HP OpenView Configuration Server Using Radia

for the AIX, HP-UX (PA-RISC 1.1 & 2.0), Solaris, Windows, and MVS operating systems

Software Version: 4.5.4

User's Guide

Manufacturing Part Number: T3424-90089

June 2005

Legal Notices

Warranty

Restricted Rights Legend

Copyright Notices

Trademark Notices

Acknowledgements

PREBOOT EXECUTION ENVIRONMENT (PXE) SERVER
Copyright © 1996-1999 Intel Corporation.

Support

Please visit the HP OpenView support web site at:

http://www.hp.com/managementsoftware/support

This web site provides contact information and details about the products, services, and support that HP OpenView offers.

HP OpenView online software support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valuable support customer, you can benefit by using the support site to:

Search for knowledge documents of interest

Submit enhancement requests online

Download software patches

Submit and track progress on support cases

Manage a support contract

Look up HP support contacts

Review information about available services

Enter discussions with other software customers

Research and register for software training

Most of the support areas require that you register as an HP Passport user and log in. Many also require a support contract.

To find more information about access levels, go to:

http://www.hp.com/managementsoftware/access_level

To register for an HP Passport ID, go to:

http://www.managementsoftware.hp.com/passport-registration.html

Revisions

Version 4.5.4 Features

This version of the HP OpenView Configuration Server Using Radia (Configuration Server) has several enhancements on the Configuration Server-supported platforms as outlined below.

AIX, HP-UX (PA-RISC 1.1 and 2.0), Solaris, and Windows

- Multiple updates were made to the ZEDMAMS utility in order to improve the functionality of the verbs. Two of the more notable updates are:
  - The CHANGE_OBJECTID verb was updated to correctly handle blank object IDs.
  - EXPORT_CLASS and EXPORT_INSTANCE have been modified so that they no longer export the _NULL_INSTANCE_.
- When inaccurate and unsuccessful object resolutions occur, the client task will be terminated.
- The database verification has been updated to recognize lowercase international characters in the resource header.
- The license utility, ZLICUTIL, has been modified to correctly delete licenses that do not meet template-size criteria.

AIX, HP-UX (PA-RISC 1.1 and 2.0), and Solaris

- The `version.nvd` file has been implemented on UNIX platforms to aid in determining the version of the Configuration Server.

  For more information, see Configuration Server Version Information, on page 24.
- Support has been added for the ZLOGSWCH and ZLOGWRAP REXX methods to include Configuration Server log information when run asynchronously via the REXX Manager.
- Support has been added for the for the execution of UNIX shell commands from a REXX method.

Windows

- The Intel Pentium 4 processor is now correctly recognized. (It is no longer incorrectly read as Xeon Processor, Model 1, Stepping 2.)
- The EDMSIGNR method has been updated to include Microsoft® Windows security code changes.

Documentation Changes

This printing of the User's Guide for the HP OpenView Configuration Server Using Radia (Configuration Server Guide) contains the following changes to information and procedures.

Chapter 2: Tuning the Configuration Server

- Page 62, the section SWITCH_TOD was revised.
- Page 63, the section THRESHOLD was revised.
- Page 69, Configuration Server Running as a Windows Service, this is a new section.
- Page 124, User Log Naming Conventions, this is a new section.

Contents

1 Introduction

About this Guide

Who this Guide is for

The User's Guide for the HP OpenView Configuration Server Using Radia (Configuration Server Guide) is for an authorized systems administrator of HP OpenView Using Radia who will be managing the Radia database.

What this Guide is about

Chapter 2 shows how to modify the Configuration Server's `edmprof` file in order to enhance system performance.

The next three chapters describe: how to customize the flow of database processing (Chapter 3); the various approaches to configuring client notification (Chapter 4); and the advantages of ODBC connectivity to a SQL database (Chapter 5).

Chapter 6 and Chapter 7 present information about using the database utilities and Access Method Services to manage the Radia database, respectively.

The performance aspects of the Configuration Server are described in Chapter 8; Chapter 9 addresses basic troubleshooting issues and recommended actions.

The last two chapters discuss multi-mode Configuration Servers (Chapter 10) and SSL Managers (Chapter 11).

Overview of Configuration Server

The HP OpenView Configuration Server Using Radia (Configuration Server) is the heart of any HP OpenView Using Radia implementation. The entire HP OpenView Using Radia environment is built around it. In conjunction with the Radia database, the Configuration Server manages software content and policy across a network environment.

The Configuration Server can be installed on a single server or on multiple servers. Once installed, it stores (in the Radia database) application data and client-computer information, and distributes application packages based on the policies that are established by an administrator.

The Configuration Server can:

- **Manage a Radia Client's Desired State**
  Dynamically generate a client's **desired state** based on situation-specific data, thereby creating a software environment that automatically adapts to user and machine changes.
- **Distribute Radia Database Objects**
  Synchronize distributed objects (such as application components, packages, computer configurations, and policy relationships) across the network, and automatically manage object transfer between HP OpenView Using Radia components.
- **Deliver and Maintain Radia Policies**
  Maintain enterprise policies in the Radia database. When a device that is being managed by HP OpenView Using Radia connects to the Configuration Server all current policies are automatically applied.
- **Issue Connects and Fulfill Requests**
  Contact devices causing them to initiate data requests and desired-state requests. These requests can be according to a schedule or upon notification from an administrator.

Radia Database

The Radia database is stored on the Configuration Server, and is accessed and modified via the System Explorer for the HP OpenView Administrator Workstation Using Radia (System Explorer). It houses Radia objects, and is where administrators save and maintain an enterprise's service-entitlement policies.

The Radia database includes the following information.

- The digital assets that are distributed by HP OpenView Using Radia.
- The policies that determine package assignment to managed devices and subscribers.
- Security and access rules for administrators of HP OpenView Using Radia.

Radia Database Documentation

Consult the following HP OpenView Using Radia documents for more information on the structure, use, and components of the Radia database.

- *Database Reference Manual for the HP OpenView Configuration Server Using Radia (Database Reference Manual)*
- *System Explorer Guide for the HP OpenView Administrator Workstation Using Radia (System Explorer Guide)*
- *Getting Started Guide for HP OpenView Configuration Server and Management Portal Using Radia (Getting Started Guide)*

Configuration Server and Client Operations Profiles

Client Operations Profiles (**COP**) allow an administrator to identify and prioritize data-access points (**DAP**) without having to use additional customized scripts. A server-access profile (**SAP**) is a generic way of defining all possible data-access points for a service. In an HP OpenView Using Radia environment, the Configuration Server can be a SAP.

In order to do so, a Configuration Server must have a role, or function, defined for the ROLE attribute of the SAP class in the Radia database. A Configuration Server can assume any of the following roles.

- **Client Operations Profiles (O)**
  This Configuration Server will get the client computer's Client Operations Profiles.
- **Service resolution (S)**
  This Configuration Server will resolve the client computer's services.
- **Client self-maintenance (M)**
  This Configuration Server will help the client perform self-maintenance.
- **Reporting (R)**
  This Configuration Server will store, in the Radia database PROFILE file, reporting objects from the client computer.

- **Data download (D)**
  This Configuration Server will download application data to the client computer.
- **All (A)**
  This Configuration Server will perform all of the functions listed here.

Configuration Server Information

This section presents the following Configuration Server information:

- Supported Operating Systems, starting below.
- Backing up the Configuration Server, starting on page 23.
- Configuration Server Version Information, starting on page 24.

Supported Operating Systems

This version of the Configuration Server supports the operating systems and levels that are listed in Table 1.

The operating systems and levels that are listed in this table are the *minimum requirements*; subsequent patches, fixes, and service packs that have been/are added to these minimum levels are/will be supported as well.

Table 1: Supported Operating Systems and Levels

Platform	Operating System and Level
Windows	NT 4.0 Server, Service Pack 6
	2000 Server, Service Pack 3
	2003 Server, Service Pack 1
	XP Professional, Service Pack 2
UNIX	HP-UX (PA-RISC 1.1 and 2.0), Version 10.20
	Red Hat Enterprise Linux, ES Version 3.0
	SuSE Enterprise Server, Version 9.0

Platform	Operating System and Level
	Solaris, Version 2.7
	AIX, Version 4.3
MVS	OS/390, level 2.5

Backing up the Configuration Server

HP recommends that the Configuration Server (and Radia database) be periodically backed up. To facilitate doing so, a list of the Configuration Server directories (that are installed by default) and their applicable platforms and contents are provided in Table 2.

HP recommends that backups be done in accordance with each environment's in-house, corporate protocol.

Table 2: Configuration Server Directories

Directory	Platforms	Contents
bin	Windows	The Configuration Server binary files and the `edmprof` file.
	UNIX	The shell scripts that enable you to start, stop, clean up, and query the Configuration Server.
DB	Windows & UNIX	The Radia database files.
exe	UNIX	The Configuration Server methods and executable files.
internet	Windows & UNIX	The Internet HTML and graphics files.
lib	Windows & UNIX	This directory contains files for proper Configuration Server operation; do not modify or delete these files.
log	Windows & UNIX	The Configuration Server log.
mail	Windows & UNIX	The Configuration Server outgoing mail messages.

Directory	Platforms	Contents
rexx	Windows & UNIX	This directory is for storing customized REXX methods. Note: Its sub-folder, `NOVADIGM`, contains the default Configuration Server REXX methods.
shell	Windows	The batch and application files, such as the un-install and query scripts, and the files that coincide with the Configuration Server options available from the **Start** menu.
	UNIX	This directory is empty.

Configuration Server Version Information

To determine which version of the Configuration Server is running on a machine, query the Configuration Server log file.

To query the Configuration Server version level

1 Go to **Start** → **Programs** → **Radia Configuration Server** → **Log Viewer**.

2 Click **Log Viewer**.

The Configuration Server activity log opens.

3 Scroll down to the **Configuration Server Information Section**.

```
==========================================================================================
Configuration Server Information Section
==========================================================================================
Configuration Server is Version <4.5.1> Build <651>
Configuration Server built on <Nov 21 2002 at 09:46:34>
version.nvd: <V4.5.2 RCS Level>
Verifying product consistency according to <C:\Novadigm\ConfigurationServer\bin\version.nvd>file
==========================================================================================
```

1st line = Installed version of the Configuration Server.

3rd line = Service-pack level of the Configuration Server.

4th line = Path to the file `version.nvd`.

4 To find out the Configuration Server version, check the line, `Configuration Server is Version <n.n.n> Build <nnn>`.

If there is a value for the line, `version.nvd`: `<V`*`n.n.n`*` RCS Level>`, then a Service Pack has been applied, and the updated Configuration Server version level is reflected.

- If there is no value for the line, `version.nvd: <Vn.n.n RCS Level>`, then either:
  - no Service Pack has been applied, or
  - the `version.nvd` file has been deleted.
- Periodically check the **Product Updates** section of the HP OpenView web site for Configuration Server Service Packs.
- HP discourages deleting and modifying the file, `version.nvd`.

Radia Documentation

Table 3 presents a list of publications that are associated with the various HP OpenView Using Radia products, and which might be referenced in this manual.

Table 3: Radia Documentation

Radia Component	**Radia Manual**
HP OpenView Configuration Server Using Radia	*Messages Guide for HP OpenView Configuration Server Using Radia (Messages Guide)*
HP OpenView Integration Server Using Radia	*Installation and Configuration Guide for the HP OpenView Inventory Manager Using Radia (Inventory Manager Guide)*
HP OpenView Management Applications Using Radia	*Installation and Configuration Guide for the HP OpenView Software Manager Using Radia (Software Manager Guide)*
	Installation and Configuration Guide for the HP OpenView Inventory Manager Using Radia (Inventory Manager Guide)
	Installation and Configuration Guide for the HP OpenView Application Manager Using Radia (Application Manager Guide)
Additional Titles	*Installation and Configuration Guide for the HP OpenView Management Portal Using Radia (Management Portal Guide)*

Radia Component	Radia Manual
	REXX Programming Guide for the HP OpenView Application Manager and Configuration Server Using Radia (REXX Programming Guide)
	Getting Started Guide for HP OpenView Using Radia (Getting Started Guide)
	System Explorer Guide for the HP OpenView Administrator Workstation Using Radia (System Explorer Guide)

2 Tuning the Configuration Server

At the end of this chapter, you will:

- Know how to tune the HP OpenView Configuration Server Using Radia (Configuration Server) for maximum performance.
- Be familiar with the Configuration Server `edmprof` file.

Understanding the Tuning Process

The performance of the Configuration Server depends on a number of factors, such as the number of clients being concurrently processed, the complexity of the configurations for those clients, the volume of the data being processed, and network bandwidth. The configuration of the Configuration Server log, which documents system status for informational purposes and problem determination, can also alter performance.

The Configuration Server operational parameters are contained in its `edmprof` file. The performance of the Configuration Server can be tuned by working with the settings of the `edmprof` file.

This chapter details the structure of the `edmprof` file and the options that can be specified.

Configuration Server Settings Overview

The Configuration Server `edmprof` file contains the parameters that determine how the Configuration Server will operate. This file is organized into **sections**—with each section containing keywords, called **settings**. The sections can be categorized into functional areas. Further detail for the settings in each of the sections is provided in this chapter.

- **Identification**
  - — MGR_STARTUP specifies the Configuration Server by ID, name, type, and communications port.

- MGR_DIRECTORIES identifies the directory paths for the Radia databases, REXX methods, and non-REXX methods.
- MGR_LICENSE contains your unique license string.

- **Specification**
  Configuration Server settings establish application-wide technical parameters.
  - MGR_CACHE contains cache-processing options.
  - MGR_TPINIT identifies communications packet sizes.
- **Initialization**
  - MGR_ATTACH_LIST determines which Configuration Server programs are initiated at startup, and has options to define their functioning.
  - MGR_CLASS specifies which classes and instances of the Radia database are cached during the Configuration Server initialization process.
  - SECTION_DELIMITERS specifies which characters are used to distinguish sections in the `edmprof` file.
- **Operations**
  A number of sections in the Configuration Server `edmprof` file contain parameters for system operations.
  - MGR_ACCESS determines Configuration Server access to administrator and console functions.
  - MGR_DB_VERIFY specifies the parameters for automatic Radia database verification at initialization.
  - MGR_DIAGNOSTIC specifies the timing, size, and logging options for verifying adequate disk space for Radia database operations.
  - MGR_DMA specifies parameters for HP OpenView Distributed Configuration Server Using Radia (Distributed Configuration Server).
  - MGR_METHODS identifies options for method processing.
  - MGR_NOTIFY specifies the Configuration Server defaults for client notifications.
  - MGR_OBJECT_RESOLUTION specifies parameters used in object resolution.

- MGR_POOLS (MVS, Windows, and OS/2 only) allocates available pools of memory (in different sizes) for system tasks.
- MGR_RETRY values define how long a client is to wait before attempting to reconnect to the Configuration Server.
- MGR_TASK_LIMIT identifies the maximum concurrent Configuration Server tasks, ongoing and deferred, system related and/or client-connect related.
- MGR_TIMEOUT specifies the amount of time a Configuration Server will wait for an inactive client.

- **Monitoring**
  - MGR_LOG and MGR_TRACE identify where the Configuration Server log is located, how flexible it is, and which individual system traces are to be captured in the log.
  - MGR_SMTP_MAIL specifies the parameters for using SMTP mail messages to support Configuration Server monitoring.
  - MGR_MESSAGE_CONTROL specifies where log messages are to be sent and if they are to be suppressed.
  - MGR_USERLOG allows an administrator to establish a user logging facility.

Viewing and Editing Configuration Server Settings

The `edmprof` file can be opened in a text editor, so that its parameters can be viewed and, if necessary, adjusted. It can be found in varying locations, based on operating system, as described below:

- **Windows NT, Windows 2000, and Windows Server 2003**
  The `edmprof` file settings are located in the `edmprof.dat` file, located in the `bin` sub-directory of the Configuration Server directory.

  Or, alternatively:

  From the system tray, go to **Start→Programs→Radia Configuration Server→Profile Editor**.

- **UNIX**
  The `edmprof` file settings are located in the `.edmprof` file,, in the `home` directory of the UNIX user ID that installs, starts, stops, and maintains the Configuration Server.

- **MVS**
  The `edmprof` file settings are located in the PARMLIB.

Although the settings in the `edmprof` file are readily accessible and easy to modify, some of the values are critical to the operation of the Configuration Server. Therefore, it is imperative that you *do not*:

- alter any `edmprof` file settings unless without consulting this guide first, and then using the recommended values.
- delete any `edmprof` file settings unless instructed to do so by a member of HP Technical Support.

Configuration Server Settings

Most of the sections of the `edmprof` file can be independently configured, whereas some are based on operating system and communications requirements. While many of the sections are optional, a number are required for proper Configuration Server functioning.

Since some of the sections in the `edmprof` file are optional, and others are platform-specific, it's possible that not all of the settings will be visible in every `edmprof` file.

The `edmprof` file is created during the installation of the Configuration Server. Much of its information is derived directly from parameters that are specified during the installation; while others are automatically entered during the installation.

Two types of values are in the `edmprof` file.

- An **as-installed** value represents a manual input, specified during installation or a derived entry for a required setting.
- A **default** value is established by the Configuration Server if there is a blank value for that setting, whether it is required or manually entered.

Format of the EDMPROF File

The `edmprof` file is organized into sections. Each section contains individual keywords called settings. Each setting receives an acceptable value, which can be numeric, alphabetic, or alphanumeric.

⚠ Some settings and values are critical to the operation of the Configuration Server. Do not alter or delete these, unless instructed to do so by a member of HP Technical Support.

The following is an example of the format of a section of the `edmprof` file.

Example

```
[MGR_SECTION]
SETTING = VALUE
SETTING = VALUE
SETTING = VALUE
```

The following table presents a list of the `edmprof` file sections, a brief description, and whether the section is required or optional.

Table 4: Sections of the EDMPROF File

Section Name Description	**Required or Optional**
MGR_ACCESS Specifies access to the Administrator and Console functions.	Optional
MGR_ATTACH_LIST Specifies the Configuration Server attach list that defines the programs to be attached when the Configuration Server is started.	Required
MGR_CACHE Specifies cache-processing options, such as cache segments, size, and statistics.	Optional
MGR_CLASS Specifies processing parameters for classes and instances.	Optional
MGR_DB_VERIFY Specifies the extent of automatic database verification.	Optional
MGR_DIAGNOSTIC Specifies the parameters for diagnostic Manager (zdiagmgr) tasks.	Optional
MGR_DIRECTORIES Specifies the path for the databases, REXX methods, and non-REXX methods.	Recommended but not required

Section Name Description	**Required or Optional**
MGR_DMA Specifies the parameters for Distributed Configuration Server operations.	Optional
MGR_ERROR_CONTROL Specifies how to process errors that are encountered while the Configuration Server is running.	Optional
MGR_LOG Specifies the logging directory and options for the Configuration Server logging facility.	Recommended, but not required
MGR_MESSAGE_CONTROL Specifies where messages are to be sent and whether they are to be suppressed.	Optional
MGR_METHODS Specifies options for method execution.	Recommended, but not required
MGR_MODULE_PRELOAD Specifies the modules that are to be loaded prior to Configuration Server startup (MVS only).	Optional
MGR_NOTIFY Specifies the parameters for notify processing.	Optional
MGR_OBJECT_RESOLUTION Specifies the parameters used in object resolution.	Optional
MGR_POLICY Specifies the IP address/name and port of the HP OpenView Policy Server Using Radia (Policy Server), if it has been enabled.	Optional
MGR_POOLS Specifies the allocation of pool sizes on startup.	Optional
MGR_RESOLUTION_FILTERS Specifies the filtering rules for resolution, based on the connection type.	Optional
MGR_RETRY Specifies when a client should attempt to re-connect to the Configuration Server, following rejection.	Optional
MGR_RIM Specifies the IP address/name and port of the Inventory Manager, if it has been enabled.	Optional

Section Name Description	**Required or Optional**
MGR_ROM Specifies the IP address/name and port of the HP OpenView Information Base Using Radia (Information Base), if it has been enabled.	Optional
MGR_RMP Specifies the IP address/name and port of the HP OpenView Management Portal Using Radia (Management Portal), if it has been enabled.	Optional
MGR_SMTP_MAIL Specifies the parameters the Configuration Server uses to interface with SMTP.	Optional
MGR_SNMP Contains parameters to specify where SNMP traps are to be sent, and controls the behavior of the built-in SNMP agent.	Optional
MGR_SSL Specifies the parameters for the HP OpenView Adapter for SSL Using Radia (Adapter for SSL).	Optional
MGR_STARTUP Specifies the Configuration Server ID and TCP/IP port number.	Required
MGR_TASK_LIMIT Specifies the number of concurrent tasks allowed.	Required
MGR_TIMEOUT Specifies how long the Configuration Server will wait for a request from a connected client before disconnecting it.	Optional
MGR_TPINIT Specifies communications packet sizes.	Required
MGR_TRACE Controls and influences diagnostic logging for the Configuration Server.	Optional
MGR_TRANSLATION Specifies customized translation tables.	Optional
MGR_USERLOG Specifies the logging directory and options for the user logging facility.	Optional

Section Name Description	**Required or Optional**
OBJECT_SIZES Specifies the number of heaps and the heap size for Radia database objects that are being created on the Configuration Server as in-storage Radia database objects.	Optional
RCS_TUNING_CONTROL Provides a mechanism to override the default values that are specified in the Configuration Server self-tuning tool.	Optional
SECTION_DELIMITERS Specifies the left and right delimiters that are to be used for enclosing the section names within the Configuration Server `edmprof` file.	Optional

In the following pages, each section of the `edmprof` file is detailed, covering the individual settings and the values for each. Also described is the impact of tunable settings on Configuration Server performance.

