tr>
<tr>
<td>232</td>
<td>1401 1402 1403 1404 1419</td>
</tr>
<tr>
<td>240</td>
<td>Common reason code for user API.</td>
</tr>
<tr>
<td>600</td>
<td>1508</td>
</tr>
<tr>
<td>601</td>
<td>1508</td>
</tr>
<tr>
<td>602</td>
<td>1419</td>
</tr>
<tr>
<td>603</td>
<td>1509</td>
</tr>
<tr>
<td>604</td>
<td>1409</td>
</tr>
<tr>
<td>605</td>
<td>1409</td>
</tr>
<tr>
<td>32768</td>
<td>1302 1303 1401 1402 1405 1407 1409 1410 1415 1419</td>
</tr>
<tr>
<td>32769</td>
<td>1302 1303 1401 1402 1405 1407 1409 1410 1419</td>
</tr>
<tr>
<td>32770</td>
<td>1302 1303 1401 1402 1405 1407 1409 1410 1419</td>
</tr>
<tr>
<td>32771</td>
<td>1401 1415 1419</td>
</tr>
<tr>
<td>32772</td>
<td>1302 1303 1401 1402 1405 1407 1409 1410 1419</td>
</tr>
</tbody>
</table>
## Functions for Each Reason Code

Table 206. Function IDs for Each Reason Code (continued)

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Function ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>45086</td>
<td>2006</td>
</tr>
</tbody>
</table>

### List of Function Names by Function ID

Table 207 lists the RODM API function names by their function ID.

<table>
<thead>
<tr>
<th>Function ID</th>
<th>Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1101</td>
<td>EKG_Connect</td>
</tr>
<tr>
<td>1102</td>
<td>EKG_Disconnect</td>
</tr>
<tr>
<td>1201</td>
<td>EKG_Checkpoint</td>
</tr>
<tr>
<td>1202</td>
<td>EKG_Stop</td>
</tr>
<tr>
<td>1302</td>
<td>EKG_CreateClass</td>
</tr>
<tr>
<td>1303</td>
<td>EKG_DeleteClass</td>
</tr>
<tr>
<td>1304</td>
<td>EKG_CreateField</td>
</tr>
<tr>
<td>1305</td>
<td>EKG_DeleteField</td>
</tr>
<tr>
<td>1306</td>
<td>EKG_CreateSubfield</td>
</tr>
<tr>
<td>1307</td>
<td>EKG_DeleteSubfield</td>
</tr>
<tr>
<td>1401</td>
<td>EKG_ChangeField</td>
</tr>
<tr>
<td>1402</td>
<td>EKG_SwapField</td>
</tr>
<tr>
<td>1403</td>
<td>EKG_ChangeSubfield</td>
</tr>
<tr>
<td>1404</td>
<td>EKG_SwapSubfield</td>
</tr>
<tr>
<td>1405</td>
<td>EKG_LinkTrigger</td>
</tr>
<tr>
<td>1406</td>
<td>EKG_LinkNoTrigger</td>
</tr>
<tr>
<td>1407</td>
<td>EKG_UnLinkTrigger</td>
</tr>
<tr>
<td>1408</td>
<td>EKG_UnLinkNoTrigger</td>
</tr>
<tr>
<td>1409</td>
<td>EKG_CreateObject</td>
</tr>
<tr>
<td>1410</td>
<td>EKG_DeleteObject</td>
</tr>
<tr>
<td>1411</td>
<td>EKG_RevertToInherited</td>
</tr>
<tr>
<td>1412</td>
<td>EKG_AddNotifySubscription</td>
</tr>
<tr>
<td>1413</td>
<td>EKG_DeleteNotifySubscription</td>
</tr>
<tr>
<td>1415</td>
<td>EKG_TriggerNamedMethod</td>
</tr>
<tr>
<td>1416</td>
<td>EKG_TriggerOIMethod</td>
</tr>
<tr>
<td>1417</td>
<td>EKG_AddObjDelSubs</td>
</tr>
<tr>
<td>1418</td>
<td>EKG_DelObjDelSubs</td>
</tr>
<tr>
<td>1419</td>
<td>EKG_ChangeMultipleFields</td>
</tr>
<tr>
<td>1501</td>
<td>EKG_QueryField</td>
</tr>
<tr>
<td>1502</td>
<td>EKG_QuerySubfield</td>
</tr>
<tr>
<td>1503</td>
<td>EKG_QueryEntityStructure</td>
</tr>
<tr>
<td>1504</td>
<td>EKG_QueryFieldStructure</td>
</tr>
<tr>
<td>1505</td>
<td>EKG_QueryFieldID</td>
</tr>
</tbody>
</table>
Table 207. Function Names by Function ID (continued)

<table>
<thead>
<tr>
<th>Function ID</th>
<th>Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1506</td>
<td>EKG_QueryFieldName</td>
</tr>
<tr>
<td>1507</td>
<td>EKG_QueryNotifyQueue</td>
</tr>
<tr>
<td>1508</td>
<td>EKG_QueryMultipleSubfields</td>
</tr>
<tr>
<td>1509</td>
<td>EKG_Locate</td>
</tr>
<tr>
<td>1510</td>
<td>EKG_QueryResponseBlockOverflow</td>
</tr>
<tr>
<td>1600</td>
<td>EKG_ExecuteFunctionList</td>
</tr>
<tr>
<td>2001</td>
<td>EKG_QueryFunctionBlockContents</td>
</tr>
<tr>
<td>2002</td>
<td>EKG_LockObjectList</td>
</tr>
<tr>
<td>2003</td>
<td>EKG_UnlockAll</td>
</tr>
<tr>
<td>2004</td>
<td>EKG_ResponseBlock</td>
</tr>
<tr>
<td>2005</td>
<td>EKG_SendNotification</td>
</tr>
<tr>
<td>2006</td>
<td>EKG_SetReturnCode</td>
</tr>
<tr>
<td>2007</td>
<td>EKG_WhereAmI</td>
</tr>
<tr>
<td>2008</td>
<td>EKG_OutputToLog</td>
</tr>
<tr>
<td>2009</td>
<td>EKG_MessageTriggeredAction</td>
</tr>
<tr>
<td>2011</td>
<td>EKG_QueryObjectName</td>
</tr>
</tbody>
</table>

Table 208 lists the NetView-supplied methods and the reason codes that are returned by each method.

Table 208. Reason Codes for NetView-Supplied Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Reason Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DUIFCATC</td>
<td>45070 45077 45078 45081 45088</td>
</tr>
<tr>
<td>DUIFCCAN</td>
<td>45077 45081 45088</td>
</tr>
<tr>
<td>DUIFCLRT</td>
<td>45070 45071 45077 45078 45081 45083 45088</td>
</tr>
<tr>
<td>DUIFCUAP</td>
<td>45065 45070 45071 45077 45081 45088</td>
</tr>
<tr>
<td>DUIFCUUS</td>
<td>45070 45077 45078 45081 45088</td>
</tr>
<tr>
<td>DUIFECDS</td>
<td>45070 45077 45078 45079 45081 45088</td>
</tr>
<tr>
<td>DUIFFAWS</td>
<td>45088</td>
</tr>
<tr>
<td>DUIFFIRS</td>
<td>45070 45077 45078 45081 45088</td>
</tr>
<tr>
<td>DUIFFRAS</td>
<td>45077 45081 45088</td>
</tr>
<tr>
<td>DUIFFSUS</td>
<td>45070 45077 45078 45081 45088</td>
</tr>
<tr>
<td>DUIFRFDS</td>
<td>45077 45081 45088</td>
</tr>
<tr>
<td>DUIFVCFT</td>
<td>45070 45077 45081 45088</td>
</tr>
<tr>
<td>EKGCTIM</td>
<td>32768 32771 32780</td>
</tr>
<tr>
<td>EKGMIMV</td>
<td>32768 32771 32780</td>
</tr>
<tr>
<td>EKGNEQL</td>
<td>32768 32769 32770 32772 32780 32954</td>
</tr>
<tr>
<td>EKGNLST</td>
<td>32768 32769 32770 32772 32780 32954</td>
</tr>
<tr>
<td>EKGNOTF</td>
<td>32768 32770 32780 32954</td>
</tr>
<tr>
<td>EKGNTHD</td>
<td>32768 32769 32770 32772 32780 32954</td>
</tr>
</tbody>
</table>
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 recommended 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 should 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.
NetView-Supplied Methods

This section provides a brief introduction to the NetView-supplied methods. These methods are provided with RODM to supply specific kinds of functions. You can replace a NetView-supplied method and add locally developed ones. NetView-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 423 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 209 is an example of the data that is returned in the data string.

Table 209. Example User_data Returned with EKGNOTF Notification Method

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>2</td>
<td>34</td>
<td>Total length of SelfDefining string</td>
</tr>
<tr>
<td>002</td>
<td>2</td>
<td>21</td>
<td>Smallint data type code</td>
</tr>
<tr>
<td>004</td>
<td>2</td>
<td>01</td>
<td>Value field indicator</td>
</tr>
<tr>
<td>006</td>
<td>2</td>
<td>10</td>
<td>Value data type flag (Integer)</td>
</tr>
<tr>
<td>008</td>
<td>4</td>
<td>value</td>
<td>Value data (Integer)</td>
</tr>
<tr>
<td>012</td>
<td>2</td>
<td>21</td>
<td>Smallint data type code</td>
</tr>
<tr>
<td>014</td>
<td>2</td>
<td>02</td>
<td>Prev_val field indicator</td>
</tr>
<tr>
<td>016</td>
<td>2</td>
<td>10</td>
<td>Prev_val data type flag (Integer)</td>
</tr>
<tr>
<td>018</td>
<td>4</td>
<td>prev_val</td>
<td>Prev_val data (Integer)</td>
</tr>
<tr>
<td>022</td>
<td>2</td>
<td>21</td>
<td>Smallint data type code</td>
</tr>
<tr>
<td>024</td>
<td>2</td>
<td>03</td>
<td>Timestamp field indicator</td>
</tr>
<tr>
<td>026</td>
<td>2</td>
<td>27</td>
<td>Timestamp data type flag (TimeStamp)</td>
</tr>
<tr>
<td>028</td>
<td>8</td>
<td>timestamp</td>
<td>Timestamp data (TimeStamp)</td>
</tr>
</tbody>
</table>

The NetView-supplied notification methods 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 should 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 executed.
NetView-Supplied Methods

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 executed. After the object’s notifications are processed, any notification subscriptions assigned to the same field in the primary parent are executed.

Methods that perform comparison operations to determine if a notification should be 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>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>2</td>
<td>8</td>
<td>Total length of SelfDefining string</td>
</tr>
<tr>
<td>002</td>
<td>2</td>
<td>21</td>
<td>Smallint data type code</td>
</tr>
<tr>
<td>004</td>
<td>2</td>
<td>1 or 2</td>
<td>Two byte integer (1=always, 2=once)</td>
</tr>
<tr>
<td>006</td>
<td>2</td>
<td>21</td>
<td>Smallint data type code</td>
</tr>
<tr>
<td>008</td>
<td>2</td>
<td>1 or 2</td>
<td>Two byte integer (1=notify only if new value is different from previous value, 2=notify always)</td>
</tr>
</tbody>
</table>

**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>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>2</td>
<td>14</td>
<td>Total length of SelfDefining string</td>
</tr>
<tr>
<td>002</td>
<td>2</td>
<td>21</td>
<td>Smallint data type code</td>
</tr>
<tr>
<td>004</td>
<td>2</td>
<td>1 or 2</td>
<td>Two byte integer (1=always, 2=once)</td>
</tr>
</tbody>
</table>
Table 211. EKGNEQL Long-lived-parameter Description (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>006</td>
<td>2</td>
<td>21</td>
<td>Smallint data type code</td>
</tr>
<tr>
<td>008</td>
<td>2</td>
<td>1 or 2</td>
<td>Two byte integer (1=notify only if new value is different from previous value, 2=notify always)</td>
</tr>
<tr>
<td>010</td>
<td>2</td>
<td>26</td>
<td>Smallint data type code (FieldID)</td>
</tr>
<tr>
<td>012</td>
<td>4</td>
<td>Field_ID</td>
<td>Field_ID of test value</td>
</tr>
</tbody>
</table>

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 212. EKGNLST Long-lived-parameter Description

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>2</td>
<td>14+(N*6)</td>
<td>Total length of SelfDefining string</td>
</tr>
<tr>
<td>002</td>
<td>2</td>
<td>21</td>
<td>Smallint data type code</td>
</tr>
<tr>
<td>004</td>
<td>2</td>
<td>1 or 2</td>
<td>Two byte integer (1=always, 2=once)</td>
</tr>
<tr>
<td>006</td>
<td>2</td>
<td>21</td>
<td>Smallint data type code</td>
</tr>
<tr>
<td>008</td>
<td>2</td>
<td>1 or 2</td>
<td>Two byte integer (1=notify only if new value is different from previous value, 2=notify always)</td>
</tr>
<tr>
<td>010</td>
<td>2</td>
<td>10</td>
<td>Smallint data type code (Integer)</td>
</tr>
<tr>
<td>012</td>
<td>4</td>
<td>N (range 0..n)</td>
<td>Number of following Field_IDs</td>
</tr>
<tr>
<td>016</td>
<td>2</td>
<td>26</td>
<td>Smallint data type code (FieldID)</td>
</tr>
<tr>
<td>018</td>
<td>4</td>
<td>Field_ID</td>
<td>Field_ID of first test value</td>
</tr>
</tbody>
</table>

: 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 should provide three options.
NetView-Supplied Methods

- 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 213. EKGNTHD Long-lived-parameter Description

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>2</td>
<td>20 or 26</td>
<td>Total length of SelfDefining string</td>
</tr>
<tr>
<td>002</td>
<td>2</td>
<td>21</td>
<td>Smallint data type code</td>
</tr>
<tr>
<td>004</td>
<td>2</td>
<td>1 or 2</td>
<td>Two byte integer (1=always, 2=once)</td>
</tr>
<tr>
<td>006</td>
<td>2</td>
<td>21</td>
<td>Smallint data type code</td>
</tr>
<tr>
<td>008</td>
<td>2</td>
<td>1 or 2</td>
<td>Two byte integer (1=notify only if new value is different from previous value, 2=notify always)</td>
</tr>
<tr>
<td>010</td>
<td>2</td>
<td>10</td>
<td>Integer data type code</td>
</tr>
<tr>
<td>012</td>
<td>4</td>
<td>1, 2, or 3</td>
<td>Option code (1=upper bound, 2=lower bound, 3=range)</td>
</tr>
<tr>
<td>016</td>
<td>2</td>
<td>26</td>
<td>Smallint data type code (FieldID)</td>
</tr>
<tr>
<td>018</td>
<td>4</td>
<td>Field_ID</td>
<td>For 1, upper bound; for 2 or 3, lower bound</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>: Next parameters for option code 3 only</td>
</tr>
<tr>
<td>022</td>
<td>2</td>
<td>26</td>
<td>Smallint data type code (FieldID)</td>
</tr>
<tr>
<td>024</td>
<td>4</td>
<td>Field_ID</td>
<td>For 3, upper bound</td>
</tr>
</tbody>
</table>

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 could be passed, in order to communicate old and new value information from the field associated with this method. This could 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 214. EKGCTIM Long-lived-parameter Description

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>2</td>
<td>12</td>
<td>Total length of SelfDefining string</td>
</tr>
<tr>
<td>002</td>
<td>2</td>
<td>26</td>
<td>Smallint data type code (FieldID)</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Field_ID</td>
<td>Field containing method name</td>
</tr>
<tr>
<td>008</td>
<td>2</td>
<td>26</td>
<td>Smallint data type code (FieldID)</td>
</tr>
<tr>
<td>010</td>
<td>4</td>
<td>Field_ID</td>
<td>Field containing Short-lived-parameter list as a SelfDefining string</td>
</tr>
</tbody>
</table>

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 could be negative causing the designated field value to be decremented.

Table 215. EKGMIMV Long-lived-parameter Description

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>2</td>
<td>12</td>
<td>Total length of SelfDefining string</td>
</tr>
<tr>
<td>002</td>
<td>2</td>
<td>26</td>
<td>Smallint data type code (FieldID)</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>Field_ID</td>
<td>Field to be incremented</td>
</tr>
<tr>
<td>008</td>
<td>2</td>
<td>26</td>
<td>Smallint data type code (FieldID)</td>
</tr>
<tr>
<td>010</td>
<td>4</td>
<td>Field_ID</td>
<td>Field containing increment value</td>
</tr>
</tbody>
</table>

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, once installed in RODM, could be used in either manner.

RODM Object-Independent Methods

EKGSPP1 - Send a command to NetView
The EKGSPP1 method is one of the services in the RODM automation platform. See "Chapter 8. Using the RODM Automation Platform" on page 191 for more information about automation tasks using NetView. An extensive RODM automation scenario using the EKGSPP1 method and the automation platform is contained in the chapter entitled the Tivoli NetView for z/OS Automation Guide.
NetView-Supplied Methods

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 DSIQTSK1 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 executed. 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSIST</td>
<td>Send the command to an operator</td>
</tr>
<tr>
<td>NOASSIST</td>
<td>Issue the command without sending it to an operator</td>
</tr>
<tr>
<td>null or blanks</td>
<td>Issue the command without sending it to an operator</td>
</tr>
</tbody>
</table>

This value has a maximum of 8 characters.

**TASKINFO_CHARVAR**
This data item specifies whether the command is executed by a specific NetView autotask. Valid values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 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 invoke the EKGSPPI method using the RODM load function. Figure 94 on page 490 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.
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.

You should only use 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
- DUIFCAAD
- DUIFCAPC
- DUIFCASB
- DUIFCATC
- DUIFCCAP
- DUIFCDTC
- DUIFCDUC
- DUIFCGRA
- DUIFCGRT
- DUIFCGR2
- DUIFCGR3
- DUIFCLSR
- DUIFCLSR2
- DUIFCLSR3
- DUIFCMUU
- DUIFCRDC
- DUIFCRTP
- DUIFCRTU
- DUIFCRUC
- DUIFCSRT
- DUIFCURA
- DUIFCUTC
- DUIFEGSN
- DUIFITKN
- DUIFRAIP
- DUIFRRTP
- DUIFVCVT

---

**Figure 94. Example RODM Load Function Primitive Statement to Invoke EKGSPPI**

```c
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 --
);```

---

NetView-Supplied Methods
DUIFCCAN - Clear All Notes
This object-independent method can be invoked 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 208 on page 481](#) 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 invoked 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.
NetView-Supplied Methods

- 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 95 is an example of triggering the DUIFCLRT method using a RODM load function primitive statement.

```c
OP DUIFCLRT INVOKED_WITH (SELFDEFINING)
   (
      (SMALLINT) 1
      (CHARVAR) 'Display_Resource_Type_Class.DUIXC_RTN_INN_DOMAIN_AGG'
      (OBJECTID) 'View_Information_Reference_Class'.
      '1.3.18.0.0.2150_Reference'
   );
```

Figure 95. 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 216, followed by a complete description of each item.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>Data Type</th>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Link or unlink</td>
<td>CHARVAR or SMALLINT</td>
<td>Required</td>
</tr>
<tr>
<td>2</td>
<td>Resource object</td>
<td>CHARVAR or OBJECTID</td>
<td>Optional ¹</td>
</tr>
<tr>
<td>3</td>
<td>Display resource type</td>
<td>CHARVAR or OBJECTID</td>
<td>Required</td>
</tr>
<tr>
<td>4</td>
<td>View information reference object</td>
<td>CHARVAR or OBJECTID</td>
<td>Optional ¹</td>
</tr>
</tbody>
</table>

**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>
<thead>
<tr>
<th>Operation</th>
<th>CHARVAR</th>
<th>SMALLINT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link resources</td>
<td>LINK</td>
<td>1</td>
</tr>
<tr>
<td>Unlink resources</td>
<td>UNLINK</td>
<td>2</td>
</tr>
</tbody>
</table>

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:

   ```c
   (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 quotes and the object name within single quotes, 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 quotes. 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 quote (') character, use two single quotes to specify the single quote. For example, if the name of an object was Greg'sObject, use the following code:

(CHARVAR)'Class.Greg'sObject'

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.

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 208 on page 481 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 invoked 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, via 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 EKGLIIILM initialization methods. Do not trigger DUIFCUAP using another method. Figure 96 is an example of triggering the DUIFCUAP method using a RODM load function primitive statement.

```c
OP DUIFCUAP INVOKED_WITH (SELFDEFINING)
  ((CHARVAR)'LINK'
   (CHARVAR)'GMFHS_Aggregate_Objects_Class.ETHERNET'
   (CHARVAR)'GMFHS_Aggregate_Objects_Class.WESTCTR');
```

**Figure 96. 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>
<thead>
<tr>
<th>Operation</th>
<th>CHARVAR</th>
<th>SMALLINT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link resources</td>
<td>LINK</td>
<td>1</td>
</tr>
<tr>
<td>Unlink resources</td>
<td>UNLINK</td>
<td>2</td>
</tr>
</tbody>
</table>

- 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 quotes and the object name within single quotes, 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 quotes. 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 quote (') character, use two single quotes to specify the single quote. 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 208 on page 481](#) 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:** This method should be used by any application that needs to change the UserStatus field value of any descendent class of GMFHS_Displayable_Objects_Parent_Class, including the GMFHS_Manged_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

Refer to the [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'8000000080000000D7C59F1404040'
'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'8000000000000000D7C59F1404040'
'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 219. Output from DUIFCUUS Method

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>Total length of SelfDefining string</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30</td>
<td>Data type AnonymousVar</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>Length of AnonymousVar data</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>Value of timestamp subfield of UserStatus field after update</td>
<td></td>
</tr>
</tbody>
</table>

If this method encounters errors, it sets a return and reason code and writes a type 1 record to the RODM log. Table 208 on page 481 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 'GMHS_Managed_Real_Objects_Class'.'TRMD401'.'ChangeDisplayStatus'
  INVOKED_WITH (SELFDEFINING)
    ((ANONYMOUSVAR)X'000000810011F9F3F0F4F0F2F1F4F3F0F0E9F0F0F0F00000');
```

**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 220. Output from DUIFECDS Method**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>2</td>
<td>12</td>
<td>Total length of SelfDefining string</td>
</tr>
<tr>
<td>002</td>
<td>2</td>
<td>30</td>
<td>Data type AnonymousVar</td>
</tr>
<tr>
<td>004</td>
<td>2</td>
<td>16</td>
<td>Length of AnonymousVar data</td>
</tr>
<tr>
<td>006</td>
<td>4</td>
<td>16</td>
<td>Integer new value of DisplayStatus field</td>
</tr>
<tr>
<td>010</td>
<td>4</td>
<td>16</td>
<td>Integer previous value of DisplayStatus field</td>
</tr>
<tr>
<td>014</td>
<td>8</td>
<td>16</td>
<td>Value of timestamp subfield of DisplayStatus field after update</td>
</tr>
</tbody>
</table>

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 208 on page 481](#) lists the reason codes that can be returned by this method.
DUIFFAWS - Aggregation Warm Start Method
This is an object-independent method that is invoked to initialize the fields related to status aggregation in the real and aggregate objects in the RODM data cache.
GMFHS invokes 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 208 on page 483 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).
NetView-Supplied Methods

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'00010010F9DC34AAF9F3F0F2F1F4F3F0F0E9F0F0F0001'

Output: If this method encounters errors, it sets a return and reason code and writes a type 1 record to the RODM log. [Table 208 on page 481] lists the reason codes that can be returned by this method.

DUIFRAS - 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 DUIFRAS 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 208 on page 481] 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’00010010F9DC34AA0F3F0F0F0F2F1F4F3F0F0E9F0F0F0F0'

**Output:** If this method encounters errors, it sets a return and reason code and writes a type 1 record to the RODM log. Table 208 on page 481 lists the reason codes that can be returned by this method.

**DUIFRFDS - Refresh DisplayStatus Change Method DUIFCRDC**

This object-independent method can be invoked 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 invoked 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 208 on page 481 lists the reason codes that can be returned by this method.

**DUIFVCFT - Change Exception State**

This object-independent method can be invoked by a user method to change the exception state of an object.

**Function:** The user method that invokes method DUIFVCFT is specified by the USRXMETH keyword in DisplayStatus mapping table DUIFSMT. Sample user methods DUIFCUXM and DUIFCUX2 invoke 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 221 lists the input parameters for method DUIFVCFT:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Length of Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total_Length</td>
<td>SMALLINT</td>
<td>2</td>
</tr>
<tr>
<td>Data_Type</td>
<td>SMALLINT</td>
<td>2</td>
</tr>
<tr>
<td>Data_Length</td>
<td>SMALLINT</td>
<td>2</td>
</tr>
<tr>
<td>Resource_Object_ID</td>
<td>OBJECTID</td>
<td>8</td>
</tr>
<tr>
<td>Requested_exception_status</td>
<td>INTEGER</td>
<td>4</td>
</tr>
</tbody>
</table>

**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 208 on page 481 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 112 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 invoked 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 222 lists the input parameters for method DUIFVINS:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Length of Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable_change_status</td>
<td>SMALLINT</td>
<td>2</td>
</tr>
<tr>
<td>rule</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>notification_method</td>
<td>OBJECTID</td>
<td>8</td>
</tr>
<tr>
<td>class</td>
<td>CLASSID</td>
<td>4</td>
</tr>
</tbody>
</table>
NetView-Supplied Methods

Table 222. Input Values for DUIFVINS (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Length of Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>field</td>
<td>FIELDID</td>
<td>4</td>
</tr>
</tbody>
</table>

**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 should be issued even if the field is changed to its previous value.
- 1: Used if the prev_val subfield does exist on the field, and if a VCN should not be issued when the field is changed to its previous value.

**rule**

The criteria used to determine if a field change should result 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 were 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. Refer to "Defining Exception View Objects and Criteria" on page 102 for more information.

5000  LU_CHANGE: Send a view update if the field changes on an LU-type object and its 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 should be installed.

**field**  The field ID on which DUIFVNOT should be installed.

The following is an example of a RODM loader statement to invoke DUIFVINS:

```c
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 208 on page 483] lists the reason codes that can be returned by this method.
NetView-Supplied Methods
Part 5. Appendixes
Appendix A. RODM Tools

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

The RODMView function is an interactive 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 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. See “RODM Unload Function” on page 544 for more information.

FLCARODM provides a fast and efficient REXX interface to RODM. (FLCARODM was formerly known as the RODM Access Facility or MultiSystem Manager Access.) FLCARODM enables you to create, update, and delete objects using a NetView CLIST written in REXX. FLCARODM provides a simple interface to RODM and it enables you to exploit the processing advantages of issuing batched requests to RODM. See “FLCARODM” on page 548 for more information.

BLDVIEWS is a tool that is used for defining custom views which match your network layout and your preferred style of monitoring it. 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 (NMC) for MultiSystem Manager discovered network resource objects. See “BLDVIEWS” on page 592 for more information.

Visual BLDVIEWS (VBV) is an application that simplifies the management of RODM views and information. VBV provides a graphical, drag-and-drop interface to the BLDVIEWS tool and the RODMView tool. See the VBV online help for more information.

Some panels in this appendix show GMFHS information.

RODMView

This section explains how to use RODMView. The following topics are covered:
- Navigating with RODMView
- RODMView restrictions
- Starting RODMView
- Using the RODMView functions
Navigating within RODMView

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 there is a command line at the bottom of each panel.

Navigating Using Menus

RODMView has a main menu panel, which is illustrated in Figure 98 on page 511. 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 223.

Table 223. Accelerator PF Keys and Options

<table>
<thead>
<tr>
<th>PF Key</th>
<th>Option</th>
<th>Panel</th>
</tr>
</thead>
<tbody>
<tr>
<td>PF13</td>
<td>Option 1</td>
<td>Access and control</td>
</tr>
<tr>
<td>PF14</td>
<td>Option 2</td>
<td>Simple query</td>
</tr>
<tr>
<td>PF15</td>
<td>Option 3</td>
<td>Compound query</td>
</tr>
<tr>
<td>PF16</td>
<td>Option 4</td>
<td>Locate actions</td>
</tr>
<tr>
<td>PF17</td>
<td>Option 5</td>
<td>Link/unlink</td>
</tr>
<tr>
<td>PF18</td>
<td>Option 6</td>
<td>Change field</td>
</tr>
<tr>
<td>PF19</td>
<td>Option 7</td>
<td>Subfield actions</td>
</tr>
<tr>
<td>PF20</td>
<td>Option 8</td>
<td>Create actions</td>
</tr>
<tr>
<td>PF21</td>
<td>Option 9</td>
<td>Delete actions</td>
</tr>
<tr>
<td>PF22</td>
<td>Option 10</td>
<td>Method actions</td>
</tr>
</tbody>
</table>

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 224 lists the PF keys and corresponding functions:

Table 224. PF Key Function

<table>
<thead>
<tr>
<th>PF Key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PF1</td>
<td>Displays help information.</td>
</tr>
<tr>
<td>PF2</td>
<td>Terminates the command and exits.</td>
</tr>
<tr>
<td>PF3</td>
<td>Returns control to the previous panel.</td>
</tr>
<tr>
<td>PF4</td>
<td>Clears query input fields.</td>
</tr>
</tbody>
</table>
**Table 224. PF Key Function (continued)**

<table>
<thead>
<tr>
<th>PF Key</th>
<th>Function</th>
</tr>
</thead>
</table>
| 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 executes 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 (wildcards) in the name.

- Only the query function supports wildcards.

- 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 will exit and the previous copy of RODMView will regain control.

- You can restrict certain keywords of the EKGVACTM command processor.

**Reference:** For a list of keywords that can be protected, see the 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 97 on page 510.
Starting RODMView

The RODMView main menu is displayed as shown in Figure 98. From the RODMView menu you can choose any of the available functions. There are three ways to choose an option:

- 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.

Figure 97. RODMView NetView Command Line Call

The RODMView main menu is displayed as shown in Figure 98.

Figure 98. RODMView Main Menu — EKGVMMNI

From the RODMView menu you can choose any of the available functions. There are three ways to choose an option:

- 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 99.

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 wildcard when issuing queries. Note that the asterisk (*) is valid as part of an object name, and might not be suitable for use as a wildcard. The connect function assigns the value to the wildcard. 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 there is a system authorization facility 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.

Figure 99. 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 wildcard when issuing queries. Note that the asterisk (*) is valid as part of an object name, and might not be suitable for use as a wildcard. The connect function assigns the value to the wildcard. 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 there is a system authorization facility 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.
Signing On To RODM

Once 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  10/18/97 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=Prev_Cmd

Figure 100. RODMView Message for a Successful Connection
```

The message line in the lower-left corner of Figure 100 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:

```
EKGV0037E 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 454 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 101 on page 513.
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 102**. Note that objects created on the EKG_User class represent users that are currently signed on to RODM.

```
<table>
<thead>
<tr>
<th>RODM name</th>
<th>RODMNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>User ID . . .</td>
<td>RODMUSER</td>
</tr>
<tr>
<td>SystemView class name</td>
<td>EKG_User</td>
</tr>
<tr>
<td>Class name</td>
<td>EKG_User</td>
</tr>
<tr>
<td>Class ID</td>
<td></td>
</tr>
<tr>
<td>Object name</td>
<td>RODMUSER</td>
</tr>
<tr>
<td>Object ID</td>
<td>(Hexadecimal value)</td>
</tr>
<tr>
<td>SystemView field name</td>
<td></td>
</tr>
<tr>
<td>Field name</td>
<td></td>
</tr>
<tr>
<td>Field ID</td>
<td></td>
</tr>
<tr>
<td>Level of field detail</td>
<td>DATA (Struct, Data, Hex)</td>
</tr>
<tr>
<td>Level of subfield detail</td>
<td>NONE (Struct, Data, Hex, None)</td>
</tr>
<tr>
<td>Maximum lines returned</td>
<td>5000</td>
</tr>
<tr>
<td>Display field IDs . . .</td>
<td>N (Y, N) Display extended field info N (Y, N)</td>
</tr>
</tbody>
</table>
```

**Figure 101. RODMView Query Panel — EKGVQUEI**

Note that, except for SystemView class and field names, RODM is case-sensitive for class, object, and field names.

```
<table>
<thead>
<tr>
<th>RODM name</th>
<th>RODMNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>User ID . . .</td>
<td>RODMUSER</td>
</tr>
<tr>
<td>SystemView class name</td>
<td>EKG_User</td>
</tr>
<tr>
<td>Class name</td>
<td>EKG_User</td>
</tr>
<tr>
<td>Class ID</td>
<td></td>
</tr>
<tr>
<td>Object name</td>
<td>RODMUSER</td>
</tr>
<tr>
<td>Object ID</td>
<td>(Hexadecimal value)</td>
</tr>
<tr>
<td>SystemView field name</td>
<td></td>
</tr>
<tr>
<td>Field name</td>
<td></td>
</tr>
<tr>
<td>Field ID</td>
<td></td>
</tr>
<tr>
<td>Level of field detail</td>
<td>DATA (Struct, Data, Hex)</td>
</tr>
<tr>
<td>Level of subfield detail</td>
<td>NONE (Struct, Data, Hex, None)</td>
</tr>
<tr>
<td>Maximum lines returned</td>
<td>5000</td>
</tr>
<tr>
<td>Display field IDs . . .</td>
<td>N (Y, N) Display extended field info N (Y, N)</td>
</tr>
</tbody>
</table>
```

**Figure 102. RODMView Querying Your User ID**
Simple Query Function

If the specified object exists, the output are displayed as shown in Figure 103.

<table>
<thead>
<tr>
<th>EKGVQEO</th>
<th>Query Output</th>
<th>AOINV OPER2</th>
<th>10/18/97 12:34:56</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyID (OBJECTID)</td>
<td>(OBJECTID) 000F000003209015</td>
<td>'RODMUSER'</td>
<td></td>
</tr>
<tr>
<td>(CLASSID) 6</td>
<td>'EKG_User'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MyPrimaryParentID (CLASSID)</td>
<td>6</td>
<td>'EKG_User'</td>
<td></td>
</tr>
<tr>
<td>EKG_Status (INTEGER)</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EKG_LogLevel (INTEGER)</td>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

EKGV00001 Request is successful (0/0)
CMD=>
F1= Help  F2= End  F3= Return  F5= RptFind  F6= Roll

Figure 103. RODMView Query Output Panel

The Query Output panel shown in Figure 103 shows (in the upper-right corner) that there are 47 lines of output 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 103 on page 514.

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>
<thead>
<tr>
<th>Command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>UP n</td>
<td>Scrolls output up one page, or optionally by n lines.</td>
</tr>
<tr>
<td>DOWN n</td>
<td>Scrolls output down one page, or optionally by n lines.</td>
</tr>
<tr>
<td>TOP</td>
<td>Scrolls output to the top.</td>
</tr>
<tr>
<td>BOTTOM</td>
<td>Scrolls output to the bottom.</td>
</tr>
<tr>
<td>F find_word</td>
<td>Search for find_word from the current panel to the end of output.</td>
</tr>
<tr>
<td>F find_word PREV</td>
<td>Search for find_word from the current panel to the beginning of output. The keyword PREV can be abbreviated as P.</td>
</tr>
</tbody>
</table>

**Note:** When 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 \texttt{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 \texttt{F find_word PREV} or \texttt{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 \texttt{appnNN} and the SystemView field name \texttt{usageState} as shown in Figure 104 on page 514, 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 105 on page 516.
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 (wildcards) 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 106.

```
EKGVQUEI Simple Query A01NV OPER2 10/18/97 12:34:56
RODM name RODMNAME
User ID . . RODMUSER
SystemView class name =
Class name EKG* =
Class ID =
Object name = (Hexadecimal value)
Object ID =
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 106. 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 107 on page 518** illustrates the output panel for a typical RODM.
As shown in Figure 107, 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 will 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: Use caution when setting the Maximum lines returned to values greater than 5000. You can increase the Maximum lines returned value to display lines that are truncated in a query report. However, if you specify a value that is too large, you can exceed NetView storage capacity. To correct this, narrow the scope of your query request. Figure 108 on page 519 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, you should increase the Maximum lines returned to a value greater than or equal to 17 to prevent the query report being truncated.

Figure 107. RODMView Query Output for Fields Containing 'Log'

As shown in Figure 107, 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 will 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: Use caution when setting the Maximum lines returned to values greater than 5000. You can increase the Maximum lines returned value to display lines that are truncated in a query report. However, if you specify a value that is too large, you can exceed NetView storage capacity. To correct this, narrow the scope of your query request. Figure 108 on page 519 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, you should increase the Maximum lines returned to a value greater than or equal to 17 to prevent the query report being truncated.
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 109 on page 520.

The criteria the simple query function uses to display classes and objects are the class and object names themselves. The compound query function not only enables you to search for classes and objects in the same manner, but also enables selection of 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 109 on page 520), to specify where to begin the search by specifying the class and object names.
- Use panel EKGVQA2I (shown in Figure 110 on page 520), to specify criteria that the classes or objects must meet to be displayed.
- Use panel EKGVQA3I (shown in Figure 111 on page 521), to specify a field that should be 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 112 on page 521), to specify which fields (or all fields, if left blank) should be 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.

Figure 108. RODMView Excessively Large Query Output
Compound Query Function

**Figure 109. RODMView Compound Query Panel 1 — EKGVQA1I**

**Figure 110. RODMView Query Criteria Panel 2 — EKGVQA2I**
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 113 on page 522.

![Figure 111. RODMView Query Traversed Criteria Panel 3 — EKGVQA3I](image)

![Figure 112. RODMView Query Field Selection Panel 4 — EKGVQA4I](image)

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 113 on page 522.
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 should have a value other than 129, as shown in Figure 114.

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

Figure 113. Starting a Compound Query on the GMFHS_Aggregate_Objects_Class

Figure 114. Selecting Only Those Entities that Have Nonsatisfactory DisplayStatus
Compound Query Function

found, press **PF8** twice to display the fourth panel EKGVQA4I, and fill in the input fields as shown in Figure 115.

![Figure 115. Selecting Only the DisplayResourceName Field to be Displayed](image)

After the compound query specification has been completed, press **Enter** to execute the query. If all of the GMFHS Sample Network aggregate objects were in unsatisfactory status, the output would be displayed as shown in Figure 116.

![Figure 116. Compound Query Example 1 Output](image)

There are 63 lines of output available, but only 17 lines are visible on the output panel at a time, as shown in Figure 116. Use PF8 to scroll through the output to display all of the entities that met the criteria.
Compound Query Function

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 117.

![Figure 117. Starting a Compound Query on the GMFHS_Aggregate_Objects_Class](image)

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 should have the value 129, as shown in Figure 118 on page 525.
To specify that the query should follow the ComposedOfPhysical link field and that those objects found on that link should have an unsatisfactory DisplayStatus, press PF8 to scroll to the third compound query panel, EKGVQA3I. The panel is filled in as shown in Figure 119.

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 120 on page 526.

Figure 118. Selecting Only Those Entities Having a Satisfactory DisplayStatus

To specify that the query should follow the ComposedOfPhysical link field and that those objects found on that link should have an unsatisfactory DisplayStatus, press PF8 to scroll to the third compound query panel, EKGVQA3I. The panel is filled in as shown in Figure 119.

Figure 119. 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 120 on page 526.
After the compound query specification has been completed, press Enter to execute 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 would be displayed as shown in Figure 121.

Figure 120. Selecting Only the DisplayResourceName Field to be Displayed

After the compound query specification has been completed, press Enter to execute 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 would be displayed as shown in Figure 121.

Figure 121. 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 122.

### Figure 122. 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 LANMGR.BRIDGE01 on a field named DisplayResourceName field, fill in the panel as shown in Figure 123 on page 528.
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, panel EKGVQUEO, Query Output, is displayed as shown in Figure 124.

Figure 123. 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, panel EKGVQUEO, Query Output, is displayed as shown in Figure 124.

Figure 124. Locate Objects Output

The next example, shown in Figure 125 on page 529, 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.
Because N was specified in the Display located entities in detail field, the output are displayed as shown in Figure 126.

**Figure 125. 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 126.

**Figure 126. 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 127 on page 530.
Using the Link/Unlink panel, you can specify two objects to link or unlink, and whether you want associated change methods to be invoked 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 128 on page 533.
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 invoke 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 493 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 493 for more information about method DUIFCUAP. For more information about aggregate objects and aggregation, see "Defining GMFHS Aggregate Objects" on page 38.

To invoke these GMFHS methods, enter G in the Trigger methods input field of the Link/Unlink panel. Also specify whether the method should link or unlink 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 should be linked or unlinked. RODMView determines which method needs to be invoked. 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 NV6000 to the GMFHS display resource type object DUIXC_RTN_MAN_AGG, the Link/Unlink panel would be filled in as shown in Figure 129 on page 532.
Because one of the objects specified the Displayable_Resource_Type_Class, method DUIFCLRT is invoked. The order in which the objects are specified is not significant.

To establish an aggregation path between two objects, the DUIFCUAP is invoked, 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 invokes 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. For example, to make the GMFHS managed real object NETVIEW.T46A an aggregation child of the GMFHS aggregate object NV6000, fill in the Link/Unlink panel as shown in Figure 130 on page 533.
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 131.

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 could 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 132.

<table>
<thead>
<tr>
<th>EKGVCHGI</th>
<th>Change Field A01NV OPER2 10/18/97 12:34:56</th>
</tr>
</thead>
<tbody>
<tr>
<td>RODM name</td>
<td>RODMNAME</td>
</tr>
<tr>
<td>User ID</td>
<td>RODMUSER</td>
</tr>
<tr>
<td>SystemView class name</td>
<td>GMFHS_Managed_Real_Objects_Class</td>
</tr>
<tr>
<td>Class name</td>
<td>GMFHS_Managed_Real_Objects_Class</td>
</tr>
<tr>
<td>Class ID</td>
<td>GMFHS_Managed_Real_Objects_Class</td>
</tr>
<tr>
<td>Object name</td>
<td>NETVIEW.T46A</td>
</tr>
<tr>
<td>Object ID</td>
<td>(Hexadecimal value)</td>
</tr>
<tr>
<td>SystemView field name</td>
<td>DisplayStatus</td>
</tr>
<tr>
<td>Field name</td>
<td>DisplayStatus</td>
</tr>
<tr>
<td>Field ID</td>
<td>DisplayStatus</td>
</tr>
<tr>
<td>Field data type</td>
<td>INTEGER (Anon, Ber, Char, Float, INDEX, INT, Small, Time)</td>
</tr>
<tr>
<td>Field data</td>
<td>129</td>
</tr>
</tbody>
</table>

The following two input fields are used ONLY with the IndexList datatype:

- Update type ADD (Add, Del, Replace) Data is CHARVAR (Anon, CharVar)

**Figure 132. 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 only used for IndexList data type fields. These input fields are ignored for all other data types, even if they are specified.
Table 226 lists, by data type, the rules for changing fields.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnonymousVar and</td>
<td>• The field data entered is interpreted as hexadecimal.</td>
</tr>
<tr>
<td>BERVar</td>
<td>• 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</td>
</tr>
<tr>
<td></td>
<td>displayed:</td>
</tr>
<tr>
<td></td>
<td>EKGV8052E The Field data value is not a valid hex value</td>
</tr>
<tr>
<td></td>
<td>• When entering hexadecimal data, do not use any special notation like X’001122’, for example. It is sufficient to enter just the numeric</td>
</tr>
<tr>
<td></td>
<td>portion 001122.</td>
</tr>
<tr>
<td></td>
<td>• AnonymousVar and BERVar field data types contain a 2-byte length before the actual data. Do not include the 2-byte length</td>
</tr>
<tr>
<td></td>
<td>when you enter a value. RODMView calculates this value after parsing the data.</td>
</tr>
<tr>
<td>CharVar</td>
<td>Accepts characters.</td>
</tr>
<tr>
<td>Floating</td>
<td>Accepts real numbers.</td>
</tr>
<tr>
<td>IndexList</td>
<td>See “Changing IndexList Fields.”</td>
</tr>
<tr>
<td>Integer</td>
<td>Accepts integers.</td>
</tr>
<tr>
<td>TimeStamp</td>
<td>• The string is interpreted as an 8-byte (16 digit) hexadecimal value, which represents the number of Lillian seconds.</td>
</tr>
<tr>
<td></td>
<td>• Query the EKG_Name field on the EKG_System class with the HEX level of subfield detail to see an example of this value.</td>
</tr>
</tbody>
</table>

**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 ‘TCP/IP’ and ‘LAN27’ to the list of exception views for the aggregate object NV6000, fill in panel EKGVCHGI as shown in Figure 133 on page 536.
Change Field Function

<table>
<thead>
<tr>
<th>EKGVCHGI</th>
<th>Change Field</th>
<th>ADINV OPER2</th>
<th>10/18/97 12:34:56</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROOM name</td>
<td>RODMNAME</td>
<td>Trigger methods</td>
<td>Y (Y, N)</td>
</tr>
<tr>
<td>User ID</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>SystemView class name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class name</td>
<td>GMFHS_Aggregate_Objects_Class</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class ID</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object name</td>
<td>NV6000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object ID</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SystemView field name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Field name</td>
<td>ExceptionViewList</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Field ID</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Field data type</td>
<td>INDEXLIST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Field data</td>
<td>'TCPIP ' 'LAN27 '</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 133. 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 134 on page 537.
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 ExtremelyImportantClass class, enter the information in the Subfield Action panel as shown in Figure 135.

![Figure 134. RODMView Subfield Actions Panel — EKGVSUBI](image)

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 ExtremelyImportantClass class, enter the information in the Subfield Action panel as shown in Figure 135.

![Figure 135. RODMView Creating a Notify Subfield](image)
Subfield Actions Function

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 136.

Table 227 on page 539 lists the information that must be provided to create a child class, an object, or a field.
Table 227. Specifications to Create Entities.

<table>
<thead>
<tr>
<th>To create this:</th>
<th>Fill in only these input fields:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Child Class</td>
<td>Class name or Class ID</td>
</tr>
<tr>
<td></td>
<td>Child Class name</td>
</tr>
<tr>
<td>Object</td>
<td>Class name or Class ID</td>
</tr>
<tr>
<td></td>
<td>Object name</td>
</tr>
<tr>
<td>Field</td>
<td>Class name or Class ID</td>
</tr>
<tr>
<td></td>
<td>Field name or Field ID</td>
</tr>
<tr>
<td></td>
<td>Field data type</td>
</tr>
<tr>
<td></td>
<td>Field inherits</td>
</tr>
</tbody>
</table>

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 137.

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 138 on page 540.

Note that no value is specified for the Object name 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 138, 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.

From the RODMView main menu, select 9. Delete Actions. The Delete Actions panel is displayed as shown in Figure 136 on page 538.
Table 228 lists the information that must be provided to delete a child class, an object, or a field.

Table 228. Specifications to Delete Entities.

<table>
<thead>
<tr>
<th>To delete this</th>
<th>Fill in only these input fields:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>Class name or Class ID</td>
</tr>
<tr>
<td>Object</td>
<td>Class name, Class ID, or Object name</td>
</tr>
<tr>
<td>Object</td>
<td>Object ID</td>
</tr>
<tr>
<td>Field</td>
<td>Class name, Class ID, Field name, or Field ID</td>
</tr>
</tbody>
</table>

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 140 on page 543.
Delete Actions Function

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 141 on page 543.
Using RODMView, object-independent methods are invoked 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 there is a field called MethodSpecField of type MethodSpec defined on the class UsefulClass, and MethodSpecField has a value that includes a method called USFLMETH. To invoke the method, enter the information on the Method Actions panel as shown in Figure 142.

Figure 141. RODMView Method Actions Panel — EKGVMETI

![Figure 141. RODMView Method Actions Panel — EKGVMETI](image1.png)

Figure 142. RODMView Triggering a Named Method

![Figure 142. RODMView Triggering a Named Method](image2.png)
Method Actions Function

The method USFLMETH is invoked 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 would be similar to the panel shown in Figure 143.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method name</td>
<td>USFLMETH</td>
</tr>
<tr>
<td>Method type</td>
<td>NAMED (Named, Object independent)</td>
</tr>
<tr>
<td>Action</td>
<td>TRIGGER (Trigger, Install, Delete, Replace)</td>
</tr>
</tbody>
</table>

In the prior example, the method that was triggered was user-written. Once 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 (8/60000)

**Note:** The method name in Figure 142 on page 543 was typed in lowercase, but when the RODMView panel is refreshed in Figure 143, 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 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 DD card. If any class field contains a locally defined value, that value is written to the CLASSVAL DD card.

The RODM unload function does not unload the values of system-defined fields on the system classes (UniversalClass and all EKxxxx 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 DD card to create it. All data contained in fields that have local values are written to the OBJVAL DD card.

To ensure that unloaded data sets load properly again, they should be concatenated in the RODM load function EKGIN3 card 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 invokes JCL would appear as shown in Figure 144.

```
//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 144. 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, it is recommended that you 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.V5R1M0.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.V5R1M0.CNMSAMP data set.
Customizing the RODM Unload Function

The RODM unload function is invoked with JCL. Input parameters are passed to the RODM unload function in a file named by the SYSIN DD card of the JCL. Figure 145 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 145. Sample SYSIN DD card of the JCL.*

Table 229 contains a description of the SYSIN DD parameters.

**Table 229. SYSIN DD Parameter Descriptions**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RODM</td>
<td>Specifies the name of the RODM to unload. This is usually the same as the z/OS procedure used to start RODM.</td>
</tr>
</tbody>
</table>
| 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 datasets specified in the start JCL.  
           • If WRITEMODE=OVERWRITE, any data that previously existed in the datasets is destroyed, and any new output created by the RODM unload function is written in its place. |
Table 229. SYSIN DD Parameter Descriptions (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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. It is recommended that the DCB specifications in the sample be used 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 146 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 146. 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 would be changed as shown in Figure 147 on page 548.
Running the RODM Unload Function

RODM=(rodmname)
CLASS=GMFHS_Monitorable_Objects_Class
DEPTH=All
REPORTONLY=No
WRITEMODE=Overwrite
WHITESPACE=Low

Figure 147. 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 148.

RODM=(rodmname)
CLASS=UniversalClass
OBJECT=DesiredObject
DEPTH=All
REPORTONLY=No
WRITEMODE=Overwrite
WHITESPACE=High

Figure 148. 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 149.

RODM=(rodmname)
CLASS=SpecificClass
OBJECT=DesiredObject
DEPTH=One
REPORTONLY=No
WRITEMODE=Overwrite
WHITESPACE=High

Figure 149. 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 150.

RODM=(rodmname)
CLASS=SpecificClass1
CLASS=SpecificClass2
DEPTH=One
REPORTONLY=No
WRITEMODE=Overwrite
WHITESPACE=Low

Figure 150. EKGKUJCL SYSIN Parameters to Determine Object Definitions for Two Classes

FLCARODM

This section explains 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.

There are two ways to use FLCARODM:

- Specify the data and operations using a low-level data stream. See "Object Data Stream Detail" on page 588 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 sample FLCSSTEM. These subroutines manipulate REXX stem variables that are used with FLCARODM. The three stem variables that are manipulated by these subroutines are:

- 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

There is also a variable called Retcode, which is used by all of the subroutines to indicate if any errors have occurred. A nonzero value in the Retcode variable indicates that processing should stop.

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 could 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.
Stem Building Subroutines

- 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 sample FLCSSTEM 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 should be 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 ALmnClass class that should be used to identify object links that are to be removed:

  ```call StartObject ALmnClass 'Demo_Lan'
call AddAttrForQuery Member
call AddAttrForQuery PhyConn
call MakeRODMCall 'DELINKA' 'XREF=1STFIELD'
```  

AddAttrForQuery Member specifies that all objects specified by the Member field should be identified and AddAttrForQuery PhyConn specifies that all links specified by the PhyicalConPP field should be 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 AddLinkForDeletefieldname
```

Operand Descriptions: Where:
- **fieldname**: 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 AddLink PhyConn ASegClass 'Segment_1' PhyConn
call AddLink PhyConn ASegClass 'Segment_2' 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 sample FLCSSTEM.

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 559 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:

```plaintext
call StartObject ALnmClass '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:**

```plaintext
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 ALnmClass named Demo_Lan:

```plaintext
call StartObject ALnmClass '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.

```plaintext
call MakeRODMCall 'BUILD'
```

### Using the Samples

To use the sample code provided by the NetView product, perform the following tasks:
About the Examples

- Change the value of RODMNAME to the name of the RODM you are using.
- Change the value of RODMAPPL to your RODM application ID.
- Append the contents of sample FLCSSTEM to the bottom of the Rexx code that you are writing. FLCSSTEM provides the subroutines and constant definitions that are used by the samples.

**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 151 shows an example of issuing the FLCARODM command:

```plaintext
PIPE STEM object_data
| COLLECT
| NETVIEW FLCARODM parameters
| stem result
```

*Figure 151. 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 151 when you are specifying data using the object data stream described in "Object Data Stream Detail" on page 588.

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 549 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:**

```plaintext
FLCARODM— ROADMNAME=name— RODUSER=user— FUNCTION=FLCarodmFunctions
```
FLCARODM Command

RODMINT=interval  RODMRETRY=number_retries

FLCARodmFunctions:

'BUILD'

'DELINKA'

'DELINKAB'

'DELOBJ'

'LOCATE'

'PURGE'

'QUERY'

'QUERYALL'

'QUERYSF'

'QUERYSTR'

'REBUILD'

'UPDATE'

Xref:

XREF= 1STFIELD  FILTER=2NDFIELD

objectname  FILTER=1STFIELD

PurgeOptions:

STATUS=ALL  TRACE=NO

STATUS=val  TIMESTAMP=val

TRACE=YES  FILTER=1STFIELD

Operand Descriptions:
CHILDREN
Specifies whether the operation should apply to the specified object’s children. CHILDREN 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 should be performed on the specified object, but not on its children.