MGR_ACCESS

This section determines access to the Administrator and Console.

Table 5: MGR_ACCESS Settings

Setting	Description
ADMIN	Specifies access to the Administrator. The values are DENY, ALLOW, and IGNORE.
CONSOLE	Specifies access to the Console. The values are DENY, ALLOW, and IGNORE.

Example

```
[MGR_ACCESS]
ADMIN      = DENY
CONSOLE    = DENY
```

Table 6: MGR_ACCESS Values

Setting	Value as Installed	Default Value
ADMIN	DENY	DENY
CONSOLE	DENY	DENY

Access can be controlled by native operating-system security features also.

Performance and Usage Considerations

- ADMIN=ALLOW will provide access to the Administrator without checking the ZACCESS domain of the Radia database. Therefore, if ADMIN=ALLOW, and an Administrator attempts an action for which no access rules have been defined, the attempted action will be allowed.
- When ADMIN=DENY, access rules governing Administrator actions are defined in the ZACCESS domain of the Radia database. Therefore, if

ADMIN=DENY, the Administrator will be unable to perform an action unless there is an access rule specifically allowing it.

- If ADMIN=IGNORE the Administrator will be able to perform any action because the access rules will be ignored by the Configuration Server. Setting ADMIN=IGNORE essentially disables all access rules as they relate to Administrator functions.
- Access to the Console is determined by local password security policy. If CONSOLE=ALLOW and local security is not configured, read-only access is granted. If CONSOLE=IGNORE, local Console security is bypassed.

Table 7: Radia Administrator Access-Level Values

Access Value	Definition
ALLOW	This value will result in the Configuration Server *not* checking the administrator access rules before granting access to an administrator to perform Radia database administrator functions. This value disables all administrator access-rule checks, even if they are defined in the database.
DENY	This value will result in the Configuration Server checking administrator access rules before granting access to an administrator to perform Radia database administrator functions. An administrator will be *unable* to perform an action unless there is an access rule defined in the database for that administrator specifically *allowing* it. This is the recommended setting when configuring administrator security. Note: If this option is specified, undefined administrators will not have access to any Radia database administrator functions, unless the ADMINID._NULL_INSTANCE_ is modified to allow such access.
IGNORE	This value will result in the Configuration Server checking administrator access rules before granting access to an administrator to perform Radia database administrator functions. An administrator will be *able* to perform an action unless there is an access rule defined in the database for that administrator specifically *prohibiting* it. Note: If this option is specified, undefined administrators will have full access to all Radia database administrator functions, unless the ADMINID instance exists for that administrator to prohibit such access.

MGR_ATTACH_LIST

In this section, specify which programs (Configuration Server tasks) are to be attached at startup, and set the options for these processes.

Table 8: MGR_ATTACH_LIST Settings

Setting	Description
ATTACH_LIST_SLOTS	Number of slots for the attach list kept in shared memory of the Configuration Server. Every entry is 132 bytes long. HP recommends that this setting be one more than the number of CMD_LINE settings used. For example, if there are seven CMD_LINE settings, set this value to 8. Note: No process starts are required. However, system ability is limited if ZTCPMGR, ZREXXMGR, and ZNFYTMGR are not attached.
CMD_LINE	Command line to use when starting processes. Blanks are not allowed in CMD_LINE= sub-string. A second format is allowed when multiple instances of the same task are required and need to be separately identified. This format is: CMD_LINE=(NAME=,ADDR=,PORT=). These names should be unique within the Configuration Server.
LIMIT (=CLOSE)	The Configuration Server will drop any client connection attempts when LIMIT=CLOSE is specified, *and* either the Task Limit or Storage Limit of the Configuration Server is reached. The Configuration Server will not return an object to the client (no return EDMLOCTP), nor will it request the client retry the connection. Note: This setting is valid with ztcpmgr only.
RESTART	Use this setting to determine if a Configuration Server task will be restarted when abnormally terminated. Blanks are not allowed in RESTART= sub-string. The default is NO.
RESTART_LIMIT	Number of attempts to restart an attached process that has terminated.

Setting	Description
VERIFY_INTERVAL	Interval (in minutes) between verifications that attached processes are still running. If this value is 0, no verification will occur. The default is 1 (minute).

The following table lists the Configuration Server tasks.

Table 9: Configuration Server Tasks

Task Name	Description
zbldpmgr	Patch Build Manager task
zdiagmgr	Diagnostic Manager task
znfytmgr	TCP Notify Manager task
zrexxmgr	REXX Manager task
zrtrymgr	Notify Retry Manager task
zsmtrmgr	SMTP receive Manager task
zsmtsmgr	SMTP send Manager task
zsnmpmgr	SNMP Manager task
zsslmgr	SSL Manager task
ztcpmgr	TCP Manager task
zutilmgr	Utility Manager task

Example

```
[MGR_ATTACH_LIST]

ATTACH_LIST_SLOTS  = 15
RESTART_LIMIT      = 7
VERIFY_INTERVAL    = 5
CMD_LINE           = (zutilmgr) RESTART=YES
CMD_LINE           = (zrexxmgr) RESTART=YES
CMD_LINE           = (zsnmpmgr) RESTART=YES
CMD_LINE          = (zsmtrmgr) RESTART=YES
CMD_LINE          = (zsmtsmgr) RESTART=YES
CMD_LINE          = (znfytmgr) RESTART=YES
CMD_LINE          = (ztcpmgr) RESTART=YES
CMD_LINE          = (ztcpmgr LIMIT=CLOSE) RESTART=YES
CMD_LINE          = (zbldpmgr)
```

Table 10: MGR_ATTACH_LIST Values

Setting	Value as Installed	Default Value	Minimum Value	Maximum Value
ATTACH_LIST _SLOTS	15	15	Number of CMD_LINEs	Number of CMD_LINEs + 1
CMD_LINE	Determined by installation options	Determined by installation options	N/A	N/A
RESTART	YES	NO	N/A	N/A
RESTART_ LIMIT	7	7	0 = No restart	3200

Performance and Usage Considerations

- If the ATTACH_LIST_SLOTS value is too low, the Configuration Server will attach as many tasks as there are slots available. Any remaining tasks will not be attached until a slot becomes vacant. A too-high value could degrade overall system performance by unnecessarily setting aside resources that remain unused.
- The RESTART_LIMIT value should be set higher when critical Configuration Server functions are being performed. Note that regardless of the value in RESTART_LIMIT, the task will not be reinitiated if RESTART=NO.
- To ensure that vital processes continue running, set VERIFY_INTERVAL lower when critical Configuration Server functions are being performed. A higher setting, on the other hand, might save CPU cycles when total demand is a critical factor.
- The ztcpmgr task supports virtual IP addresses. It accepts the IP address and port number on the command line, as in:

```
CMD_LINE = (ztcpmgr addr=1.1.1.94,port=4438)
```

If the address is not specified, the machine address is used.

MGR_CACHE

This section specifies cache-processing options, such as cache size, statistics, load type, and error response.

The settings in this section should be established based on operating-system environment and performance needs.

Table 11: MGR_CACHE Settings

Setting	Description
AVERAGE_OBJECT_SIZE	Average size of an object that will be cached.
CACHE_SEGMENTS	Number of cache segments.
CACHE_SIZE	Size of each cache segment.
CACHE_STATS	A YES/NO switch to accumulate statistics.
ICACHE_COUNT_ERROR	Instructs the Configuration Server on what action to take (shut down or log a warning) if the ICACHE instance counts don't match the DMA instance counts when it (the Configuration Server) is starting up. The default is SHUTDOWN. Note: The default is SHUTDOWN because running the Configuration Server without the correct number of cached items it is not recommended. See Performance and Usage Considerations, starting on page 43. • SHUTDOWN directs the Configuration Server program (ZTOPTASK) to shut down. • WARN instructs the Configuration Server program to continue loading and record the event in the Configuration Server log. Notes: ICACHE_COUNT_ERROR=SHUTDOWN is the discrete, default behavior in pre-4.5.2 Configuration Server versions. If your Configuration Server was upgraded to version 4.5.2 from 4.5.1 *and* you want to change the default behavior, this setting must be manually added to the `edmprof` file.

Setting	Description
ICACHE_LOAD_TYPE	The DOMAIN.CLASS entries that are specified in the MGR_CLASS section are loaded, one class at a time, into Index cache via one of the following methods. • **AUTOMATIC** instructs the Configuration Server to auto-determine the loader method (FULL or CHUNKY) to be used. This is the default. • **FULL** instructs the Configuration Server to always use the FULL loader method—loading the database as a single entity. • **CHUNKY** instructs the Configuration Server to always use the CHUNKY loader method—loading each Radia database service one at a time. For more information, see ICACHE_LOAD_TYPE Considerations on page 44.
ICACHE_SIZE	Size of the Index cache.

Example

```
[MGR_CACHE]

AVERAGE_OBJECT_SIZE    = 2048

CACHE_SEGMENTS         = 2

CACHE_SIZE             = 5242880

CACHE_STATS            = NO

ICACHE_COUNT_ERROR     = SHUTDOWN

ICACHE_LOAD_TYPE       = AUTOMATIC

ICACHE_SIZE            = 0
```

Table 12: MGR_CACHE Values

Setting	Value as Installed	Default Value	Minimum Value	Maximum Value
AVERAGE_ OBJECT_SIZE	2048 Bytes	2048	2048	6144

Setting	Value as Installed	Default Value	Minimum Value	Maximum Value
CACHE_SEGMENTS	2	0 (No caching)	0 (No caching)	System resource dependent
CACHE_SIZE	5,242,880 Bytes	0	0	System resource dependent
CACHE_STATS	NO	NO	N/A	N/A
ICACHE_ COUNT_ERROR	SHUTDOWN	SHUTDOWN	N/A	N/A
ICACHE_ LOAD_TYPE	AUTOMATIC	AUTOMATIC	N/A	N/A
ICACHE_SIZE	N/A	N/A	0 (No Index caching)	N/A

Performance and Usage Considerations

When modifying any of the cache parameters in this section, take care not to exceed the amount of virtual memory that is available.

Also, sufficient virtual memory must be available to handle the maximum concurrent-resolutions workload.

- Classes that are going to be cached are defined in the MGR_CLASS section.

  At a minimum, the ZSYSTEM.ZPROCESS and ZSYSTEM.ZMETHOD classes should be cached.

- If the value of AVERAGE_OBJECT_SIZE is too low, the Configuration Server will have to reconfigure the cache until the object is accommodated.

  If the value of AVERAGE_OBJECT_SIZE is too high, the caching function might not be used efficiently.

- The ICACHE_SIZE setting should be used in conjunction with the CACHE_SEGMENTS setting and the MGR_CLASS section.

  ICACHE_SIZE will be enabled only if the value of CACHE_SEGMENTS is greater than zero.

UNIX Configuration Servers
The recommended value for ICACHE_SIZE is the *total _number_of_instances_in_the_classes_to_be_cached* (see the MGR_CLASS section) multiplied by **64**.

- If ICACHE_COUNT_ERROR=SHUTDOWN and the Configuration Server shuts down, areas to check are:
  - Check the Configuration Server's log file,
  - Check the integrity of the connection to the database,
  - Verify the load type for the database connection method, and
  - Verify the validity of the database.

ICACHE_LOAD_TYPE Considerations

This section presents information that should be considered when specifying a value for ICACHE_LOAD_TYPE.

- If ICACHE_LOAD_TYPE=FULL, the Configuration Server will attempt to load into ICACHE, as a single element, all of the Radia database classes that are specified in the MGR_CLASS section.

  The Configuration Server sequentially accesses all instances in the specified directories.
- If ICACHE_LOAD_TYPE=CHUNKY, the Configuration Server will attempt to load all of the Radia database classes that are specified in the MGR_CLASS section—but as individual entities.
  - **Component Classes**
    The Configuration Server uses the PACKAGE class in the current domain to control the loading of Component class instances—that is, all instances that are associated with a PACKAGE instance will get loaded into ICACHE, but orphan instances (those not associated with a PACKAGE instance) will not get loaded into ICACHE.
  - **Non-Component Classes**
    The Configuration Server uses a systematic approach to load the instances of instances of non-Component classes.

HP recommends the CHUNKY method for large databases because it minimizes the load on the network, thereby decreasing the likelihood of problems and the possibility of the database integrity being compromised.

- If a Configuration Server has a UNC connection (database path starts with \\) to the Radia database and the DMA count exceeds 300K instances, ICACHE_LOAD_TYPE=AUTOMATIC will choose the CHUNKY load method. See UNC Connectivity Issues on page 55.

Purging Dynamic Cache

This section provides precautionary information about purging the dynamic cache on a Configuration Server. It includes information about protection that HP has introduced in order avoid purging the Radia database when an HP OpenView Proxy Server Using Radia (Proxy Server) is co-located with the Configuration Server and dynamic cache is enabled.

HP recommends not using dynamic cache for a co-located Proxy Server.

If the dynamic-cache root is a Radia database then, by default, this parameter (default=0) prevents automatic dynamic-cache purging of aged files when the dynamic-cache index is saved. By default, the new parameter automatically safeguards against purging dynamic-cache files from a Radia database.

To remove the safeguard and allow a purge of shared-resource, dynamic-cache files, set the parameter to 1, as shown in the following example.

Example

```
-dynamic -allow -shared -resource -purge 1
```

MGR_CLASS

This section specifies which classes and instances will be cached during the initialization of the Configuration Server.

Table 13: MGR_CLASS Settings

Setting	Description
CLASS	Specifies which classes and instances will be cached at initialization.
	Format: DOMAIN.CLASS={VALUE1,VALUE2,VALUE3,VALUE4} Note: If multiple domains have identical class names, the class templates must be identical. If they are not, performance will be adversely impacted and objects, larger than necessary, might be created from the resolution process. *`Cache_Class_Template_&_Base_Instance`*, *`Cache_All_Instances`*, *`Heap_Size`*, *`Max._Number_of_Heaps`*.
	VALUE1 To cache (at Configuration Server startup) the _BASE_INSTANCE_ and class template of the associated class, specify Y. Otherwise, specify N.
	VALUE2 To cache (at Configuration Server startup) all instances in the associated class, specify Y. Otherwise, specify N.
	VALUE3 This value is numeric and represents the initial size of the resolved object for the associated class. This is the size that each heap in the associated DOMAIN.CLASS will occupy in persistent objects. It should include any attributes that have been classed into the in-storage object as the result of resolution. Typically, the size of the transient class instance can be used for transient objects (such as, ZLOCMGR, ZLOCCLNT, ZSCHEDULE). The default (2048) can be refined for ZSERVICE and ZRSOURCE, which are typically 3–4 KB in size. Examine the client object resulting from a representative resolution in order to determine the most appropriate value. Values specified are generally in 512-byte increments.

Setting	Description
	VALUE4 This is a numeric value that indicates an estimate of the number of heaps that are required for resolution. As each client begins resolution, memory is allocated in blocks equal to the sum of all of the products of VALUE3*VALUE4. When memory is exhausted for a persistent class that is being cached, the next increment of storage is obtained in a block determined by the product of VALUE3*VALUE4 for the associated class. For example, if SYSTEMX.ZRSOURCE = Y,Y,2048,100 were specified, the class template and _BASE_INSTANCE_ are cached at startup. All of the instances of the SYSTEMX.ZRSOURCE class are cached in memory (if there is enough room in CACHE_SEGMENTS*CACHE_SIZE for the whole class). The working value for the size of each ZRSOURCE instance after resolution is estimated to be 2048 bytes. An initial allocation for the in-storage ZRSOURCE object is made for 100 heaps that will require 100*2048 bytes, or 20 KB. If the resolution mode for the *average* client required 1000 ZRSOURCE instances, then one initial allocation of 20 KB and nine subsequent allocations of 20 KB would be required to satisfy the entire 1000 ZRSOURCE instance in-storage objects.

At startup, classes are cached in the order they appear if VALUE1=Y and VALUE2=Y.

Example

```
[MGR_CLASS]

ZSYSTEM.ZPROCESS   = Y,N,1024,1

ZSYSTEM.ZMETHOD    = Y,N,1024,1

SYSTEMX.ZLOCMGR    = Y,Y,0450,1

SYSTEMX.ZLOCCLNT   = Y,Y,230,1

SYSTEMX.ZRSOURCE   = Y,N,2048,511

SYSTEMX.ZSERVICE   = Y,Y,2048,60
```

Table 14: MGR_CLASS Values

Setting	Value as Installed	Default Value
CLASS	N/A	N/A

Performance and Usage Considerations

- You can also specify your own unique classes in this section.
- Note that class instance size and number of instances specified in MGR_CLASS have an impact on storage and performance. A class instance size larger than the actual class size represents wasted storage. A class instance size smaller than the actual class size results in continual resolution performance degradation.
- Only the classes listed will be cached. All instances of listed classes are icached, if required, regardless of the values of the first two parameters in the list.

MGR_DB_VERIFY

This section establishes the level of Radia database verification and whether errors are automatically corrected.

If this function is configured, the Configuration Server will scan the Radia database for Y2K compliance, expanded format (ZBASE), and appropriate Radia database ownership (Configuration Server IDs) at startup. If errors are found, the applicable database utility can be run against the Radia database to correct the error. See Chapter 6, Radia Database Utilities starting on page 273 for more information.

Table 15: MGR_DB_VERIFY Settings

Setting	Description
DB_AUTOFIX	A YES/NO switch to determine if the Radia database should be automatically fixed (where possible) if errors are discovered. The default is NO.
VERIFY_DEPTH	Specifies the level of Radia database verification. The values are DOMAIN, CLASS, INSTANCE, and RESOURCE for this option. The default is CLASS. • If DOMAIN is selected, the verification process will verify down to the DOMAIN level. • If CLASS is selected, the verification process will verify down to the CLASS level. • If INSTANCE is selected, the verification process will verify down to the INSTANCE level. • If RESOURCE is selected, the verification process will verify the entire Radia database.

Example

```
[MGR_DB_VERIFY]

MGR_VERIFY_DEPTH   = CLASS

DB_AUTOFIX         = YES
```

Table 16: MGR_DB_VERIFY Values

Setting	Value as Installed	Default Value
VERIFY_DEPTH	N/A	CLASS
DB_AUTOFIX	N/A	NO

Performance and Usage Considerations

- A value of VERIFY_DEPTH=RESOURCE will result in longer start-up times, as the level of detail is deeper. Conversely, VERIFY_DEPTH=CLASS will result in a much quicker start-up time, because the level of verification is decreased.
- Refer to the MGR_ERROR_CONTROL section, which offers handling parameters for errors found during the Radia database verification process.

MGR_DIAGNOSTIC

This section establishes the values the Configuration Server will use to verify that sufficient disk space is available for Radia database operations and logging, as well as the frequency of verification occurrences.

Table 17: MGR_DIAGNOSTIC Settings

Setting	Description
DIAGNOSTIC_INTERVAL	Interval (in seconds) for the diagnostic Configuration Server to verify that sufficient disk space is available for the Radia database and log. If set to 0, monitoring is disabled.
DIAGNOSTIC_MIN_DB_BYTES	The minimum number of bytes established for Radia database operations. The maximum value is 2 GB.
DIAGNOSTIC_MIN_LOG_BYTES	The minimum number of bytes established for logging operations. The maximum value is 2 GB.

Example

```
[MGR_DIAGNOSTIC]

DIAGNOSTIC_INTERVAL       = 900

DIAGNOSTIC_MIN_DB_BYTES   = 50

DIAGNOSTIC_MIN_LOG_BYTES  = 25
```

Table 18: MGR_DIAGNOSTIC Values

Setting	Value as Installed	Default Value
DIAGNOSTIC_INTERVAL	N/A	900
DIAGNOSTIC_MIN_DB_BYTES	N/A	50M
DIAGNOSTIC_MIN_LOG_BYTES	N/A	25M

Performance and Usage Considerations

- Include CMD_LINE=(zdiagmgr) in the MGR_ATTACH_LIST section in order to configure and use this setting.
- On a Radia database:

- When the DIAGNOSTIC_MIN_DB_BYTES threshold (2GB) is reached, the following will be issued:
  - an SNMP trap of 2040.
  - a message to the Configuration Server log:

    ```
    (9282 - Warning: The volume containing the Configuration
    Server Database has only %.0lf free bytes).
    ```
- When the DIAGNOSTIC_MIN_LOG_BYTES threshold (2GB) is reached, the following will be issued:
  - an SNMP trap of 2045.
  - a message to the Configuration Server log:

    ```
    (9283 - Warning: The volume containing the
    Configuration Server log has only %.0lf free bytes).
    ```

- On the Configuration Server, you can program a REXX that calls the EDMMAILQ method to send a notification e-mail.
- Set the following line in the MGR_MESSAGE_CONTROL section of the `edmprof` file,

  ```
  9282, 9283=LOG,EVENTLOG
  ```

  (This triggers messages 9282 and 9283 to be generated in the Configuration Server log and Windows Event Log.)

MGR_DIRECTORIES

This section specifies the path for the databases, REXX methods, and non-REXX methods. See MGR_LOG to specify the directory path for the Configuration Server log.

- The EXPORT_PATH setting can be used to define a directory for export operations.
- The five USER_PATH*n* settings can be customized for each HP OpenView Using Radia environment.

MVS Note
These six parameters are not applicable to MVS.

Table 19: MGR_DIRECTORIES Settings

Setting	Description
DBPATH	Fully qualified directory path for object databases.
EXPORT_PATH	Fully qualified directory path for EXPORT operations, accessible via REXX.
METHOD_PATH	Fully qualified directory path for non-REXX methods.
REXX_PATH	Fully qualified directory path for REXX methods.
USER_PATH1	Working directory to be used by users, accessible via REXX.
USER_PATH2	Working directory to be used by users, accessible via REXX.
USER_PATH3	Working directory to be used by users, accessible via REXX.
USER_PATH4	Working directory to be used by users, accessible via REXX.
USER_PATH5	Working directory to be used by users, accessible via REXX.

Examples

Intel Example:

```
[MGR_DIRECTORIES]

DBPATH        = D:/MGR/DB
EXPORT_PATH   = D:/MGR/BIN/EXPORT
METHOD_PATH   = D:/MGR/BIN
REXX_PATH     = D:/MGR/REXX
USER_PATH1    = D:/MGR/BIN/USER1
USER_PATH2    = D:/MGR/BIN/USER2
```

UNIX Example:

```
[MGR_DIRECTORIES]

DBPATH        = /radia/db
EXPORT_PATH   = /radconfigsrvr/bin/export
METHOD_PATH   = /radconfigsrvr/bin
REXX_PATH     = /radconfigsrvr/rexx
USER_PATH1    = /radconfigsrvr/bin/user1
USER_PATH2    = /radconfigsrvr/bin/user2
```

MVS Example:

```
[MGR_DIRECTORIES]

DBPATH        = DDN:
METHOD_PATH   = DDN:STEPLIB
REXX_PATH     = DDN:SYSEXEC
```

Table 20: MGR_DIRECTORIES Values

Setting (REXX Name)	**Value as Installed**	**Default Value**
DBPATH	As specified during installation	*Current_directory*
EXPORT_PATH (EXPTPATH)	As specified during installation	*Current_directory*
METHOD_PATH	As specified during installation	*Current_directory*

Setting (REXX Name)	**Value as Installed**	**Default Value**
REXX_PATH	As specified during installation	*Current_directory*
USER_PATH1 (USRPATH1)	As specified during installation	*Current_directory*
USER_PATH2 (USRPATH2)	As specified during installation	*Current_directory*
USER_PATH3 (USRPATH3)	As specified during installation	*Current_directory*
USER_PATH4 (USRPATH4)	As specified during installation	*Current_directory*
USER_PATH5 (USRPATH5)	As specified during installation	*Current_directory*

Performance and Usage Considerations

- The REXX directory specified in this section is further defined by the samples sub-directory, which contains a set of sample REXX methods.
- HP supports Universal Naming Code (**UNC**), which allows paths in the `edmprof` file to be specified as shown below:

  ```
  \\VFH_LAPTOP\DRIVE_D\RADIADB
  ```

When using UNC, the address must be preceded by two backslashes (\\), with a single backslash (****) used to separate each sub-folder.

UNC Connectivity Issues

Interruptions in UNC connectivity to a Radia database might result in critical database classes not being accessed during the resolution process. This failed access could lead to incomplete and erroneous resolution of Radia services and, possibly, the inadvertent removal of applications. If this happens, the following errors will appear in the Configuration Server log:

```
NVD7005E 06:19:52 <172.26.132.24  /1930>   Radia Client  --!
ERROR:  CLASS    <PRIMARY.POLICY.USER> NOT FOUND

NVD7005E 06:19:52 <172.26.132.24  /1930>   Radia Client  --!
ERROR:  CLASS    <PRIMARY.POLICY.ZBASE> NOT FOUND

ERROR:  CLASS
<LICENSE.80d40ad0fd2a4ded8c9d26254e8daf0c.MDEVICE> NOT FOUND

NVD5113E 02:25:17 <172.31.18.5     /668>   Radia Client  --!
ERROR RC=<4>    CREATING
INSTANCE<LICENSE.80d40ad0fd2a4ded8c9d26254e8daf0c.MDEVICE
   .3B06A9B26C5047D886942D1E2EC4A46E>
```

Also, during Configuration Server startup, the following will be seen in the Configuration Server log:

```
Configuration Server Database is on <Remote> <NTFS>
<Uncompressed> Drive   <\\example\RCSDB\>

NVD9268I 02:17:34 <ztoptask   /FB4>   System Task --- Drive
   <\\example\RCSDB\> supports <255> character file names

NVD9269I 02:17:34 <ztoptask   /FB4>   System Task   --- Drive
<\\example\RCSDB\> supports <Case-Sensitivity Case-
Preservation Unicode   File-Compression>

NVD9271I 02:17:34 <ztoptask   /FB4>   System Task   ---
Database resides in    <\\example\RCSDB\DB\>
```

The UNC-mapped drive will be reflected in the DBPATH setting of the MGR_DIRECTORIES section in the `edmprof` file.

Recommended Preventive Measures

HP recommends, in addition to the preventive measures detailed here, that the Configuration Server be monitored—especially if UNC disconnects are frequently occurring.

HP recommends the following measures be taken in order to minimize the impact of UNC connectivity interruptions.

- At Configuration Server startup, cache all Radia database classes that are used for resolution.

  (See MGR_CACHE on page 41, and MGR_CLASS on page 46.)
- In the MGR_CACHE section of the `edmprof` file, type **`ICACHE_LOAD_TYPE=CHUNKY`**.

  (See MGR_CACHE on page 41.)
- Make the following Radia database changes to the PRIMARY.SYSTEM.ZMETHOD.LDAP_RESOLVE method:
  - If ZMTHTYPE (method type) is set to EXE, change it to REXX.
  - If ZMTHNAME (method name) is set to RADISH, change it to NVDMTCL.

If policy resolution is being driven by a method other than LDAP_RESOLVE, and the configuration is uncertain, contact HP Technical Support before making any changes to the Radia database.

These settings will result in the client-connects and client-resolutions being stopped—preventing the clients from doing anything, thereby protecting them in the event they are presented an empty catalog. This also prevents the removal of applications.

MGR_DMA

This section specifies the time-out parameter and the directory path for the Distributed Configuration Server (formerly known as the Distributed Manager Adapter, DMA), and enables you to specify values for these options.

You must have the Distributed Configuration Server installed to enable these parameters.

Table 21: MGR_DMA Settings

Setting	Description
ADMIN_LIST	If SECURITY_METHOD is specified this setting is required. It defines the list of administrator user IDs that are allowed to do login. The format is comma-separated, no spaces, and case-sensitive. For EDMSIGNR, the user IDs must be defined in the Configuration Server's native security system.
DMA_TIMEOUT	Maximum interval (in seconds) that the Distributed Configuration Server will wait for tasks to complete before allowing a commit. A value of 0 means, wait indefinitely.
DMA_STAGE_PATH	Path in which staging directories will be created.
SECURITY_METHOD	Name of the security method to be used to verify logins. (Optional) If Distributed Configuration Server security is wanted, the recommended value is EDMSIGNR. If not specified, no login is required.

Examples

Intel Example:

```
[MGR_DMA]

DMA_TIMEOUT     = 0

DMA_STAGE_PATH  = D:\MGR\
```

UNIX Example:

```
[MGR_DMA]
```

```
DMA_TIMEOUT     = 600
DMA_STAGE_PATH  = /radconfigsrvr/
```

Table 22: MGR_DMA Values

Setting	Value as Installed	Default Value
ADMIN_LIST	N/A	None
DMA_TIMEOUT	N/A	600
DMA_STAGE_PATH	N/A	ZTOPTASK Path
SECURITY_METHOD	N/A	None

Performance and Usage Considerations

- In order to decrease the chance of the Distributed Configuration Server synchronization timing out without having committed database updates, increase the value of DMA_TIMEOUT.

MGR_ERROR_CONTROL

This section specifies error-handling parameters for the Configuration Server.

Table 23: MGR_ERROR_CONTROL Settings

Setting	Description
DBERROR	Describes the type of error and the response. The format is: (*Error Type*) = (*Response*) Valid values for (Response) are: [SHUTDOWN, IGNORE], [NOEMAIL, EMAIL], [NOSNMP, SNMP]
UserEmailErrorsTo	E-mail address of the administrator to which error messages should be sent.

The actions taken by the settings in this section depend on the levels specified for VERIFY_DEPTH and the errors discovered during MGR_DB_VERIFY processing.

Example

```
[MGR_ERROR_CONTROL]

DBERROR                  = IGNORE,EMAIL

UserEmailErrorsTo        = administrator@yourcompany.com
```

Table 24: MGR_ERROR_CONTROL Values

Setting	Value as Installed	Default Value
DBERROR	N/A	[SHUTDOWN] [NOEMAIL] [NOSNMP]
UserEmailErrorsTo	N/A	N/A

Performance and Usage Considerations

- Currently, the only DBERROR supported is DBError. Examples of DBErrors are instances with 0 length, attempting to load a template that does not exist, and so forth.
- The UserEmailErrorsTo value must be a valid e-mail address.

MGR_LOG

This section specifies the logging directory and logging options for the Configuration Server logging facility. It also provides detailed information about reclaiming dormant client-device licenses.

Table 25: MGR_LOG Settings

Setting	Description
DIRECTORY	Fully qualified directory path where the Configuration Server log is written.
DISABLE_NT_ EVENT_LOGGING	If YES is specified, the Configuration Server logger will disable its Windows Event logging support. This means that messages sent to the Configuration Server log will not be sent to the Windows Event log, even if EVENTLOG is specified in the section MGR_MESSAGE_CONTROL. If NO is specified, such messages will be echoed to the Windows Event log if EVENTLOG is specified in the MGR_MESSAGE_CONTROL section. Note: This parameter affects only the event logging of Configuration Server log messages. Some Windows Event log records are written without any corresponding Configuration Server log messages. *Windows only*.
DISABLE_SNMP _TRAP_LOGGING	If YES is specified, the Configuration Server logger will disable its SNMP trapping support. This means that messages sent to the Configuration Server log will not be sent to the primary SNMP Manager as traps, even if SNMPTRAP is specified in the section MGR_MESSAGE_CONTROL. If NO is specified, such messages will be sent to the SNMP Manager as traps, if SNMPTRAP is specified in the MGR_MESSAGE_CONTROL section. Note: This parameter affects only the trapping of Configuration Server log messages. Some SNMP traps are issued without any corresponding Configuration Server log messages.
FLUSH_SIZE	The number of bytes between automatic flushes of operating system buffers for Configuration Server log file.

Setting	Description
MESSAGE_DATE	Allows insertion of the date into every line in the log. Values are JULIAN (YYYYDDD), GREGORIAN (DDMMYYYY), and ISO (YYYYMMDD).
MESSAGE _DELIMITER	The left and right delimiters that are used for enclosing the task name and task ID in the Configuration Server log messages. The format is MESSAGE_DELIMITER=xy, where *x* is the left delimiter and *y* is the right delimiter. The options are: [], (), <>, and {}. The default is [].
MESSAGE _PREFIX	The 3-digit, alphabetic, Configuration Server identifier that will precede Configuration Server log messages. Specify any 3-digit, alphabetic value. The default is NVD. Note: If you have log scrapers, verify that they work properly with the NVD setting. If not, change this to RAD.
MESSAGE _WIDTH	The maximum width (in characters) of the messages in the Configuration Server log.
PIPE_SIZE	The maximum amount (in bytes) of log messages that can be queued by the Configuration Server logging facility while the log file is busy. When the value of PIPE_SIZE is reached, any task that issues a log message will freeze until the pipe starts emptying.
SWITCH_TOD	A time-of-day specification that will, each day, automatically trigger log switching (see Log Switching on page 65). The value must be 5 characters, expressed in base-24 time (00:00–23:59), with a colon separator, as in HH:MM. Notes: This is a scheduled event that causes the log to switch; it is independent of, and runs regardless of, any previous log-switching activity. This setting can be used to activate license reclamation using `ZLICUTIL.EXE`. For more information, see License Reclamation, on page 67. This setting is not part of the Configuration Server installation; it must be manually added. HP recommends that this be set to an off-peak, low-activity time.

Setting	Description
THRESHOLD or THRESHHOLD (both accepted)	The maximum number of lines that will be written to the Configuration Server log before it is automatically switched to the next log. When the limit is reached, a new log file is created, regardless of SWITCH_TOD setting. Note: For a more detailed description of this setting and how it can be used for license reclamation, see Log Switching, on page 65.

Examples

Intel Example:

```
[MGR_LOG]

DIRECTORY            = D:\Novadigm\ConfigurationServer\LOG

FLUSH_SIZE           = 100000

MESSAGE_DATE         = JULIAN

MESSAGE_DELIMITER    = []

MESSAGE_PREFIX       = NVD

MESSAGE_WIDTH        = 256

PIPE_SIZE            = 1000000

SWITCH_TOD           = 23:38

THRESHHOLD           = -5000000
```

UNIX Example:

```
[MGR_LOG]

DIRECTORY            = /opt/Novadigm/ConfigurationServer/log

FLUSH_SIZE           = 100000

MESSAGE_DATE         = ISO

MESSAGE_DELIMITER    = ()

MESSAGE_WIDTH        = 256

MESSAGE_PREFIX       = NVD

PIPE_SIZE            = 1000000

SWITCH_TOD           = 23:38

THRESHHOLD           = -5000000
```

MVS Example:

```
[MGR_LOG]
DIRECTORY           = DDN:EDMLOGA
FLUSH_SIZE          = 100000
MESSAGE_DATE        = GREGORIAN
MESSAGE_DELIMITER   = < >
MESSAGE_PREFIX      = NVD
MESSAGE_WIDTH       = 500
PIPE_SIZE           = 1000000
SWITCH_TOD          = 23:38
THRESHOLD           = -5000000
```

Table 26: MGR_LOG Values

Setting	Value as Installed	Default Value	Minimum Value
DIRECTORY	`Novadigm \Configuration Server\log`	*`Current_directory`*	N/A
DISABLE_NT _EVENT_LOGGING	Varies across platforms.	NO	N/A
DISABLE_SNMP _TRAP_LOGGING	Varies across platforms.	NO	N/A
FLUSH_SIZE	1000 bytes	5000 bytes	1
MESSAGE_DATE	N/A	N/A	N/A
MESSAGE _DELIMITER	N/A	[]	N/A
MESSAGE_PREFIX	NVD	NVD	N/A
MESSAGE_WIDTH	256	90	80
PIPE_SIZE	1000000 bytes	1 MB (Win, UNIX) 2 MB (MVS)	1 MB (Win, UNIX) 2 MB (MVS)
SWITCH_TOD	N/A	N/A	N/A
THRESHOLD	-5000000 lines	100000 lines	1

Performance and Usage Considerations

- Increasing the FLUSH_SIZE will enhance performance, but will delay messages flushed to the log file.
- Increase MESSAGE_WIDTH if log messages are being truncated.
- When closely monitoring system status using the Configuration Server log, set THRESHOLD to a positive value to create and save successive portions of the Configuration Server log. If disk-storage space is critical, set the value to a negative number in order to reuse the allocated log disk space.
- If numerous Configuration Server methods are being invoked, use the MGR_METHODS.LOG_LIMIT setting to control the size of the Configuration Server log for each method. Additionally, the MGR_TASK_LIMIT.TASK_LOG_LIM setting controls the number of messages printed by the execution of each task.
- When modifying parameters in this section as they relate to memory or disk utilization, take care not to exceed the maximum amount of memory or storage space available.

Log Switching

By default, the Configuration Server log will be switched according to the log-size value of THRESHOLD and the amount of Configuration Server activity. However, log switching can be configured to automatically occur on a daily basis, at a scheduled time. This is done by specifying the SWITCH_TOD setting and modifying the THRESHOLD setting. THRESHOLD can also dictate what happens to the log that gets rolled over. These sections describe how to do this.

SWITCH_TOD

Specify the time-of-day for the Configuration Server log to automatically roll over. This setting is independent of THRESHOLD, but can be used with it to trigger either of two REXX methods (`ZLOGSWCH.REX` or `ZLOGWRAP.REX`) to launch the user-implemented license-reclamation utility, ZLICUTIL, as described on page 67 in the section, License Reclamation.

THRESHOLD

This setting determines how large the Configuration Server log can be before it is switched to a new log. The value of this setting will dictate what happens to the log that is rolled out, as described in this section.

- If THRESHOLD ≥ 0, the old log will be renamed and saved, and the Configuration Server REXX method, ZLOGSWCH, will run.

The Configuration Server's log directory (`ConfigurationServer\log`) will have multiple log files whose names conform to the following operating system-specific conventions.

ISO is the *International Standards Organization*.

Configuration Server ID is the value of the `edmprof` file's setting MGR_STARTUP.MGR_NAME.

Windows

— The *standard* log-naming format is the Configuration Server log prefix (**nvd**), followed by the letter **r**, and the **Configuration Server ID**:

```
nvdmr001.log
```

— The *log-switch* format is the **r** replaced by an **s**, followed by the **Configuration Server ID**, and the ISO-formatted date and time (each preceded by an underscore) appended:

```
nvdms001_20050427_075835.log
```

UNIX

— The *standard* log-naming format is the Configuration Server log prefix (**nvd**), followed by the **Configuration Server ID**, an underscore, and the **manager** designation:

```
nvd001_manager.log
```

— The *log-switch* format is an **s** added to the prefix, followed by the **manager** designation, and the ISO-formatted date and time (each preceded by an underscore) appended:

```
nvds001_manager_20050427_083357.log
```

This THRESHOLD setting will result in multiple Configuration Server log files. To differentiate between them, check the date on which they were modified.

HP recommends this setting because it results in saved, rather than deleted, log files.

- If THRESHOLD < 0, the old log will be overwritten (but not deleted), and the Configuration Server REXX method, ZLOGWRAP, will run.

  The Configuration Server log directory will have just one old log file whose name conforms to the following operating system-specific conventions.

ISO is the International Standards Organization.

Configuration Server ID is the value of the `edmprof` file's setting MGR_STARTUP.MGR_NAME.

Windows

— The *standard* log-naming format is the Configuration Server log prefix (**nvd**), followed by the letter **r**, and the **Configuration Server ID**:

```
nvdmr001.log
```

— The *log-wrap* (dump) format is the **r** replaced by a **d**:

```
nvdmd001.log
```

UNIX

— The *standard* log-naming format is the Configuration Server log prefix (**nvd**), followed by the **Configuration Server ID**, an underscore, and the **manager** designation:

```
nvd001_manager.log
```

— The *log-wrap* (dump) format sees the **manager** designation being replaced by **mgrdump**:

```
nvd001_mgrdump.log
```

At the next log switch, the previous log (from the last wrap) will be replaced/overlaid by the newly switched log.

ZLICUTIL

The Configuration Server REXX utility, ZLICUTIL, can be used with ZLOGSWCH and ZLOGWRAP for daily **license reclamation**. For more information, see License Reclamation below.