ONLY
Indicates that the function should be performed on the specified object’s children, but not on the object itself.

YES
Indicates that the function should be 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.

FIELDID
Indicates whether the QUERY function should return field identifiers with the field names.

NO
Indicates that the field identifiers should not be returned.

YES
Indicates that the field identifiers should be 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 was specified and an object existed that has the value Seg, the filter would match and the object would be 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:
FLCAROMD Command

1. Field name
2. Field data type
3. Field value

FUNCTION
Specifies the function that is performed. For a description of each function, see "FLCAROMD Functions" on page 559.

LOCATE
Specifies that the first field definition should be 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 FLCAROMD should wait between retrying requests when RODM is checkpointing. The default value is five seconds.

RODMRTRY
The number of times FLCAROMD should retry a request when RODM is checkpointing. The default value is three. If RODM is still checkpointing after FLCAROMD 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 FLCAROMD. However, Access cannot use the same RODMUSER value as other applications (for example, RODMVIEW) that connect to RODM.

It is recommended that you create a RODMUSER value by concatenating the NetView domain name with a three-character identifier. For example, MultiSystem Manager concatenates the NetView domain name CNM01 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 should be purged by the PURGE function.
ALL
Indicates that an object should be 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 should run in trace mode. In trace mode, a message is issued for every object that is purged.

NO
Indicates that the PURGE function should not run in trace mode.

YES
Indicates that the PURGE function should run in trace mode.

XREF
Specifies that a function should be 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.
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:

<table>
<thead>
<tr>
<th>Date Type</th>
<th>Data Type Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHARVAR</td>
<td>4</td>
</tr>
<tr>
<td>INTEGER</td>
<td>10</td>
</tr>
<tr>
<td>SELFDEFINING</td>
<td>19</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>21</td>
</tr>
<tr>
<td>FIELDID</td>
<td>26</td>
</tr>
<tr>
<td>ANONYMOUSVAR</td>
<td>30</td>
</tr>
</tbody>
</table>

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 */
IPPING(*********************************************************************/
call StartObject NetClass 'Hometown'
  /* Start creating Hometown object */
call AddAttr Annotate CharVar 'This is the Hometown City View'
  /* Add an Annotation or label */

IPPING(*********************************************************************/
 /* 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. */
IPPING(*********************************************************************/
call StartObject AggClass 'Main_Street'
  /* Add an Annotation or label */
call AddAttr Annotate CharVar 'This is the Hometown City View'
  /* Add an Annotation or label */

IPPING(*********************************************************************/
 /* 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. */
IPPING(*********************************************************************/
call StartObject AggClass 'Main_Street'
  /* Add an Annotation or label */
call AddAttr Annotate CharVar 'This is the Hometown City View'
  /* Add an Annotation or label */

IPPING(*******************************************************************************/
 /* Now add a label which says 'Constructed in 1889 to */
 /* the object. */
IPPING(*******************************************************************************/
call AddAttr DispOther CharVar 'Constructed in 1889'

IPPING(*******************************************************************************/
 /* Add a link to the object which tells the Display */
 /* ResourceType and Display_Resource_Type_Class are */
 /* linked to the DUIXC_RTN_HOST_AGG */
IPPING(*******************************************************************************/
call AddLink DispType DispClass HtAgg_Icon 'Resources'

IPPING(*******************************************************************************/
 /* Now add another link to link the object to the */
 /* Hometown view */
IPPING(*******************************************************************************/
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'
/**********************************************************/
call AddAttr DispOther CharVar '3 Bedroom Ranch'
call AddAttr DispStat Integer Active
/**********************************************************/
call AddLink DispType DispClass HtAgg_Icon 'Resources'
call AddLink PartOf AggClass 'Main_Street' COMPPHY

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.

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
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:

```c
call StartObject AggClass 'Main_Street' /*Which object we are referring to. */
call AddAttrForQuery DispOther /*Query contents of */
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 executed.

```
3
FLCARODM:0,0,0
4
Constructed In 1889
```

Table 230 describes the result stem that was returned for sample FLCSX3:

<table>
<thead>
<tr>
<th>Element Number</th>
<th>Element Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>Indicates that the result stem contains 3 elements</td>
</tr>
<tr>
<td>1</td>
<td>FLCARODM:0,0,0</td>
<td>The FLCARODM return code and the RODM return and reason code</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>Indicates the data type of the field (charvar)</td>
</tr>
<tr>
<td>3</td>
<td>Constructed In 1889</td>
<td>The value of the field</td>
</tr>
</tbody>
</table>

Sometimes it is 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 should not be 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:

```lisp
...call StartObject AggClass 'Main_Street' /*Which object we are referring to. */
call AddAttrForQuery DispOther /*Query contents of */
call MakeRODMCall 'QUERY' 'FIELDID=YES' /*DisplayResourceOtherData*/
...;
```

The following is a partial example of the result stem that is returned when the modified sample FLCSX3 is executed.

4 FLCARODM:0,0,0
4 60 Constructed In 1889 :

Table 231 describes the result stem that was returned for the modified sample FLCSX3:

<table>
<thead>
<tr>
<th>Element Number</th>
<th>Element Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>Indicates that the result stem contains 3 elements</td>
</tr>
<tr>
<td>1</td>
<td>FLCARODM:0,0,0</td>
<td>The FLCARODM return code and the RODM return and reason code</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>Indicates the data type of the field (charvar)</td>
</tr>
<tr>
<td>3</td>
<td>60</td>
<td>Indicates the field ID of the field</td>
</tr>
<tr>
<td>4</td>
<td>Constructed In 1889</td>
<td>The value of the field</td>
</tr>
</tbody>
</table>

Execute samples FLCSX1, FLCSX2, and FLCSX3 before you execute 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 executed. The first part of sample FLCSX4 queries all of GMFHS_Aggregate_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 232 describes the result stem that was returned for the first part of sample FLCSX4:

<table>
<thead>
<tr>
<th>Element Number</th>
<th>Element Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>Indicates that the result stem contains 4 elements</td>
</tr>
<tr>
<td>1</td>
<td>FLCARODM:0,0,0</td>
<td>The FLCARODM return code and the RODM return and reason code</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>Indicates the data type of the field (objectidlist)</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>The number of object IDs in the list</td>
</tr>
<tr>
<td>4</td>
<td>00010012E05C2A1E</td>
<td>The hexadecimal object ID of the object</td>
</tr>
</tbody>
</table>

The second part of sample FLCSX4 queries the name and status of the object ID that was returned from the first query:
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 233 describes the result stem that was returned for the second part of sample FLCSX4:

<table>
<thead>
<tr>
<th>Element Number</th>
<th>Element Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>Indicates that the result stem contains 6 elements</td>
</tr>
<tr>
<td>1</td>
<td>FLCARODM:0,0,0</td>
<td>The FLCARODM return code and the RODM return and reason code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>for the first field that was queried</td>
</tr>
<tr>
<td>2</td>
<td>18</td>
<td>Indicates the data type of the field</td>
</tr>
<tr>
<td>3</td>
<td>Main_Street</td>
<td>The number of object IDs in the list</td>
</tr>
<tr>
<td>4</td>
<td>FLCARODM:0,0,0</td>
<td>The FLCARODM return code and the RODM return and reason code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>for the second field that was queried</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>The data type of the field (integer)</td>
</tr>
<tr>
<td>6</td>
<td>132</td>
<td>The value of the field</td>
</tr>
</tbody>
</table>

Note: The query functions in FLCSX4 were performed by two calls to FLCARODM using the MakeRODMCall subroutine. Both functions could be performed using one call to FLCARODM by using the XREF parameter. See "FLCARODM Command" on page 553 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 that enables you to protect objects associated with other applications from being deleted.

The following code from sample FLCSX5 uses the DELOBJ to delete the 1000 Main Street object.

```
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 will determine which links exist and remove all of them.

For an example of using the DELINKA function, see "Delinking Objects" on page 576.

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 will first remove the old link and then define 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 576.

PURGE Function
Use the PURGE function to remove objects from RODM. PURGE uses the RemvObjs command to remove objects from RODM. Refer to the Tivoli NetView for z/OS MultiSystem Manager User’s Guide 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.

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 would be returned:

4
FLCARODM:0,0,0
15
1
000100012E05C2A1E

Table 234 describes the result stem that was returned for sample FLCSXL01 if one object met the search criteria.

<table>
<thead>
<tr>
<th>Element Number</th>
<th>Element Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>Indicates that the result stem contains 4 elements</td>
</tr>
<tr>
<td>1</td>
<td>FLCARODM:0,0,0</td>
<td>The FLCARODM return code and the RODM return and reason code</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>Indicates the data type of the return data (objectidlist)</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>The number of matches found</td>
</tr>
<tr>
<td>4</td>
<td>000100012E05C2A1E</td>
<td>The object ID of the object that matched the search criteria</td>
</tr>
</tbody>
</table>

If there were no objects in RODM with a field that matched the comparison criteria, FLCARODM would return an Object ID List with zero elements as follows.

3
FLCARODM:0,0,0
15
0

Table 233 describes the result stem that would be returned for sample FLCSXL01 if no objects met the search criteria.

<table>
<thead>
<tr>
<th>Element Number</th>
<th>Element Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>Indicates that the result stem contains 3 elements</td>
</tr>
<tr>
<td>1</td>
<td>FLCARODM:0,0,0</td>
<td>The FLCARODM return code and the RODM return and reason code</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>Indicates the data type of the return data (objectidlist)</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>Indicates that no matches were found</td>
</tr>
</tbody>
</table>

| 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 executed.

212
FLCARODM:0,0,0
51
FLCARODM:0,0,0
IsPartOf
17
0
FLCARODM:0,0,0
IsBusNode
17
0

Table 236 describes the result stem that was returned for sample FLCSXQ2.

Table 236.

<table>
<thead>
<tr>
<th>Element Number</th>
<th>Element Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>212</td>
<td>Indicates that the result stem contains 212 elements</td>
</tr>
<tr>
<td>1</td>
<td>FLCARODM:0,0,0</td>
<td>The FLCARODM return code and the RODM return and reason code</td>
</tr>
<tr>
<td>2</td>
<td>51</td>
<td>Indicates the number of fields defined on the object</td>
</tr>
<tr>
<td>3</td>
<td>FLCARODM:0,0,0</td>
<td>The FLCARODM return code and the RODM return and reason code</td>
</tr>
<tr>
<td>4</td>
<td>IsPartOf</td>
<td>The name of the first field defined on the object.</td>
</tr>
<tr>
<td>5</td>
<td>17</td>
<td>The data type of the IsPartOf field. (objectlinklist)</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>The value of the IsPartOf field</td>
</tr>
<tr>
<td>7</td>
<td>FLCARODM:0,0,0</td>
<td>The FLCARODM return code and the RODM return and reason code</td>
</tr>
</tbody>
</table>
The previous example describes the first two fields in the result stem. Elements 11 through 212 would 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 executed.

```
214
FLCARODM:0,0,0
53
AggregationChild
121
17
00
UpdateAggregationCounters
122
13
00
```

Table 237 on page 574 describes the result stem that was returned for sample FLCSXQ1.

<table>
<thead>
<tr>
<th>Element Number</th>
<th>Element Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>IsBusNode</td>
<td>The name of the second field defined on the object</td>
</tr>
<tr>
<td>9</td>
<td>17</td>
<td>The data type of the IsBusNode field. (objectlink)</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>The value of the IsBusNode field</td>
</tr>
</tbody>
</table>
Table 237.

<table>
<thead>
<tr>
<th>Element Number</th>
<th>Element Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>214</td>
<td>Indicates that the result stem contains 214 elements</td>
</tr>
<tr>
<td>1</td>
<td>FLCARODM:0,0,0</td>
<td>The FLCARODM return code and the RODM return and reason code.</td>
</tr>
<tr>
<td>2</td>
<td>53</td>
<td>Indicates the number of fields defined on the object</td>
</tr>
<tr>
<td>3</td>
<td>AggregationChild</td>
<td>The name of the first field defined on the object.</td>
</tr>
<tr>
<td>4</td>
<td>121</td>
<td>The field identifier</td>
</tr>
<tr>
<td>5</td>
<td>17</td>
<td>The data type of the field (objectlinklist)</td>
</tr>
<tr>
<td>6</td>
<td>00</td>
<td>The inheritance status bitmap</td>
</tr>
<tr>
<td>7</td>
<td>UpdateAggregationCounters</td>
<td>The name of the second field defined on the object</td>
</tr>
<tr>
<td>8</td>
<td>122</td>
<td>The field identifier</td>
</tr>
<tr>
<td>9</td>
<td>13</td>
<td>The data type of the field (methodspec)</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>The inheritance status bitmap</td>
</tr>
</tbody>
</table>

The previous example describes the first two fields in the result stem. Elements 11 through 214 would 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:

```plaintext
  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: Execute samples FLCSX1 and FLCSX2 before you execute sample FLCSXQ3.
The following is an example of the result stem that is returned when sample FLCSXQ3 is executed.

```
3
FCARODM:0,0,0
10
129
```

Table 238 describes the result stem that was returned for sample FLCSXQ3.

<table>
<thead>
<tr>
<th>Element Number</th>
<th>Element Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>Indicates that the result stem contains three elements</td>
</tr>
<tr>
<td>1</td>
<td>FLCARODM:0,0,0</td>
<td>The FLCARODM return code and the RODM return and reason code</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>The data type of the subfield (integer)</td>
</tr>
<tr>
<td>3</td>
<td>129</td>
<td>The previous value of the field</td>
</tr>
</tbody>
</table>

**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 549.

**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
Putting It All Together

```
call StartObject NetClass 'Advanced' /*Which object? */
/*******************************/
/* Start creating LAN object in the Advanced Operations View */
/*******************************/
call StartObject ALnmClass '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 ALnmClass '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 ALnmClass '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 ALnmClass 'Demo_Lan' Member
call AddLink PhyConn ABrgClass 'Bridge_1' Phyconn

call MakeRODMCall 'BUILD' /*Call RODM */
```
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 154** uses the UPDATE function to accomplish the same task as **Figure 153**; 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).

**Putting It All Together**

```plaintext

call StartObject ALnmClass '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 153. Sample FLCSX7**

**Figure 155** uses the UPDATE function with the XREF parameter to specify that the links defined by the ComposedOfPhysical field should be used to determine the list of objects to be updated.

**Figure 154. Sample FLCSX14**

**Figure 155. Sample FLCSX15**

**Figure 156 on page 574** 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 should be used to determine the list of objects to be updated. For example, sample FLCSX16 specifies the following:
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.

Figure 156. Sample FLCSX16

Figure 157 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.

Figure 157. Sample FLCSX17

Quering 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 578 for more information about result stems.

Figure 158 on page 575 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).
Putting It All Together

Figure 158. 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 159 queries all objects in RODM to determine which objects have a display name of LNM_NETWORKS. Note that call StartObject '' '' means all objects in RODM.

Figure 159. 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=LN_M_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 160 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.

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 161 also uses the DELINKA function to delete all of the links defined by the PhysicalConnPP field of the Bridge_1 object.

Figure 161. Sample FLCSX10

Like Figure 161, Figure 162 on page 577 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.
Putting It All Together

```plaintext
call StartObject ALnmClass 'Demo_Lan' /*Which object we are referring to. */
call AddLinkForDelete PhyConn

call MakeRODMCall 'DELINKA' 'CHILDREN=ONLY' /*Call RODM Only do the CHILDREN */
```

Figure 162. Sample FLCSX9

```plaintext
call StartObject ALnmClass 'Demo_Lan' /*Which object we are referring to. */
call AddAttrForQuery Member

call AddAttrForQuery PhyConn

call MakeRODMCall 'DELINKA' 'XREF=1STFIELD' /*Call RODM */
```

Figure 163. Sample FLCSX19

**Figure 164** uses the DELINKAB function to remove specific links to the Bridge_1 object.

```plaintext
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 164. Sample FLCSX11

**Deleting Objects**

**Figure 165** uses the DELOBJ function to delete the Demo_Lan object. The CHILDREN parameter specifies that the child objects of the Demo_Lan object should also be deleted.

```plaintext
call StartObject ALnmClass 'Demo_Lan' /*Which object we are referring to. */
call MakeRODMCall 'DELOBJ' 'CHILDREN=YES' /*Call RODM */
```

Figure 165. Sample FLCSX8
Working with IndexList Fields
Use the SetIndexList subroutine to change IndexList fields.

Figure 166 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 ALnmClass 'Demo_Lan' /*Which object we are referring to.*/
my_String = 'testing'
call SetIndexList my_String ExceptionViewList

call MakeRODMCall 'UPDATE' /*Call RODM*/
```

Figure 166. Sample FLCSX22

Result Stem
A result stem is returned each time the FLCARODM command is invoked. 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 would be 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 586. For a description of the RODM return and reason codes, see "RODM Return and Reason Codes" on page 454.

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 582 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:

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RodmResult.0</td>
<td>1</td>
</tr>
<tr>
<td>RodmResult.1</td>
<td>FLCARODM:0,0,0</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>RodmResult.0</td>
<td>x</td>
<td>The number of elements in the result stem</td>
</tr>
<tr>
<td>RodmResult.1</td>
<td>FLCARODM:0,0,0</td>
<td>The FLCARODM return code and the RODM return and reason code</td>
</tr>
<tr>
<td>RodnResult.1</td>
<td>10</td>
<td>The data contained in the field</td>
</tr>
</tbody>
</table>

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 there is an additional return code to indicate 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
```

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:**

\[ \text{return_code_data} \rightarrow \text{field_type} \rightarrow \text{field_value} \rightarrow \]

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 could result in:

```plaintext
FLCARODM:0,0,0
4
Constructed In 1889
```

**OBJECTLINK:** For fields of OBJECTLINK data types, the format of the result stem is:

\[ \text{return_code_data} \rightarrow \text{field_type} \rightarrow \text{object_id} \rightarrow \text{field_id} \rightarrow \]

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 could result in:
For fields of OBJECTLINKLIST data types, the format of the result stem is:

For fields of OBJECTLINKLIST data types, the format of the result stem is:

\[\text{Stem.f} = \text{return_code_data} \quad \text{Stem.f+1} = \text{field_type} \quad \text{Stem.f+2} = \text{relations}\]

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:

\[\text{Stem.l} = \text{object_id} \quad \text{Stem.l+1} = \text{field_id}\]

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 could result in:

\[\text{FLCARODM:0,0,0} \quad 17 \quad 2 \quad 00010012E05C2A1E \quad 5 \quad 00010012E05C2A1F \quad 6\]
Result Stem

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.

\[ \text{Stem.} r = \text{return code data} - \text{Stem.} r + 1 = \text{operation code} - \text{Stem.} r + 2 = \text{object id} \]

\[ \text{Stem.} r + 3 = \text{object class} - \text{Stem.} r + 4 = \text{object name} \]

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 should be inspected. See "Return Codes" on page 588 for other return code value definitions. Refer to the 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 will have null values.

**MultiSystem Manager Operations**

The operations that FLCARODM performs are:

<table>
<thead>
<tr>
<th>Operation Id</th>
<th>Operation</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>No Operation Determined</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>Create An Object</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>Delete An Object</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>Delete An Object And Its Children</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>Delete An Object's Children</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>Execute Purge Against An Object</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>Change A Field, Creating The Object If Necessary</td>
<td></td>
</tr>
<tr>
<td>201</td>
<td>Change A Field, Only If The Object Exists</td>
<td></td>
</tr>
<tr>
<td>202</td>
<td>Query A Field On An Object</td>
<td></td>
</tr>
<tr>
<td>203</td>
<td>Change A Field On A Child Object</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>Define A Relation, Creating The Object If Necessary</td>
<td></td>
</tr>
</tbody>
</table>
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).

```plaintext
call StartObject 'NoClass' 'Dave' /*Which object we are */
   /*referring to. */
call MakeRODMCall 'BUILD' /*Call RODM */
```

The following error stem would be returned:

```plaintext
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:

```
+<.-VVVV)+<.-VVVV
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 could be:

```plaintext
FLCARODM:1048,0,0
200
0000000000000000
GMFHS_Managed_Real_Objects_Class
```
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 invalid. This is because the field type must be a decimal value representing the data type, The word Integer was specified which is incorrect. The decimal value 10 should have been used.

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 could be returned when one tries to query a field that does not exist.

FLCARODM:2000,4,56
02
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:

```
<table>
<thead>
<tr>
<th>field_name</th>
<th>object_id</th>
<th>object_class</th>
<th>object_name</th>
<th>linked_field_name</th>
</tr>
</thead>
</table>
```

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 could 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 could 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 would indicate that three errors occurred while processing the FLCARODM request. The calling application would be able to decode this information since 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 were 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
```
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 invalid 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 invalid, 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 invalid, 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 invalid number, or was too large.

1032  The number of fields specified was either an invalid number, or was too large.
The number of relations specified was either an invalid number or was too large.

The field specified was invalid, 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.

The field type specified was either an invalid number, or was too large.

The field value specified was invalid. If the field type indicates that the field value is numeric, then the field value was either an invalid 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.

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.

The specified field name to link to was either null, or its length was greater than 64 characters.

The specified class name to link to was either null, or its length was greater than 64 characters.

The specified object name to link to was either null, or its length was greater than 254 characters.

The specified field name to link to was either null, or its length was greater than 64 characters.

For the function specified, no fields are allowed.

For the function specified, no relations are allowed.

The data type returned for the field that was queried is not supported by FLCARODM.

The value supplied for the RODMRTRY parameter is invalid.

The value supplied for the RODMINT parameter is invalid.

The value supplied for the CHILDREN parameter is invalid.

The value supplied for the STATUS parameter is invalid.

The value supplied for the TIME parameter is invalid.

The value supplied for the TRACE parameter is invalid.

A parameter specified is invalid or unauthorized for the function specified.

The number of object definitions found was less than the number of objects specified.

All expected data has been processed, but more data still exists.

The object definition was not complete.

The number of field definitions found was less than the number of fields specified.

The field definition was not complete.

The number of relations found was less than the number specified.

The relation definition was incomplete.
Return Codes

1144 The number of fields specified was incorrect for the XREF function.
1148 The value supplied for the LOCATE parameter is invalid.
1152 The value supplied for the SF parameter is invalid.
1156 The value supplied for the FIELDID parameter is invalid.
1160 The value supplied for the FILTER parameter is invalid.
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 should 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 There is no Member or MemberArcs field 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

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

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
Object Data Stream Detail

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

<table>
<thead>
<tr>
<th>Field Used For Relation</th>
<th>Class Of Related Object</th>
<th>Name Of Related Object</th>
<th>Field On Related Object</th>
</tr>
</thead>
</table>

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:

```
*,<.-VVVV),<.-VVVV
Stem.0= elements
Stem.1= objects
)*/[VjVVVV
```

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.

```
*,<.-VVVV),<.-VVVV
Stem.o=object_class
Stem.o+1=object
Stem.o+2=fields
Stem.o+3=relations
)*/[VjVVVV
```

Where:

- **object_class**
  The object class to be operated on. This should be blank if an object id is specified. This should be null if the Locate function is specified.

- **object**
  The name or object id of the object to be operated on. 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 should 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.

\[
\begin{align*}
\text{Field Definition: } & \quad \text{Stem.f=field} \\
& \quad \text{Stem.f+1=field_type} \\
& \quad \text{Stem.f+2=field_value}
\end{align*}
\]

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.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Data Type Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHARVAR</td>
<td>4</td>
</tr>
<tr>
<td>INTEGER</td>
<td>10</td>
</tr>
<tr>
<td>SELFDEFINING</td>
<td>19</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>21</td>
</tr>
<tr>
<td>FIELDID</td>
<td>26</td>
</tr>
<tr>
<td>ANONYMOUSVAR</td>
<td>30</td>
</tr>
</tbody>
</table>

For a list of data types supported by the BUILD and UPDATE functions,
see “BUILD Function” on page 560.

For a list of data types supported by the QUERY function, see “QUERY
Function” on page 561.

field_value
The value that should be 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 that should be related to the object being defined.

**object_to_link_to**
The name or object id of the object that should 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 that should 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.