License Reclamation

A Configuration Server license string supports a certain number of client devices. To keep only active, current machines in a license count, the Configuration Server allows the reclamation of client-device licenses. If a client device hasn't connected to the Configuration Server for a specified number of days, its license becomes inactive and can be reclaimed in the license count by the Configuration Server utility, ZLICUTIL.

Considerations

- By default, license reclamation occurs only at Configuration Server shutdown.
- To make license reclamation occur daily:
  - Specify values for SWITCH_TOD and THRESHOLD (see SWITCH_TOD and THRESHOLD in Table 25 on page 61).
  - Modify the appropriate REXX (either ZLOGSWCH or ZLOGWRAP) so that a log-switching REXX method will launch the license utility, ZLICUTIL, when it gets called.
- Licenses that are aged out via ZLICUTIL are not immediately available. They are reclaimed only at the next midnight.

  The exception to condition is that aged-out licenses will be immediately available when the Configuration Server is re-started

The following instructions detail how to implement license reclamation on a Configuration Server.

To reclaim licenses daily

1. Add the log switch setting, SWITCH_TOD, to the MGR_LOG section of the `edmprof` file, and specify a valid value (see SWITCH_TOD in Table 25 on page 61), such as:

   ```
   [MGR_LOG]
   SWITCH_TOD = 23:45
   ```

2. Specify a value for THRESHOLD, as described on page 63.
3. Navigate to `Novadigm\ConfigurationServer\rexx\NOVADIGM`, and copy either ZLOGSWCH or ZLOGWRAP (or both) up one level to the `rexx` directory.

   During the Configuration Server installation, the REXX methods are, by default, installed to `Novadigm\ConfigurationServer\rexx\NOVADIGM`.

   Before making changes to either ZLOGSWCH or ZLOGWRAP, be sure to copy it up one level to the `rexx` directory

4. Open the ZLOGSWCH (or ZLOGWRAP) file, and add the commands shown here:

   ```
   call edmget zcvt

   ADDRESS CMD "ZLICUTIL" zcvt.dbpath zcvt.rptpath
   zcvt.uuid zcvt.mgrid zcvt.syspath || license.nvd
   ```

See the section, Configuration Server Running as a Windows Service, for Windows-specific configuration information.

5 Save and close ZLOGSWCH (or ZLOGWRAP).

From this point on, the license reclamation feature will be started asynchronously when the log switch is invoked.

MVS User's Note

The license utility, ZLICUTIL, cannot run via ZLOGSWCH or ZLOGWRAP. It is a separate job step, and must be run while the Configuration Server is shut down.

Configuration Server Running as a Windows Service

This section describes the changes that are necessary if the Configuration Server is running as a **service** on a Windows machine.

This information is relevant *only* if the Configuration Server is configured as a Windows **service**; if it is not, this information is not applicable.

The directory in which Windows always first looks for its "service" programs is `Windows\system32`. Therefore, in order for a program to be run as a service, it must be located here; if it is not, it won't be found by Windows—or Windows has to be instructed to look elsewhere.

Even though the Configuration Server might be set up to run as a Windows service, it probably isn't in this directory—rather, it is likely in the directory that was specified during the installation (the default of which is `*SystemDrive*:\Novadigm\ConfigurationServer`). Therefore, Windows has to be told where to find the program files. Instructions for doing this are in the following section.

Directing Windows to the Configuration Server Program Files

To resolve the issue of Windows not being able to find the Configuration Server files, make the following changes to the ZLOGSWCH (or ZLOGWRAP) REXX method.

1 Specify the *fully qualified path name* of the directory in which ZLICUTIL resides so that Windows will know where to look. (See the example that follows.)

2 Add a fifth argument—the *fully qualified path name* of the directory in which the file `license.nvd` resides.

Examples

Original ZLOGSWCH (or ZLOGWRAP) REXX method command:

```
ADDRESS CMD "START ZLICUTIL" zcvt.dbpath "H:\Novadigm
\ConfigurationServer\log" "05c87c3e95194a9d9251fee5cbfddafb" zcvt.mgrid
```

Revised ZLOGSWCH (or ZLOGWRAP) REXX method command:

```
ADDRESS CMD "START FULLY_QUALIFIED_BIN_PATH\ZLICUTIL.EXE" zcvt.dbpath
"H:\Novadigm\ConfigurationServer\log" "05c87c3e95194a9d9251
fee5cbfddafb" zcvt.mgrid "FULLY_QUALIFIED_BIN_PATH\LICENSE.NVD"
```

MGR_MESSAGE_CONTROL

This section specifies which log messages are to be sent and to where, or if they are to be suppressed.

Table 27: MGR_MESSAGE_CONTROL Settings

Setting	Description
`nnnnn` = (*`Destination`*)	(*`Message_Number`*) = (*`Message_Destination`*)

The left side of the equals sign (=) specifies which messages are to be affected by the command. There is an "ALL" directive that can be used to affect all messages (0001-9999). Also, if there is no destination after the equals sign, the message has no associated destination, so it is suppressed.

The following table presents a list of the six destinations for log messages.

Table 28: MGR_MESSAGE_CONTROL Log Message Destinations

Destination	Description
EVENTLOG	Write log messages to the Windows Event log. (*Windows only*)
LOG	Write log messages to the Configuration Server log.
REXX	Write log messages to a pre-determined REXX (named MGRLOG). Important: MGRLOG is not an existing REXX, and therefore, must be created in order to use this facility. Note: The MGRLOG REXX will receive the message number and the message text as its first and second input parameters. You can then process the information in whatever manner you chose.
SNMPTRAP	Write log messages as traps to the current SNMP Manager.
USERLOG	Write log messages to the user log. Note: You must first activate the user log feature in the MGR_USERLOG section of the `edmprof` file, by specifying ACTIVATE=YES.
WTO	Write log messages to the MVS Console. (*MVS only*)

Example

```
[MGR_MESSAGE_CONTROL]

ALL                          = SNMPTRAP

500                          =

520-600                      = WTO, LOG

225, 300                     =

220-299, 400, 542-545, 803   = EVENTLOG,REXX,USERLOG
```

In this example:

- The first line will send all messages to the SNMP Manager as traps.
- The second line will send 500 (to all destinations).
- The third line will cause messages 520 through 600 (inclusive) to be written to the MVS console (via WTO) and the Configuration Server log.
- The fourth line will suppress messages 225 and 300 only.
- The fifth line will cause messages 220 through 299 (inclusive), 400, 542 through 545 (inclusive), and 803 to be written to the Windows Event log, the pre-determined REXX, and the user log.

Table 29: MGR_MESSAGE_CONTROL Values

Setting	Value as Installed	Default Value
nnnnn = (*Destination*)	N/A	ALL=LOG

Performance and Usage Considerations

- SNMPTRAP should be used as a destination only if the address of the SNMP Manager has been configured in the SNMP section.
- Any line that does not contain an equals sign (=) is treated as a comment. In addition, lines that begin with an asterisk (*), double forward slashes (//), or a slash (/) are treated as comments. Blanks and tabs can occur anywhere on the line, even ahead of the comment specifications.
- ALL= will suppress all messages.

- Any errors encountered in parsing a line cause the entire line to be ignored and an error message to be written to STDERR (the SYSTEM DD for an MVS Configuration Server).

MVS User's Note

There is a modify console command that can be used to refresh the MESSAGE_CONTROL specifications as well as the TRACE and POOL specifications. It is the command:

```
F  jobname,R,memname
```

and will process any MGR_MESSAGE_CONTROL sections found in member `memname` of parmlib.

MGR_METHODS

This section specifies options for method execution.

Table 30: MGR_METHODS Settings

Setting	Description
LOG_LIMIT	Maximum number of messages a method can issue to the Configuration Server log. When this limit is reached, a message will be written stating that the message limit has been reached and that all other messages from the method will be ignored.
TIMEOUT	Configuration Server method time-out parameter (sec).

Example

```
[MGR_METHODS]

LOG_LIMIT                                        = 0

TIMEOUT                                          = 300
```

Table 31: MGR_METHODS Value

Setting	Value as Installed	Default Value	Minimum Value	Maximum Value
LOG_LIMIT	0	0 = No Limit	0	2,147,483,647
TIMEOUT	300 (seconds)	60 (seconds)	0	32000

Performance and Usage Considerations

- The number of messages generated by the execution of a method is also affected by the TASK_LOG_LIM setting in MGR_TASK_LIMIT.
- Tune system resource usage with the TIMEOUT setting. If system resources are critical, lower the TIMEOUT setting to free up unused processing cycles.
- When the TIMEOUT value is reached, the method is terminated and messages are written to the Configuration Server log.
- If the TIMEOUT=0, the method will continue in effect until its conclusion.

MGR_MODULE_PRELOAD (MVS only)

This section specifies certain load modules that are to be pre-loaded by the Configuration Server during startup to ensure that they are loaded only once, and to thereby secure a performance improvement.

This section is considered *critical* for MVS.

Table 32: MGR_MODULE_PRELOAD Settings

Setting	Description
IEFBR14	A sample module name that is to be preloaded.

Example

```
[MGR_MODULE_PRELOAD]
IEFBR14 = YES
```

Table 33: MGR_MODULE_PRELOAD Values

Setting	Value as Installed	Default Value
IEFBR14	N/A	N/A

MGR_NOTIFY

This section specifies defaults for the Configuration Server's client notification.

Table 34: MGR_NOTIFY Settings

Setting	Description
ISSUE_WAKE_ON_LAN	Enables support for TCP Wake-On-LAN. (*Windows and UNIX only*)
NFY_RETRY	Number of times to attempt re-notification.
NFYT_TIMEOUT	Notify timeout (in seconds) for TCP/IP clients.
RECOVERY_DOMAIN	The domain that the Retry Manager accesses in order to reinitiate incomplete notifies after the Configuration Server has shut down prematurely.
SUBNET_MASK	This value is used to convert a destination IP address into a subnet address for EDMWAKE. Note: Applicable only to version 4.3 Configuration Server, with Service Pack 2.
WAKE_ON_LAN_TTL	*xxxxx* is the number of routers that the WOL broadcast is allowed to pass. The default is 99. Note: Before establishing this value, consult your network administrator.

Example

```
[MGR_NOTIFY]

NFYT_TIMEOUT           = 120
NFY_RETRY              = 5
SUBNET_MASK            = 255.255.0.0
ISSUE_WAKE_ON_LAN      = YES
RECOVERY_DOMAIN        = ALL
WAKE_ON_LAN_TTL        = 99
```

In some network addressing schemes, a class A IP address is used with a class B or class C subnet mask. This often results in problems, caused by parameters passed by a Configuration Server while trying to invoke EDMWAKE. If specified, the SUBNET_MASK parameter will cause the Notify Manager to use this mask from the `edmprof` file, rather than generate the subnet mask according to the network type.

For example, if the notify for IP address, 10.241.5.5 failed, EDMWAKE will be issued for subnet address, 10.241.255.255.

Table 35: MGR_NOTIFY Values

Setting	Value as Installed	Default Value	Value Range
ISSUE_WAKE_ON_LAN	N/A	NO	YES/NO
NFY_RETRY	N/A	0	32000
NFYT_TIMEOUT	120 seconds	0	32000
RECOVERY_DOMAIN	N/A	ALL	RETRY/ALL
SUBNET_MASK	N/A	N/A	N/A
WAKE_ON_LAN_TTL	99	99	99

Performance and Usage Considerations

- Establish MGR_NOTIFY settings based on network operations parameters.
- The MGR_NOTIFY settings should be coordinated with values in the MGR_RETRY and MGR_TASK_LIMIT sections.
- In order for the Wake-On-LAN Notify function to operate, and to allow the retrying of failed operations, zrtrymgr must be specified under MGR_ATTACH_LIST.

MGR_OBJECT_RESOLUTION

This section specifies the parameters to use during object resolution.

Table 36: MGR_OBJECT_RESOLUTION Settings

Setting	Description
ALLOW_DUPLICATE _INSTANCES	Allow or disallow duplicate instances to be used during object resolution. Values are YES and NO.
ALWAYS_CALL _ZADMIN	Force object resolution to call the ZADMIN method to process the ZADMIN object. Values are YES and NO.
ZERRORM_MAX _ERRORS	Maximum number of errors associated with a ZERRORM event.
ZERRORM_MAX _WARNINGS	Maximum number of warnings associated with a ZERRORM event.

Example

```
[MGR_OBJECT_RESOLUTION]

ALWAYS_CALL_ZADMIN              = YES

ZERRORM_MAX_WARNINGS            = 50

ZERRORM_MAX_ERRORS              = 50

ALLOW_DUPLICATE_INSTANCES       = YES
```

Table 37: MGR_OBJECT_RESOLUTION Values

Setting	Value as Installed	Default Value
ALWAYS_CALL_ZADMIN	YES	NO
ALLOW_DUPLICATE_INSTANCES	N/A	YES
ZERRORM_MAX_WARNINGS	N/A	50
ZERRORM_MAX_ERRORS	N/A	50

Performance and Usage Considerations

- ALLOW_DUPLICATE_INSTANCES=NO will cause the Configuration Server to eliminate duplicate services at resolution time. This might result in elimination of duplicate ZRSOURCE instances on the client. As the size of the ZRSOURCE object grows, the resolution time will increase.
- HP recommends that you do not modify or remove ALWAYS_CALL_ZADMIN, as doing so might prohibit administrator type functions.

MGR_POLICY

This section specifies the IP address/name and port of the Policy Server, if it has been enabled.

This section will be present only if the Policy Server option was selected during the Configuration Server installation.

Table 38: MGR_POLICY Settings

Setting	Description
HTTP_HOST	The IP address of the Radia Policy Server. If the RPS is on the same machine as the Configuration Server, specify `localhost`.
HTTP_PORT	The port of the Radia Policy Server. The default is `3466`.

Example

```
[MGR_POLICY]

HTTP_HOST   = localhost

HTTP_PORT   = 3466
```

Table 39: MGR_POLICY Values

Setting	Value as Installed	Default Value	Minimum Value	Maximum Value
HTTP_HOST	None	N/A	N/A	N/A
HTTP_PORT	N/A	3466	N/A	N/A

MGR_POOLS (MVS, Windows, and OS/2 only)

This section allows you to specify allocations (pools) of memory in different sizes, prior to Configuration Server startup. These allocations can be changed dynamically while the Configuration Server is running by using modify commands. In addition, you can establish percentage-of-waste tolerances (the amount of memory that is not used by a specific requirement) for each of the allocations.

You can establish pools of any size. The pool sizes specified, however, should be multiples of eight. If not, they will be rounded up to the next multiple of eight by the system. Likewise, you can specify any number of pools.

➤ The sizes and number of pools you establish, as well as the percentage of waste tolerated, should be based on the workload and operating requirements of your environment.

When the Configuration Server requires memory, the pools are searched from smallest to largest until a pool is found in which the memory requirement fits. When that pool is found, the first item on the free list is used to resolve the request – providing that the percent of space wasted is not above the value specified for that pool. If there is no item on the free list, then the memory allocation is redirected to the standard C heap. If the C heap is also exhausted, a host system native storage request (for example, MVS GETMAIN) will be performed to satisfy the memory request.

The format for specific pool allocation is:

(Pool Size) = (number of elements for that pool size),
(specific percentage of waste tolerated),(expansion increments).

➤ The specific percentage of waste tolerated parameter is optional. The expansion-increments default is 1.

Table 40: MGR_POOLS Settings

Setting	Description
WASTE _TOLERATED	The general percentage of waste that will be tolerated to fit a specific requirement to an available pool. The default is 100 (%).
ALLOCATION_SIZE _ERROR_THRESHOLD	Deny memory allocations above this size.

Setting	Description
ALLOCATION_SIZE_ REPORTING _THRESHOLD	Report memory allocations above this size.
XXXXXX	The allocation for the *XXXXXX* byte pool. Any reasonable number of these can be specified.

By default, the MGR_POOLS section establishes the pool sizes shown in the example below.

Example

```
[MGR_POOLS]

WASTE_TOLERATED                       = 100

ALLOCATION_SIZE_ERROR_THRESHOLD       = 4194304

ALLOCATION_SIZE_REPORTING_THRESHOLD   = 65536

168                                   = 150

296                                   = 50

552                                   = 20

1064                                  = 10

2100                                  = 5000

2700                                  = 100

4136                                  = 4

8232                                  = 2

12500                                 = 2
```

Table 41: MGR_POOLS Values

Setting	Value as Installed	Default Value	Minimum Value	Maximum Value
WASTE_TOLERATED	100	100 (%)	0	100 (%)
ALLOCATION_SIZE _ERROR_THRESHOLD	4 MB	4 MB	0 (disabled)	2 GB

Setting	Value as Installed	Default Value	Minimum Value	Maximum Value
ALLOCATION_SIZE_ REPORTING_THRESHOLD	64 KB	64 KB	0 (disabled)	2 GB
XXXXXX	N/A	(See Table 40 on page 81)	0	32767

Performance and Usage Considerations

- *MVS Only*: The sum of all of the pools requested must not exceed the MVS Regions size specified for the JOB. Also, additional local address space must be left available for MVS operations that require local storage in the address space to complete. These operations include file opens and closes, such as the Configuration Server would perform if it was configured to automatically switch between logs under certain conditions; as well as REXXs that might dynamically allocate and open MVS data sets.
- Provisions should be made for pools that are the product of the MGR_CLASS section of the STARTUP member, as requests for these memory elements will automatically occur at the initiation of resolution (generally, the receipt of the ZMASTER object). Special attention should be given to the pools that will be identified by the product of the third and fourth values specified in MGR_CLASS, as in Y,N,3072,500 for the MGR_CLASS entry for SYSTEMX.ZRSOURCE.
- The product of the third and fourth values of this is a value exactly equal to 1.5 MB, and one of these will be allocated at the initiation of each resolution. Additional storage is required for storage management of these queues, so a storage pool of elements size (3072) * (500) + 500 = 1536500 should be created. The number of elements to assign to this pool is dependent on the MGR_TASK_LIMIT values specified (for example, one element for each concurrent task), and the number of ZRSOURCE instances resolved for each client.
- When using the Distributed Configuration Server, the default setting of ALLOCATION_SIZE_ERROR_THRESHOLD must be changed in order to prevent a destination Radia database being only partially updated. If the Distributed Configuration Server transfers a resource file that is larger than the default (4 MB), and the setting has not been changed, the ‘commit’ operation will halt, leaving the destination Radia database only partially updated. See the examples below for completing this:

- If you use memory pooling, set ALLOCATION_SIZE_ERROR_THRESHOLD to zero, as below:

  ```
  [MGR_POOLS]
  ALLOCATION_SIZE_ERROR_THRESHOLD = 0
  ```

  Here, 0 will disable the memory allocation limit.
- If the MGR_POOLS section does not exist in the `edmprof` file, there are no issues and no changes are required.

MVS User's Note

You can refresh options that are set from the MVS parmlib with the following command:

```
F  jobname,REFRESH,memname
```

or just:

```
F  jobname,R,memname
```

where *`memname`* is a member in the parmlib with which the Configuration Server was started. This is the data set on the PARMLIB DD card in the Configuration Server JCL. The trace options, message filtering routing options, and pools options will be updated from any corresponding MGR_TRACE, MGR_MESSAGE_CONTROL, and MGR_POOLS sections found in that member. Use the REFRESH command to revert to the original Configuration Server start-up options. Monitor, modify, and refresh pool allocations on an ongoing basis by using the following MVS Console commands:

```
F  jobname,R,memname
```

This command refreshes the three above-noted sections from the member `memname` of the current parmlib data set.

There is also a POOLS modify command that enables pool usage monitoring, adding or deleting pools, and pool allocations dynamically. The commands are:

```
F  jobname,POOLS,STATS
```

This will display a line on the Console (and in the log) for each pool you are using. An example follows in Table 42.

Table 42: Pool Values

Size	64	400	632	2048	4120	6504	8200	12000
Number	100	100	100	60	60	30	15	20
Waste %	100	100	100	100	100	89	65	75

InUse	21	2	12	0	0	0	1	0
HighWM	21	15	12	1	1	0	1	4
Allocs	30	656	19	1	5	0	1	218
Empty	0	0	0	0	0	0	0	0
Waste	0	0	0	0	0	0	0	0

- **Size** is the size of the elements of the pool.
- **Number** is the number of elements currently in the pool.
- **Waste %** is the percentage of waste tolerated at storage element allocation time.
- **InUse** is the number of elements in use when the stats call was made.
- **HighWM** is the high watermark that shows the maximum number of elements that have been allocated at any one time since startup or a clear command.
- **Allocs** is the total number of elements that were allocated since startup or since the last clear command on that pool.
- **Empty** is the number of times we failed to get an element because there were no free elements available.
- **Waste** is the number of times we failed to get an element because the waste was higher than the tolerated waste percent.

In the example in Table 42 on page 84, the pools of 64, 400, 632, 2048, and 4120 do not have a specified waste tolerance. This is because these pool sizes are so small that the waste in them will not be an issue.

In the previous command example in the MVS User's Note, POOLS can be abbreviated as POOL, POO, PO, or just P, and STATS can be abbreviated as STAT, STA, ST or just S. Therefore:

```
F  jobname,P,S
```

is equivalent to the command in the MVS note on page 84.

There is a DISABLE command that looks like:

```
F  jobname,DISABLE,2048
```

or just:

```
F  jobname,D,2048
```

This would disable pool 2048, that is, it would prevent new allocations from using that pool. Allocations between 633 and 4120 bytes would then come from the 4120 pool, provided the tolerated waste percentage is met.

A disabled pool will show up on the command with a "D" after the size. A disabled pool can be re-enabled with the ENABLE command, as follows:

```
F  jobname,E,2048
```

The pool counts can be cleared with the CLEAR command:

```
F  jobname,C,2048
```

Note: The *HighWM*, *Allocs*, *Empty*, and *Waste* counters are cleared by the above command.

Lastly, there is an ADJUST command, which looks like:

```
F  jobname,A,2048,80,95
```

This command would add 20 free elements to the 2048 pool (60 + 20 = 80) and would set its waste tolerated percentage to 95. The last parameter (percentage of waste tolerated) is optional. The "A" denoting the ADJUST subcommand can be replaced with ADJUST, ADJUS, ADJU, ADJ, or AD. The Enable, Disable, and Clear commands can also be specified in a longer form.

If you use the ADJUST command to set the number of elements to 0, then the pool is completely deleted if all the elements can be freed. However, the pool count will not go to 0 as long as there are allocated elements. New pools can be added on the fly with the ADJUST command.

Finally, storage trace (STORAGE=YES) can be used to produce a message in the log each time a storage allocation is made and freed. This can be used to tune the pool, and to understand the Configuration Server’s use of dynamic memory.

MGR_RESOLUTION_FILTERS

This section defines filtering rules for resolution, based on connection type.

This section is not included in the `edmprof` file at installation—it must be manually added.

Table 43: MGR_RESOLUTION_FILTERS Settings

Setting	Description
DOMAIN.CLASS	Specify the domain and class of the Radia database for which to define filtering rules for resolution.

Valid values for MGR_RESOLUTION_FILTERS are:

- RADIA – Allow RADIA resolution only.
- EDM – Allow EDM resolution only.
- ANY – Allow all resolutions. This is the default.

Example

```
[MGR_RESOLUTION_FILTERS]

SOFTWARE.ZSERVICE   = RADIA

SYSTEMX.ZSERVICE    = EDM

SYSTEMX.ZRSOURCE    = EDM

SYSTEMX.WORKGROUP   = ANY

PRIMARY.*           = RADIA
```

Table 44: MGR_RESOLUTION_FILTERS Values

Setting	Value as Installed	Default Value	Minimum Value	Maximum Value
DOMAIN.CLASS	N/A	N/A	N/A	N/A

Performance and Usage Considerations

- CLASS can be specified as a valid Radia database class, or with a wildcard (*) to specify all classes in a domain.

MGR_RETRY

This section specifies how soon (in minutes) a client can attempt another connection to the Configuration Server, after being rejected due to an exceeded task limit or a disabled connection.

Table 45: MGR_RETRY Settings

Setting	Description
BUSY_RETRY	Number of minutes for a client to wait before reconnecting when the Configuration Server is at its task limit (TASK_LIMIT).
DISA_RETRY	Number of minutes for a client to wait before reconnecting when the Configuration Server has logons halted.

Example

```
[MGR_RETRY]

BUSY_RETRY   = 1

DISA_RETRY   = 999
```

HP recommends values of at least 1 for these settings in order to avoid unnecessarily tying up the Configuration Server.

Table 46: MGR_RETRY Values

Setting	Value as Installed	Default Value	Minimum Value	Maximum Value
BUSY_RETRY	7 minutes	0 (allow connect now)	0 (allow connect now)	999 (do not retry)
DISA_RETRY	999 (do not retry)	999 (do not retry)	0 (allow connect now)	999 (do not retry)

Performance and Usage Considerations

- You can raise these values if processing resources are critical. Lowering these values will provide greater assurance that connections will be re-established if broken during client connects.

- The MGR_RETRY settings should be coordinated with values in the MGR_NOTIFY and MGR_TASK_LIMIT sections.

MGR_RIM

This section specifies the IP address/name and port of the Inventory Manager if it has been enabled.

This section will be present only if the Inventory Manager option was selected during the Configuration Server installation.

Table 47: MGR_RIM Settings

Setting	Description
HTTP_HOST	The IP address of the Radia Inventory Manager. If the RIM is on the same machine as the Configuration Server, specify *localhost*.
HTTP_PORT	The port of the Radia Inventory Manager. The default is 3466.

Example

```
[MGR_RIM]
HTTP_HOST = localhost
HTTP_PORT = 3466
```

Table 48: MGR_RIM Values

Setting	Value as Installed	Default Value	Minimum Value	Maximum Value
HTTP_HOST	None	N/A	N/A	N/A
HTTP_PORT	N/A	3466	N/A	N/A

MGR_RMP

This section specifies the IP address/name and port of the Management Portal, if it has been enabled.

This section will be present only if the Management Portal option was selected during the Configuration Server installation.

Table 49: MGR_RMP Settings

Setting	Description
HTTP_HOST	The IP address of the Management Portal. If the Management Portal is on the same machine as the Configuration Server, specify *localhost*.
HTTP_PORT	The port of the Management Portal. The default is 3466.

Example

```
[MGR_RMP]
HTTP_HOST = localhost
HTTP_PORT = 3466
```

Table 50: MGR_RMP Values

Setting	Value as Installed	Default Value	Minimum Value	Maximum Value
HTTP_HOST	None	N/A	N/A	N/A
HTTP_PORT	N/A	3466	N/A	N/A

MGR_ROM

This section specifies the IP address/name and port of the Information Base, if it has been enabled.

Table 51: MGR_ROM Settings

Setting	Description
DSML_HOST	The IP address of the Information Base. If the Information Base is on the same machine as the Configuration Server, specify *localhost*.
DSML_PORT	The port of the Information Base. The default is 3468.
BASEDN	The domain name of the root for the computer class.

Example

```
[MGR_ROM]
DSML_HOST = localhost
DSML_PORT = 3468
BASEDN    = cn=machine
```

Table 52: MGR_ROM Values

Setting	Value as Installed	Default Value	Minimum Value	Maximum Value
HTTP_HOST	None	N/A	N/A	N/A
HTTP_PORT	N/A	3468	N/A	N/A
BASEDN	N/A	N/A	N/A	N/A

MGR_SMTP_MAIL

This section specifies SMTP-related parameters, including the ID of the Configuration Server, and the TCP port number of the Configuration Server to be used.

Table 53: MGR_SMTP_MAIL Settings

Setting	Description
DNS_SERVER	The Domain Name Service (**DNS**) server that is used to query the mail server address. Note: Specify a value for this setting to ensure that mail is delivered.
MAIL_DIR	Directory to spool and queue outgoing mail from the SMTP send Manager.
MAIL_TIMEOUT	Time-out interval (in seconds) for establishing communications with the mail server.
MAX_TIME_IN_SPOOL	The interval (in minutes) to wait before deleting undelivered mail in the spool. The default is 4320 minutes (3 days).
MGR_MAIL_ID	Mail ID for the Configuration Server. This setting takes the form of a fully qualified address (for example, **`radia@HP.com`**) and has a maximum length of 255 characters. The mail-receiving Configuration Server will reject mail addressed to any other user ID.
RETRY_INTERVAL	The interval (in seconds) to wait before re-attempting to deliver mail in the spool. The default is 300 seconds (5 minutes).
SMTP_PORT	Port on which to wait for incoming mail.

Examples

Intel Example:

```
[MGR_SMTP_MAIL]

DNS_SERVER          = 192.168.1.20

MAIL_DIR            = D:\MGR\MAIL

MAIL_TIMEOUT        = 60
```

```
MAX_TIME_IN_SPOOL   = 4320
MGR_MAIL_ID         = radia@HP.com
RETRY_INTERVAL      = 300
SMTP_PORT           = 25
```

UNIX Example:

```
[MGR_SMTP_MAIL]

DNS_SERVER          = 192.168.1.20
MAIL_DIR            = /radiaconfigsrvr/MAIL
MAIL_TIMEOUT        = 60
MAX_TIME_IN_SPOOL   = 4320
MGR_MAIL_ID         = radia@hp.com
RETRY_INTERVAL      = 300
SMTP_PORT           = 25
```

Table 54: MGR_SMTP_MAIL Values

Setting	**Value as Installed**	**Default Value**
DNS_SERVER	N/A	NONE
MAIL_DIR	as specified during installation	*current_directory*
MAIL_TIMEOUT	N/A	60
MAX_TIME_IN_SPOOL	4320 minutes (3 days)	4320 minutes (3 days)
MGR_MAIL_ID	as specified during installation	*sending_address*
RETRY_INTERVAL	300 seconds (5 minutes)	300 seconds (5 minutes)
SMTP_PORT	as specified during installation	25

Performance and Usage Considerations

- If storage capacity is an issue, decrease the MAX_TIME_IN_SPOOL value. This will decrease the length of time messages are retained.
- Increase the RETRY_INTERVAL value to use fewer system-processing resources.

- DNS_SERVER: For cross-platform compatibility, operating-system network settings for DNS servers are not used by the Configuration Server's mail delivery. Therefore, in order to ensure delivery of any e-mail (including license-warning messages) sent by the Configuration Server, DNS_SERVER must be defined.

MGR_SNMP

This section contains SNMP-related parameters that include where SNMP traps are to be sent and how to control the behavior of the built-in SNMP agent.

Table 55: MGR_SNMP Settings

Setting	**Description**
RUN_AS_EXTENSION	• If YES, NT's SNMP service is the primary SNMP agent and HP SNMP transactions are processed by a HP SNMP extension DLL. If so, SNMP_PORT, and SNMP_IP_ADDR are not used, since SNMP port access is handled by NT's SNMP service. The SNMP_COMMUNITY will be used for insertion into traps issued by the Configuration Server, but will not be used to authenticate GET and SET commands. • If NO, the Configuration Server will act as the primary SNMP agent, and SNMP_COMMUNITY should be specified, while SNMP_IP_ADDR and SNMP_PORT can be specified to override their defaults.
SNMP_COMMUNITY	This is a password that incoming SNMP transactions must match. It should be set to a character string. The string will be used as the SNMP community name by the agent. The default is public. Note: This keyword is effective only when RUN_AS_EXTENSION=NO.
SNMP_IP_ADDR	The TCP/IP address of the local network adapter card on which the agent is to receive SNMP transactions. The default is 0.0.0.0, meaning any adapter on the machine can be used. Note: This keyword is effective only when RUN_AS_EXTENSION=NO, and there are several adapters on the machine, and a specific adapter is to receive SNMP transactions.

Setting	Description
SNMP_MANAGER_IP _ADDR SNMP_MANAGER_IP _ADDR2 SNMP_MANAGER_IP _ADDR3	The SNMP Managers at the IP addresses specified here are authorized to issue GET and SET commands for variables supported by the agent. The SNMP Manager specified for SNMP_MANAGER_IP_ADDR is considered the primary SNMP Manager. This field is required because it specifies two things: 1) the receiving location of traps generated by the Configuration Server, and 2) the address of the authorized source for SNMP commands.
SNMP_MANAGER _PORT	This parameter is used to specify the remote TCP/IP port to which the Configuration Server sends its traps. The default is port 162.
SNMP_PORT	This parameter is used to specify the TCP/IP port on which the agent receives SNMP transactions. If not specified, the default is port 161. Note: This keyword is only effective when RUN_AS_EXTENSION=NO.
SNMP_SET _COMMUNITY	This parameter can be set to a character string. The agent will use this string as the SNMP community name when it is attempting to authorize SET commands. If this keyword is not specified, the community name given by SNMP_COMMUNITY is used for SET commands. Note: This keyword is only effective when RUN_AS_EXTENSION=NO.
SNMP_ZERROR _SEVERITY	This parameter is used to specify the severity of ZERROR instances to send as SNMP traps. The trap is sent when the Configuration Server adds an error instance to its ZERRORM for an error whose severity is greater than or equal to the value specified by this parameter. The parameter can be set to a positive value between 0 and 99; the default is 12.

Example

```
[MGR_SNMP]

RUN_AS_EXTENSION       = NO

SNMP_COMMUNITY         = public

SNMP_IP_ADDR           = 0.0.0.0
```

```
SNMP_PORT              = 162
SNMP_MANAGER_IP_ADDR   = 183.235.246.32
SNMP_ZERROR_SEVERITY   = 12
```

Table 56: MGR_SNMP Values

Setting	Value as Installed	Default Value
RUN_AS_EXTENSION	YES	NO
SNMP_COMMUNITY	public	public
SNMP_IP_ADDR	0.0.0.0	0.0.0.0
SNMP_MANAGER_IP_ADDR	N/A	N/A
SNMP_MANAGER_PORT	N/A	162
SNMP_PORT	161	161
SNMP_SET_COMMUNITY	N/A	N/A
SNMP_ZERROR_SEVERITY	12	12

Performance and Usage Considerations

There are no performance or usage issues with any of the SNMP parameters. If you do not start zsnmpmgr, you are not starting the agent, and are running one less task in the Configuration Server.

MGR_SSL

This section specifies the operational settings for an SSL Manager. For more information on configuring and using an SSL Manager, see Chapter 11,SSL Managers.

Table 57: MGR_SSL Settings

Setting	Description
CA_FILE	Sets the Certificate Authority's certificate. The CA certificate is usually stored in a file in Privacy Enhanced Mail (PEM) format. The SSL Manager needs a CA certificate to start up. If it's expired or corrupt it prevents the SSL Manager from starting up.
CERTIFICATE_FILE	Sets the Configuration Server or the server certificate. The certificate is usually stored in a file in PEM format. This value must be a valid and existing certificate file. The SSL Manager needs a certificate to start up. An expired or corrupt certificate will prevent the SSL Manager from starting up.
KEY_FILE	Sets the private key. The private key is usually stored in a file in PEM format. This value must be a valid and existing key file. The private key is usually stored in the same file as the server certificate; in which case, you don't have to specify any value for the key file.
KEY_PASSWORD	The password used to encrypt the private key, the one specified in the KEY_FILE keyword. This is usually needed if the private key is encoded. If the private key is not encoded, you don't need to specify this parameter.
SSL_PORT	Sets the port that the SSL Manager will listen on for client connects.
VERIFY_CLIENT	Specifies whether the Configuration Server should verify the client by requesting a certificate from the client. If the client doesn't have a certificate, the connection will be dropped.

Example

```
[MGR_SSL]

CA_FILE            = w:\openssl\ms\cacert.pem

CERTIFICATE_FILE   = w:\openssl\ms\srvcert.pem
```

```
KEY_FILE           = w:\openssl\ms\srvprvk.pem
KEY_PASSWORD       = violin
SSL_PORT           = 3456
VERIFY_CLIENT      = Y
```

Performance and Usage Considerations

- In order for an SSL Manager to function, in the MGR_ATTACH_LIST section, you must have CMD_LINE=(zsslmgr) RESTART=YES.
- The settings are placeholders, and, as such, are not used until an SSL add-on is installed.

MGR_STARTUP

This section specifies start-up information for the Configuration Server. In addition, if you are processing EDM and Radia clients with a single Configuration Server (Multi-Mode Configuration Server), you can define a different resolution starting point for each type of client. Use the EDM_STARTUP and RADIA_STARTUP sections for EDM and Radia clients, respectively.

Table 58: MGR_STARTUP Setting

Setting	Description
ALLOW_DUPLICATE_IP_ADDRESS	NO will cause the Configuration Server to reject a second log on if one from the same IP address is already active. YES will allow multiple concurrent IP connections from the same client IP address.
BYTE_LEVEL_DIFF	Turns on byte-level differencing during resolution.
MANAGER_TYPE	Type of Configuration Server. Values are DISTRIBUTED, SERVER, and STANDALONE.
MEMORY_TYPE	Reserved for future development. *Do not* alter this setting unless instructed to do so by HP Technical Support.
MGR_ID	A three-character, alphanumeric hexadecimal qualifier for the Configuration Server log file. This ID is also passed to the client as the ZOBJMID variable, and is used in the Radia database to identify the source Configuration Server in a Distributed Manager environment. Valid values are any combination of 001 to EFF.
MGR_NAME	Configuration Server name.
MGR_UUID	The Configuration Server's Universal Unique Identification Number (UUID). This 32-byte ID is generated automatically during the installation, and is used exclusively for licensing. Important Note: Do not change or delete this value.

Setting	Description
OBJECTID_ FORMAT	This setting determines which format will be used for generating object IDs. • Specify **1** to use the established algorithm. • Specify **2** to use the new algorithm. • If the value is absent or invalid, the default of **1** is assumed and the established algorithm is used. Note: For more information on this setting, see the section, OBJECTID_FORMAT, on page 105.
SHOW_VERINFO	Displays version information in the Configuration Server log at startup.
TASK_TYPE	Reserved for future development. *Do not* alter this setting unless instructed to do so by HP Technical Support.
TCP_PORT	Port to listen on for TCP/IP client connects. Note: This setting will be overridden by specifying a TCP/IP port in the MGR_ATTACH_LIST section. See MGR_ATTACH_LIST on page 37 for more details.
VERBOSE	Sends messages to the screen (stderr) when the Configuration Server starts up, verifies the license, verifies the Radia database, loads cache, and readies IP Managers (tcp and ssl) for processing and when shutting down.

Do not change MANAGER_TYPE, MEMORY_TYPE, MGR_ID, MGR_NAME, or TASK_TYPE after a successful installation, unless advised to do so by HP Technical Support.

Do not change or delete MGR_UUID unless instructed to do so by HP Technical Support.

Example

```
[MGR_STARTUP]

BYTE_LEVEL_DIFF          = NO

MANAGER_TYPE             = DISTRIBUTED

MGR_ID                   = 001

MGR_NAME                 = RCS
```

```
MGR_UUID              = ssed111454d6kgh4eh7g3md90f94d6k8
OBJECTID_FORMAT       = 2
SHOW_VERINFO          = YES
TASK_TYPE             = THREAD
TCP_PORT              = 3460
VERBOSE               = NO
```

Table 59: MGR_STARTUP Values

Setting	Value as Installed	Default Value	Minimum Value	Maximum Value
ALLOW_DUPLICATE _IP_ADDRESS	YES	NO	N/A	N/A
BYTE_LEVEL_DIFF	N/A	NO	N/A	N/A
MANAGER_TYPE	As specified	DISTRIBUTED	N/A	N/A
MEMORY_TYPE	SHARED	SHARED	N/A	N/A
MGR_ID	As specified	001	000	EFF (F00-FFF are reserved)
MGR_NAME	As specified	EDM	N/A	N/A
MGR_UUID	N/A	N/A	N/A	N/A
OBJECTID_FORMAT	1	1	N/A	N/A
SHOW_VERINFO	N/A	YES	N/A	N/A
TASK_TYPE	THREAD	THREAD	N/A	N/A
TCP_PORT	3460	1029	System dependent	64 KB
VERBOSE	N/A	NO	N/A	N/A

Performance and Usage Considerations

- If MGR_TYPE is not set to DISTRIBUTED, you will not be able to use the Distributed Configuration Server functionality.

- MGR_ID must be a three-character, alphanumeric, hexadecimal string. (Valid values are 001 to EFF.)
- If running the Configuration Server as a Windows Service, VERBOSE will be disabled.

OBJECTID_FORMAT

A new time-based format of object ID generation has been introduced. This new format eliminates the possibility of randomly generating a duplicate OID, because it prefixes each object ID with the letter A, B, C, E, or F—differing from the old format, which used only the letter D as a prefix.

Specifying the new object ID generation format affects new object ID generation only; it has no impact on existing OIDs.

When the time-based generator counter wraps, the new format will automatically start prefixing the OIDs with the next sequential letter.

EDM_STARTUP

This section specifies the resolution starting point for EDM clients in a Multi-Mode Configuration Server environment.