**BLDVIEWS**

BLDVIEWS is a REXX exec that enables you to create aggregate objects and customized views. 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 cards 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 cards 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 wildcard name.
BLDVIEWS

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 cards 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 ESPGAF.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 enables you 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 cards. 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 cards are contained in member FLCVBLDS which resides in the CNMSAMP dataset. FLCVBLDS has examples of coding control cards 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 cards. Refer to the NetView Command online help for FLCV2RCM and the NMC 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
BLDVIEWS enables you to specify any supported view layout type, but only uses the following view layout types:

- Hierarchical
- Ellipse
- Grid

**Aggregate Objects**

Use the AGGRegate control card and AGGChild control cards 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 Cards**

The following control cards are supported:

<table>
<thead>
<tr>
<th>Control Card</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADaPter</td>
<td>Specifies LNM adapter resources.</td>
</tr>
<tr>
<td>AGENT</td>
<td>Specifies MultiSystem Manager agent that has created objects in RODM.</td>
</tr>
<tr>
<td>AGG</td>
<td>Specifies the aggregate resources (GMFHS aggregate objects). The aggregate resources you specify can be existing resources, or you can create an aggregate and link the resources on the AGGChild control cards that follow to the aggregate resource.</td>
</tr>
<tr>
<td>AGGCHILD</td>
<td>Specifies the aggregation children that you want linked to the aggregate resource that was previously defined.</td>
</tr>
<tr>
<td>BBVIEW</td>
<td>Defines a configuration backbone view, which contains the resources on the control cards that follow it.</td>
</tr>
<tr>
<td>BRidge</td>
<td>Specifies LNM bridge real or aggregate resources.</td>
</tr>
<tr>
<td>CAU</td>
<td>Specifies LNM CAU real or aggregate resources.</td>
</tr>
<tr>
<td>CDRM</td>
<td>Specifies VTAM CDRM resources.</td>
</tr>
<tr>
<td>CDRSC</td>
<td>Specifies VTAM CDRSC resources.</td>
</tr>
<tr>
<td>CIRCUIT</td>
<td>Specifies the APPN transmission group circuits, and subarea circuits.</td>
</tr>
<tr>
<td>CLASS</td>
<td>Specifies the global RODM class which contains the resources on the OTHER control cards that follow it. This control card is only used for the OTHER control card and enables you to specify the RODM class globally without having to specify it on each OTHER control card.</td>
</tr>
<tr>
<td>CLUSTER</td>
<td>Specifies the MultiSystem Manager or APPNTAM cluster aggregate resource.</td>
</tr>
<tr>
<td>DOMAIN</td>
<td>Specifies APPN domains.</td>
</tr>
<tr>
<td>ENODE</td>
<td>Specifies APPN end nodes.</td>
</tr>
<tr>
<td>EVIEW</td>
<td>Defines an exception view.</td>
</tr>
<tr>
<td>GW_NCP</td>
<td>Specifies NCP gateway resources.</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>HOST_NODE</td>
<td>Specifies the host PUs (PU Type 5 nodes).</td>
</tr>
<tr>
<td>IC_NODE</td>
<td>Specifies APPN interchange nodes.</td>
</tr>
<tr>
<td>INTERFACE</td>
<td>Specifies TCP/IP adapter resources.</td>
</tr>
<tr>
<td>IP_BRIDGE</td>
<td>Specifies TCP/IP bridge aggregate resources.</td>
</tr>
<tr>
<td>IP_HOST</td>
<td>Specifies TCP/IP host aggregate resources.</td>
</tr>
<tr>
<td>IP_HUB</td>
<td>Specifies TCP/IP hub aggregate resources.</td>
</tr>
<tr>
<td>IP_LINK</td>
<td>Specifies TCP/IP interface link resources.</td>
</tr>
<tr>
<td>IP_LOCATION</td>
<td>Specifies TCP/IP location aggregate resources.</td>
</tr>
<tr>
<td>IP_ROUTER</td>
<td>Specifies TCP/IP router aggregate resources.</td>
</tr>
<tr>
<td>IP_SEGMENT</td>
<td>Specifies TCP/IP segment aggregate resources.</td>
</tr>
<tr>
<td>IP_SUBNET</td>
<td>Specifies TCP/IP subnetwork aggregate resources.</td>
</tr>
<tr>
<td>IPSpname</td>
<td>Specifies the VTAM PU, LU, or CP name for the NetView for AIX service point which manages the resources on the control cards that follow it.</td>
</tr>
<tr>
<td>LAN_PORT</td>
<td>Specifies LNM Port. This is for LNM V2.</td>
</tr>
<tr>
<td>LANSpname</td>
<td>Specifies the VTAM PU, LU, or CP name for the LNM service point which manages the resources on the control cards that follow it.</td>
</tr>
<tr>
<td>LC VIEW</td>
<td>Defines a configuration logical connectivity view, which contains the resources on the control cards that follow it.</td>
</tr>
<tr>
<td>LINE</td>
<td>Specifies VTAM lines.</td>
</tr>
<tr>
<td>LLINK</td>
<td>Specifies logical links.</td>
</tr>
<tr>
<td>LNODE</td>
<td>Specifies APPN len nodes.</td>
</tr>
<tr>
<td>LU</td>
<td>Specifies VTAM logical units.</td>
</tr>
<tr>
<td>LU_GROUP</td>
<td>Specifies VTAM logical unit groups.</td>
</tr>
<tr>
<td>MAJNODE</td>
<td>Specifies VTAM major nodes.</td>
</tr>
<tr>
<td>MDL VIEW</td>
<td>Defines a more detailed logical view, which contains the resources on the control cards that follow it.</td>
</tr>
<tr>
<td>MDP VIEW</td>
<td>Defines a more detailed physical view, which contains the resources on the control cards that follow it.</td>
</tr>
<tr>
<td>MIG_DATA_HOST</td>
<td>Specifies Migration Data Hosts.</td>
</tr>
<tr>
<td>NCP</td>
<td>Specifies NCP resources.</td>
</tr>
<tr>
<td>NETWORK</td>
<td>Specifies the MultiSystem Manager or APPNTAM network aggregate resource.</td>
</tr>
<tr>
<td>NNODE</td>
<td>Specifies APPN network nodes.</td>
</tr>
<tr>
<td>NONSNA</td>
<td>Specifies Non-SNA (GMFHS managed real) resources.</td>
</tr>
<tr>
<td>NTF_GROUPAGG</td>
<td>Specifies NetFinity Group Aggregate resources.</td>
</tr>
</tbody>
</table>
### BLDVIEWS Control Cards

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTF_MONITOR</td>
<td>Specifies NetFinity Monitor resources.</td>
</tr>
<tr>
<td>NTF_OS</td>
<td>Specifies NetFinity Operating System resources.</td>
</tr>
<tr>
<td>NTF_PROCESS</td>
<td>Specifies the NetFinity Process resources.</td>
</tr>
<tr>
<td>NTF_SECMON</td>
<td>Specifies the NetFinity Security Monitor resources.</td>
</tr>
<tr>
<td>NTFS_Pname</td>
<td>Specifies the VTAM PU, LU, or CP name for the NetFinity service point which manages the resources on the control cards that follow it.</td>
</tr>
<tr>
<td>OTHER</td>
<td>Specifies a resource from a user-created or MultiSystem Manager open class.</td>
</tr>
<tr>
<td>PCVIEW</td>
<td>Defines a configuration physical connectivity view, which contains the resources on the control cards that follow it.</td>
</tr>
<tr>
<td>PU</td>
<td>Specifies VTAM physical units.</td>
</tr>
<tr>
<td>PVIEW</td>
<td>Defines a configuration peer view, which contains the resources on the control cards that follow it.</td>
</tr>
<tr>
<td>SEGment</td>
<td>Specifies the LNM segment real or aggregate resources.</td>
</tr>
<tr>
<td>SNA</td>
<td>Specifies VTAM SNA shadow resources.</td>
</tr>
<tr>
<td>SNA_DOMAIN</td>
<td>Specifies the global VTAM domain which owns the resources on the control cards that follow it.</td>
</tr>
<tr>
<td>SNA_PORT</td>
<td>Specifies the SNA port.</td>
</tr>
<tr>
<td>SNA_LOCALTOPO</td>
<td>Specifies the APPN SNA local topology resources.</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>Specifies system aggregate resources.</td>
</tr>
<tr>
<td>TG</td>
<td>Specifies APPN transmission groups.</td>
</tr>
<tr>
<td>TME_MONITOR</td>
<td>Specifies TME Monitor resources.</td>
</tr>
<tr>
<td>TME_POLICYREGION</td>
<td>Specifies TME Policy Region resources.</td>
</tr>
<tr>
<td>TME_TMR</td>
<td>Specifies the TME Managed Region resources that you to process.</td>
</tr>
<tr>
<td>TMES_Pname</td>
<td>Specifies the IP name or address for the TME service point which manages the resources on the control cards that follow it.</td>
</tr>
<tr>
<td>VIEW</td>
<td>Defines a network view, which contains the resources on the control cards that follow it.</td>
</tr>
<tr>
<td>VRN</td>
<td>Specifies APPN virtual routing nodes.</td>
</tr>
<tr>
<td>WILDCARD</td>
<td>Defines wildcard characters to use when coding wildcard names on the control statements.</td>
</tr>
</tbody>
</table>

### Control Card Syntax

BLDVIEWS control cards 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 card can span 1 or more lines. There are two types of continuation available:

- A control card 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 card 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.

Control cards can be coded in a:
- 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 cards 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 cards 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 cards 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 will always be translated to upper case:
- ADaPter
- 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
- BRidge
- CAU
- IPSPname
- LANSPname
- NTFSPname
- TMESPname
- SEGment
- SNA
- SNA_DOMAIN

For all other resource names, code them in the same case, because that is how they are displayed by NMC 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 Card Parameters

The following parameters are common to many of the BLDVIEWS control cards and are documented here and referenced later in the documentation by the control cards 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). This enables you 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 aggregate’s TotalRealResourceCount field and will then 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 will multiply it by the total number of real resources linked to the aggregate to come up with the threshold.
BLDVIEWS Control Cards

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 card which follows a view control card with a layout type of 9 (grid).
- ROW must be specified if COLUMN is specified.

CONSOLE:

Description: You can exploit remote console support, which enables you to click on a resource and then issue a command, such as TELNET, or to remotely logon to a resource. Although this is referred to as remote console support, any command can be specified. The command runs on the NMC 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 Tivoli NetView for z/OS NetView Management Console User’s Guide 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 executed 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 only supported for the following 
resources identified by the following control 
cards:  
−  ADAPTER  
−  CAU  
−  IP_HOST  
−  INTERFACE

%IPADDRESS%  Is substituted with the internet address of the 
resource.

%CMDRTEADDRESS%  Is substituted with the command routing address 
of the resource. This variable is only supported 
for the NetFinity NTF_OS, NTF_MONITOR, and 
NTF_PROCESS control cards.

• BLDVIEWS enables the following NetView variables to be coded anywhere in 
the command text: 

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>netid()</td>
<td>VTAM network identifier</td>
</tr>
<tr>
<td>domain()</td>
<td>Current NetView domain</td>
</tr>
<tr>
<td>opid()</td>
<td>NetView operator or task ID</td>
</tr>
<tr>
<td>cursys()</td>
<td>Current operating system name</td>
</tr>
<tr>
<td>vtam()</td>
<td>VTAM version and release</td>
</tr>
<tr>
<td>netview()</td>
<td>NetView version and release</td>
</tr>
<tr>
<td>mvslevel()</td>
<td>MVS version and release</td>
</tr>
<tr>
<td>opsystem()</td>
<td>Type of operating system</td>
</tr>
<tr>
<td>sysplex()</td>
<td>Name of the MVS sysplex</td>
</tr>
</tbody>
</table>

• 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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>netid()</td>
<td>VTAM network identifier</td>
</tr>
<tr>
<td>domain()</td>
<td>Current NetView domain</td>
</tr>
<tr>
<td>opid()</td>
<td>NetView operator or task ID</td>
</tr>
<tr>
<td>cursys()</td>
<td>Current operating system name</td>
</tr>
<tr>
<td>vtam()</td>
<td>VTAM version and release</td>
</tr>
<tr>
<td>netview()</td>
<td>NetView version and release</td>
</tr>
<tr>
<td>mvslevel()</td>
<td>MVS version and release</td>
</tr>
<tr>
<td>opsystem()</td>
<td>Type of operating system</td>
</tr>
<tr>
<td>sysplex()</td>
<td>Name of the MVS sysplex</td>
</tr>
</tbody>
</table>

**Syntax:**  
CORRELATER=’USA VA RICHMOND’
CORRELATER=’text’

**Usage Notes:** Single quotation marks (’) or double quotation marks (″) can be used 
as a delimiter.
BLDVIEWs Control Cards

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. This enables you to reformat the names of multiple resources at once with one control card. 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 NMC Data1 field.

Syntax:
OTHER_DATA='other_data'

Example:
**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()` : Type 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 card which follows a view control card 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 NMC 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:
BLDVIEWs Control Cards

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>netid()</td>
<td>VTAM network identifier</td>
</tr>
<tr>
<td>domain()</td>
<td>Current NetView domain</td>
</tr>
<tr>
<td>opid()</td>
<td>NetView operator ID or task ID</td>
</tr>
<tr>
<td>cursys()</td>
<td>Current operating system name</td>
</tr>
<tr>
<td>vtam()</td>
<td>VTAM version and release</td>
</tr>
<tr>
<td>netview()</td>
<td>NetView version and release</td>
</tr>
<tr>
<td>mvlevel()</td>
<td>MVS version and release</td>
</tr>
<tr>
<td>opsystem()</td>
<td>Type of operating system</td>
</tr>
<tr>
<td>sysplex()</td>
<td>Name of the MVS sysplex</td>
</tr>
</tbody>
</table>

**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 cards to create the resource and then code the control cards 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 cards to create the resource and then code the control cards to update the resource.

The SUSPEND and SUSPEND_WITH_AUTO_CLEAR keywords enable you 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 cards to create the resource and then code the control cards 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 the NMC console or use BLDVIEWs.
The state of the UserStatus bits can be displayed in the Resource Information pop-up window.

**Generic Commands:**

*Description:* BLDVIEWS enables you to set generic commands for objects. The NMC generic commands function enables an NMC operator to 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 only supported for the following resources identified by the following control cards:
    - ADAPTER
    - CAU
    - IP_HOST
    - INTERFACE
  - `%IPADDRESS%`: Substituted with the internet address of the resource. This variable is only supported for the NWSERVER control card.
- 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
BLDVIEWS Control Cards

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 card to define wildcard characters.

**WILDCARD Control Card:**

*Description:* The WILDCARD control card defines wildcard characters to use when coding a wildcard pattern matching name on a RESOURCE control card.

*wildcard_single_character* and *wildcard_string* are special characters used to define a wildcard pattern matching name.

The default value of *wildcard_single_character* and *wildcard_string* is an *

**WildCard**

\[
\text{WildCard} = \left( \text{wildcard_single_character} \right) \left( \text{wildcard_string} \right)
\]

*Parameters:*

*wildcard_single_character* and *wildcard_string*

- Special characters used to define a wildcard 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* is an asterisk (*).

*wildcard_single_character*

- Used when you wish to perform a wildcard match on 1 character. The *wildcard_single_character* coded in a position in the wildcard pattern matching name will always match the character in the corresponding position in the resource name.

  If *wildcard_string* is specified in a wildcard pattern matching name in any position but the last position of the wildcard pattern matching name, then it will be treated as a *wildcard_single_character* (perform a wildcard match on 1 character in the position specified).

*wildcard_string*

- Used when you wish to perform a wildcard match on a remaining string of characters at the end of a resource name. The *wildcard_string* coded at the end of a wildcard pattern matching name will always match the string of characters at the corresponding position in the resource name.

  If *wildcard_string* is specified in a wildcard pattern matching name in any position except the last position of the wildcard pattern matching name, it is treated as a *wildcard_single_character* (perform a wildcard match on 1 character in the position specified).
Defining Wildcard Characters

Examples: Assume WILDCARD=(?,*) for the following pattern matching examples:

<table>
<thead>
<tr>
<th>Pattern Match Example</th>
<th>Matches Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRIDGE=A001B*</td>
<td>Matches all bridge resources whose names begin with A001B.</td>
</tr>
<tr>
<td>BRIDGE=????B001</td>
<td>Matches all bridge resources whose names are eight characters in length and end with B001.</td>
</tr>
<tr>
<td>SEGMENT=?C?0</td>
<td>Matches all segment resources whose names have a C in position 2 and a 0 in position 4.</td>
</tr>
<tr>
<td>ADP=??SERV*</td>
<td>Matches all adapter resources whose names are 6 or more characters in length, and have SERV in positions 3 through 6.</td>
</tr>
<tr>
<td>ADP=??PRINTER0?</td>
<td>Matches all adapter resources whose names are 10 or 11 characters in length, and have PRINTER0 in positions 3 through 10.</td>
</tr>
</tbody>
</table>

Assume WILDCARD = *,* (the default), for the following pattern matching examples:

<table>
<thead>
<tr>
<th>Pattern Match Example</th>
<th>Matches Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRIDGE=A001B*</td>
<td>Matches all bridge resources whose names begin with A001B.</td>
</tr>
<tr>
<td>BRIDGE=****B001</td>
<td>Matches all bridge resources whose names are eight characters in length and end with B001.</td>
</tr>
<tr>
<td>SEGMENT=<em>C</em>0</td>
<td>Matches all segment resources whose names have a C in position 2 and a 0 in position 4.</td>
</tr>
<tr>
<td>ADP=*<em>SERV</em></td>
<td>Matches all adapter resources whose names are 6 or more characters in length, and have SERV in positions 3 through 6.</td>
</tr>
<tr>
<td>ADP=*<em>PRINTER0</em></td>
<td>Matches all adapter resources whose names are 10 or more characters in length, and have PRINTER0 in positions 3 through 10.</td>
</tr>
</tbody>
</table>

Selective Control Cards
The following selective control cards enable you to be more selective in specifying resources to be processed by BLDVIEWS, or enables you to specify common information to be used to locate certain resources in RODM. Wildcard is not valid for these types of control cards.

Service Point Control Card:

Description: The service point control card specifies the service point that manages the resources on the control cards following the service point control card. The service point control card enables you to be more selective in specifying resources to be processed by BLDVIEWS. This service point name can be overridden on individual control cards using the SPname keyword.

The following service point control cards are enabled:
- LANSPname
- NTFSSPname
- TMESPname
- IPSPname
Selective Control Cards

Syntax:

ATMSPname

ATMSPname=ALL

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 cards with a name of ALL or a resource name, the resources that get processed depend on whether a service point control card was previously specified.
• If no prior service point statement was specified and a resource control card was coded with ALL for a resource name, all resources are processed.
• If no prior service point statement was specified and a resource control card was coded with a wildcard resource name, all resources that match the wildcard name are processed.
• If a prior service point statement was specified and a resource control card 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 card was coded with a wildcard resource name, all resources that match the wildcard name, and are managed by that service point, are processed.

SNA_DOMAIN Control Card:

Description: The SNA_DOMAIN control card specifies the SNA domain that owns the SNA topology manager resources on the control cards following the SNA_DOMAIN control card. 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 cards using the SNA_DOMAIN keyword.

Syntax:

SNA_DOMAIN

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)
**Selective Control Cards**

**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 card)
- CDRMs (CDRM control card)
- CDRSCs (CDRSC control card)
- Logical Units (LU control card)
- Logical Unit Groups (LU_GROUP control card)

The SNA Domain Name can also be specified on those control cards using the SNA_DOMAIN keyword in which case it overrides the SNA_DOMAIN control card.

**View Control Cards**

The following view control cards define the types of views to be created.

**VIEW Control Card:**

*Description:* The VIEW control card defines a network view which contains the resources on the control cards that follow it.

*Syntax:*

```
VIEW
```

```
VIEW=view_name[,ANNOTATION=annotation][,LAYOUT=layout_type][,CREATE=YES][,CREATE=BUILD][,LAYOUT_WIDTH=layout_width]
```

*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
View Control Cards

layout_width
An integer which specifies how many resource objects should appear horizontally on one line in the view. The default 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 Card:

Description: The EVIEW control card defines an exception view.

Syntax:

EVIEW

```
EVIEW=view_name,ANNOTATION=annotation
,LAYOUT_WIDTH=layout_width,EVIEW_NAME=exceptionviewname
,DSF=IGNORE,NOXCPT
,MF=IGNORE,OFF,ON
,SAPF=IGNORE,OFF,ON
,TIF=IGNORE,OFF,ON
,AIPF=IGNORE,OFF,ON
,CREATE=YES,BUILD
```

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 should appear horizontally on one line in the view. The default 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.
**View Control Cards**

**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. 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.

**PVIEW Control Card:**

Description: The PVIEW control card defines a configuration peer view, which contains the resources on the control cards that follow it.

Syntax:

```
PVIEW
```

```
PVIEW=view_name, LAYOUT=layout_type, LAYOUT_WIDTH=layout_width, CREATE=YES
```

```
PVIEW=view_name, LAYOUT=layout_type, LAYOUT_WIDTH=layout_width, CREATE=BUILD
```

```
PVIEW=view_name, LAYOUT=layout_type, LAYOUT_WIDTH=layout_width, CREATE=NO
```

```
PVIEW=view_name, LAYOUT=layout_type, LAYOUT_WIDTH=layout_width, CREATE=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 should appear horizontally on one line in the view. The default 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.
**View Control Cards**

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 Card:**

*Description:* The BBVIEW control card defines a configuration backbone view, which contains the resources on the control cards that follow it.

*Syntax:*

```
BBVIEW

+<.-VVVV)+<.-VVVV

BBVIEW=view_name,ANNOTATION=annotation,,LAYOUT=layout_type

+<.-VVVV

+<.-VVVV

,CREATE=NO

,CREATE=YES

,LAYOUT_WIDTH=layout_width

```

*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 should appear horizontally on one line in the view. The default is 0 which will result 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.
View Control Cards

**Build**  
Create a new object for this view if it does not exist. If it does exist, update the object.

**LCVIEW Control Card:**

*Description:* The LCVIEW control card defines a Configuration Logical Connectivity View which contains the resources on the control cards that follow it.

*Syntax:*

```
LCVIEW

LCVIEW=view_name,,LAYOUT=layout_type,,LAYOUT_WIDTH=layout_width,,CREATE=CREATE
```

*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 should appear horizontally on one line in the view. The default is 0 which will result 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 Card:**

*Description:* The PCVIEW control card defines a configuration physical connectivity view, which contains the resources on the control cards that follow it.

Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer’s Guide
Syntax:

PCVIEW

Syntax:

MDLVIEW

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. BLDVIEW 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 should appear horizontally on one line in the view. The default is 0 which will result 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.

No Do not create a new object for this view. Update the existing object. If the object does not exist, an error will occur.

Build Create a new object for this view if it does not exist. If it does exist, update the object.

MDLVIEW Control Card:

Description: The MDLVIEW control card defines a more detail logical view, which contains the resources on the control cards that follow it.

Syntax:

MDLVIEW
View Control Cards

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 should appear horizontally on one line in the view. The default 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 Card:

Description: The MDPVIEW control card defines a more detail physical view, which contains the resources on the control cards that follow it.

Syntax:

MDPVIEW
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 should appear horizontally on one line in the view. The default 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 Cards

The following resource control cards specify resources to be processed by BLDVIEWS.

ADAPter Control Card:

Description: The ADAPter control card specifies the MultiSystem Manager LNM adapter resource to be processed. For LNM ports (supported in Lan Network Manager V2), see the LAN_PORT control card.

Syntax:

ADAPter
Resource Control Cards

Parameters:

adapter_name
  STATION, BRIDGE, CAU or LAN
  • 1–12 character adapter mac address or the
  • 1–16 character adapter mac name

    ALL or a wildcard name can be specified.

TYPE
  Specifies the type of adapter resource. The values are:
  • STATION - adapter is a station adapter (default)
  • BRIDGE - adapter is a bridge adapter
  • CAU - adapter is a Controlled Access Unit adapter
  • LAN - adapter is a LNM adapter (can be a STATION, BRIDGE or CAU)

segment_name
  STATION, BRIDGE, CAU or LAN, segment number (3–4 characters) or
  segment name (for example, SEGxxxx).

    ALL can be specified and is the default.
AGENT Control Card:

**Description:** The AGENT control card specifies the MultiSystem Manager agent resource to be processed.

**Syntax:**

AGENT

Parameters:

**application**

The 1–8 character service point application name.

- The Lan Network Manager agent application name is LANMGR.
- The agent application name for NetView for AIX is the name registered to AIX NetView Service Point.
- The NetFinity agent application name is REMOTEOP.FLCHNETF.
The TME agent application name is MSMTME.
ALL or a wildcard name can be specified.

**TYPE**
Specifies the type of agent. TYPE is ignored if the agent name specified is ALL or a wildcard name.

- **LAN**  
  Lan Network Manager IBM agent
- **IP**  
  NetView for AIX IBM agent
- **NTF**  
  NetFinity IBM agent
- **TME**  
  TME IBM agent

**service_point**
The VTAM PU, LU, or CP name for the agent.
ALL is the default.

See "Common Control Card Parameters" on page 598 for a description of the other supported keywords.

**BRidge Control Card:**

**Description:** The BRidge control card specifies the MultiSystem Manager LNM bridge resource to be processed.

**Syntax:**

```plaintext
BRidge
```

- `bridge_name`  
  ALL, TYPE= AGG
- `service_point`
  ALL, AGGPRI=aggregation_priority
- `AGGTHRESH=(xxx,yyy,zzz)`
- `ACTivate=activate_command`
- `DEACTivate=deactivate_command`
- `RECYcle=recycle_command`
- `DISPLAY=display_command`
  CONSOLE=command
  USER_DATA=user_data
  CORRELATER=text
- `DISPLAY_STATUS=status_integer`
  UNLINK
  ROW=row_on_view`
Parameters:

bridge_name
The 1–8 character bridge name. ALL or a wildcard name can be specified.

TYPE
Specifies the type of bridge resource. The values are:

- REAL  real bridge resource
- AGG    aggregate bridge resource

See "Common Control Card Parameters" on page 598 for a description of the other supported keywords.

CAU Control Card:

Description: The CAU control card specifies the MultiSystem Manager LNM Controlled Access Unit resource to be processed.

Syntax:

CAU

```plaintext
CAU=cau_name
,TYPE=REAL
,SPname=service_point

,SEGMENT=REAL
,SEGMENT=segment
,AGGPRI=aggregation_priority

,AGGTHRESH=(xxx,yyy,zzz)
,ACTivate=activate_command

,DEACTivate=deactivate_command
,RECYcle.recycle_command
```
Resource Control Cards

Parameters:

cau_name
   The 1–8 character Controlled Access Unit name. ALL or a wildcard name can be specified.

TYPE
   Specifies the type of CAU resource. The values are:
   REAL real CAU resource
   AGG aggregate CAU resource

segment_name
   The segment number (3–4 characters) or segment name (for example, SEGxxxx). ALL can be specified and is the default.

See “Common Control Card Parameters” on page 598 for a description of the other supported keywords.

CDRM Control Card:

Description: The CDRM control card specifies the VTAM CDRM resource to be processed.

Syntax:

CDRM
Parameters:

**name**
The 1–17 character VTAM CDRM name in the format of: snaNetID.snaNodeName. ALL or a wildcard 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 Card Parameters" on page 598 for a description of the other supported keywords.

**CDRSC Control Card:**

*Description:* The CDRSC control card specifies the VTAM CDRSC resource to be processed.

*Syntax:*

```
CDRSC
```

Resource Control Cards
Resource Control Cards

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 wildcard 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 Card Parameters” on page 598 for a description of the other supported keywords.

CIRCUIT Control Card:

Description: The CIRCUIT control card 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
Resource Control Cards

Parameters:

name
The SNA Circuit name in the format of snaNetID.circuitID. The name is in the same format that is displayed on the NMC for the resource (DisplayResourceName). ALL or a wildcard 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 Card Parameters" on page 598 for a description of the other supported keywords.

CLUSTER Control Card:

Description: The CLUSTER control card specifies the MultiSystem Manager or APPN Cluster aggregate resource to be processed. This aggregate can contain 1 or more network aggregates.

Syntax:
Parameters:

**cluster_name**

The name of the CLUSTER aggregate resource.

For TYPE=LAN, TYPE=IP, TYPE=NTF, or TYPE=TME, the name is the value specified for the NETWORK_AGG_OBJECT on the GETTOPO command or statement.

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 wildcard name can be specified.

**TYPE**

Specifies the type of CLUSTER aggregate resource. The values are:

- **LAN**  
  Lan Network Manager (LNM)
- **IP**  
  TCP/IP
- **APPN**  
  APPN
- **NTF**  
  NetFinity
- **TME**  
  Tivoli Managed Enterprise

See ["Common Control Card Parameters" on page 598](#) for a description of the other supported keywords.
DOMAIND Control Card:

Description: The DOMAIND control card specifies the APPN Domain resource to be processed.

Syntax:

\[ \text{DOMAIND} \]

Parameters:

name

The 1–17 character APPN network node domain name in the format: snaNetID.snaNodeName. ALL or a wildcard name can be specified.

See ["Common Control Card Parameters" on page 598](#) for a description of the other supported keywords.

ENODE Control Card:

Description: The ENODE control card specifies the APPN End Node resource to be processed.

Syntax:

\[ \text{ENODE} \]
Resource Control Cards

Parameters:

name

The 1–17 character SNA end node resource name in the format: snaNetID.snaNodeName. ALL or a wildcard name can be specified.

See "Common Control Card Parameters" on page 598 for a description of the other supported keywords.

GW_NCP Control Card:

Description: The GW_NCP control card specifies the SNA Communication Controller node resource functioning as gateways to be processed.

Syntax:

GW_NCP

GW_NCP= 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

AUTO_IN_PROGRESS=OFF

SUSPEND=OFF,ROW=row_on_view

SUSPEND_WITH_AUTO_CLEAR=OFF
Parameters:

**name**

The 1–17 character SNA Communication Controller node in the format of: snaNetID.snaNodeName. ALL or a wildcard name can be specified.

See “Common Control Card Parameters” on page 598 for a description of the other supported keywords.

**HOST_NODE Control Card:**

*Description:* The HOST_NODE control card 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=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

,AUTO_IN_PROGRESS=OFF

,SUSPEND=OFF

,SUSPEND_WITH_AUTO_CLEAR=OFF
```

Resource Control Cards
Parameters:

name
The 1–17 character SNA Host Node name in the format of:
snaNetID.snaNodeName. ALL or a wildcard name can be specified.

See "Common Control Card Parameters" on page 598 for a description of the other supported keywords.

IC_NODE Control Card:

Description: The IC_NODE control card specifies the SNA Interchange Node resources to be processed.

Syntax:

IC_NODE

IC_NODE=

name

ALL

CDS

EBN

GWS

ICN

IRS

PBN

CONSOLE=command

USER_DATA=user_data

CORRELATER=text

DISPLAY_STATUS=status_integer

UNLINK

ROW=row_on_view

COLUMN=column_on_view

MARK=OFF

AUTO_IN_PROGRESS=OFF

SUSPEND=OFF

SUSPEND_WITH_AUTO_CLEAR=OFF
Resource Control Cards

Parameters:

name
The 1–17 character SNA Interchange Node name in the format of:
snaNetID.snaNodeName. ALL or a wildcard 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 only supported for a
name of ALL or a wildcard name.

See "Common Control Card Parameters" on page 598 for a description of the other
supported keywords.

INTERFACE Control Card:

Description: The INTERFACE control card specifies the MultiSystem Manager
TCP/IP adapter resource to be processed.

Syntax:

INTERFACE

\[
\text{INTERFACE} = \text{adapter\_name} \leftarrow \text{SPname} = \text{ALL}\rightarrow
\text{SEGMENT} = \text{segment} \leftarrow \text{AGGPRII} = \text{aggregation\_priority}\rightarrow
\text{ACTivate} = \text{activate\_command} \leftarrow \text{DEACTivate} = \text{deactivate\_command}\rightarrow
\text{RECYcle} = \text{recycle\_command} \leftarrow \text{DISPLAY} = \text{display\_command}\rightarrow
\text{CONSOLE} = \text{command} \leftarrow \text{USER\_DATA} = \text{user\_data}\rightarrow
\text{DISPLAY\_STATUS} = \text{status\_integer} \leftarrow \text{UNLINK} \leftarrow \text{ROW} = \text{row\_on\_view}\rightarrow
\]
Resource Control Cards

Parameters:

adapter_name
The TCP/IP interface adapter name.
ALL or a wildcard name can be specified.

segment_name
The segment name.
ALL can be specified and is the default.

See "Common Control Card Parameters" on page 598 for a description of the other supported keywords.

IP_BRIDGE Control Card:

Description: The IP_BRIDGE control card specifies the MultiSystem Manager TCP/IP bridge aggregate resource to be processed.

Syntax:

IP_BRIDGE

IP_BRIDGE= name

SPname= ALL

AGGTHRESH= (xxx, yyy, zzz)

ACTivate= activate_command

DEACTivate= deactivate_command

RECYcle= recycle_command

DISPLAY= display_command

CONSOLE= command

USER_DATA= user_data

CORRELATER= text
Parameters:

**name**

The TCP/IP bridge name. ALL or a wildcard name can be specified.

See [“Common Control Card Parameters” on page 598](#) for a description of the other supported keywords.

**IP_HOST Control Card:**

**Description:** The IP_HOST control card specifies the MultiSystem Manager TCP/IP host aggregate resource to be processed.

**Syntax:**

```
IP_HOST
```

```
IP_HOST=name
```

```
SPname=ALL
```

```
AGGTHRESH=(xxx,yyy,zzz)
```

```
ACTivate=activate_command
```

```
DEACTivate=deactivate_command
```

```
RECYcle.recycle_command
```

```
DISPLAY=display_command
```

```
CONSOLE=command
```

```
USER_DATA=user_data
```

```
CORRELATER=text
```

```
DISPLAY_STATUS=status_integer
```

```
UNLINK
```

```
ROW=row_on_view
```
Parameters:

**name**

The TCP/IP host name. ALL or a wildcard name can be specified.

See ["Common Control Card Parameters" on page 598](#) for a description of the other supported keywords.

**IP_HUB Control Card:**

**Description:** The IP_HUB control card specifies the MultiSystem Manager TCP/IP hub aggregate resource to be processed.

**Syntax:**

```
IP_HUB
```

```
IP_HUB= name

,SPname= ALL

,AGGTHRESH=(xxx,yyy,zzz)

,ACTivate= activate_command

,DEACTivate= deactivate_command

,RECYcle= recycle_command

,DISPLAY= display_command

,CONSOLE= command

,USER_DATA= user_data

,CORRELATER= text

,DISPLAY_STATUS= status_integer

,UNLINK

,ROW= row_on_view
```
Parameters:

name

The TCP/IP hub name. ALL or a wildcard name can be specified.

See “Common Control Card Parameters” on page 598 for a description of the other supported keywords.

IP_LINK Control Card:

Description: The IP_LINK control card specifies the MultiSystem Manager TCP/IP interface link aggregate resource to be processed.

Syntax:

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
```

```
USER_DATA=user_data
```

```
CORRELATER=text
```

```
DISPLAY_STATUS=status integer
```

```
UNLINK
```

```
ROW=row on view
```

Resource Control Cards

Parameters:

name
The TCP/IP Link name. ALL or a wildcard name can be specified.

See “Common Control Card Parameters” on page 598 for a description of the other supported keywords.

IPLOCATION Control Card:

Description: The IP_LOCATION control card specifies the MultiSystem Manager TCP/IP location resource to be processed.

Syntax:

IP_LOCATION

IP_LOCATION=name

,SPname=ALL

,service_point=

,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

,AUTO_IN_PROGRESS=OFF

,SUSPEND=OFF

,SUSPEND_WITH_AUTO_CLEAR=OFF
Parameters:

**name**

The TCP/IP location name. ALL or a wildcard name can be specified.

See "[Common Control Card Parameters](#) on page 598" for a description of the other supported keywords.

**IP_ROUTER Control Card:**

**Description:** The IP_ROUTER control card specifies the MultiSystem Manager TCP/IP router aggregate resource to be processed.

**Syntax:**

```plaintext
IP_ROUTER

name

SPname=ALL

service_point

AGGTHRESH=(xxx,yyy,zzz)

ACTivate=activate_command

DEACTivate=deactivate_command

RECYcle=recycle_command

DISPLAY=display_command

CONSOLE=command

USER_DATA=user_data

CORRELATER=text

DISPLAY_STATUS=status_integer

UNLINK

display_command

ROW=row_on_view

COLUMN=column_on_view

MARK=OFF

AUTO_IN_PROGRESS=OFF

SUSPEND=OFF

ALL

OFF

ON
```
Resource Control Cards

Parameters:

name
The TCP/IP router name. ALL or a wildcard name can be specified.

See "Common Control Card Parameters" on page 598 for a description of the other supported keywords.

IP_SEGMENT Control Card:

Description: The IP_SEGMENT control card specifies the MultiSystem Manager TCP/IP Segment aggregate resource to be processed.

Syntax:

IP_SEGMENT

Parameters:

name
The TCP/IP segment name. ALL or a wildcard name can be specified.

See "Common Control Card Parameters" on page 598 for a description of the other supported keywords.
IP_SUBNET Control Card:

Description: The IP_SUBNET control card specifies the MultiSystem Manager TCP/IP Subnetwork aggregate resource to be processed.

Syntax:

```
IP_SUBNET

IP_SUBNET=ALL

SPname=ALL

service_point

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

AUTO_IN_PROGRESS=OFF

SUSPEND=OFF

SUSPEND_WITH_AUTO_CLEAR=OFF
```

Parameters:

name

The TCP/IP Subnetwork name. ALL or a wildcard name can be specified.

See “Common Control Card Parameters” on page 598 for a description of the other supported keywords.

LAN_PORT Control Card:

Description: The LAN_PORT control card specifies the LAN resource to be processed. It is for LNM V2.

Syntax:

```
LAN_PORT

LAN_PORT=ALL

SPname=ALL

service_point
```
Resource Control Cards

- SEGMENT=segment
- ALL
- AGGPR1=aggregation_priority

- ACTivate=activate_command
- DEACTivate=deactivate_command

- RECYcle.recycle_command
- DISPLAY=display_command

- CONSOLE=command
- USER_DATA=user_data
- CORRELATER=text

- DISPLAY_STATUS=status_integer
- UNLINK
- ROW=row_on_view

- COLUMN=column_on_view
- MARK=OFF

- AUTO_IN_PROGRESS=OFF
- SUSPEND=OFF

- SUSPEND_WITH_AUTO_CLEAR=OFF

Parameters:

name
The LNM Port resource name as determined by Lan Network Manager V2 (and displayed on the NMC for the resource using DisplayResourceName). ALL or a wildcard name can be specified.

See "Common Control Card Parameters" on page 598 for a description of the other supported keywords.

LINE Control Card:

Description: The LINE control card specifies the SNA Line resource to be processed.

Syntax:

LINE

- LINE= name
- ALL
- AGGPR1=aggregation_priority
- CONSOLE=command
- USER_DATA=user_data
Parameters:

**name**

The 1–17 character SNA line name in the format of: snaNetID.snaNodeName. ALL or a wildcard name can be specified.

See “Common Control Card Parameters” on page 598 for a description of the other supported keywords.

**LLINK Control Card:**

Description: The LLINK control card specifies the Logical Link resource to be processed.

Syntax:

**LLINK**

```
LLINK=name,TYPE=ALL,AGGPR=aggregation_priority
```

Consoles: command

User data: user_data

Correlator: text

Display status: status_integer

Row: row_on_view

Unlink: ON

Auto in progress: OFF

Suspend: OFF

Suspend with auto clear: OFF
Resource Control Cards

Parameters:

name
The SNA Logical Link resource name in the format: network.resource.link. ALL or a wildcard name can be specified.

TYPE
Specifies the type of Logical LInk. TYPE is ignored when you specify an exact resource name. It is only supported for a name of ALL or a wildcard 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 Card Parameters" on page 598 for a description of the other supported keywords.

LNODE Control Card:

Description:  The LNODE control card specifies the APPN Len Node resource to be processed.

Syntax:

LNODE

-LNODE= name
-ALL
-AGGPR1=aggregation_priority

-CONSOLE=command
-USER_DATA=user_data
-CORRELATER=text

-DISPLAY_STATUS=status_integer
-UNLINK
-ROW=row_on_view
Parameters:

name
The 1–17 character SNA LEN node resource name in the format: snaNetID.snaNodeName. ALL or a wildcard name can be specified.

See "Common Control Card Parameters" on page 598 for a description of the other supported keywords.

LU Control Card:

Description: The LU control card specifies the SNA Logical Unit resource to be processed.

Syntax:

LU

```plaintext
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

AUTO_IN PROGRESS=OFF
SUSPEND=OFF
SUSPEND_WITH_AUTO_CLEAR=OFF
```
Resource Control Cards

Parameters:

name
The 1–17 character SNA logical unit name in the format of:
snaNetID.snaNodeName. ALL or a wildcard 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 Card Parameters" on page 598 for a description of the other supported keywords.

LU_GROUP Control Card:

Description: The LU_GROUP control card specifies the SNA Logical Unit group resources to be processed.

Syntax:

LU_GROUP

Parameters:
Resource Control Cards

name
The 1–17 character SNA logical unit group name the format of: luGroupName.
ALL or a wildcard 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 Card Parameters" on page 598 for a description of the other supported keywords.

MAJNODE Control Card:

Description: The MAJNODE control card specifies the VTAM Major Node resource to be processed.

Syntax:

MAJNODE

Parameters:

name
The 1–8 character VTAM Major node name in the format of: snaNodeName.
ALL or a wildcard name can be specified.
**Resource Control Cards**

### 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 Card Parameters” on page 598](#) for a description of the other supported keywords.

### MIG_DATA_HOST Control Card:

**Description:** The MIG_DATA_HOST control card specifies the SNA Migration Data Host node resource to be processed.

**Syntax:**

```
MIG_DATA_HOST
```

```
 MIG_DATA_HOST = name
 MIG_DATA_HOST = ALL
 MIG_DATA_HOST = AGGPRI = aggregation_priority
```

```
 MIG_DATA_HOST = CONSOLE = command
 MIG_DATA_HOST = USER_DATA = user_data
```

```
 MIG_DATA_HOST = DISPLAY_STATUS = status_integer
 MIG_DATA_HOST = UNLINK
 MIG_DATA_HOST = ROW = row_on_view
```

```
 MIG_DATA_HOST = COLUMN = column_on_view
 MIG_DATA_HOST = MARK = OFF
```

```
 MIG_DATA_HOST = AUTO_IN_PROGRESS = OFF
```

```
 MIG_DATA_HOST = SUSPEND = OFF
```
Parameters:

name
The 1–17 character SNA Migration Data Host node in the form of network.name. ALL or a wildcard name can be specified.

See “Common Control Card Parameters” on page 598 for a description of the other supported keywords.

NCP Control Card:

Description: The NCP control card specifies the SNA Communication Controller node resource to be processed.

Syntax:

NCP

Parameters:

name
The 1–17 character SNA Communication Controller node in the format of: snaNetID.snaNodeName. ALL or a wildcard name can be specified.
Resource Control Cards

**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 Card Parameters" on page 598 for a description of the other supported keywords.

**NETWORK Control Card:**

*Description:* The NETWORK control card specifies the MultiSystem Manager or APPN Network aggregate resource to be processed. This aggregate represents the network managed by one service point.

*Syntax:*

```
NETWORK
```

- `NETWORK=network_name`
  - `SPname=ALL`
  - `service_point`
  - `TYPE=APPN`
  - `IP`
  - `LAN`
  - `LMU`
  - `NTF`
  - `TME`
  - `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/ON`
  - `SUSPEND=OFF/ON`
  - `SUSPEND_WITH_AUTO_CLEAR=OFF/ON`

*Parameters:*

- `network_name`
  - The name of the network aggregate resource.
  - For TYPE=LAN, TYPE=IP, or TYPE=LMU, network_name is the 1–8 character service point application name.
Resource Control Cards

- The Lan Network Manager application name is LANMGR.
- The agent application name for NetView for AIX is the name registered to AIX NetView Service Point.
- The LMU application name is REMOTEOP.LMU.

For TYPE=APPN the name is in the format of snaNetid.n where n is a numeric increment. ALL or a wildcard name can be specified.

service_point

- The VTAM PU, LU, or CP name for the LAN, IP, or LMU agent. It is not supported for TYPE=APPN and is ignored.
- ALL is the default.

TYPE

- Specifies the type of NETWORK aggregate resource. The values are:
  - LAN: Lan Network Manager (LNM)
  - IP: TCP/IP
  - APPN: APPN
  - NTF: NetFinity
  - TME: TME

See "Common Control Card Parameters" on page 598 for a description of the other supported keywords.

NNODE Control Card:

Description: The NNODE control card specifies the APPN Network Node resource to be processed.

Syntax:

NNODE

```
\[ name \],\[ TYPE= \]ALL,\[ TYPE= \]CDS,\[ AGGPRI= \]aggregation_priority
\[ EBN \],\[ GWS \],\[ ICN \],\[ IRS \],\[ PBN \]
\[ CONSOLE= \]command,\[ USER_DATA= \]user_data,\[ CORRELATER= \]text
\[ DISPLAY_STATUS= \]status_integer,\[ UNLINK \],\[ ROW= \]row_on_view
\[ COLUMN= \]column_on_view,\[ MARK= \]OFF,\[ ON \]
```
Parameters:

name
The 1–17 character SNA network node resource name in the format: snaNetID.snaNodeName. ALL or a wildcard 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 wildcard 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 Card Parameters" on page 598 for a description of the other supported keywords.

NONSNA Control Card:

Description: The NONSNA control card 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_resource_name
```

```
ALL,AGGPR1=aggregation_priority
```

```
ACTivate=activate_command,DEACTivate=deactivate_command
```

```
RECYcle.recycle_command,DISPLAY=display_command
```
Parameters:

nonsna_resource_name
The Non SNA resource name. ALL or a wildcard name can be specified for CREATE=NO

DISPLAY_NAME
Specifies the RODM DisplayResourceName for the object. This value is displayed on the NMC workstation for the resource instead of the RODM 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.

TYPE
Specifies the type of non-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 non-SNA object. You can specify any valid non-SNA DisplayResourceType value documented in the RODM Programming Guide.
Resource Control Cards

**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 card.

**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 Card Parameters" on page 598 for a description of the other supported keywords.

**NTF_GROUPAGG Control Card:**

*Description:* The NTF_GROUPAGG control card specifies the MultiSystem Manager NetFinity Group Aggregate resource to be processed.

*Syntax:*

```
NTF_GROUPAGG
```

```
+<.-VVVV)+<.-VVVV

NTF_GROUPAGG= name

,SPname= ALL,service_point

,AGGTHRESH=(xxx,yyy,zzz),ACTivate=activate_command

,DEACTivate=deactivate_command,RECYcle=recycle_command

,DISPLAY=display_command,CONSOLE=command

,USER_DATA=user_data,CORRELATER=text

,DISPLAY_STATUS=status_integer,UNLINK

,ROW=row_on_view
```
Parameters:

name
The NetFinity Group Aggregate name.
ALL or a wildcard name can be specified.

See "Common Control Card Parameters" on page 598 for a description of the other supported keywords.

NTF_MONITOR Control Card:

Description: The NTF_MONITOR control card specifies the MultiSystem Manager NetFinity Monitor resource to be processed.

Syntax:

NTF_MONITOR

\[\text{NTF\_MONITOR} = \text{name} \], \text{ALL} \], \text{SPname} = \text{ALL} \], \text{service\_point}\]

\[\text{AGGTHRESH} = (xxx, yyy, zzz) \], \text{ACTivate} = \text{activate\_command}\]

\[\text{DEACTivate} = \text{deactivate\_command}\], \text{RECYcle} = \text{recycle\_command}\]

\[\text{DISPLAY} = \text{display\_command}\], \text{CONSOLE} = \text{command}\], \text{USER\_DATA} = \text{user\_data}\]

\[\text{DISPLAY\_STATUS} = \text{status\_integer}\], \text{UNLINK}\], \text{ROW} = \text{row\_on\_view}\]
Resource Control Cards

Parameters:

name
The Netfinity Monitor resource name.
ALL or a wildcard name can be specified.

See "Common Control Card Parameters" on page 598 for a description of the other supported keywords.

NTF_OS Control Card:

Description: The NTF_OS control card specifies the MultiSystem Manager Netfinity Operating System resource to be processed.

Syntax:

**NTF_OS**

```
NTF_OS=
```

```
SPname= ALL
```

```
Activate= activate_command
```

```
DEACTivate= deactivate_command
```

```
RECYcle= recycle_command
```

```
Display= display_command
```
Resource Control Cards

Parameters:

name
The NetFinity Operating System computer name.
ALL or a wildcard name can be specified.

TYPE specifies the type of NetFinity Operating System managed resource.
NTF_OS2 OS/2 Operating Systems
NTF_WINNT Windows NT Operating Systems
NTF_WIN95 Windows 95 Operating Systems
NTF_WIN Windows Operating Systems
NTF_NETWARE Netware Operating Systems
NTF_ALL All Operating Systems

NTF_ALL is the default if not specified.

See "Common Control Card Parameters" on page 598 for a description of the other supported keywords.

NTF_PROCESS Control Card:
Description: The NTF_PROCESS control card specifies the MultiSystem Manager NetFinity Process resource to be processed.

Syntax:

**NTF_PROCESS**

```
--NTF_PROCESS=  name
    ALL
     SPname=  ALL
       service_point
```
Parameters:

name
The NetFinity Process resource name.
ALL or a wildcard name can be specified.

See "Common Control Card Parameters" on page 598 for a description of the other supported keywords.

NTF_SECMON Control Card:

Description: The NTF_SECMON control card specifies the MultiSystem Manager NetFinity Security Monitor resource to be processed.

Syntax:

```
NTF_SECMON
```

Resource Control Cards

Parameters:

name

The NetFinity Security Monitor resource name. ALL or a wildcard name can be specified.

See "Common Control Card Parameters" on page 598 for a description of the other supported keywords.

OTHER Control Card:

Description: The OTHER control card specifies a Real or Aggregate resource from a user defined or MultiSystem Manager Open RODM class to be processed.

Syntax:

OTHER

OTHER=name,CLASS=name,AGGPRI=aggregation_priority

Resource Control Cards

- AGGTHRESH=(xxx,yyy,zzz), ACTivate=activate_command
- DEACTivate=deactivate_command, RECYcle=recycle_command
- DISPLAY=display_command, CONSOLE=command, USER_DATA=user_data, CORRELATER=text
- DISPLAY_STATUS=status_integer, UNLINK, ROW=row_on_view
- COLUMN=column_on_view, MARK=ON
- AUTO_IN_PROGRESS=OFF, SUSPEND=OFF
- SUSPEND_WITH_AUTO_CLEAR=OFF

Appendix A. RODM Tools 657
Parameters:

name
The RODM MyName or DisplayResourceName of the resource in the user defined or MultiSystem Manager Open class.

classname
The name of the user defined or MultiSystem Manager Open class.

See [“Common Control Card Parameters” on page 598](#) for a description of the other supported keywords.

PU Control Card:

Description: The PU control card specifies the SNA Physical Unit resource to be processed.

Syntax:

PU
Parameters:

name
   The 1–17 character SNA physical unit name in the format of: snaNetID.snaNodeName. ALL or a wildcard 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 wildcard name.

See "Common Control Card Parameters" on page 598 for a description of the other supported keywords.

SEGment Control Card:

Description: The SEGment control card specifies the MultiSystem Manager LNM segment resource to be processed.

Syntax:

SEGMENT
Resource Control Cards

Parameters:

*segment_name*

The segment number (3–4 characters) or segment name (for example, SEGxxxx). ALL or a wildcard name can be specified.

**TYPE**

Specifies the type of segment resource. The values are:

- **REAL** Real segment resource
- **AGG** Aggregate segment resource

See [“Common Control Card Parameters” on page 598](#) for a description of the other supported keywords.

**SNA Control Card:**

*Description:* The SNA control card specifies the SNA (GMFHS Shadow) resource to be processed.

*Syntax:*
SNA

Parameters:

sna_resource_name

The 1–17 character SNA resource name in the format: network.resource. ALL or a wildcard 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 RODM Programming Guide.

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
Resource Control Cards

**LU**

**DISPLAY_NAME**

Specifies the RODM DisplayResourceName for the object. This value is displayed on the NMC 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 Card Parameters" on page 598 for a description of the other supported keywords.

**SNA_PORT Control Card:**

**Description:** The SNA_PORT control card specifies the SNA resource to be processed.

**Syntax:**

```
SNA_PORT
```

```
>>> SNA_PORT = name
"","",ALL=ALL
"","",AGGPRI=aggregation_priority

","",CONSOLE=command
"","",USER_DATA=user_data

","",CORRELATER=text

","",DISPLAY_STATUS=status_integer
"",UNLINK=UNLINK
"",ROW=row_on_view

","",COLUMN=column_on_view
"",MARK=OFF
"",MARK=ON
```
Resource Control Cards

Parameters:

name
The SNA Port resource name in the format: snaNetID.portId. ALL or a wildcard name can be specified.

SNALOCALTOPO Control Card:

Description: The SNALOCALTOPO control card specifies the APPN SNA Local Topology resource to be processed.

Syntax:

SNALOCALTOPO

Parameters:

name
Is the APPN SNA Local Topology resource name in the format: snaNetID.snaNodeName. ALL or a wildcard name can be specified.
**Resource Control Cards**

See [“Common Control Card Parameters” on page 598](#) for a description of the other supported keywords.

**SYSTEM Control Card:**

*Description:* The SYSTEM control card specifies the workstation System aggregate resource to be processed.

*Syntax:*

```plaintext
SYSTEM
```

```plaintext
SYSTEM= name
,AGGTHRESH=(xxx,yyy,zzz)

,ACTivate=activate_command
,DEACTivate=deactivate_command

,RECYcle=recycle_command
,DISPLAY=display_command

,CONSOLE=command
,USER_DATA=user_data
,USER_DATA=text

,DISPLAY_STATUS=status_integer
,UNLINK
,ROW=row_on_view

,COLUMN=column_on_view
,MARK= OFF

,AUTO_IN_PROGRESS= OFF
,SUSPEND= OFF

,SUSPEND_WITH_AUTO_CLEAR= OFF
```

*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 wildcard name can be specified.
TG Control Card:

Description: The TG control card specifies the APPN Transmission Group resource to be processed.

Syntax:

\[
\text{TG} = \begin{array}{c}
\text{name} \\
\text{ALL} \\
\text{AGGPR}=\text{aggregation\_priority} \\
\text{CONSOLE}=\text{command} \\
\text{USER\_DATA}=\text{user\_data} \\
\text{CORRELATER}=\text{text} \\
\text{DISPLAY\_STATUS}=\text{status\_integer} \\
\text{UNLINK} \\
\text{ROW}=\text{row\_on\_view} \\
\text{COLUMN}=\text{column\_on\_view} \\
\text{MARK}=\text{OFF} \\
\text{AUTO\_IN\_PROGRESS}=\text{OFF} \\
\text{SUSPEND}=\text{OFF} \\
\text{SUSPEND\_WITH\_AUTO\_CLEAR}=\text{OFF} \\
\end{array}
\]

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 the NMC for the resource (DisplayResourceName). ALL or a wildcard name can be specified.

See "Common Control Card Parameters" on page 598 for a description of the other supported keywords.

TME_MONITOR Control Card:

Description: The TME_MONITOR control card specifies the MultiSystem Manager TME Monitor resource to be processed.

Syntax:
Resource Control Cards

**TME_MONITOR**

```
TME_MONITOR= name 
,SPname= ALL 

,MANNODEname= ALL 
,MANNODEname= managed_node_name 

,AGGTHRESH= (xxx, yyy, zzz) 

,ACTivate= activate_command 
,DEACTivate= deactivate_command 

,RECYcle= recycle_command 
(DISPLAY= display_command) 

,CONSOLE= command 
,USER_DATA= user_data 

,DISPLAY_STATUS= status_integer 

,COLUMN= column_on_view 

,MARK= OFF 

,AUTO_IN_PROGRESS= OFF

,SUSPEND= OFF 

,SUSPEND_WITH_AUTO_CLEAR= OFF 
```

**Parameters:**

**name**

The TME Monitor resource name.

ALL or a wildcard name can be specified.

**managed_node_name**

The name defined to the TMR as a managed node.

ALL or a wildcard name can be specified.