Table 60: EDM_STARTUP Settings

Setting	Description
START_CLASS	The class starting point of resolution for EDM clients.
START_DOMAIN	The domain starting point of resolution for EDM clients.

Example

```
[EDM_STARTUP]
START_DOMAIN  = ZSYSTEM
START_CLASS   = ZPROCESS
```

Table 61: EDM_STARTUP Values

Setting	Value as Installed	Default Value
START_DOMAIN	N/A	ZSYSTEM
START_CLASS	N/A	ZPROCESS

RADIA_STARTUP

This section specifies the resolution starting point for Radia clients in a Multi-Mode Configuration Server environment.

Table 62: RADIA_STARTUP Settings

Setting	Description
START_CLASS	The class starting point of resolution for Radia clients.
START_DOMAIN	The domain starting point of resolution for Radia clients.

Example

```
[RADIA_STARTUP]
START_DOMAIN  = SYSTEM
START_CLASS   = PROCESS
```

Table 63: RADIA_STARTUP Values

Setting	Value as Installed	Default Value
START_DOMAIN	N/A	SYSTEM
START_CLASS	N/A	PROCESS

MGR_TASK_LIMIT

This section specifies the maximum number of concurrent tasks allowed (including Configuration Server tasks and client connects), lines that can be written to the Configuration Server log per client, and resolutions allowed per client.

Table 64: MGR_TASK_LIMIT Settings

Setting	Description
TASK_HEAP_SIZE	This setting controls the heap size of a task (used in conjunction with MGR_POOLS). (MVS only)
TASKLIM	The maximum number of concurrent tasks for the Configuration Server. Note: These tasks include all Configuration Server system tasks, MGR_ATTACH_LIST tasks, and all tasks created by client connects initiated by clients and administrators. Once this limit is reached, the Configuration Server will accept no more connections.
TASK_LOG_LIM	The maximum number of lines, per client, that can be written to the Configuration Server log.
TASK_RESO_LIM	The maximum number of resolutions allowed per client.
TASK_STACK_SIZE	This setting defines how many variables can be used per task.

Example

```
[MGR_TASK_LIMIT]

TASK_HEAP_SIZE   = 0
TASKLIM          = 20
TASK_LOG_LIM     = 0
TASK_RESO_LIM    = 64000
TASK_STACK_SIZE  = 64000
```

Table 65: MGR_TASK_LIMIT Values

Setting	Value as Installed	Default Value	Minimum Value	Maximum Value
TASK_HEAP_SIZE	0	0	0	4 GB
TASKLIM	50	0	0	32 KB
TASK_LOG_LIM	0	100000 Lines	0	N/A
TASK_RESO_LIM	64000	64000 Resolutions	1	64000
TASK_STACK_SIZE	64000	64000	16384	64000

Do not change the TASK_STACK_SIZE setting in this section unless advised to do so by a member of HP Technical Support.

Performance and Usage Considerations

- The MGR_TASK_LIMIT settings should be coordinated with values in the MGR_NOTIFY and MGR_RETRY sections.
- TASK_HEAP_SIZE applies only to MVS.
- TASK_STACK_SIZE will affect the depth of resolution. Higher values will result in deeper resolution.

MGR_TIMEOUT

This section specifies how long the Configuration Server will wait for a request from a connected client before disconnecting that client due to inactivity (no requests/responses from the client).

Table 66: MGR_TIMEOUT Settings

Setting	Description
ADMIN_TIMEOUT	Timeout (in seconds) for administrator functions.
SEND_THROTTLE	Timeout (in milliseconds) before each send.
TIMEOUT_COMM	Communications (receive) timeout (in seconds).

Example

```
[MGR_TIMEOUT]
ADMIN_TIMEOUT= 0
SEND_THROTTLE= 0
TIMEOUT_COMM = 1800
```

Table 67: MGR_TIMEOUT Values

Setting	Value as Installed	Default Value	Minimum Value	Maximum Value
ADMIN_TIMEOUT	0 (never time out)	0 (never time out)	0	32767
SEND_THROTTLE	N/A	0	0	4 GB
TIMEOUT_COMM	1800 (seconds)	0 (never time out)	0	32767

Performance and Usage Considerations

- If processing resources are critical, increase these MGR_TIMEOUT values.
- The MGR_TIMEOUT settings should be coordinated with values in the MGR_RETRY section.

- The SEND_THROTTLE value can be overridden by bandwidth throttling variables sent up in a client object.
- TIMEOUT_COMM is the Configuration Server analog to the client ZTIMEO. If a connect is active, and the Configuration Server has not received any data from a client for the TIMEOUT_COMM value, the Configuration Server will terminate the session.

MGR_TPINIT

This section specifies packet sizes to send to clients.

Table 68: MGR_TPINIT Settings

Setting	Description
BUFTCP	TCP buffer size used for send/receive.
GET_REMOTE _HOST_NAME	Controls the Configuration Server's attempt to obtain the `host_name` from the DNS server. The default is NO. Note: Do not set to YES if the Configuration Server is not in a dynamic TCP/IP environment (such as DNS and DHCP).
MAXREC	Maximum record size.

⚠ *Do not* change any settings in this section unless advised to do so by a member of HP Technical Support.

Example

```
[MGR_TPINIT]

BUFTCP                  = 12288

GET_REMOTE_HOST_NAME    = NO

MAXREC                  = 6144
```

Table 69: MGR_TPINIT Values

Setting	Value as Installed	Default Value
BUFTCP	12288	12288
GET_REMOTE_HOST_NAME	NO	NO
MAXREC	6144	6144

Performance and Usage Considerations

- The buffer size reflected in the MGR_TPINIT settings should be the same for the Configuration Server and clients.
- Any buffer-size setting in the MGR_TPINIT section should only be changed in coordination with equivalent changes to clients and after having been directed to do so by a member of HP Technical Support.
- Do not use GET_REMOTE_HOST_NAME if the Configuration Server is not in a dynamic TCP/IP environment. This will cause the Configuration Server to expend unnecessary processing time attempting to associate the remote host name.

  If GET_REMOTE_HOST_NAME=YES, the Configuration Server obtains the remote host name using standard library calls. The IPNAME is received on the Configuration Server (via DNS) and stored in ZMASTER.ZIPNAME, which is stored in the appropriate domain of the PROFILE file.

  The remote host name will appear in each associated line of the Configuration Server log in place of the IP address.

MGR_TRACE

This section contains a list of keywords (settings) that you can specify in order to control and influence diagnostic logging for the Configuration Server. All diagnostic output produced by TRACE settings is written to the active Configuration Server log. To activate a TRACE keyword, specify YES. To de-activate a TRACE keyword, specify NO.

TRACE keywords specified in this section are invoked at Configuration Server initialization, and remain in effect:

- until they are changed by altering the MGR_TRACE setting and restarting the Configuration Server.
- while the Configuration Server is running, using the Console.
- until a specified REXX overrides the setting.
- unless a ZCVT value overrides the setting.

The trace settings that are in effect at Configuration Server initialization are displayed at the beginning of the log.

The value for each setting is evaluated in the order in which it is presented in the `edmprof` file. The results can be non-intuitive. For example:

- If ALL=YES is the first setting specified, and each following settings are specified as NO, the effect is to turn off *tracing*.
- If ALL=YES is the last setting specified, all tracing will be active.
- If ALL=NO is the last setting, all tracing is turned off.

Table 70: MGR_TRACE Settings

Setting	Description
ADMIN	Traces ADMIN transaction flow.
ADMPROM	Not used.
ALL	Turns on all other traces.
ALLOC	Traces file allocations.
AUDIT	Traces audit file activity.

Setting	Description
BUFF	Traces data buffers (without transformation).
CMPR	Traces data compression.
COMM	Traces data stream buffers.
COMMCBS	Traces communications control block (CCB) activity.
COMMDATA	Traces data communications.
COMMRPLS	Traces communications control blocks (CCBS).
CONFIG	Traces configuration file activities.
DATA	Traces data buffers to or from the client.
DES	Traces encryption/decryption.
DMA	Traces Distributed Configuration Server activity.
DSCOMP	Traces data stream compression. (MVS only)
ENQDEQ	Traces serialization activity (enqueues/dequeues).
EXPL	Traces data transformation (explode).
FILE	Traces file I/O.
IMPL	Traces data transformation (implode).
LOOKASID	Traces cache activity for classes/instances.
METHOD	Traces Configuration Server method execution/return codes.
NOTIFY	Traces notify processing.
OBJCRC	Traces object CRC processing.
OBJRES	Traces object resolution (very detailed).
OBJRES1	Traces object resolution (medium detail).
OBJRESO	Traces high-level object resolution flow (light detail).
OBJXFER	Traces object transfer.
PASSWORD	Traces passwords.
POOLMISS	Traces memory pool allocation.
PROFILE	Traces profile database activity.
PROMOTE	Traces file promotion.

Setting	Description
RESOURCE	Traces resource file activity.
REXX	Traces REXX environment.
REXXOFF	Suppresses all REXX activity.
STORAGE	Traces storage in conjunction with the MGR_LOG's STORAGE_INTERVAL setting.
STATS	Traces statistics.
SUBST	Traces variable substitution.
TCP	Traces TCP/IP activity.
TEST	Reserved.
TRAN	Traces data transformation buffers. (MVS only)
VAR	Traces the variable references.
VARSTG	Traces variable processing storage usage.
VARSUB	Traces variable substitution activity.
VSAM	Traces VSAM I/O. (MVS only)
VSAMDATA	Traces VSAM data. (MVS only)
VSAMRPLS	Traces VSAM request parameter list (RPL). (MVS only)
YEAR2000	Traces a database's Year 2000 compliance.

Example

```
[MGR_TRACE]

ADMIN      = NO

ADMPROM    = NO

ALLOC      = NO

AUDIT      = NO

BUFF       = NO

CMPR       = NO

COMM       = NO

COMMCBS    = NO
```

```
COMMDATA   = NO
COMMRPLS   = NO
CONFIG     = NO
DATA       = NO
DES        = NO
DMA        = NO
DSCOMP     = NO
ENQDEQ     = NO
EXPL       = NO
FILE       = NO
IMPL       = NO
LOOKASID   = NO
METHOD     = NO
NOTIFY     = NO
OBJCRC     = NO
OBJRES     = NO
OBJRES0    = NO
OBJRES1    = NO
OBJXFER    = NO
PASSWORD   = NO
POOLMISS   = NO
PROFILE    = NO
PROMOTE    = NO
RESOURCE   = NO
REXX       = NO
REXXOFF    = NO
STATS      = YES
STORAGE    = NO
SUBST      = NO
TCP        = NO
TEST       = NO
TRAN       = NO
```

```
VAR         = NO
VARSTG      = NO
VSAM        = NO
VSAMDATA    = NO
VSAMRPLS    = NO
YEAR2000    = NO
ALL         = YES
```

Performance and Usage Considerations

- Turning on trace flags generates a large number of Configuration Server log messages. This will degrade the performance of the Configuration Server due to the disk I/O load. However, this might be necessary at times for problem resolution. Ensure that the Configuration Server log is properly configured.
- Tracing can be clustered in order to troubleshoot a particular aspect of Configuration Server operations, while simultaneously preserving an appropriate level of logging activity. For example, turning on flags, such as OBJCRC, OBJRES, OBJRES1, OBJRES0, and OBJXFER, can help identify problems that might be occurring during object resolution. Likewise, CPIC, DATA, TCP, and TRAN focus on communications activities. (Note that some of these traces pertain to specific protocols.) Special purpose trace flags include DMA, NOTIFY, REXX, and METHOD.
- STORAGE=YES can be used to produce a message in the log each time a storage allocation is made and freed. This can be used to tune the pool and to understand the Configuration Server's use of dynamic memory.
- ALL=YES does not affect the REXXOFF, VSAMDATA, or VSAMRPLS trace settings.

MVS User's Note

You can refresh options set from the MVS parmlib with the following command:

```
F  jobname,REFRESH,memname
```

or just

```
F  jobanme,R,memname
```

where *memname* is a member in the parmlib with which the Configuration Server was started. This parmlib is the dataset on the PARMLIB DD card in the Configuration Server JCL. The Configuration Server's trace options, message filtering, routing options, and pools options will be updated from any corresponding MGR_TRACE, MGR_MESSAGE_CONTROL, and MGR_POOLS sections found in that member.

ALL=YES and VSAM=YES include the VSAMAPI trace. This is because the VSAMAPI trace is no longer as large as it was. It issues only one single-line message on entry to, and one on exit from, a VSAM module. Since it shows the FDCI key, the VSAM requests for it are quite useful in understanding a program's database access.

The other VSAM traces, VSAMRPLS and VSAMDATA, are not set by ALL=YES and should be used only when specifically requested. Note that VSAMRPLS has an alias name of VSAMCB.

MGR_TRANSLATION (MVS only)

This section specifies customized ASCII-to-EBCDIC and EBCDIC-to-ASCII translation tables.

Table 71: MGR_TRANSLATION Settings

Setting	Description
A2E XX	Specifies conversion from ASCII to EBCDIC. The hex values (yy) will replace the standard default translation table values beginning at offset xx.
E2A XX	Specifies conversion from EBCDIC to ASCII. The hex values (yy) will replace the standard default translation table values beginning at offset xx.

Notes

- Offsets from 00 to FF can be specified in any order.
- Any individual character or groups of characters can be replaced in the default table.
- In the cases of duplicate and overlapping settings, the last specified value will be used.
- Translations should always be symmetrical (for example, ASCII 30 translates to EBCDIC F0 and EBCDIC F0 translates back to ASCII 30).

Table 72: MGR_TRANSLATION Values

Setting	Value as Installed	Default Value
A2E XX	See example.	See example.
E2A XX	See example.	See example.

Example

```
[MGR_TRANSLATION]* The default ASCII-to-EBDCIC translation table.

A2E 00 = 00010203372D2E2F1605250B0C400E0F
A2E 10 = 101112133C3D322618193F27221D351F
A2E 20 = 405A7F7B5B6C507D4D5D5C4E6B604B61
A2E 30 = F0F1F2F3F4F5F6F7F8F97A5E4C7E6E6F
A2E 40 = 7CC1C2C3C4C5C6C7C8C9D1D2D3D4D5D6
A2E 50 = D7D8D9E2E3E4E5E6E7E8E9BAE0BBB06D
A2E 60 = 79818283848586878889919293949596
A2E 70 = 979899A2A3A4A5A6A7A8A9C06AD0A107
A2E 80 = 040608090A1415171A1B1C1E20212324
A2E 90 = 28292A2B2C303133343638393A3B3E41
A2E A0 = 42434445464748494A51525354555657
A2E B0 = 58596263646566676869707172737475
A2E C0 = 767778808A8B8C8D8E8F909A9B9C9D9E
A2E D0 = 9FA0AAABACAEAFB0B1B2B3B4B5B6B7B8
A2E E0 = B9BABBBCBEBFCACBCCCDCECFDADBDCDD
A2E F0 = DEDFE1EAEBECEDEEEFFAFBFCFDFEFF40

* The default EBDCIC-to-ASCII translation table.

E2A 00 = 000102038009817F8283840B0C0D0E0F
E2A 10 = 101112138586088718198889 8A1D8B1F
E2A 20 = 8C8D1C8E8F0A171B9091929394050607
E2A 30 = 95961697981E99049A9B9C9D14159E1A
E2A 40 = 209FA0A1A2A3A4A5A6A7A82E3C282B7C
E2A 50 = 26A9AAABACADAEAFB0B121242A293B5E
E2A 60 = 2D2FB2B3B4B5B6B7B8B97C2C255F3E3F
E2A 70 = BABBBCBDBEBFC0C1C2603A2340273D22
E2A 80 = C361626364656667686 9C4C5C6C7C8C9
E2A 90 = CA6A6B6C6D6E6F707172CBCCCDCECFD0
E2A A0 = D17E737475767778797AD2D3D45BD5D6
E2A B0 = 5ED8D9DADBDCDDDEDFE05B5DE35DE4E5
E2A C0 = 7B414243444546474849E6E7E8E9EAEB
E2A D0 = 7D4A4B4C4D4E4F505152ECEDEEEFF0F1
E2A E0 = 5CF2535455565758595AF3F4F5F6F7F8
E2A F0 = 30313233343536373839F9FAFBFCFDFE
```

Performance and Usage Considerations

- All Configuration Servers in a Distributed Configuration Server group must use the same MGR_TRANSLATION.

Once a MGR_TRANSLATION is established, use extreme care in changing it; adding to or updating anything in the Radia database could cause data to become un-readable or un-viewable.

MGR_USERLOG

This section specifies the logging directory and logging options for the user logging facility.

Table 73: MGR_USERLOG Settings

Setting	Description
ACTIVATE	Activate user log at Configuration Server startup. Values are YES and NO.
DIRECTORY	Fully qualified directory path where the user log is written.
FLUSH_SIZE	The number of bytes between automatic flushes of operating-system buffers for Configuration Server log file.
MESSAGE_WIDTH	The maximum width (in characters) of the messages in the user log.
PIPE_SIZE	The maximum amount (in bytes) of log messages that can be queued by the Configuration Server logging facility while the log file is busy. When this value is reached, any task that issues a log message will freeze until the pipe starts emptying.
THRESHOLD THRESHHOLD (both spellings accepted)	Maximum number of messages that will be written to a log before automatically switching to the next log. When the limit is reached, new log files are created. Specify a negative number to overwrite the log file when the limit is reached.

Examples

UNIX Example

```
[MGR_USERLOG]

ACTIVATE        = NO

DIRECTORY       = /radiaconfigsrvr/log

FLUSH_SIZE      = 256

MESSAGE_WIDTH   = 256

PIPE_SIZE       = 1000000

THRESHOLD       = 5000000
```

Intel Example

```
[MGR_USERLOG]

ACTIVATE         = NO

DIRECTORY        = D:\MGR\LOG

FLUSH_SIZE       = 256

MESSAGE_WIDTH    = 256

PIPE_SIZE        = 1000000

THRESHOLD        = 5000000
```

MVS Example

```
[MGR_USERLOG]

ACTIVATE       = YES

DIRECTORY      = //DDN:EDMULOGA

FLUSH_SIZE     = 1000

MESSAGE_WIDTH  = 500

PIPE_SIZE      = 16834

THRESHOLD      = 150000
```

Table 74: MGR_USERLOG Values

Setting	**Value as Installed**	**Default Value**	**Minimum Value**	**Maximum Value**
ACTIVATE	NO	NO	N/A	N/A
DIRECTORY	`/edmmgr/log`	*`current_directory`*	N/A	N/A
FLUSH_SIZE	256 bytes	100000 bytes	1	N/A
MESSAGE_WIDTH	256 characters	90	80	N/A
PIPE_SIZE	1000000 bytes	65535 bytes	1	N/A
THRESHHOLD	-5000000 bytes	5000 bytes	1	N/A

Performance and Usage Considerations

- Increasing the FLUSH_SIZE will enhance performance, but will delay messages flushed to the log file.
- Increase MESSAGE_WIDTH if log messages are being truncated.

- When modifying parameters in this section as they relate to memory or disk use, be sure that the maximum amount of memory or storage space is available.

User Log Naming Conventions

The user log names conform to the following operating system-specific conventions.

ISO is the *International Standards Organization*.

Configuration Server ID is the value of the `edmprof` file's setting MGR_STARTUP.MGR_NAME.

Windows

— The *standard* log-naming format is the user log prefix (**nvd**), followed the user designation (**ur**), and the **Configuration Server ID**:

```
nvdur001.log
```

— The *log-switch* format is similar to the standard but an **us** (rather than **ur**) is added to the prefix, and the ISO-formatted date and time (each preceded by an underscore) appended:

```
nvdus001_20050427_083357.log
```

— The *log-wrap* (dump) format is similar to the standard but sees the **r** being replaced by a **d**:

```
nvdud001.log
```

UNIX

— The *standard* log-naming format is the user log prefix (**nvd**), followed by the **Configuration Server ID**, then an underscore, and the **user** designation:

```
nvd001_user.log
```

— The *log-switch* format is similar to the standard but an **s** is added to the prefix, and the ISO-formatted date and time (each preceded by an underscore) appended:

```
nvds001_user_20050427_075835.log
```

— The *log-wrap* (dump) format sees **dump** being added to the **user** designation:

```
nvd001_userdump.log
```

OBJECT_SIZES

The OBJECT_SIZES section must be manually added to the `edmprof` file.

Additionally, this section must be added to the `edmprof` file in order for the Configuration Server self-tuning tool to operate properly.

This section accommodates specifying the number of heaps and the heap size for Radia database objects that are being created on the Configuration Server as in-storage Radia database objects.