See "Common Control Card Parameters" on page 598 for a description of the other supported keywords.

**TME_POLICYREGION Control Card:**
**Resource Control Cards**

*Description:* The TME_POLICYREGION control card specifies the MultiSystem Manager TME Policy Region resource to be processed.

*Syntax:*

```
TME_POLICYREGION
```

Parameters:

**name**

The TME Policy Region name.

ALL or a wildcard name can be specified.

See "Common Control Card Parameters" on page 598 for a description of the other supported keywords.

**TME_TMR Control Card:**

*Description:* The TME_TMR control card specifies the MultiSystem Manager TME Managed Region resource to be processed.
Resource Control Cards

Syntax:

TME_TMR

\[
\text{TME_TMR}= \begin{cases} 
\text{name} = \text{ALL}, \text{SPname} = \text{ALL} \\
, \text{AGGTHRESH} = (\text{xxx}, \text{yyy}, \text{zzz}) \\
, \text{ACTivate} = \text{activate\_command} \\
, \text{DEACTivate} = \text{deactivate\_command} \\
, \text{RECYcle} = \text{recycle\_command} \\
, \text{DISPLAY} = \text{display\_command} \\
, \text{CONSOLE} = \text{command} \\
, \text{CORRELATER} = \text{text} \\
, \text{DISPLAY\_STATUS} = \text{status\_integer} \\
, \text{UNLINK} \\
, \text{ROW} = \text{row\_on\_view} \\
, \text{COLUMN} = \text{column\_on\_view} \\
, \text{MARK} = \text{OFF} \\
, \text{AUTO\_IN\_PROGRESS} = \text{OFF} \\
, \text{SUSPEND} = \text{OFF} \\
, \text{SUSPEND\_WITH\_AUTO\_CLEAR} = \text{OFF} \\
\end{cases}
\]

Parameters:

name

The TME Managed Region name.

ALL or a wildcard name can be specified.

See [Common Control Card Parameters on page 593] for a description of the other supported keywords.

VRN Control Card:

Description: The VRN control card specifies the APPN Virtual Routing Node resource to be processed.

Syntax:
Parameters:

**name**

The 1–17 character SNA Virtual Routing Node resource name in the format: snaNetID.snaNodeName. ALL or a wildcard name can be specified.

See "Common Control Card Parameters" on page 598 for a description of the other supported keywords.

**Aggregation Control Cards**

The following control cards specify the aggregate resources to be created or updated and the resources that compose the aggregate resource.

**AGGregate Control Card:**

*Description:* The AGGregate control card specifies the Aggregate (GMFHS Aggregate) resource to be processed.

*Syntax:*

AGGregate

\[ AGG\text{regate} = \text{aggregate\_name} \]

\[ , \text{TYPE} = \text{aggregate\_resource\_type} \]

\[ , \text{AGGTHRESH} = (\text{xxx,yyy,zzz}) \]

\[ , \text{ACTivate} = \text{activate\_command} \]
Aggregation Control Cards

Parameters:

aggregate_name
The aggregate resource name.

ALL or a wildcard 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
NTF_CLUSTER           DUIXC_RTN_GROUP_AGG
NTF_NETWORK           DUIXC_RTN_NETWORK_AGG
NTF_GROUPAGG          DUIXC_RTN_NTF_DOM_AGG
NTF_SECMON            DUIXC_RTN_APPL
NTF_PROCESS           DUIXC_RTN_PROGRAM
NTF_MONITOR           DUIXC_RTN_MONITOR
NTF_WINDOWS           DUIXC_RTN_WIN_SYSTEM
NTF_WINNT             DUIXC_RTN_WINNT_SYSTEM
NTF_OS2               DUIXC_RTN_OS2_SYSTEM
NTF_WIN95             DUIXC_RTN_WIN_95_SYSTEM
Aggregation Control Cards

<table>
<thead>
<tr>
<th>Resource</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTF_NETWARE</td>
<td>DUIXC_RTN_NETWARE_SYSTEM</td>
</tr>
<tr>
<td>TME_TMR</td>
<td>DUIXC_RTN_MANAGED_REGION_AGG</td>
</tr>
<tr>
<td>TME_POLICYREGION</td>
<td>DUIXC_RTN_POLICY_REGION_AGG</td>
</tr>
<tr>
<td>TME_MONITOR</td>
<td>DUIXC_RTN_MONITOR</td>
</tr>
<tr>
<td>SEGMENT</td>
<td>DUIXC_RTN_TR_SEGMENT_AGG</td>
</tr>
<tr>
<td>BRIDGE</td>
<td>DUIXC_RTN_BRIDGE_AGG</td>
</tr>
<tr>
<td>CAU</td>
<td>DUIXC_RTN_LAN_CONCENT_AGG</td>
</tr>
<tr>
<td>IP_CLUSTERS</td>
<td>DUIXC_RTN_INTERNET_CLUSTER</td>
</tr>
<tr>
<td>IP_NETWORK</td>
<td>DUIXC_RTN_INTERNET_MGMT_DOMAIN_AGG</td>
</tr>
<tr>
<td>IP_SUBNET</td>
<td>DUIXC_RTN_INTERNET_SUBNET_AGG</td>
</tr>
<tr>
<td>IP_SEGMENT</td>
<td>DUIXC_RTN_INTERNET_SEGMENT_AGG</td>
</tr>
<tr>
<td>IP_LOCATION</td>
<td>DUIXC_RTN_INTERNET_LOCATION_AGG</td>
</tr>
<tr>
<td>IP_ROUTER</td>
<td>DUIXC_RTN_INTERNET_ROUTER_AGG</td>
</tr>
<tr>
<td>IP_HUB</td>
<td>DUIXC_RTN_INTERNET_HUB_AGG</td>
</tr>
<tr>
<td>IP_BRIDGE</td>
<td>DUIXC_RTN_INTERNET_BRIDGE_AGG</td>
</tr>
<tr>
<td>IP_HOST</td>
<td>DUIXC_RTN_INTERNET_HOST_AGG</td>
</tr>
<tr>
<td>IP_LINK</td>
<td>DUIXC_RTN_LTN_IP_LINK_AGG</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>DUIXC_RTN_OPEN_SYSTEM_AGG</td>
</tr>
<tr>
<td>APPN_DOMAIN</td>
<td>DUIXC_RTN_NN_DOMAIN_AGG</td>
</tr>
<tr>
<td>APPN_NETWORK</td>
<td>DUIXC_RTN_NN_DOMAIN_NETWORK</td>
</tr>
<tr>
<td>APPN_CLUSTER</td>
<td>DUIXC_RTN_NN_DOM_NET_CLUSTER</td>
</tr>
<tr>
<td>SNALOCALTOPO</td>
<td>DUIXC_RTN_NN_LOCAL_TOP_AGG</td>
</tr>
<tr>
<td>USER</td>
<td>DUIXC_RTN_NODE_AGG_USER1</td>
</tr>
</tbody>
</table>

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 Card Parameters” on page 598 for a description of the other supported keywords.

Note: The AGGregate control card creates new aggregates or references existing aggregates which belong to the GMFHS_Aggregate_Objects_Class class.

If any AGGChild control cards follow the AGGregate control card, the resources specified on the AGGChild control cards are linked to the aggregate specified on the AGGregate control card, unless the AGGCHILD control cards specify UNLINK=YES.

AGGChild Control Card:

Description: The AGGChild control card specifies the aggregation children resource that you want linked or unlinked to the aggregate resource on the AGGregate statement that precedes the AGGChild control cards.

Syntax:
Parameters:

**name**
The name of the resource. The name formats and lengths depend upon the type of resource.
ALL or a wildcard name can be specified.

**TYPE**
Specifies the type of resource. The types correspond to the specific resource control cards.
- LAN_CLUSTER
- LAN_NETWORK
- LAN_AGENT
- NTF_CLUSTER
- NTF_NETWORK
- NTF_AGENT
- TME_CLUSTER
- TME_NETWORK
- TME_AGENT
- BRIDGE
- BRIDGE_AGG
- SEGMENT
- SEGMENT_AGG
- CAU
- CAU_AGG
- ADAPTER | ADP
- STATION_ADAPTER
- BRIDGE_ADAPTER
• CAU_ADAPTER
• LAN_ADAPTER
• NTF_GROUPAGG
• NTF_SECMON
• NTF_PROCESS
• NTF_MONITOR
• NTF_OS2
• NTF_WINNT
• NTF_WIN95
• NTF_WIN
• NTF_NETWARE
• NTF_ALL
• TME_TMR
• TME_POLICYREGION
• TME_MONITOR
• IP_CLUSTER
• IP_NETWORK
• IP_AGENT
• IP_SUBNET
• IP_LOCATION
• IP_SEGMENT
• IP_ROUTER
• IP_HUB
• IP_BRIDGE
• IP_HOST
• IP_LINK
• INTERFACE
• SYSTEM
• NONSNA
• APPN_CLUSTER
• APPN_NETWORK
• SNALOCALTOPO
• NNODE
• ENODE
• LNODE
• LINE
• SNA_PORT
• LAN_PORT
• DOMAIN
• LLINK
• TG
• APPN_VRN
• APPN_TG_CIRCUIT
• INTER_DOMAIN_CIRCUIT
• INTER_SUBNETWORK_CIRCUIT
Aggregation Control Cards

- 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
(INTEGER) 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 card.

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 Card Parameters" on page 598 for a description of the other supported keywords.

Running BLDVIEWS
Code the BLDVIEWS control cards which direct BLDVIEWS to build the views and aggregates you specify. The control cards 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 Cards in a NetView DSIPARM Member
If the control cards 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 cards.

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 cards. No actions are performed. RODM does not need to be active. The default is TEST=NO.

ECHO=YES
Results in BLDVIEWS displaying the control cards 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 Cards in a fully Qualified Dataset

If the control cards are coded in a cataloged data set then the syntax is:

```plaintext
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 cards. 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 cards. No actions are performed. RODM does not need to be active. The default is TEST=NO.

ECHO=YES

Results in BLDVIEWS displaying the control cards 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 will save 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:

```plaintext
BLDVIEWS ESP.NV24.BLDVIEWS(MYVIEWS)
BLDVIEWS ESP.NV24.BLDVIEWS.DATA1
```

Coding Control Cards in REXX Stem Arrays

If the control cards are coded in a REXX stem array, the syntax is:

```plaintext
'PIPE STEM stem_array | COLLECT | NETV BLDVIEWS', '({RODM=rodmname})', '({TEST=YES|NO})',
```

Examples:

```plaintext
BLDVIEWS ESP.NV24.BLDVIEWS(MYVIEWS)
BLDVIEWS ESP.NV24.BLDVIEWS.DATA1
```
BLDVIEWS Command Syntax

'{ECHO=YES|NO}',
'{QUIET=YES|NO}',
'{OPTIMIZE=CPU|STORage} | ....'

*stem_array*

The name of the REXX stem array variable that contains the BLDVIEWS control cards. *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 cards. No actions are performed. RODM does not need to be active. The default is TEST=NO.

**ECHO=YES**

Results in BLDVIEWS displaying the control cards 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
/* REXX */

card.1="VIEW='My View',ANNOTATION='This is my View',",
card.2='CREATE=YES'
card.3='NONSNA='resource',CREATE=YES,'
 || 'TYPE=DUIXC_RTN_HOST'

card.0=3

'PIPE STEM card. | COLLECT | NETV FLCVBLDV | CONSOLE'
exit
```

**BLDVIEWS Control Card Examples**

This section contains examples of coding BLDVIEWS control cards. Use the descriptions in Table 240 on page 678 to determine which example best matches
### BLDVIEWS Control Card Examples

Table 240. BLDVIEWS Control Card Examples

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
<th>Page Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Change the aggregation thresholds for MultiSystem Manager objects.</td>
<td>678</td>
</tr>
<tr>
<td>2</td>
<td>Set the generic commands in RODM for MultiSystem Manager objects.</td>
<td>678</td>
</tr>
<tr>
<td>3</td>
<td>Set the generic commands in RODM for MultiSystem Manager objects, the DisplayStatusCommandText to do an rping, and the DisplayResourceUserData to do a TELNETPM.</td>
<td>679</td>
</tr>
<tr>
<td>4</td>
<td>Set the DisplayResourceName for a Non-SNA resource.</td>
<td>679</td>
</tr>
<tr>
<td>5</td>
<td>Create a view that contains all bridge aggregate resources managed by a service point.</td>
<td>679</td>
</tr>
<tr>
<td>6</td>
<td>Create a view that contains specific bridge and segment resources managed by service point A19SRVCP.</td>
<td>679</td>
</tr>
<tr>
<td>7</td>
<td>Create a view that contains two new aggregate objects.</td>
<td>679</td>
</tr>
<tr>
<td>8</td>
<td>Create a view that contains multiple real, shadow, and aggregate resources.</td>
<td>679</td>
</tr>
<tr>
<td>9</td>
<td>Create a view with a layout type of 6 (hierarchical) and resources on the view on specific rows.</td>
<td>681</td>
</tr>
<tr>
<td>10</td>
<td>Unlink a bridge resource from a view.</td>
<td>682</td>
</tr>
<tr>
<td>11</td>
<td>Create a view that contains an aggregate object with OS/2 operating system resources.</td>
<td>682</td>
</tr>
<tr>
<td>12</td>
<td>Create a view that contains an aggregate resource with Windows NT operating system resources.</td>
<td>682</td>
</tr>
<tr>
<td>13</td>
<td>Create a view that contains all NetFinity group aggregate resources that begin with msmmr.</td>
<td>682</td>
</tr>
<tr>
<td>14</td>
<td>Set the DisplayResourceUserData field (Remote Console) to do a NETFIN command for all NetFinity operating system resources.</td>
<td>682</td>
</tr>
<tr>
<td>15</td>
<td>Create a view that contains all TME managed region aggregate resources.</td>
<td>682</td>
</tr>
<tr>
<td>16</td>
<td>Create a view that contains all TME policy region resources that begin with RTP.</td>
<td>682</td>
</tr>
</tbody>
</table>

**BLDVIEWS Example 1:**

This example changes the aggregation thresholds for all the MultiSystem Manager cluster and network aggregates for LNM and TCP/IP resources. The aggregation thresholds are changed to 25%, 50% and 75%.

```
NETWORK=ALL,AGGTHRESH=(25%,50%,75%),TYPE=LAN
CLUSTER=ALL,AGGTHRESH=(25%,50%,75%),TYPE=LAN
```

**BLDVIEWS Example 2:**

The example sets the generic commands in RODM for the MultiSystem Manager adapters, bridges, and controlled access units. Note that for MultiSystem Manager LNM resources, BLDVIEWS appends the commands with an operator and correlator.

```
ADP=ALL,
    DISPLAY='ADP QUERY ADP=%NAME% SEG=%SEGMENT%',
    DEACTIVATE='ADP REMOVE ADP=%NAME% SEG=%SEGMENT%'
BRIDGE=ALL,TYPE=REAL,
    DISPLAY='BRG QUERY NAME=%NAME%',
    ACTIVATE='BRG LINKNAME=%NAME%'
```
BLDVIEWS Example 3:
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 the remote console support will 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 4:
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 5:
This example creates a view that contains all bridge aggregate resources managed by service point A19SRVCP.

```
VIEW=GAF_ALLBridgesA,ANNOTATION='All Bridge Aggregates'
LANSNAME=A19SRVCP
BRIDGE=ALL,TYPE=AGG
```

BLDVIEWS Example 6:
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'
LANSNAME=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
```
BLDVIEWs Control Card Examples

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 7:
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 8:
This example creates a view that contains the following resources:
- Aggregate resource created previously by BLDVIEWs
- Non-sna resource
- MultiSystem Manager LNM bridge aggregate resource
- MultiSystem Manager LNM segment aggregate resource
- MultiSystem Manager LNM station adapter resource
- MultiSystem Manager TCP/IP Subnetwork resource
- MultiSystem Manager TCP/IP Location resource
- MultiSystem Manager TCP/IP Segment resource
- MultiSystem Manager TCP/IP Router resource
- MultiSystem Manager TCP/IP Hub resource
- MultiSystem Manager TCP/IP Bridge resource
- MultiSystem Manager TCP/IP Host resource
- MultiSystem Manager TCP/IP Adapter resource
BLDVIEWs Control Card Examples

- Create a NetWare server aggregate resource containing all NetWare Servers managed by service point A19NWAPU. Set the aggregation thresholds for the aggregate to 30%, 50%, and 75%.
- Create a NetWare server aggregate resource containing three NetWare Servers managed by service point A19NWAPU and set the aggregation thresholds for the aggregate to 30%, 50%, and 75%.

```
VIEW=GAF_Specific,ANNOTATION='Specific Resources'
LANSPNAME=A19SRVCP
NWSPNAME=A19NWAPU
IPSPNAME=A19RSCCP.NV6000
AGG=GAF_MS_Bridges,CREATE=NO
NONSNA=NV6000.CODEBUST.BUDDY
BRIDGE=A059C06,TYPE=AGG
SEGMENT=0C16,TYPE=AGG
CAU=5A982D60,TYPE=AGG
ADP=GARY
NWSERVER=ESP_A86A,TYPE=IBM_AGENT
IP_SUBNET=9.67.96
IP_LOCATION=raleigh
IP_SEGMENT=9.67.96.Segment1
IP_ROUTER=mercury.raleigh.ibm.com
IP_BRIDGE=austin.raleigh.ibm.com
IP_HUB=boca.raleigh.ibm.com
IP_HOST=forghetti.raleigh.ibm.com
INTERFACE=9.67.96.163
AGG=ALL_NW_SERVER_AGG,type=nwserver,aggthresh=(30%,50%,75%),create=yes
AGGCHILD=ALL,TYPE=NWSERVER_AGG
AGG=My_NW_SERVER_AGG,type=nwserver,aggthresh=(30%,50%,75%),create=yes
AGGCHILD=SYDNEY,TYPE=NWSERVER_AGG
AGGCHILD=NEPTUNE,TYPE=NWSERVER_AGG
AGGCHILD=H52_NW,TYPE=NWSERVER_AGG
```

BLDVIEWs Example 9:
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=NV6000.CODEBUST.BUDDY,ROW=1
BRIDGE=A059C06,TYPE=AGG,ROW=2
SEGMENT=0C16,TYPE=AGG,ROW=3
```
BLDVIEWS Control Card Examples

CAU=5A982D60,TYPE=AGG,ROW=4
ADP=GARY,ROW=5
NWSERVER=ESP_A86A,TYPE=IBM_AGENT,ROW=5

BLDVIEWS Example 10:
This example unlinks a bridge resource from a view.
VIEW=GAF_BLDG_500,CREATE=NO
LANSPNAME=A19SRVCP
BRIDGE=A085C17,TYPE=AGG,UNLINK

BLDVIEWS Example 11:
This example creates a view that contains an aggregate resource with OS/2
operating system resources.
VIEW=NETFINITY_OS2_SYSTEMS,ANNOTATION='OS2 VIEW',
CREATE=YES,LAYOUT=9
AGG=OS2_SYSTEMS,CREATE=YES,TYPE=NTF_NETWORK,
DISPLAYNAME='OS2_SYSTEMS'
AGGC=ALL,TYPE=NTF_OS2

BLDVIEWS Example 12:
This example creates a view that contains an aggregate resource with Windows NT
operating system resources.
VIEW=NETFINITY_WINNT_SYSTEMS,ANNOTATION='WINNT VIEW',
CREATE=YES,LAYOUT=9
AGG=WINNT_SYSTEMS,CREATE=YES,TYPE=NTF_NETWORK,
DISPLAYNAME='WINNT_SYSTEMS'
AGGC=ALL,TYPE=NTF_WINNT

BLDVIEWS Example 13:
This example creates a view that contains all NetFinity group aggregate resources
that begin with msmmgr in the view.
VIEW=NETFINITY_MSMGR_GROUP,ANNOTATION='GROUP VIEW',
CREATE=YES,LAYOUT=9
NTFGROUPAGG=msmmgr*

BLDVIEWS Example 14:
This example sets the DisplayResourceUserData field (remote console) to do a
NETFIN command for all of the NetFinity operating system resources.
NTFOS=ALL,TYPE=NTF_ALL,
CONSOLE='NETFIN /N:%CMOREADDRESS% /L:%RANDOM% /R:%RANDOM%'

BLDVIEWS Example 15:
This example creates a view that contains all port resources managed by service
point ESPA86CP.
VIEW=ATM_PORTS,ANNOTATION='ATM PORT VIEW',CREATE=YES,
LAYOUT=9
ATMSPNAME=ESPA86CP
ATM_PORT=ALL,ATMCLUSTER=ALL

BLDVIEWS Example 16:
This example creates a view that contains all TME managed region aggregate
resources.
VIEW=TME_MANAGED_REGIONS,ANNOTATION='MANAGED REGION VIEW',
CREATE=YES,LAYOUT=9
TME_TMR=ALL
BLDVIWES Example 17:
This example creates a view that contains all TME policy region resources that begin with RTP.

```
VIEW=TME_POLICY_REGION_RTP,ANNOTATION='POLICY REGION VIEW',
CREATE=YES,LAYOUT=9
TME_POLICYREGION=RTP*
```

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 and terminate it 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 *Tivoli NetView for z/OS Data Model Reference* for more information.
Examples of Deleting Views
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 167 on page 686)
- Radial layout for user-defined clusters by cluster ID (see Figure 167 on page 686)
- Radial layout for broad-band networks (see Figure 167 on page 686)
- Radial layout for token-ring networks (see Figure 168 on page 686)
- Radial layout for local area networks (see Figure 169 on page 687)
- Radial layout for local area networks with a central bus (see Figure 170 on page 687)
- Elliptical layout with a single ellipse (see Figure 171 on page 688)
- Hierarchical layout (see Figure 172 on page 688)
- Connectivity tree layout (see Figure 173 on page 689)
- Grid layout for exception, configuration, and network views (see Figure 174 on page 689)

For a list of the advantages and disadvantages of each layout type, see Table 241 on page 690.
Figure 167. Radial Layout Example

Figure 168. Token-Ring Layout Example
Figure 169. LAN Net Layout Example

Figure 170. LAN Bus Layout Example
Figure 171. Ellipse Layout Example

Figure 172. Hierarchical Graph Layout Example
Figure 173. Connectivity Tree Layout Example

Figure 174. Grid Layout Example
Choosing a View Layout Type

[Table 241 describes some of the advantages and disadvantages for each layout type.]

**Table 241. Advantages and Disadvantages of View Layout Types**

<table>
<thead>
<tr>
<th>View Layout Type</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radial by link type</td>
<td>Efficiently uses presentation space on workstation.</td>
<td>The mental picture of the user might not correspond to the view layout.</td>
</tr>
<tr>
<td></td>
<td>Can effectively show groupings of resources at physical sites.</td>
<td>Does not convey parent-child relationships well.</td>
</tr>
<tr>
<td></td>
<td>Can lay out any view regardless of connectivity.</td>
<td></td>
</tr>
<tr>
<td>Radial by cluster ID</td>
<td>Same advantages as radial layout by link type.</td>
<td>Requires you to assign a cluster ID to each node in the view.</td>
</tr>
<tr>
<td></td>
<td>Gives you complete control of how nodes are grouped.</td>
<td></td>
</tr>
<tr>
<td>Single ellipse</td>
<td>Makes optimal use of the presentation space.</td>
<td>Can only represent a single site or grouping.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>You must set sequence numbers for link-crossing reduction.</td>
</tr>
<tr>
<td>LAN network layout</td>
<td>Well suited to laying out views containing a broad band LAN.</td>
<td>The view must meet connectivity requirements for a LAN view as defined by the view layout facility.</td>
</tr>
<tr>
<td>LAN token-ring layout</td>
<td>Well suited to laying out views containing a token-ring LAN.</td>
<td>The view must meet connectivity requirements for a token-ring view as defined by the view layout facility.</td>
</tr>
<tr>
<td>LAN bus layout</td>
<td>Well suited to laying out views containing a LAN with a central bus.</td>
<td>The view must meet connectivity requirements for a LAN bus view as defined by the view layout facility.</td>
</tr>
<tr>
<td>Connectivity tree layout</td>
<td>Quick layout.</td>
<td>The view must meet connectivity requirements for a connectivity tree view as defined by the view layout facility.</td>
</tr>
<tr>
<td></td>
<td>Shows the parent-child relationships among resources.</td>
<td></td>
</tr>
<tr>
<td>Hierarchical graph by node priority</td>
<td>Shows the parent-child relationships among network resources.</td>
<td>You must assign a hierarchical priority to each node in the view.</td>
</tr>
<tr>
<td></td>
<td>Can lay out any view regardless of connectivity.</td>
<td></td>
</tr>
<tr>
<td>Grid layout</td>
<td>Quick layout.</td>
<td>Does not display network topology unless you define the rows and columns.</td>
</tr>
<tr>
<td></td>
<td>Good for displaying lists of related or unrelated network objects.</td>
<td>Does not show connectivity.</td>
</tr>
</tbody>
</table>
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 NMC, 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.
Radial Layout View by Link Type

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 could 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 would result 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 should be 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.
Hierarchical Layout View

**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 should 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 would be to set the hierarchical priority according to the node object type, so that all nodes of a type would be 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 X 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

**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 should 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 should 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.
Grid Layout

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 would be placed would be one-tenth of the distance between the rows on which nodes 2 and 3 were 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 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 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
Grid Layout

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 242 lists examples of differently defined links and the results of each definition:

Table 242. Link Definitions and Results

<table>
<thead>
<tr>
<th>A link is defined with row and column layout parameters. No end points are defined for the link.</th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A link is defined without row and column layout parameters. No end points are defined for the link.</td>
<td>The link is drawn with two null nodes at random locations. To control the location of the node, specify coordinates on the link.</td>
</tr>
<tr>
<td>A link is defined with row and column layout parameters. Only 1 end point is defined with row and column layout parameters.</td>
<td>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.</td>
</tr>
<tr>
<td>A link is defined with row and column layout parameters. Only 1 end point is defined, but without row and column layout parameters.</td>
<td>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.</td>
</tr>
<tr>
<td>A link is defined with row and column layout parameters. Two end points are defined with row and column layout parameters specified for both.</td>
<td>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.</td>
</tr>
</tbody>
</table>
Index

Special Characters
%APPL% replacement parameter 63
%DOMAIN% replacement parameter 63
%RESOURCE% replacement parameter 63
%SPNAME% replacement parameter 63
%TYPE% replacement parameter 63