- These values affect only the Configuration Server self-tuning tool; they have no impact on other Configuration Server processing.
- The format for specifying these values is:

  OBJECT_NAME = *heap_size*,*number_of_heaps*

For example:

`ZCONTROL = 512,100000`

In this example, whenever an object named ZCONTROL is created in the Configuration Server as an in-storage object, its initial allocation will be for 100,000 heaps of 512 bytes each.

`ZERROR = 400,1000`

In this example, whenever an object named ZERROR is created in the Configuration Server as an in-storage object, its initial allocation will be for 1000 heaps of 400 bytes each.

For a detailed description of how these settings affect the Configuration Server self-tuning tool, see Configuration Server Self-Tuning Tool on page 136.

Example

```
[OBJECT_SIZES]

FILE      = 1536,2000
RELEASE   = 420,1536
WBEMAUDT  = 6000,100
MSIFEATS  = 512,100
ZOBJSTAT  = 512,270
PRODUCT   = 1024,200
ZREQDATA  = 200,100
ASERVICE  = 3000,60
ZSERVICE  = 4000,60
```

```
ZREQNEWI  = 150,15
MSIPROPS  = 1024,30
ZERROR    = 512,10
UMFLTCRI  = 1536,23
WBEM      = 1024,20
APPEVENT  = 1024,5
SAPSTATS  = 1024,10
SAP       = 1024,10
CLISTATS  = 512,5
UMFLTRUL  = 600,4
DESKTOP   = 2000,3
MSI       = 2000,5
REGISTRY  = 1536,5
PATH      = 1024,3
ZCONFIG   = 4000,1
ZMASTER   = 3000,1
TIMER     = 1536,2
```

RCS_TUNING_CONTROL

The RCS_TUNING_CONTROL section must be manually added to the `edmprof` file.

Additionally, this section must be added to the `edmprof` file in order for the Configuration Server self-tuning tool to operate properly.

This section provides a mechanism to override the default values that are specified in the Configuration Server self-tuning tool and these are described in the following table. (For a look at the Configuration Server self-tuning tool, see Configuration Server Self-Tuning Tool on page 136.)

Table 75: RCS_TUNING_CONTROL Settings

Setting	Description
MAXIMUM_MEG_ CACHE	A four-digit integer value that specifies the maximum number of MBs that are to be allocated to CONTENT CACHE. Note: This setting provides protection when the size of a class (or classes) would exceed the virtual storage of the process space. In the case that one class exceeds this value, that class is forced to be cached on first reference (=Y,N).
MINIMUM_MEG_ CACHE	A four-digit integer value that specifies the minimum number of MBs that are to be allocated for CONTENT CACHE.
MAXIMUM_SHARED _MEMORY_SEGMENTS	A two-digit integer value that specifies the *`maximum_number_of_segments`* times the *`largest_shared_memory_size`*. Note: This value must be less than 15 because the value cannot exceed the size of the process space (generally, 2 GB [2*1024*1024*1024]).
MAXIMUM_SHARED _MEMORY_SIZE	A four-digit integer value that specifies the maximum number of MBs that are be allocated to any single CONTENT CACHE segment. Notes: If the aggregate size of CONTENT CACHE exceeds this value, it should be allocated as multiple segments—each less than this value in size. This value anticipates the UNIX requirements for tuning where shared memory sizes are a kernel parameter and not easily changed.

Setting	Description
MAXIMUM_CLASS_ INSTANCES	A ten-digit integer value that specifies the maximum number of instance elements that can exist in a class that might be cached at startup. Note: Any class with more instances will be marked (=Y,N) as "cache on first reference."
MINIMUM_ INSTANCES	A five-digit integer value that specifies the minimum number of instance elements for which ICACHE and CONTENT CACHE are intended to be sized.
UPDATE_RCS_ STARTUP	A YES-NO toggle for updating the startup of the Configuration Server. The default (NO) leaves the `edmprof` file unchanged and creates the file `EDMPROF_TUNED.DAT_`.

Example

```
[RCS_TUNING_CONTROL]

MAXIMUM_MEG_CACHE       = 1500
MINIMUM_MEG_CACHE       = 50
MAXIMUM_SHARED_MEMORY_SEGMENTS = 6
MAXIMUM_SHARED_MEMORY_SIZE = 32
MAXIMUM_CLASS_INSTANCES= 100000
MINIMUM_INSTANCES       = 10000
UPDATE_RCS_STARTUP      = YES
```

Table 76: RCS_TUNING_CONTROL Values

Setting	Value as Installed	Default Value	Minimum Value	Maximum Value
MAXIMUM_MEG _CACHE	None	1500	None	None
MINIMUM_MEG _CACHE	None	200	None	None
MAXIMUM_SHARED _MEMORY_SEGMENTS	None	6	None	None
MAXIMUM_SHARED _MEMORY_SIZE	None	384	None	None

Setting	**Value as Installed**	**Default Value**	**Minimum Value**	**Maximum Value**
MAXIMUM_CLASS _INSTANCES	None	100000	None	None
MINIMUM_INSTANCES	None	50000	None	None
UPDATE_RCS_STARTUP	None	NO	None	None

SECTION_DELIMITERS

This section is not a true section of the Configuration Server `edmprof` file. It is used only to specify the symbols that will be used to delimit section names in the `edmprof` file.

⚠ If used, SECTION_DELIMITERS must be the very first entry and the first non-blank line in the `edmprof` file. If it is not, the Configuration Server will not be able to read-in the license string and will not start up.

Table 77: SECTION_DELIMITERS Settings

Setting	Description
SECTION_ DELIMITERS	The type of characters to be used to separate sections of the `edmprof` file. The format is: SECTION_DELIMITERS=xy, where *x* and *y* are the left and right delimiters. The options are: **[]**, **()**, **<>**, and **{}**.

Examples

```
SECTION_DELIMITERS  = <>

<MGR_LICENSE>

LICENSE_STRING                              = FFCDAB

SECTION_DELIMITERS                          = []


[MGR_LICENSE]
LICENSE_STRING   = FFCDAB
```

Table 78: SECTION_DELIMITERS Values

Setting	Value as Installed	Default Value
SECTION_DELIMITERS	N/A	[]

3 Managing Configuration Server Processing

At the end of this chapter, you will:

- Have a better understanding of the HP OpenView Configuration Server Using Radia (Configuration Server) processing.
- Know how to customize the processing flow using REXX programs and methods.

Configuration Server Operations

Configuration Server operations has three basic phases:

- Startup
- Processing Requests
- Shutdown

Configuration Server startup is initiated by icon, command line, console, or control panel, depending on the platform and installation. In the Startup phase, the Configuration Server initializes ZTOPTASK. Using the Configuration Server `edmprof` file to provide the working parameters for configuring the Configuration Server, ZTOPTASK then starts the Task Manager. After the various tasks specified in MGR_ATTACH_LIST are activated, the Configuration Server is ready to process requests.

A request can be sent to the Configuration Server as a system command, an in-bound object, a client connect, an administrative transaction, or a Distributed Configuration Server command. During the Processing Requests phase, the Configuration Server performs the requests (tasks) that are submitted to it. There are four types of tasks: System tasks, Client tasks, Admin tasks, and Distributed Configuration Server tasks. System tasks are Configuration Server functions, Client tasks are client requests, Admin tasks pertain to System Explorer and Publisher operations, and Distributed Configuration Server tasks are Distributed Configuration Server functions.

The type of task is shown in each line of the Configuration Server log, as shown below.

```
NVD1069I 13:49:52 [208.244.225.166 /16B] Radia Client
Max size of local memory allocated        : 230805
```

-
  ```
  NVD8115I 13:50:30 [ztoptask/17B]   System Task
  REXX Method <D:\DEV\MGR\REXX\ZSHUTDWN> with parms <%s> started
  at <13:50:30:574>
  ```

Like startup, Configuration Server shutdown can be initiated in a number of ways. In the Shutdown phase, the Configuration Server performs some basic housekeeping, then essentially reverses the startup flow. The Configuration Server is now down, allowing you to run a backup, use the database utilities, apply maintenance, or perform other operating system tasks.

Customizing Configuration Server Processing

The values in your Configuration Server `edmprof` file allow you to customize its overall configuration. There are two main ways of customizing the flow of Configuration Server processing:

- Configuration Server REXX programs and
- Configuration Server methods.

The primary difference is that REXX programs are pre-configured, while the methods can be inserted anywhere in the processing flow.

This chapter describes the Configuration Server REXX programs and Configuration Server methods. An overview of the methods is presented in Configuration Server Methods, on page 158. Also Appendix A, Configuration Server Methods details each method, providing an example of its use, a description, its associated parameters, and possible return codes.

Configuration Server REXX Programs

REXX Directories

The Configuration Server installation creates two REXX-related directories,

```
\radia\configurationserver\rexx
```

and

`\radia\configurationserver\rexx\NOVADIGM`.

The `\rexx` directory is empty, and the `\rexx\NOVADIGM` directory contains all the HP-related REXX programs.

The `\radia\configurationserver\rexx` directory can be renamed to further distinguish it from the HP REXX directory.

Customizing the HP REXX Programs

The HP REXX programs can be customized in order to adapt to, and enhance, various computing environments. To customize any of these REXX programs, copy them from the `\rexx\NOVADIGM` directory to the `\rexx` directory, then modify them as needed.

Do not make any changes to the REXX programs in the `\rexx\NOVADIGM` directory. Doing so will adversely affect the performance of the Radia database.

There are two reasons that HP REXX programs have to be copied to the `\rexx` directory prior to being modified:

- If a REXX is customized and left in the `\rexx\NOVADIGM` directory, the customizations could be lost (overwritten) if a database update is applied, thereby affecting the behavior and execution of the database operations.
- During processing, the database reads the `\rexx` directory first. Therefore, place any customized REXXs in that directory.

Event Points

There are eight event points at which the Configuration Server issues calls to ten major REXX programs. Table 79 below lists these REXX programs and the points at which they are called.

Table 79: Configuration Server REXX Programs

REXX Name	When Called (Event Point)
ZSTARTUP	Configuration Server Startup
ZPCACHE	Configuration Server Startup
ZINIT	Configuration Server Startup
ZTASKSTA	Task Start
ZTASKEND	Task End

REXX Name	When Called (Event Point)
ZNFYxSTA	Notify Start
ZNFYxEND	Notify End
ZLOGSWCH	Log Switch
ZLOGWRAP	Log Wrap
ZSHUTDOWN	Configuration Server Shutdown

REXX Programs

ZSTARTUP

This is called just before the Configuration Server is enabled to accept and process client connections. ZSTARTUP does not pass any parameters to the REXX, nor does it accept any information from this REXX. It can be used to perform user-defined specific processing prior to allowing connections to be initiated.

ZPCACHE

This can be called after cache is loaded during Configuration Server startup.

ZINIT

This is called during Configuration Server startup. It runs REXXs and tests for certain conditions. If the conditions are not met (return code = 16), Configuration Server startup will be halted.

This is not configurable. Consult HP Technical Support before using this REXX.

ZTASKSTA

This is called when each connection is first accepted. It is passed a single parameter that contains the protocol level identifier for the client.

ZTASKEND

This is called when each connection has ended, while storage and objects associated with the connection are still available. It is passed a single parameter that contains multiple sub-parameters that can be parsed and that are position dependent.

> If the Management Portal is installed on a machine other than that which houses the Configuration Server, ZTASKEND on the Configuration Server must be modified (as below) so that it specifies the IP address for the Management Portal.
>
> For more information, refer to the *Installation and Configuration Guide for the HP OpenView Management Portal Using Radia* (*Management Portal Guide*).

- Connect termination reason.
- User ID of the current connection.
- Total number of object instances, of any type, processed during this connection.
- Total number of object instances, of any type, resulting from resolution.
- The maximum depth of transient objects processed in any single instance resolution.
- Count of communications protocol sends and receives originating from this connection.
- Total number of characters transmitted from the Configuration Server to the client during this connection (non-compressed count).
- Total number of files transferred from the Configuration Server to the client during this connection.

ZNFYxSTA

This is called at the beginning of Notify processing for each client being notified. The x defines the type of Notify (T for TCP/IP and D for Dial-up). The set of parameters passed includes:

- UINF=<value1>
  user info passed via EDMMPUSH method. If no information is provided, COMMON will be set as a value.
- RETRS=<value2>
  number of retries for this destination.
- STATUS=<value3>
  RETRY, SUCCESS, FAILURE.

- MSG=<value4>
  message describing the result of notification.

ZNFYxSTA can generate a return code that controls the execution of the Notify, RC value 1956 (RC_SKIP_NOTIFY). If this is set, it will cause the Notify request to be written for retry without execution of current notification.

ZNFYxEND

This is called at the end of Notify processing for each client being notified. The x defines the type of Notify (T for TCP/IP and D for Dial-up). The set of parameters passed includes:

- UINF=<value1>
  user info passed via EDMMPUSH method. If no information is provided, COMMON will be set as a value.
- RETRS=<value2>
  number of retries for this destination.
- STATUS=<value3>
  RETRY, SUCCESS, FAILURE.
- MSG=<value4>
  message describing the result of notification.

ZNFYxEND can generate a return code that controls the execution of the Notify, RC value 1955 (RC_NEVER_RETRY). If this is set, it will prevent the Notify request from being rescheduled for retry.

ZLOGSWCH

This can be called when log switch occurs (a new log file is created) to insert a user-defined command (for example, zip the old log file and save it).

ZLOGWRAP

This can be called when log wrap occurs (the log file is reused) to insert a user-defined command (for example, zip the old log file and save it).

ZSHUTDWN

This is called just before the Configuration Server shuts down.

Configuration Server Self-Tuning Tool

The Configuration Server self-tuning tool enables Configuration Server instances to automate the tuning of the `edmprof` file's MGR_CACHE and MGR_CLASS sections.

Two `edmprof` file sections, OBJECT_SIZES (see page 125) and RCS_TUNING_CONTROL (see page 127), are needed in order for this tool to function properly.

This tool does not dynamically alter the MGR_CACHE and MGR_CLASS sections while the Configuration Server is active; rather, while the Configuration Server is shutdown, it examines the database and creates an updated dataset that can be compared to the existing (master) `edmprof` file, and possibly used to automatically replace it.

The self-tuning tool can be configured to rename the master `edmprof` file and replace it with the newly generated one, although this is not the default behavior (which is to overwrite the master `edmprof` file). If the non-default (rename-and-replace) option is selected, the only active `edmprof` file settings that should differ between the two `edmprof` files are those in the MGR_CACHE and MGR_CLASS sections (because these settings are generated at shutdown and are based on the size of the database and the number of instances).

The MGR_CLASS Section

If the master `edmprof` file does not have a MGR_CLASS section, the self-tuning tool attempts to content-cache all the database instances. It does this by calculating the amount of space that will be necessary to support content-caching all the instances, and by generating the caching directives for each class that is to be set for caching in its entirety at Configuration Server initiation and/or cache refresh.

If the MGR_CLASS section in the master `edmprof` file has caching directives for individual classes, they will be preserved. However, if these caching directives are intended for internally generated objects, or are received by the Configuration Server from a session partner, they will not be preserved in the automatically generated values because the new MGR_CLASS section is based on the Radia database.

OBJECT_SIZES

The OBJECT_SIZES section will be preserved (as it was in the master `edmprof` file) in the new `edmprof` file, and will result in additional MGR_CLASS section entries, in the form of:

```
"NONDB." OBJECT_NAME = N,N,heap_size,number_of_heaps
```

These entries will be listed first in the updated MGR_CLASS section.

They are followed by the MGR_CLASS entries from the master `edmprof` file, which are listed based on the Radia processing preference in which priority is placed on processing the ZSERVICE and PACKAGE instances in order to facilitate the initial catalog resolution, followed by component instances, and then non-component instances.

Revise and Overwrite

The default behavior for the Configuration Server self-tuning tool is to create a new `edmprof` file in the directory that contains the master (or original) `edmprof` file. That format for naming the file is:

`EDMPROF_TUNED.DAT_REVISED_`*`YYYYMMDD_HH_MM_SS_uuuuuu`*

Where:

- `EDMPROF_TUNED.DAT_REVISED_` is a literal value,
- *`YYYYMMDD`* is the year, month, and date of the revise action, and
- *`HH_MM_SS_uuuuuu`* is the local hour, minute, second, and microsecond at which the new file is established.

Only one file matching the filter EDMPROF_TUNED.DAT_REVISED_* will be retained in the directory—that file will be the most recent one that was created by this tool.

Rename and Replace

If the non-default behavior of replacing the master `edmprof` file with the newly created one is selected, the `edmprof` file that existed when the Configuration Server was started will be renamed to:

`EDMPROF.DAT_REPLACED_`*`YYYYMMDD_HH_MM_SS_uuuuuu`*

Where:

- *`YYYYMMDD`* is the year, month, and date of the replace action, and
- *`HH_MM_SS_uuuuuu`* is the local hour, minute, second, and microsecond at which the file is replaced.

Up to 15 files matching the filter EDMPROF.DAT_REPLACED_* will be retained in the directory that contains the `edmprof` file.

The 15 files are the 14 most-recent datasets, and the oldest `edmprof` file (so that an original `edmprof` file is retained for reference).

All of these files will be in the directory in which the original `edmprof` file was found. Therefore, the account under which the Configuration Server is executing must have update and create authority for that directory.

The MGR_CACHE Section

In addition to the updated MGR_CLASS section, a new MGR_CACHE section gets generated and replaces the section in the input `edmprof` file.

While processing the original input `edmprof` file, the existing MGR_CLASS and MGR_CACHE sections are ignored, while all other sections are copied in their entirety. The newly created MGR_CACHE and MGR_CLASS sections will be appended to the end of the copied input `edmprof` file.

If, in the original `edmprof` file, there are comments preceding the MGR_CLASS and/or MGR_CACHE sections, then, in the newly generated `edmprof` file, the comments might appear to be dislocated. However, since these are comments, their position does not impact the Configuration Server startup.

Reporting Files

A reporting file (`DB_EDMPROF_REPORT.TXT`) will be created in the Configuration Server log directory. It documents the MGR_CACHE and MGR_CLASS settings and the fact that the newly created values represent the actual amount of required storage. Only the most recent copy of each file will be retained, and each execution of the self-tuning tool will delete prior copies.

The self-tuning tool will attempt to allocate enough ICACHE and CONTENT CACHE in order to accommodate a database growth of 30%. However, the results might change due to default sizes for minimum number of database instances, maximum amount of virtual storage to commit to content cache, and the impact of very large classes.

- A file named `EDMPROF_SECTION.DAT` is also created. It contains a replica of the MGR_CLASS and MGR_CACHE sections that were merged into the master `edmprof` file and replaced the existing MGR_CACHE and MGR_CLASS sections when the self-tuning tool started.
- A file (`LIST_PREFIX.CSV`) is created, but currently not used. When implemented, this will contain a snapshot of database domain- and class-information, including the size and number of instances in each class at shutdown.

To implement the Configuration Server self-tuning tool

1 Install the new ZSHUTDWN REXX method into the `rexx` directory.

If a previous, customized version of ZSHUTDWN REXX exists in the `rexx` directory, be sure to copy any customizations from it to the new one. Then delete the previous one, or move it down to the `rexx\NOVADIGM` directory.

2 Shutdown the Configuration Server.

(After the Configuration Server has shutdown, in the location that housed the `edmprof` file when the Configuration Server was started, there should be either: an updated `edmprof` file along with the original (but now renamed) `edmprof` file, or the revised `edmprof` file that overwrote the original.)

3 Restart the Configuration Server.

HP REXX Extensions

Five HP REXX extensions perform specific functions. These functions are object resolution, and the getting and setting of objects and variables. The REXX extensions are:

- EDMRESO
- EDMGET (objects)
- EDMGETV
- EDMSET (objects)
- EDMSETV

EDMRESO

(*`resolvestring`*)

Function: To perform object resolution.

Returns: `OK: rc=0, eval-string=obj.var value`

`err: rc>0`

Example: `rc = EDMRESO('PRIMARY.SYSTEMX.USER.USER1(EDMSETUP)')`

EDMGET

(*`objname,heapnumber`*)

Function: To get an object.

➤ `heapnumber` is an optional parameter, and defaults to 0, meaning the current heap.

Returns: `OK: rc=0, a REXX Stem Variable named objname (that depicts the object) is created.`

`err: rc>0`

Example: `rc = EDMGET('TESTREXX',1)`

EDMGETV

(*objname,varname,heapnumber,resolveflag*)

Function: To get a variable from an object.

➤ `heapnumber` and `resolveflag` are optional parameters. `heapnumber` defaults to 0, meaning the current heap.