A
about this book xi
abstract data types 223, 225
access block
definition 8
description 307
RODM_name parameter 308
Sign_on_token parameter 308
User_appl_ID parameter 308
access functions 371
accessibility information xiii
accessing and changing GMFHS-defined fields 184
action functions 372
adding NMGs and domains, GMFHS 60
adding objects, GMFHS 57
administrative functions 371
aggregate, suspending aggregation 130
aggregate objects
defining 38
definition 25
aggregation
aggregation child description 132
aggregation hierarchy, building 134
aggregation hierarchy, creating 133
aggregation hierarchy
description 133
aggregation hierarchy loop
description 134
aggregation level 132
aggregation parent description 132
aggregation path description 132
aggregation priority 139
aggregation thresholds 138
DisplayStatus field, role in 136
DUIFCUAP method, role in 135
events that start process 141
introduction 132
parent status, calculating 138
problems 141
process overview 132
ResourceTraits field 132
rules 140
status, affects on aggregation 136
status group 144
status group customization 139
suspending resources from 137
updating status 135
aggregation limits 25
AggregationChild connectivity relationship 27
AggregationParent connectivity relationship 27
AGGRST parameter 498
alert processing, DUIFDEF 174
alert translation tables 178
alerts
down 82
GMFHS data model 169
INIT, DOMS010 protocol 80
monitoring 169
receiving 76
resolving 169
session termination 82
timing 76
ANONYMOUSVAR
load function data type 302
null value 262
API_version parameter, transaction information block 310
application programs
API query field control block
call 307
asynchronous error notification 328
calls, RODM 303
compiling programs 305
control block relationships 307
EKGUAPI module 304
error conditions, user API transactions 319
languages, RODM user applications 222
link-editing programs 305
object deletion notification 328
parameters, user API calls to RODM 306
program calls 303
programming reference 371
register conventions 304
RODM system (x/OS), illustration 223
user API calls to RODM 306
using control blocks 306
using user APIs 304
writing RODM application programs 303
APPLICATIONID
load function data type 302
null value 262
applying policy to views 123
ASSIST_CHANGEDATA data item 488
asterisk 284
asynchronous error
definition 202
notification 328
ATTRLIST high-level syntax
keyword 279
authorization
function calls 376
load function statements 254
automation
accessing and changing
GMFHS-defined fields 184
advantages 183
CNMSNIIFF sample application 188
EKGSNIIFF sample method 188
GMFHS 183
GMFHS example 187
notifying applications, field changes 183
overview 183
RODM 191
sample application 188
sample method 188
using GMFHS methods 185
writing automation code, data model 183
B
backbone 31
BackboneConnPP 28
BAS_FCP attribute 230
batch job, RODM load function 252
BERVAR
load function data type 302
null value 262
Bit_map function parameter 447
books
feedback xii
online xii
ordering xii
building views 91
C
c, definition 6
calling load function 253
calls
call statement format 358
program 303, 358
cause undetermined subvector, INIT alert 80
CE keyword, DOMP010 protocol 66
change method
change subfield 347
description 346
parameters 346
procedure interface 348
restrictions 366
Change_status function parameter 447
change subfield
definition 216
description 347
changes, non-SNA domain 23
changing
views 41
changing objects, GMFHS 57
char_literal data type 294
characters 197, 211, 212
characters (continued)
class name 197
field name 212
object name 211
characters, allowed 197
class name 197
field name 212
object name 211
characters, double-byte 232
chars data type 294
CHARVAR
load function data type 302
null value 262
CHARVADDR
load function data type 302
null value 262
checking output listings 255
checkpoint
coding checkpoint control 386
definition 5
process 384
TRANSPARENT_CHECKPOINT
keyword 386
ChildAccess connectivity relationship 27
class
names, in RODM 197
class, common syntactic element 294
Class_access_info_ptr parameter 447
Class_access_info_ptr function parameter 423
EKG_CreateClass function 388
EKG_CreateField function 389
EKG_CreateSubfield function 392
EKG_DeleteClass function 393
EKG_DeleteField function 394
EKG_DeleteSubfield function 398
Class_ID function parameter 448
Class_ID parameter
EKG_QueryNotifyQueue function 423
EKG_WhereAml function 446
entity access-information block 312
class_list, common syntactic element 295
class locking, RODM 222
Class_name function parameter 448
Class_name_length parameter, entity access-information block 312
Class_name parameter,
EKG_QueryObjectName function 425
Class_name_ptr parameter, entity access-information block 312
class structure definitions 247
classes 197
CLASSID
load function data type 302
using data type 261
CLASSIDLIST load function data type 302
classlink_list, common syntactic element 295
CLASSLINK load function data type 302
CLASSLINKLIST load function data type 302
CM keyword, DOMP010 protocol 67
CMD_CHARVAR data item 489
CMD_DESC_CHARVAR data item 489
CMNQAPI service routine,
description 191
CMNS4402 sample application 188
CMNSJH12, sample 57
CMNSNF sample application 188
CODEPAGE parameter 272
coding high-level load function statements 275
coding installation-written methods 362
coding primitive statements 284
cold start, definition 5
collection definition object 145
collection definition object fields 146
collection definition objects 146
collection definition objects, examples 157
collection specification, using 147
collection specification syntax 151
collection specification values 152
command, protocol
INIT_ACCEPT 69
INIT_ACCEPT_ACCEPT 69
SESSION_REQUEST 68
SESSION_REQUEST_ACCEPT 69
SET_CLOCK 69
SET_CLOCK_ACCEPT 69
command responses, timing 77
command session 77
command transport envelope, PPI 86
common operations services (COS) NMGs 85
common syntactic elements
class 294
class_list 295
classlink_list 295
field 296
method_spec 298
object 298
objectid_list 298
objectlink_list 299
recipient_spec 299
sd_parm 299
subfield 300
subspec 300
subs_spec_list 300
type 301
typed_value 302
communicating, NMGs 61
compiling application programs 305
compiling programs 363
complex conditional statements 148
CompositeOfLogical connectivity relationship 26
CompositeOfPhysical connectivity relationship 26
Concat_of_strings function parameter 448
Concat_of_strings parameter
EKG_TriggerNamedQuery function 441
EKG_TriggerOIMethod function 442
conditional statements 147
CONFIG DOMAIN command,
GMFHS 58
CONFIG NETWORK command,
GMFHS 58
CONFIG VIEW command, GMFHS 58
configuration, RODM 33
configuration view
description 31
types of 31
configuration views
defining backbone 43
logical 43
peer 43, 44
physical 43
connecting, RODM 330
connectivity relationships
defining 40
identifying 26
connectors
NSL_B202 32
NSL_ENET 32
OEMLAB 32
control blocks
access block 307
API query field control block, sample 307
entity access information block 311
field access information block 314
function block 311
relationships 307
response block 316
transaction information block 309
using 306
control functions 371
control table
modifying 262
sample 262
correlation
aggregate object names 337
customization
changing display name priority 341
disabling correlation for specific resources 342
enabling 333
extending for MultiSystem Manager and SNA topology manager objects 339
methods 334
object display labels 337
object field values 338
object relationships 337
objects enabled 335
types
free-form 335
network 335
using objects you created 339
Correlation_ID function parameter 448
Correlation_ID parameter,
EKG_QueryResponseBlockOverflow function 426
COS NMGs 85
COS transport protocol, definition 83
CP keyword, DOMP010 protocol 67
CREATE high-level statement 280
creating
class structure and object definitions 247
high-level load function statements 275
creating (continued) 284
RODM data models 241
creating a notification queue 322
crossing user APIs 304
Customer Support xiii
customizing fast path to failing resource views 97
customizing locate failing resource views 97

D
data definitions
initialization 268
object load 268
statements 266
structure load 268
Data function parameter 448
Data model, system class definitions 198
Data parameter
EKG_QueryField function 413
EKG_QueryResponseBlockOverflow function 427
EKG_QuerySubfield function 428
Data_to_be_returned function parameter 448
Data_to_be_returned parameter, EKG_QueryMultipleSubfields function 421
Data_Type function parameter 448
Data_type parameter
EKG_ChangeField function 380
EKG_ChangeMultipleFields function 381
EKG_ChangeSubfield function 383
EKG_CreateField function 389
EKG Locate function 406
EKG_QueryEntityStructure function 412
EKG_QueryField function 413
EKG_QueryFieldStructure function 417
EKG_QueryMultipleSubfields function 421
EKG_QuerySubfield function 428
EKG_SwapField function 437
EKG_SwapSubfield function 439
data types
abstract data types 225
fields 224
identifiers 224
null values 224
reserved data types 225
subfields 217
data types, values and 155
Data_value parameter, EKG_QueryMultipleSubfields function 421
Date/Time subvector, INIT alert 81
dbcs_literal data type 295
DD List Structure 270
deciding load type 248
deciding method type 356
defining
configuration, RODM 33
network elements 22
non-SNA domain 35
DELETE high-level statement 281
deleting notification queues 327
deleting objects, GMFHS 57
delimiters, load function statements 276
designing RODM data models 241
digits data type 296
disability information xiii
disconnecting, RODM 330
Display_Resource_Type_Class 92
DisplayStatus field
defining exception criteria 103
ExceptionViewFilter field 105
DisplayStatus method, creating
DUIFVCFCT method 113
sample method DUIFCUX2 114
sample method DUIFCUXM 112
USRXMETH keyword 110
DM keyword, DOMP010 protocol 68
DOMP010 presentation protocol 62
DOMP010 protocol
CE keyword 66
CM keyword 67
CP keyword 67
DM keyword 68
packet definition 65
packet format 65
PT keyword 68
RN keyword 69
RP keyword 70
SN keyword 70, 71
ST keyword 71
TM keyword 73
TX keyword 73
DOMP020 presentation protocol 63
DOMS010 protocol 78
double-byte characters 232, 296
down alert 82
DSINOR service routine, description 191
DUIFCAAP method 490
DUIFCADT method 490
DUIFCAPC method 490
DUIFCASB method 490
DUIFCATC method 490
DUIFCCAT method, description 185
DUIFCCAN method
description 491
DUIFCCAN method, description 185
DUIFCCAP method 490
DUIFCDTC method 490
DUIFCDUC method 490
DUIFCGGR2 method 490
DUIFCGGR3 method 490
DUIFCGRA method 490
DUIFCGRT method 490
DUIFCFLRT method
description 491
overview 185
DUIFCLLS method 490
DUIFCLLS3 method 490
DUIFCNSR method 490
DUIFCMU method 490
DUIFCRDC method 490
DUIFCRTP method 490
DUIFCRTU method 490
DUIFCRUC method 490
DUIFCSRT method 490
DUIFCUAP method
description 493
overview 185
DUIFCURA method 490
DUIFCUTC method 490
DUIFCUXS method
description 495
overview 186
DUIFECDS method
description 496
overview 186
DUIFECMV 169
DUIFEFED 173
DUIFEFED AlertProc 174
DUIFEDST, DUIFEBIM, and DUIFEUSR alert translation tables 178
DUIFEGSN method 490
DUIFFAWS method
description 498
overview 186
DUIFFRS method
description 498
overview 186
DUIFFRAS method
description 499
overview 186
DUIFFSUS method
description 499
overview 186
DUIFITKN method 490
DUIFRAIP method 490
DUIFREDS method
description 500
not triggering DisplayStatus recalculation 107
overview 187
DUIFRTC method 490
DUIFSMT
DUIFSMT statement syntax 106
DUIFSMT macro
keywords
CLASS 107
CLASS, alias values for 108
MYNAME 110
RESOURCE 110
USRXMETH 110
XCPT 108
sample table DUIFSMT 107
syntax for 107
DUIFVCFT method
description 500
overview 187
using 113
DUIFVCYT method 490
DUIFVDRT method 491
DUIFVEFC method 491
DUIFVEVF method 491
DUIFVEXV method 491
DUIFVFPM method 491
DUIFVGEM method 491
DUIFVIEW method 491
DUIFVINS method
description 501
overview 187
DUIFVLST method 491
DUIFVLT method 491
DUIFVMDR method 491
Index 701
EKG_LinkNoTrigger function 220, 404
EKG_LinkTrigger function 220, 404
EKG_Link function 406
EKG_Name field, EKG_System class 201
EKG_NotificationQueue class 206
EKG_NotificationQueue function 407
EKG_LogLevel field, EKG_User class 205
EKG_LogLevel function 407
EKG_LogLevel field, EKG_System class 201
EKG_Maximum_Q_Entries field, EKG_NotificationQueue class 208
EKG_MessageOnQueue field, EKG_NotificationQueue class 208
EKG_MessageTriggeredAction function 409
EKG_LastCheckpointResult field, EKG_LastCheckpointID field, EKG_LastAsyncError field, EKG_InstallerID field, EKG_Method EKG_ExternalLogState field, EKG_System EKG_ExecuteFunctionList function 402
EKG_ECBPostedStatus field,
EKG_ECBAddress field,
EKG_Disconnect function 401
EKG_Connect function 387
EKG_CreateClass function 388
EKG_CreateField function 389
EKG_CreateObject function 391
EKG_CreateSubfield function 392
EKG_DeleteClass function 393
EKG_DeleteField function 394
EKG_DeleteNotifySubscription function 395
EKG_DeleteObject function 397
EKG_DeleteSubfield function 398
EKG_DeleteObjectDelSubs function 399
EKG_Disconnect function 401
EKG_ECBAddress field,
EKG_NotificationQueue class 207
EKG_ECBPostedStatus field,
EKG_NotificationQueue class 207
EKG_ExecuteFunctionList function 402
EKG_ExternalLogState field, EKG_System class 201
EKG_InstallerID field, EKG_Method class 209
EKG_LastAsyncError field
EKG_System class 202
EKG_User class 205
EKG_LastCheckPointID field,
EKG_System class 202
EKG_LastCheckPointResult field,
EKG_System class 202
EKG_LastCheckpointResult field, EKG_System class 202
EKG_TransSegment field, EKG_System class 203
EKG_User class 203
EKG_User class 201
EKG_LastCheckpointResult field, EKG_LastCheckpointID field, EKG_LastAsyncError field, EKG_InstallerID field, EKG_Method EKG_ExternalLogState field, EKG_System EKG_ExecuteFunctionList function 402
EKG_ECBPostedStatus field,
EKG_ECBAddress field,
EKG_Disconnect function 401
EKG_Connect function 387
EKG_CreateClass function 388
EKG_CreateField function 389
EKG_CreateObject function 391
EKG_CreateSubfield function 392
EKG_DeleteClass function 393
EKG_DeleteField function 394
EKG_DeleteNotifySubscription function 395
EKG_DeleteObject function 397
EKG_DeleteSubfield function 398
EKG_DeleteObjectDelSubs function 399
EKG_Disconnect function 401
EKG_ECBAddress field,
EKG_NotificationQueue class 207
EKG_ECBPostedStatus field,
EKG_NotificationQueue class 207
EKG_ExecuteFunctionList function 402
EKG_ExternalLogState field, EKG_System class 201
EKG_InstallerID field, EKG_Method class 209
EKG_LastAsyncError field
EKG_System class 202
EKG_User class 205
EKG_LastCheckPointID field,
EKG_System class 202
EKG_LastCheckPointResult field,
EKG_System class 202
EKG_LinkNoTrigger function 220, 404
EKG_LinkTrigger function 220, 404
EKG_Link function 406
EKG_Name field, EKG_System class 201
EKG_NotificationQueue class 206
EKG_NotificationQueue function 407
EKG_LogLevel field, EKG_User class 205
EKG_LogLevel function 407
EKG_LogLevel field, EKG_System class 201
EKG_Maximum_Q_Entries field, EKG_NotificationQueue class 208
EKG_MessageOnQueue field, EKG_NotificationQueue class 208
EKG_MessageTriggeredAction function 409
EKG_LastCheckpointResult field, EKG_LastCheckpointID field, EKG_LastAsyncError field, EKG_InstallerID field, EKG_Method EKG_ExternalLogState field, EKG_System EKG_ExecuteFunctionList function 402
EKG_ECBPostedStatus field,
EKG_ECBAddress field,
EKG_Disconnect function 401
EKG_Connect function 387
EKG_CreateClass function 388
EKG_CreateField function 389
EKG_CreateObject function 391
EKG_CreateSubfield function 392
EKG_DeleteClass function 393
EKG_DeleteField function 394
EKG_DeleteNotifySubscription function 395
EKG_DeleteObject function 397
EKG_DeleteSubfield function 398
EKG_DeleteObjectDelSubs function 399
EKG_Disconnect function 401
EKG_ECBAddress field,
EKG_NotificationQueue class 207
EKG_ECBPostedStatus field,
EKG_NotificationQueue class 207
EKG_ExecuteFunctionList function 402
EKG_ExternalLogState field, EKG_System class 201
EKG_InstallerID field, EKG_Method class 209
EKG_LastAsyncError field
EKG_System class 202
EKG_User class 205
EKG_LastCheckPointID field,
EKG_System class 202
EKG_LastCheckPointResult field,
EKG_System class 202
EKG_LinkNoTrigger function 220, 404
EKG_LinkTrigger function 220, 404
EKG_Link function 406
EKG_Name field, EKG_System class 201
EKG_NotificationQueue class 206
EKG_NotificationQueue function 407
EKG_LogLevel field, EKG_User class 205
EKG_LogLevel function 407
EKG_LogLevel field, EKG_System class 201
EKG_Maximum_Q_Entries field, EKG_NotificationQueue class 208
EKG_MessageOnQueue field, EKG_NotificationQueue class 208
EKG_MessageTriggeredAction function 409
EKG_LastCheckpointResult field, EKG_LastCheckpointID field, EKG_LastAsyncError field, EKG_InstallerID field, EKG_Method EKG_ExternalLogState field, EKG_System EKG_ExecuteFunctionList function 402
EKG_ECBPostedStatus field,
EKG_ECBAddress field,
EKG_Disconnect function 401
EKG_Connect function 387
EKG_CreateClass function 388
EKG_CreateField function 389
EKG_CreateObject function 391
EKG_CreateSubfield function 392
EKG_DeleteClass function 393
EKG_DeleteField function 394
EKG_DeleteNotifySubscription function 395
EKG_DeleteObject function 397
EKG_DeleteSubfield function 398
EKG_DeleteObjectDelSubs function 399
EKG_Disconnect function 401
EKG_ECBAddress field,
element management system communicating session 77
Entity_access_info_ptr parameter 448
Entity_access_info_ptr_1 parameter EKG_LinkNoTrigger function 404
EKG_LinkTrigger function 404
EKG_UnlinkNoTrigger function 443
EKG_UnlinkTrigger function 443
Entity_access_info_ptr_2 parameter 448
Entity_access_info_ptr_2 function EKG_LinkNoTrigger function 404
EKG_LinkTrigger function 404
EKG_UnlinkNoTrigger function 444
EKG_UnlinkTrigger function 444
Entity_access_info_ptr_1 function parameter 448
Entity_access_info_ptr function parameter 448
Entity_access_info_ptr_2 parameter EKG_LinkNoTrigger function 404
EKG_LinkTrigger function 404
EKG_UnlinkNoTrigger function 444
EKG_UnlinkTrigger function 444
Entity_access_info_ptr function parameter 448
Entity_access_info_ptr parameter EKG_AddNotifySubscription function parameter 448
EKG_AddNotifySubscription function 377
EKG_AddObjDelSubs function 379
EKG_ChangeField function 380
EKG_ChangeMultipleFields function 381
EKG_ChangeSubfield function 382
EKG_CreateObject function 391
EKG_DeleteNotifySubscription function 396
EKG_DeleteObject function 397
EKG_DeleteObjDelSubs function 400
EKG_QueryEntityStructure function 411
EKG_QueryField function 413
EKG_QueryFieldName function 416
EKG_QueryFieldStructure function 417
EKG_QueryMultipleSubfields function 421
EKG_QuerySubfield function 428
EKG_RevertToInherited function 431
EKG_SwapField function 437
EKG_SwapSubfield function 439
EKG_TriggerNamedMethod function 441
entity access information block Class_ID parameter 312
Class_name_length parameter 312
Class_name_ptr parameter 312
definition 8
description 311
Naming_count parameter 312
Object_ID parameter 312
Object_name_length parameter 312
Object_name_ptr parameter 312
envelope, PPI command transport 86
errors asynchronous error notification 328
reporting error conditions 319
user API transactions 319
ESTAE routines, method restrictions 367
ESTAX routines, method restrictions 367
Ethernet network 20
example layout parameters, detailed views 51
layout parameters, objects 54
examples of collection definition objects 157
exception state affect on exception views 106
defining exception criteria 103
defining ExceptionViewFilter field 105
element, mapping DisplayStatus to 111
fast path to failing resource view 97
user methods 113
exception view customizing DisplayStatus mapping table DUIFSMT
CNMSFH13 106
DUIFSMT statements 106
examples of 111
syntax of DUIFSMT macro 107
Customizing DisplayStatus mapping table DUIFSMT
creating DisplayStatus method 112
defining 41, 102
defining candidates for 105
defining the ExceptionViewFilter field 105
description 28
DUIFSMTV 
example of 41
using 102
illustration 29
layout parameter used 46
mapping display status using sample table DUIFSMT 103
object connectivity process 102
object discovery process 101
open view, creating objects for 105
open view, deleting objects from 105
sample DUIFSMT 102
user method, not triggered 113
using table DUIFSMT 106
exception views implementing 114
ExceptionViewFilter field
defining 105
defining exception criteria 103
DisplayStatus filter 106
role in exception view object discovery process 102
UserStatus filter 106
ExceptionViewList field in sample DUIFSMT 103
role in exception view object discovery process 102
ExceptionViewName field
role in exception view object discovery process 101
execute command major vector 86
fast path to failing resource views, customizing 97
feedback about publications xiii
field identifiers, RODM 213
names, RODM 212
field, common syntactic element 296
Field_access_info_ptr_1 function parameter 448
Field_access_info_ptr_1 parameter EKG_LinkNoTrigger function 404
EKG_LinkTrigger function 404
EKG_UnlinkNoTrigger function 443
EKG_UnlinkTrigger function 443
Field_access_info_ptr_2 function parameter 448
Field_access_info_ptr_2 parameter EKG_LinkNoTrigger function 404
EKG_LinkTrigger function 404
EKG_UnlinkNoTrigger function 444
EKG_UnlinkTrigger function 444
Field_access_info_ptr function parameter 448
Field_access_info_ptr parameter EKG_AddNotifySubscription function 377
EKG_ChangeField function 380
EKG_ChangeMultipleFields function 381
EKG_ChangeSubfield function 382
EKG_CreateObject function 391
EKG_DeleteNotifySubscription function 396
EKG_DeleteObject function 397
EKG_DeleteObjDelSubs function 400
EKG_QueryEntityStructure function 411
EKG_QueryField function 413
EKG_QueryFieldName function 416
EKG_QueryFieldStructure function 417
EKG_QueryMultipleSubfields function 421
EKG_QuerySubfield function 428
EKG_RevertToInherited function 431
EKG_SwapField function 437
EKG_SwapSubfield function 439
EKG_TriggerNamedMethod function 441
field access information block
definition 8
description 314
Field_ID parameter 315
Field_name_length parameter 315
Field_name_ptr parameter 315
Naming_count parameter 315
Naming_count parameter 312
Object_ID parameter 312
Object_name_length parameter 312
Object_name_ptr parameter 312
envelope, PPI command transport 86
errors asynchronous error notification 328
reporting error conditions 319
user API transactions 319
ESTAE routines, method restrictions 367
ESTAX routines, method restrictions 367
Ethernet network 20
example layout parameters, detailed views 51
layout parameters, objects 54

function_block_ptr function parameter 448
Field_info_count function parameter 448
Field_info_count parameter, EKG_QueryEntityStructure function 411
Field_info_element_size function parameter 448
Field_info_element_size parameter, EKG_QueryEntityStructure function 411
Function_block_ptr parameter, field access information block 315
Function_block_origin parameter, Function_block_origin function 449
Function_block_copy parameter, Function_block_copy function 449
function block freeing methods 360
Function_ID parameter (continued)
EKG_ChangeSubfield function 382
EKG_Checkpoint function 384
EKG_Connect function 387
EKG_CreateClass function 388
EKG_CreateField function 389
EKG_CreateObject function 391
EKG_CreateSubfield function 392
EKG_DeleteClass function 393
EKG_DeleteField function 394
EKG_DeleteNotifySubscription function 396
EKG_DeleteObject function 397
EKG_DeleteSubfield function 398
EKG_DeleteObjDeSubs function 400
EKG_Disconnect function 401
EKG_ExecuteFunctionList function 402
EKG_LinkNoTrigger function 404
EKG_LinkTrigger function 404
EKG_Locate function 406
EKG_LockObjectList function 408
EKG_MessageTriggeredAction function 409
EKG_OutputToLog function 410
EKG_QueryEntityStructure function 411
EKG_QueryField function 413
EKG_QueryFieldID function 414
EKG_QueryFieldName function 416
EKG_QueryFieldStructure function 417
EKG_QueryFunctionBlockContents function 419
EKG_QueryMultipleSubfields function 421
EKG_QueryNotifyQueue function 423
EKG_QueryObjectName function 425
EKG_QueryResponseBlockOverflow function 426
EKG_QuerySubfield function 428
EKG_ResponseBlock function 430
EKG_RevertToInherited function 431
EKG_SendNotification function 433
EKG_SetReturnCode function 434
EKG_Stop function 436
EKG_SwapField function 437
EKG_SwapSubfield function 439
EKG_TriggerNamedMethod function 441
EKG_TriggerOMethod function 442
EKG_UnlinkNoTrigger function 443
EKG_UnitLinkTrigger function 443
EKG_UnclockAll function 445
EKG_WhereAml function 446
Function_info_array function parameter 449
Function_info_array parameter, EKG_ExecuteFunctionList function 402
function parameters (continued)
Correlation_ID 448
Data 448
Data_to_be_returned 448
Data_type 448
Entity_access_info_ptr 448
Entity_access_info_ptr_1 448
Entity_access_info_ptr_2 448
Field_access_info_ptr 448
Field_access_info_ptr_1 448
Field_access_info_ptr_2 448
Field_ID 448
Field_info_array 448
Field_info_count 448
Field_info_element_size 448
Field_name 449
Field_type_flag 449
Function_block_copy 449
Function_block_origin 449
Function_block_ptr 449
Function_ID 449
Function_info_array 449
Indexed_data_length 449
Indexed_data_ptr 449
Inheritance_state 449
Last_checkpoint_ID 449
Local_copy_map 449
Local_inherited_flag 450
Log_message 450
Long_lived_parm 450
Message_CCSID 450
Method_name 450
Method_output_message 450
Method_parms 450
New_char_data_length 450
New_data_ptr 450
Notification_queue 450
Notification_queue_count 450
Notify_method 451
Number_of_fields 451
Number_of_functions 451
Number_of_subfields 451
Object_array 451
Object_ID 451
Object_list_length 451
Object_name 451
Old_char_data_length 451
Old_data_ptr 451
Parent_access_info_ptr 451
Private_public_flag 451
Reason_code 451
Requesting_method_ID 451
Response_block_length 451
Response_block_reference 451
Response_block_type 452
Response_block_used 452
Response_data 452
Return_code 452
Stop_EC 452
Stop_type 452
Subfield 452
Subfield_map 453
Subscription_info 453
User_appl_ID 453
User_area 454
User_password 453
User_word 454

704  Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer’s Guide
functions
  access functions 371
  action functions 372
  administrative functions 371
  control functions 371
  EKG_AddNotifySubscription function 377
  EKG_AddObjDelSubs function 378
  EKG_ChangeField function 380
  EKG_ChangeMultipleFields function 381
  EKG_ChangeSubfield function 382
  EKG_Checkpoint function 384
  EKG_Connect function 387
  EKG_CreateClass function 388
  EKG_CreateField function 389
  EKG_CreateObject function 391
  EKG_CreateSubfield function 392
  EKG_DeleteClass function 393
  EKG_DeleteDeleteSubfield function 394
  EKG_DeleteObject function 397
  EKG_DeleteSubfield function 398
  EKG_DeleteObjectList function 399
  EKG_Disconnect function 401
  EKG_ExecuteFunctionList function 402
  EKG_LinkNoTrigger function 404
  EKG_LinkTrigger function 404
  EKG_Locate function 406
  EKG_LockObjectList function 407
  EKG_MessageTriggeredAction function 409
  EKG_OutputToLog function 410
  EKG_QueryEntityStructure function 411
  EKG_QueryField function 413
  EKG_QueryFieldID function 414
  EKG_QueryFieldName function 416
  EKG_QueryFieldStructure function 417
  EKG_QueryFunctionBlockContents function 419
  EKG_QueryMultipleSubfields function 421
  EKG_QueryNotifyQueue function 423
  EKG_QueryObjectName function 425
  EKG_QueryResponseBlockOverflow function 426
  EKG_QuerySubfield function 428
  EKG_ResponseBlock function 430
  EKG_RevertToInherited function 431
  EKG_SendNotification function 433
  EKG_Stop function 436
  EKG_SwapField function 437
  EKG_SwapSubfield function 439
  EKG_TriggerNamedMethod function 440
  EKG_TriggerOMMethod function 442
  EKG_UnlinkNoTrigger function 443
  EKG_UnlinkTrigger function 443
  EKG_UnlockAll function 445
  EKG_WHERE_AM I function 446
  query functions 373
  reason codes 472, 474
  reference 375
  user API services 374

G
gateways 61
GENALERT command, monitoring
  non-network devices 85
generic alert data subvector, INIT
  alert 80
generic commands using DOMP010
  protocol 62
global character 284
GMFHS 25, 116
GMFHS_Aggregate_Objects_Class
  objects 25
GMFHS fields used by the view layout
  facility 691
GMFHS_Managed_Real_Objects_Class
  objects 24
gmfs methods, restricted
  DUIFCAAP method 490
  DUIFCADC method 490
  DUIFCACP method 490
  DUIFCASB method 490
  DUIFCATC method 490
  DUIFCACP method 490
  DUIFCDTC method 490
  DUIFCDUC method 490
  DUIFCGR2 method 490
  DUIFCGR3 method 490
  DUIFCGRA method 490
  DUIFCGRT method 490
  DUIFCLS2 method 490
  DUIFCLS3 method 490
  DUIFCLSR method 490
  DUIFCMUU method 490
  DUIFCRDC method 490
  DUIFCRTP method 490
  DUIFCRTU method 490
  DUIFCRUC method 490
  DUIFCSRT method 490
  DUIFCURA method 490
  DUIFCTUC method 490
  DUIFEGSN method 490
  DUIFITKN method 490
  DUIFRAIP method 490
  DUIFRTCR method 490
  DUIFVCT method 490
  DUIFVCT method 490
  DUIFVDCRT method 490
  DUIFVCT method 490
  DUIFVCT method 490
  DUIFVCF method 490
  DUIFVF method 490
  DUIFVEX method 490
  DUIFVF method 490
  DUIFVF method 490
  DUIFVGET method 490
  DUIFVIEW method 490
  DUIFVLSL method 490
  DUIFVLIT method 490
  DUIFVMDR method 490
  DUIFVNGI method 490
  DUIFVNGI method 490
  DUIFVNOT method 490
  DUIFVFPR method 490
  DUIFVSUB method 490
  DUIFVTKN method 490
  DUIFVUNS method 490

Index 705
gmfs methods, restricted (continued)  
DIUFVUPD method 491  
DIUFVCL method 491  
GMFHS parameter, AGGRST 498  
GMFHS_Shadow_Objects_Class objects 24  
GMT offset 76, 82  
GRAPHICVAR  
load function data type 302  
null value 262  
linkage conventions 268  
loading modules 253  
loading class structure and method names 252  
loading data cache 246  
loading data models, RODM 246  
loading definitions and method names 253  
loading modules 253  
H  
HAS_FIELD load function primitive 286  
HAS_INDEXED_FIELD load function primitive 286  
HAS_INSTANCE load function primitive 286  
HAS_NO_FIELD load function primitive 287  
HAS_NO_INSTANCE load function primitive 287  
HAS_NO_SUBFIELD load function primitive 288  
HAS_PARENT load function primitive 288  
HAS_PRV_FIELD load function primitive 288  
HAS_SUBFIELD load function primitive 289  
HAS_VALUE load function primitive 289  
hex_chars data type 297  
hex_literal data type 297  
hierarchy resource list subvector, INIT alert 81  
high-level load function statements  
CREATE 280  
definition 9  
DELETE 281  
description 244  
MANAGED OBJECT CLASS 278  
SET 282  
syntax 277  
syntax rules 275  
how GMFHS builds views 91  
I  
identifiers, data type fields 224  
identifying installation methods 247  
il_parm data type 297  
Indexed_data_length function 449  
Indexed_data_length parameter, EKG Locate function 406  
Indexed_data_ptr function 449  
Indexed_data_ptr parameter, EKG Locate function 406  
index fields 222, 482  
INDEXLIST load function data type 302  
information, accessibility xiii  
information, disability xiii  
inheritance in methods 354  
Inheritance_state function parameter 449  
Inheritance_state parameter, EKG QueryFieldStructure function 417  
INHERITS load function primitive 290  
INIT_ACCEPT_ACCEPT protocol command 69  
INIT ACCEPT protocol command 69  
INIT alert  
cause undetermined subvector 80  
Date/Time subvector 81  
DOMS010 protocol 80  
first product set ID subvector 81  
generic alert data subvector 80  
hierarchy resource list subvector 81  
probable cause subvector 80  
second product set ID subvector 81  
self-defining text message subvector 82  
INIT high-level syntax keyword 279  
INITIAL high-level syntax keyword 279  
initialization load  
cold start 251  
description 248  
warm start 251  
initialization method  
coding 345  
definition 6  
services available 368  
installing methods 360  
INTEGER load function data type 302  
interfaces 222  
INVOKED_WITH load function primitive 290  
INVOKER high-level syntax keyword 280  
IS_LINKED_TO load function primitive 291  
IS_NOT_LINKED_TO load function primitive 292  
IsPartOf connectivity relationship 26  
K  
keyboard, shortcut keys xiii  
L  
languages, methods 223  
languages, RODM user applications 222  
Last_checkpoint_ID function parameter 449  
Last_checkpoint_ID parameter, EKG Connect function 387  
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 369  
link action functions 220  
link-editing application programs 305  
link-editing RODM programs 364  
linkage conventions 268  
links between objects 218, 219  
LISTLEVEL parameter 272  
load function  
ATTRLIST high-level syntax keyword 279  
authorization level 254  
batch job 252  
calling load function, module 253  
checking output listings 255  
class structure definitions 247  
coding high-level load function statements 275  
coding primitive statements 284  
common syntactic elements 294  
CREATE high-level statement 280  
creating class structure and object definitions 247  
data definitions necessary initialization 268  
object load 268  
structure load 268  
data types 302  
deciding load type 248  
DELETE high-level statement 281  
delimiters 276  
EKGLOAD sample job 252  
EKGLOADP sample procedure 252  
FORCE HAS NO INSTANCE primitive 285  
FORCE NOT A CLASS primitive 285  
HAS_FIELD primitive 286  
HAS_INDEXED_FIELD primitive 286  
HAS_INSTANCE primitive 286  
HAS_NO_FIELD primitive 287  
HAS_NO_INSTANCE primitive 287  
HAS_NO_SUBFIELD primitive 288  
HAS_PARENT primitive 288  
HAS_PRV_FIELD primitive 288  
HAS_SUBFIELD primitive 289  
HAS_VALUE primitive 289  
initialization load  
cold start 251  
description 248  
warm start 251  
initialization 248  
Introduction 242  
INVOKED_WITH primitive 290  
INVOKER high-level syntax keyword 280  
IS_LINKED_TO primitive 291  
IS_NOT_LINKED_TO primitive 292  
link-edit restriction, calling module 253  
MANAGED OBJECT CLASS 278  
definition 9  
description 244  
SET 282  
syntax 277  
syntax rules 275  
how GMFHS builds views 91  
links between objects 218, 219  
LISTLEVEL parameter 272  
load function  
ATTRLIST high-level syntax keyword 279  
authorization level 254  
batch job 252  
calling load function, module 253  
checking output listings 255  
class structure definitions 247  
coding high-level load function statements 275  
coding primitive statements 284  
common syntactic elements 294  
CREATE high-level statement 280  
creating class structure and object definitions 247  
data definitions necessary initialization 268  
o
load function (continued)
loading object definitions 252
loading RODM data cache 243
MANAGED OBJECT CLASS high-level statement 278
MODE high-level syntax keyword 282
MODLST high-level syntax keyword 282
NOT_A_CLASS primitive 292
OBJCLASS high-level syntax keyword 280
object definitions 247
object load 250
OBJINST high-level syntax keyword 280
operations 242
parameters
  CODEPAGE 272
  LISTRLEVEL 272
  LOAD 273
  NAME 273
  OPERATION 274
  SEVERITY 275
parameters, invoking 272
PARENT IS high-level syntax keyword 279
PL/I and C 253
primitive statements 244
primitive statements, definition 9
PRIVATE high-level syntax keyword 279
processing logic 284
PUBLIC high-level syntax keyword 279
PUBLIC_INDEXED high-level syntax keyword 279
reference 260
SET high-level statement 282
statements 242
structure load 249
SUBFIELD_HAS_VALUE primitive 292
SUBFIELD_INHERITS primitive 293
submitting jobs, invoking load functions 252
syntax, primitive statements 284
syntax rules
  high level statements 275
  primitive statements 284
  using CLASSID data type 261
  using OBJECTID data type 261
verify operation 260
LOAD parameter 273
loading class structure and method names 252
loading data models, RODM 246
loading definitions and method names 253
loading modules 253
loading object definitions 252
loading RODM data cache 243
loading the data models, RODM 246
loading RODM data cache 243
loading object definitions 252
loading modules 253
LOAD parameter 273
loading class structure and method names 252
loading data models, RODM 246
loading definitions and method names 253
loading modules 253
loading object definitions 252
loading RODM data cache 243
loading the data models, RODM
  Using sample CNMSSH12 57
Local_copy_map function parameter 449
Local_copy_map parameter,
  EKG_QueryFieldStructure function 417
Local_inherited_flag function parameter 450
locate failing resource views, customizing 97
locate resource function 115
Log_message function parameter 450
Log_message parameter,
  EKG_OutputToLog function 410
logging
  controlling, EKG_LogLevel field 205
  controlling, EKG_MLogLevel field 206
logical 31
logical connectivity, defining 40
LogicalConnDownstream connectivity relationship 28
LogicalConnFP connectivity relationship 28
LogicalConnUPstream connectivity relationship 28
long-lived parameters 360
Long_lived_parm function parameter 450
Long_lived_parm parameter
  EKG_AddNotifySubscription function 377
  EKG_AddObjDelSubs function 379
  EKG_DeleteNotifySubscription function 396
  EKG_DeleteObjDelSubs function 400
M
MACRO preprocessor option, PL/I applications 305
macros 364
major vector
  execute 86
  reply, execute command 86
text data parameter 86
MANAGED OBJECT CLASS high-level statement 278
managed objects
  defining 36
  definition 4
  identifying 23
managed real objects, definition 24
management objects
  defining 33
  definition 4
  identifying 22
manuals
  feedback xii
  online xii
  ordering xii
maximizing RODM performance
  customizing parameters and system fields 482
  data model structure and size 482
  method design 482
  user application design 482
  using indexed fields 482
Message_CCSID function parameter 450
Message_CCSID parameter,
  EKG_OutputToLog function 410
method, class 208
method API
  asynchronous error notification 328
  call statement format 358
  coding installation-written methods 362
  compiling programs 363
  control block relationships 307
  deciding method type 356
description 343
designing, performance 482
function reference 375
genre restrictions 366
  general restrictions 366
  grouping, API services 358
  languages 223
  linking programs 364
  long-lived parameters 360
  method API services 358, 374
  method parameters 359
  programming language-specific preprocessor statements 363
  programming reference 371
  query field control block, sample 307
  query field control block sample restrictions 364, 366
  RODM system (z/OS), illustration 223
  services available, initialization methods 368
  services available, object-specific methods 368
  services available, RODM methods 367, 368
  short-lived parameters 360
  tasks best performed 343
  writing installation-written methods 362
  writing RODM methods 343
Method_name function parameter 450
Method_name parameter
  EKG_QueryNotifyQueue function 423
  EKG_TriggerOMethod function 442
method name table
  bypassing load 263
description 263
Method_output_message function parameter 450
Method_output_message parameter,
  EKG_SendNotification function 433
Method_parms function parameter 450
Method_parms parameter
  EKG_ChangeField function 380
  EKG_ChannelMultipleFields function 381
  EKG_CreateClass function 388
  EKG_CreateObject function 391
  EKG_DeleteClass function 393
  EKG_DeleteObject function 397
  EKG_LinkTrigger function 404
  EKG_QueryField function 413
  EKG_SwapField function 437
  EKG_TriggerNamedMethod function 441
  EKG_TriggerOMethod function 444
methods
change method 346
deciding method type 356
definition 6
description 343
DUIFCAAP method 490
DUIFCADC method 490
DUIFCPC method 490
DUIFCCS method 490
DUIFCCIC method 490
DUIFCUCAP method 493
DUIFCUAP method 493
DUIFCUAR method 493
DUIFCUCR method 493
DUIFCUED method 495
DUIFCUE42 method 496
DUIFF GN method 498
DUIFFA method 498
DUIFFFR method 499
DUIFF FAS method 499
DUIFF S method 499
DUIFFITK method 499
DUIFFRRA method 499
DUIFFRDFS method 500
DUIFFRRTC method 490
DUIFFVF method 500
DUIFFVCT method 490
DUIFFVCTR method 491
DUIFFVFCE method 491
DUIFFVVF method 491
DUIFFVFEX method 491
DUIFFVFV method 491
DUIFFVFF method 491
DUIFFVFLST method 491
DUIFFVFLT method 491
DUIFFVMDR method 491
DUIFFVING method 491
DUIFFVNGN method 491

methods (continued)
DUIFVNO method 491
DUIFVNOT method 491
DUIFVPFF method 491
DUIFVFS method 491
DUIFVTK method 491
DUIFVUN method 491
DUIFVUPD method 491
DUIFVVLG method 491
DUIFVEXV method 491
DUIFVGET method 491
DUIFVIEW method 491
DUIFVINS method 501
DUIFVFLST method 491
DUIFVFLT method 491
DUIFVMDR method 491
DUIFVING method 491
DUIFVNGN method 491

more detail views (continued)
defining (continued)
physical 46
more detail views, defining layout
parameters 49
multiple policies, resources
belonging 126
multiple-response presentation
protocol 75
MultiSystem Manager
exception views 114
multi-valued fields
description 218
example 219
MyClassChildren field 214
MyID field 214
MyName field 214
MyObjectChildren field 214
MyPrimaryParentID field 214
MyPrimaryParentName field 214

N
NAME parameter 273
named method
description 353
parameters 353
procedure interface 354
restrictions 366
Naming_count parameter, entity access
information block 312
Naming_count parameter, field access
information block 315
navigating using menus 508
NETCENTER
converting status keywords 72
internal status values 72
migrating 87
protocols, migrating 87
NetView/6000, establishing session
using sample CNMS4406 79
NetView for AIX, establishing session
using sample CNMS4406 79
NetView interface, RODM 191
NetView Resource Manager (NRM) 161
NetView-supplied methods
change methods 486
descriptions 362
EGCGTIM method 486, 487
EGKCMIV method 487
EGKNECL notification method 484
EGKNLST notification method 485
EGKNOTF notification method 484
EGKNTHD notification method 485
EGKOTIPI notification method 488
EGKSPPII notification method 487
general restrictions 366
identifying installation methods 247
inheritance 354
initialization method 345
installation-written methods 362
installing and freeing methods 360
long-lived parameters 360
method library 369
method parameters 359
name table 247
named methods 353
NetView-supplied methods 362, 483
not for customer use, list of 490
notification methods 350
null method 356
object-independent methods 344
object-specific methods 346
obtaining storage 362
query methods 349
reason codes 481
restrictions 364, 367
return and reason codes 454
services available, initialization
methods 368
services available, object-specific
methods 368
services available, RODM
methods 367, 368
short-lived parameters 360
types 344
using object-independent
methods 356
using object-specific methods 356
writing RODM methods 343
METHODSPEC load function data
type 302
MODE high-level syntax keyword 282
MODLIST high-level syntax
keyword 282
monitoring alerts 169
monitoring non-network devices 85
more detail view
description 32
types of 32
more detail views
defining
logical 46
overflow, response block 317

P

parameter mapping table modifying 264
sample 266
parameter substitution using DOMP010 protocol 63
parameters customizing, performance 482
long-lived 360
method 359
short-lived 360
Parent_access_info_ptr function parameter 451
Parent_access_info_ptr parameter, EKG_CreateClass function 388
parent-child relationships, defining 41
PARENT IS high-level syntax keyword 279
ParentAccess connectivity relationship 27
PASSTHRU presentation protocol, definition 64
PASSTHRU session protocol, definition 78
path, resource owner 27
peer 31, 44
performance 482
physical 31
physical connectivity, defining 40
PhysicalConnDownstream connectivity relationship 28
PhysicalConnPP connectivity relationship 27
PhysicalConnUpstream connectivity relationship 28
PL/I, definition 6
pointer, null 304
policies, resources belonging to multiple 126
policy, resources suspended from aggregation due to 129
policy, system status updates no longer sent to resources due to 131
postfix notation in conditional statements 148
PPI command transport envelope 86
PPI NMGs 85
PPI transport protocol, definition 84
presentation protocol defining 62
multiple-response 75
single-response 74
PresentationProtocolName defining 62
DOMP010 62
DOMP020 63
PASSTHRU 64
typical values 61
prev_val subfield, definition 217
primitive statements description 244
FORCE_HAS_NO_INSTANCE 285
FORCE_NOT_A_CLASS 285
global character 284
primitive statements (continued)
HAS_FIELD 286
HAS_INDEXED_FIELD 286
HAS_INSTANCE 286
HAS_NO_FIELD 287
HAS_NO_INSTANCE 287
HAS_NO_SUBFIELD 288
HAS_PARENT 288
HAS_PRV_FIELD 288
HAS_SUBFIELD 289
HAS_VALUE 289
INHERITS 290
INVOKED_WITH 290
IS_LINKED_TO 291
IS_NOT_LINKED_TO 292
NOT_A_CLASS 292
processing logic 284
SUBFIELD_HAS_VALUE 292
SUBFIELD_INHERITS 293
syntax rules 284
PRIVATE high-level syntax keyword 279
Private_public_flag function parameter 451
probable cause subvector, INIT alert 80
process, loading data cache 246
processing logic, primitive statements 284
program calls, RODM 303
program-to-program interface (PPI) NMGs 85
programming languages C 5, 9
PL/I 5, 9
protocol multiple-response 75
single-response 74
protocol command INIT_ACCEPT 69
INIT_ACCEPT_ACCEPT 69
SESSION_REQUEST 68
SESSION_REQUEST_ACCEPT 69
SET_CLOCK 69
SET_CLOCK_ACCEPT 69
protocol specification, migrating 87
PT keyword, DOMP010 protocol 68
PUBLIC high-level syntax keyword 279
PUBLIC_INDEXED high-level syntax keyword 279
publications feedback xii
ordering xii
Q
query field control block sample, method API 359
query functions 373
query method description 349
parameters 349
procedure interface 349
query subfield, definition 216
RCVRID_CHARVAR data item 488
Reason_code function parameter 451
Reason_code parameter
EKG_ChangeMultipleFields function 381
EKG_ExecuteFunctionList function 403
EKG_LockObjectList function 408
EKG_QueryMultipleSubfields function 421
Reason_code parameter, transaction information block 310
reason codes each function 472
functions 474
NetView-supplied methods 481
return code 0 455
return code 12 469
return code 4 456
return code 8 459
RODM 454
recipient_spec, common syntactic element 299
RECIPIENTSPEC load function data type 302
register conventions 304
reply, execute command major vector 86
representing policy definitions in RODM 124
Requested_data parameter, EKG Locate function 406
Requested_info_array parameter, EKG QueryMultipleSubfields function 421
Requesting_method_ID function parameter 451
Requesting_method_ID parameter, EKG WhereAmI function 446
RODM 446
reserved data types 225
resource owner path 27
resources, updates no longer sent 131
resources belonging to multiple policies 126
resources suspended from aggregation due to policy 129
ResourceTraits field changing with a user method 113
defining exception criteria 104
response block definition 8
description 316
overflow 317
response block, error message 317
Response_block_length function parameter 451
Response_block_length parameter
EKG_ExecuteFunctionList function 403
EKG_Locate function 406
EKG_QueryEntityStructure function 411
EKG_QueryField function 413
EKG_QueryFieldID function 415
EKG_QueryFieldName function 416
EKG_QueryFieldStructure function 417

710  Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer’s Guide
RODM (Resource Object Data Manager)  (continued)
notification process, definition 9
object definitions 247
object deletion notification 328
object identifiers 212
object locking 222
object names 211
objects 210
program calls 303
reserved data types 225
return and reason codes 454
structure 197
subfields 215
system-defined classes 198
system-defined fields 213
system structure (z/OS),
illustration 223
user API services 374
using load functions 241
using user APIs 304
writing RODM application
programs 303
writing RODM methods 343

RODM_name parameter, access
block 308

RODM unload function
customizing 545
description 544
running 547
starting 545

RODMView 507
change field function 533
compound query function 519
create actions function 538
delete actions function 540
link function 529
locate objects function 526
method actions function 542
navigating within RODMView 508
restrictions 509
signing on to 511
simple query function 512
starting 509
subfields actions function 536
unlink function 529

RP keyword, DOMP010 protocol 70
running RODM load functions 250

S
sample network
illustration 18
loading 57
samples
EKG5VDCL sample variable
declarations 225
EKG5WAIT sample PL/I call,
EKGWAIT 324
EKG6VDCL sample variable
declarations 225
EKG6WAIT sample C call,
EKGWAIT 324
EKGLLOAD sample job, load
function 252
EKGLOADP sample procedure, load
function 252

Index 711
samples (continued)
FLCSEXV 114
FLCSSMT 114
sd_parm, common syntactic element 299
second product set ID subvector, INIT
alert 81
self-defining text message subvector, INIT
alert 82
SELFDEFINING
load function data type 302
null value 262
SENDER_CHARVAR data item 489
service point 5, 19
services, method API 358
session
element management system 77
establishing for NetView/6000 79
establishing for NetView for AIX 79
establishing with DOMS010
protocol 78
termination 82
session protocol 77
SESSION_REQUEST_ACCEPT protocol
command 69
SESSION_REQUEST protocol
command 68
session termination alert 82
SessionProtocolName
defining 77
DOMS010 77
NONE 78
PASSTHRU 78
typical values 61
SET_CLOCK_ACCEPT protocol
command 69
SET_CLOCK protocol command 69
SET_CLOCK protocol command 69
SET high-level statement 282
SEVERITY parameter 275
shadow objects 36
definition 24
NMC support for 24
short-lived parameters 360
shortcut keys, keyboard xiii
SHORTNAME
load function data type 302
null value 262
Sign_on_token parameter, access
block 308
single-response presentation protocol 74
SMALLINT load function data type 302
SN keyword, DOMP010 protocol 70, 71
SDA domain
defining 33
definition 22
SNA resources, defining 36
SNA topology manager
loading the data model, RODM 57
spans
defining dynamically built views to 117
defining predefined views to 117
DisplayResourceName, use with spans 119
examples of defining views to 118
elements of restricting resources in views 121
time stamp 76
GMFHS processing 116
spans (continued)
MyName field, use with spans 119
RACF 123
resolving problems for views 122
restricting resources in views 119
set and clear operator status 123
UserSpanName, use with spans 119
SPIE routines, method restrictions 367
ST keyword, DOMP010 protocol 71
stack model postfix processing 150
STAE routines, method restrictions 367
statements, complex conditional 148
statements, conditional 147
statements, postfix notation in conditional 148
status, NETCENTER internal 72
status groups
description 144
using 144
using to customize DisplayStatus 145
STEPLIB DD statement 266
Stop_ECB parameter, EKG_Connect
function 387
Stop_type parameter function 452
Stop_type parameter, EKG_Stop
function 436
storage key 364
structure load, load function 249
subfield, common syntactic element 300
Subfield function parameter 452
SUBFIELD_HAS_VALUE load function
primitive 292
SUBFIELD_INHERITS load function
primitive 293
Subfield_map function parameter 453
Subfield_map parameter
EKG_CreateFunctionField 389
EKG_CreateSubfield function 392
EKG_DeleteSubfield function 398
EKG_QueryFieldStructure
function 417
Subfield parameter
EKG_ChangeField function 380
EKG_ChangeSubfield function 382
EKG_QueryNotifyQueue
function 423
EKG_QuerySubfield function 428
EKG_RevertToInherited function 431
EKG_SwapSubfield function 439
EKG_WhereAmI function 446
subfields
associated fields 221
change subfield 216
data types 217
notify subfield 216
prev_val subfield 217
query subfield 216
RODM fields 215
time-stamp subfield 217
value subfield 215
submitting jobs, invoking load
functions 252
subs_spec, common syntactic
element 300
subs_spec_list, common syntactic
element 300
SUBSCRIBEID
load function data type 302
null value 262
subscribing 321
subscription, definition 9
Subscription_info function
parameter 453
Subscription_info parameter,
EKG_DeleteNotifySubscription
function 396
SUBSCRIPTSPEC load function data
type 302
SUBSCRIPTSPECLIST load function data
type 302
substitution, parameter, using DOMP010
protocol GMFHS replaces the 63
subvector
cause undetermined 80
Date/Time 81
first product set ID 81
generic alert data 80
hierarchy resource list 81
probable cause 80
second product set ID 81
self-defining text message 82
supporting data correlation 86
supporting data correlation subvector 86
suspending aggregation via an
aggregate 130
syntax
collection specification 151
syntax rules
collection specification 151
syntax, collection specification 151
target, definition 7
TASKINFO_CHARVAR data item 489
TASKNAME_CHARVAR data item 489
tasks best performed with methods 343
text data parameter major vector 86
time-stamp keyword 73
TIMESTAMP load function data
type 302
timestamp subfield, definition 217
timeout
alerts 76
command responses 77
considerations 76
time stamp 76
Tivoli Customer Support xiii
TM keyword, DOMP010 protocol 73
token-ring network layout 21

Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer's Guide
Index 713