`resolveflag` is a one-character alphanumeric field that instructs the EDMGETV REXX to perform a substitution on the variable. `resolveflag` defaults to 0, which means no substitution.

Returns: `OK: rc=0, eval-string=obj.var value`

`err: rc>0`

Example: `rc = EDMGETV('ZMASTER',ZOS,1)`

EDMSET

(*objname,heapnumber*)

Function: To set an object.

➤ `heapnumber` is an optional parameter, and defaults to 0, meaning the current heap.

Returns: `OK: rc=0`

`err: rc>0`

Example: `rc = EDMSET('TESTREXX',1)`

EDMSETV

(*objname,varname,value,heapnumber*)

Function: To set a variable in an object.

heapnumber is an optional parameter, and defaults to 0, meaning the current heap.

Returns: `OK: rc=0`

`err: rc>0`

Example: `rc = EDMSETV('ZMASTER',ZOS,NT,1)`

If you are in a REXX, you can change the communications timeout of the current Configuration Server task by specifying:

`EDMSETV('SESSION','TIMEOCOM',nnnn,1)`

where *nnnn* is the value (in seconds), and 1 is the *heapnumber*.

ZCVT and ZTCBG

These two objects assist you in determining current values for Radia database tasks and connects, whether or not they are running. ZCVT performs a global function—determining values of tasks and connects for the entire Radia database, whereas ZTCBG determines the values that are relevant to the specified task or connect.

The ZCVT and ZTCBG objects (`objname`) work with the EDMGET, EDMGETV, EDMSET, and EDMSETV REXX extensions, as described in the previous section.

Table 80 and Table 81 (on page 153) present the variables associated with ZCVT and ZTCBG, respectively.

ZCVT Table of Variables

Table 80: ZCVT Variables

Variable Name	Variable Type	Description
VERMAJ	USHORT	Configuration Server major version number
VERMIN	USHORT	Configuration Server minor version number
VERREVNO	USHORT	Configuration Server revision number
VERREVLE	UCHAR	Configuration Server revision letter
VERBLDNO	ULONG	Configuration Server build number

Variable Name	**Variable Type**	**Description**
OSNAME	STR	Configuration Server’s operating system
TOLOGONS	LONG	Total number of logons to the Configuration Server
TRCOMMCB	FLAG	COMM Trace value
TRCOMDAT	FLAG	COMM trace value
TRDSCOMP	FLAG	DSCOMP trace value
TRTEST	FLAG	TEST trace value
TRDYNALO	FLAG	ALLOC trace value
TRVARSTG	FLAG	VARSTG trace value
TRDBCB	FLAG	ODBCBS trace value
TRDBDATA	FLAG	ODBDATA trace value
TRAUDIT	FLAG	AUDIT trace value
TRPROFIL	FLAG	PROFLE trace value
TRRESRCE	FLAG	RESOURCE trace value
TRPROMOT	FLAG	PROMOTE trace value
TRCONFIG	FLAG	CONFIG trace value
TRMETHOD	FLAG	METHOD trace value
TRCPIC	FLAG	CPIC trace value
TRADMIN	FLAG	ADMIN trace value
TRRESLV0	FLAG	OBJRES0 trace value
TRBINDFL	FLAG	DATA trace value
TRDAXFRM	FLAG	TRAN trace value
TR3270BU	FLAG	BUFF trace value
TRCOMM	FLAG	COMM trace value
TRFILPRO	FLAG	FILE trace value
TROBJXFR	FLAG	OBJXFER trace value
TROBJCRC	FLAG	OBJCRC trace value

Variable Name	Variable Type	Description
TRREXX	FLAG	REXX trace value
TRVARS	FLAG	VAR trace value
TRSUBST	FLAG	SUBST trace value
TRDESENC	FLAG	DES trace value
TRCOMPR	FLAG	CMPR trace value
TROBJRES	FLAG	OBJRES trace value
TRIMPLOD	FLAG	IMPL trace value
TREXPLOD	FLAG	EXPL trace value
TRLASIDE	FLAG	LOOKASID trace value
TRENQUE	FLAG	ENQDEQ trace value
TRSTATS	FLAG	STATS trace value
TRRESLV1	FLAG	OBJRES1 trace value
TRTCPIP	FLAG	TCP trace value
TRADMPRM	FLAG	ADMPROM trace value
TRNOTIFY	FLAG	NOTIFY trace value
TRSESBLK	FLAG	SESSBLK trace value
TRSTORAG	FLAG	Storage trace value
TRY2K	FLAG	YEAR2000 trace value
TRDMA	FLAG	DMA trace value
TRVSAPI	FLAG	VSAM trace value
TRVSCB	FLAG	VSAMRPL trace value
TRVSDATA	FLAG	VSAMDATA trace value
TRREXOFF	FLAG	REXXOFF trace value
SHTINDIC	UCHAR	Shut-down indicator
SHTLGMGR	UCHAR	Log Configuration Server shut-down indicator
REXALLOC	LONG	REXX allocate count

Variable Name	Variable Type	Description
TOPTSKID	LONG	ZTOPTASK ID
LOGMGRID	LONG	ZLOGMGR ID
ULGMGRID	LONG	ZULOGMGR ID
CLKMGRID	LONG	ZCLKMGR ID
TSKMGRID	LONG	ZTASKMGR ID
MGRTYPE	UCHAR	MANAGER_TYPE value
TASKTYPE	UCHAR	TASK_TYPE value
MEMTYPE	UCHAR	MEMORY_TYPE value
DBLUDATE	STR	Date of last Radia database update
DBLUTIME	STR	Time of last Radia database update
DBLCKCNT	USHORT	Radia database lock count
DBSTATUS	UCHAR	Radia database status
BYTELEVD	UCHAR	Byte level differencing on value
DNYDUPIP	FLAG	ALLOW_DUPLICATE_IP_ADDRESS value
TRUSTEDP	FLAG	Authorized trusted partner value
TSOSRVCE	FLAG	TSO service value
LICERROR	FLAG	License violation error value
ACALLADM	FLAG	ALWAYS_CALL_ZADMIN value
OKDUPINS	FLAG	ALLOW_DUPLICATE_INSTANCES value
QUITASKS	FLAG	Quiesce task
QUITRANS	FLAG	Quiesce transaction
LALLTCP	FLAG	TCP/IP logons allowed value
REXDISAB	FLAG	REXX off
MODNAMLO	FLAG	SHOW_MODULE value
MODVERLO	FLAG	SHOW_VERINFO value
TCPRHNAM	FLAG	GET_REMOTE_HOST_NAME value

Variable Name	Variable Type	Description
HISTORY	UCHAR	History file value
ACCESSA	UCHAR	MGR_ACCESS ADMIN value
ACCESSC	UCHAR	MGR_ACCESS CONSOLE value
ZTIME	STR	Time in HH:MM
ZTIME24	STR	Time in HH:MM on 24-hour clock
ZAMPM	STR	AM or PM value
ZDATE	STR	MM/DD/YYYY date form
ZDATEDMY	STR	DD/MM/YYY date form
ZMONTH	STR	Short value for month (e.g., Jan)
ZMONTHLNG	STR	Long value for month (e.g., January)
ZDATEJUL	STR	Julian date
ZDATEYMD	STR	YYYY/MM/DD form for sorting
ZDAT2YMD	STR	YYYY/MM/DD form for sorting – full MM, DD value
MGRID	STR	Configuration Server ID (MGR_ID)
MGRNAME	STR	Configuration Server name (MGR_NAME)
DOMAIN	STR	Configuration Server domain name (DOMAIN)
STDOMANE	STR	EDM start domain (START_DOMAIN)
STCLASSE	STR	EDM start class (START_CLASS)
STDOMANR	STR	RADIA start domain (START_DOMAIN)
STCLASSR	STR	RADIA start class (START_CLASS)
STDOMANA	STR	ATM start domain (START_DOMAIN)
STCLASSA	STR	ATM start class (START_CLASS)
DBASE	STR	Radia database used at startup (DBASE)
AGENT	STR	Agent ACB (AGENT)
TCPPORT	STR	TCP PORT value (TCP_PORT)
TCPUSRID	STR	USERID of TCP/IP address space

Variable Name	Variable Type	Description
TORESO	LONG	Number of object resolutions
DEEPRESO	LONG	Deepest object resolution
TOOBJI	LONG	Number of inbound objects
TOOBJO	LONG	Number of outbound objects
ZERMXWRN	SHORT	Max number of warnings heaps
ZERMXERR	SHORT	Max number of error heaps
TOCOMP	LONG	Number of times compression done
TOCOMPI	LONG	Compression total bytes in
TOCOMPO	LONG	Compression total bytes out
TODCMPI	LONG	Decompression total bytes in
TODCMPO	LONG	Decompression total bytes out
TODCMP	LONG	Number to times decompression done
COMPSEED	LONG	Compression seed
TODBGETS	LONG	Number of GET operations
TODBPUTS	LONG	Number of PUT operations
TODBADDS	LONG	Number of ADD operations
TODBDELE	LONG	Number of DELETE operations
TOFILEIO	LONG	Global file i/o counts
TOFALLOC	LONG	Global file allocations
TSKLIMAX	SHORT	Max TASKLIM
TSKLIMIT	SHORT	TASKLIM (TASKLIM)
TSKLHARD	SHORT	HARD TASKLIM (TASKLIM_HARD)
TSKLSOFT	SHORT	SOFT TASKLIM
TSKLDLTA	SHORT	TASKLIM Delta
MAXRESCL	LONG	Max number of resource per client
MAXRESAL	LONG	Max number of resource all clients

Variable Name	Variable Type	Description
TOMETBIN	LONG	Number of methods run – ASM and C
TOMETREX	LONG	Number of methods run – REXX
TOXNRCVD	LONG	Number of transaction received
TOXNRJCT	LONG	Number of transaction rejected
PLSTATUS	PTR	Pointer to disable Pools heap
PLGLBHEP	FLAG	Global/local pools in used flag
PLCONTIG	FLAG	Pools are allocated contiguous flag
PLEXPSIZ	ULONG	Pool expansion size
PLCSCRED	ULONG	Current storage credit
PLPSGUAR	ULONG	Percent Storage Guard
PLCSCUSH	ULONG	Current storage cushion
PLMSCUSH	ULONG	Minimum storage cushion
PLPOLHWM	ULONG	Largest polar bytes used
TSKPRIV	ULONG	Task private size
TSKSTSIZ	ULONG	Task stack size
TSKSTHWM	ULONG	Task stack size overuse HWM
TSKHPSIZ	ULONG	Task heap size
TSKHPHWM	ULONG	Task heap size overuse HWM
TIMEOCOM	LONG	TIMEOUT_COMM
TIMEONCM	LONG	TIMEOUT_NCOMM
TIMEOADM	LONG	ADMIN_TIMEOUT
TIMEODMA	LONG	DMA_TIMEOUT
RETRYBUS	LONG	BUSY_RETRY
RETRYDIS	LONG	DISABLE_RETRY
DSCOMPI	LONG	Data stream compression total bytes in
DSCOMPO	LONG	Data stream compression total bytes out

Variable Name	Variable Type	Description
DSCOMPT	LONG	Number of times data stream compression done
SNDTHRTL	ULONG	SEND_THROTTLE
TIMEONFT	SHORT	NFYT_TIMEOUT
TIMEONFS	SHORT	NFYS_TIMEOUT
TIMEONFD	SHORT	NFYD_TIMEOUT
OBJNMASK	STR	Object name mask for traces
VARNMASK	STR	Variable name mask for trace
LOGLNTSK	ULONG	TASK_LOG_LIM
MAXRSTSK	ULONG	TASK_RESO_LIM
MAXREC	LONG	MAXREC
BUFTCP	LONG	BUFTCP
LOGDIR	STR	DIRECTORY
LOGTHRES	LONG	THRESHOLD
LOGLNCNT	LONG	Log file line count
LOGSTINT	LONG	STORAGE_INTERVAL
LOGFSIZE	LONG	FLUSH_SIZE
LOGMWIDT	USHORT	MESSAGE_WIDTH
LOGMPREF	STR	MESSAGE_PREFIX
LOGMDATE	UCHAR	MESSAGE_DATE
LOGMLDEL	UCHAR	MESSAGE_DELIMITER
LOGMRDEL	UCHAR	MESSAGE_DELIMITER
LOGPSIZE	LONG	PIPE_SIZE
LOGFLUSH	FLAG	Log flush
LOGSWITC	FLAG	Log switch
LOGELOFF	FLAG	DISABLE_NT_EVENT_LOGGING
LOGSLOFF	FLAG	DISABLE_SNMP_TRAP_LOGGING

Variable Name	Variable Type	Description
LOGSFREQ	STR	SWITCH_TOD
LOGFFREQ	STR	FLUSH_INTERVAL
LOGBPIPE	LONG	Log bytes in pipe
ULGDIR	STR	USERLOG DIRECTORY
ULGTHRES	LONG	USERLOG THRESHOLD
ULGLNCNT	LONG	USERLOG LINECOUNT
ULGFSIZE	LONG	USERLOG FLUSH_SIZE
ULGMWIDT	USHORT	USERLOG MESSAGE_WIDTH
ULGACTIV	FLAG	USERLOG ACTIVATE
ULGPSIZE	LONG	USERLOG PIPE_SIZE
ULGFLUSH	FLAG	USERLOG flush
ULGSWITC	FLAG	USERLOG switch
SIMTSKPC	LONG	Simulation TASKS_PER_CONNECT
STATPATH	STR	Stats path
STATINTV	LONG	Stats interval
MTHPATH	STR	METHOD_PATH
MTHMLIMI	LONG	LOG_LIMIT
MTHTIMEO	LONG	TIMEOUT
DBPATH	STR	DBPATH
DBPPRIM	STR	PRIMARY DB path
DBPSECO	STR	SECONDARY DB path
DBPPROF	STR	PROFILE DB path
DBPRESO	STR	RESOURCE DB path
DBPHIST	STR	HISTORY DB path
DBPNOTI	STR	NOTIFY DB path
REXXPATH	STR	REXX_PATH

Variable Name	Variable Type	Description
SYSPATH	STR	System path
DMASPATH	STR	DMA_STAGE_PATH
MGRSETFI	STR	PROFILE path
EXPTPATH	STR	EXPORT_PATH
USRPATH1	STR	USER_PATH1
USRPATH2	STR	USER_PATH2
USRPATH3	STR	USER_PATH3
USRPATH4	STR	USER_PATH4
USRPATH5	STR	USER_PATH5
USZTCBGS	SHORT	Number of used ZTCBGs
CACSEGS	USHORT	CACHE_SEGMENTS
CACSIZE	ULONG	CACHE_SIZE
CACMAXE	ULONG	Max cache entries
CACSTATS	USHORT	CACHE_STATS
CACFULL	USHORT	CACHE full
CACCLOSE	USHORT	CACHE closed
ICASIZE	ULONG	ICACHE_SIZE
ICACLOSE	UCHAR	ICACHE closed
SMMAILDR	STR	MAIL_DIR
SMMGRMID	STR	MGR_MAIL_ID
SMLOCHST	STR	Local host NAME
SMDNSSRV	STR	DNS_SERVER
SMSMTPRT	USHORT	SMTP_PORT
SMSPLCNT	ULONG	Spooled mail count
SMTIMEO	USHORT	MAIL_TIMEOUT
SMMAXSPL	USHORT	MAX_TIME_IN_SPOOL

Variable Name	Variable Type	Description
SMRETRYI	USHORT	RETRY_INTERVAL
SNPORT	USHORT	SNMP_PORT
SNLOGPRT	USHORT	SNMP_LOGGER_PORT
SNMGRPRT	USHORT	SNMP_MANAGER_PORT
SNRUNEXT	FLAG	RUN_AS_EXTENSION
SNIPADD	STR	SNMP_IP_ADDR
SNMIPAD	STR	SNMP_MANAGER_IP_ADDR
SNMIPAD2	STR	SNMP_MANAGER_IP_ADDR2
SNMIPAD3	STR	SNMP_MANAGER_IP_ADDR3
SNCMNT	STR	SNMP_COMMUNITY
SNSTCMNT	STR	SNMP_SET_COMMUNITY
SNZERSEV	SHORT	SNMP_ZERROR_SEVERITY
ALSLOTS	LONG	ATTACH_LIST_SLOTS
ALVINTVL	LONG	VERIFY_INTERVAL
ALRLIMIT	LONG	RESTART_LIMIT
DMSECMTH	STR	SECURITY_METHOD
DIAINTVL	ULONG	DIAGNOSTIC_INTERVAL
DIADBYTE	ULONG	DIAGNOSTIC_MIN_DB_BYTES
DIALBYTE	ULONG	DIAGNOSTIC_MIN_LOG_BYTES
DBVERIFY	STR	VERIFY_DEPTH
DBAUTOFIX	FLAG	DB_AUTOFIX
METHDLLS	FLAG	Run methods as DLLS
ACTSKMON	ULONG	Number of monitors
ACTSKCON	ULONG	Number of consoles
ERREMAIL	STR	UserEmailErrorsTo
DBEEMAIL	FLAG	DBERROR e-mail

Variable Name	Variable Type	Description
DBESHTDN	FLAG	DBERROR SHUTDOWN
DBESNMP	FLAG	DBERROR SNMP
POLCYSVR	STR	Status of the Policy Server (enabled/disabled)
RIM	STR	Status of the Inventory Manager (enabled/disabled)
RMP	STR	Status of the Management Portal (enabled/disabled)

ZTCBG Table of Variables

Table 81: ZTCBG Variables

Variable Name	Variable Type	Description
AUDFLAG	FLAG	Audit trace flag
TSKNAME	STR	Name of this task or client
TSKSTTIM	STR	Time when task started
TSKSTDAT	STR	Date when task started
TSKLSTCO	STR	Time of last communication transaction
FREEMAIN	FLAG	Return storage on last free call
SHORTSTO	FLAG	Retry due to Short on storage
ZTERMINI	FLAG	Terminal task initiated
STOCRCUR	ULONG	Current storage credit
STOCRHEP	ULONG	Heap credit
STOCRSTK	ULONG	Stack credit
STOCRPVT	UONG	Private credit
STOCRPOO	ULONG	Zpools credit
STOCRCUS	ULONG	Credit cushion
MTHNAME	STR	Child name
MTHLIBNA	STR	Method library name
MTHLIBHA	ULONG	Method library handle

Variable Name	**Variable Type**	**Description**
MTHLIBEN	ULONG	Method library entry point
MTHTHPRM	PTR	Method thread parameter pointer
TASKID	ULONG	Unique ID of this task
TASKPAR	ULONG	Unique ID of the parent task
USERID	STR	Task's user ID
TASKTYPE	STR	TASK TYPE
DEEPRESO	LONG	DEEPSET OBJECT RESOLUTION
OBJSRESO	LONG	NUMBER OF OBJECTS RECEIVED
OBJSRECV	LONG	NUMBER OF OBJECTS RECEIVED
OBJSSENT	LONG	NUMBER OF OBJECTS SENT
TRCOMDAT	FLAG	COMM TRACE FLAG
TRDSCOMP	FLAG	DSCOMP TRACE FLAG
TRTEST	FLAG	TEST TRACE FLAG
TRDYNALO	FLAG	ALLOC TRACE FLAG
TRVARSTG	FLAG	VARSTG TRACE FLAG
TRAUDIT	FLAG	AUDIT TRACE FLAG
TRPROFIL	FLAG	PROFILE TRACE FLAG
TRRESRCE	FLAG	RESOURCE TRACE FLAG
TRPROMOT	FLAG	PROMOTE TRACE FLAG
TRCONFIG	FLAG	CONFIG TRACE FLAG
TRMETHOD	FLAG	METHOD TRACE FLAG
TRCPIC	FLAG	CPIC TRACE FLAG
TRADMIN	FLAG	ADMIN TRACE FLAG
TRRESLVL	FLAG	OBJRES0 TRACE FLAG
TRBINDFL	FLAG	DATA TRACE FLAG
TRDAXFRM	FLAG	TRAN TRACE FLAG

Variable Name	**Variable Type**	**Description**
TR3270BU	FLAG	BUFF TRACE FLAG
TRCOMM	FLAG	COMM TRACE FLAG
TRFILPRO	FLAG	FILE TRACE FLAG
TROBJXFR	FLAG	OBJXFER TRACE FLAG
TROBJCRC	FLAG	OBJCRC TRACE FLAG
TRREXX	FLAG	REXX TRACE FLAG
TRVARS	FLAG	VAR TRACE FLAG
TRSUBST	FLAG	SUBST TRACE FLAG
TRDESENC	FLAG	DES TRACE FLAG
TRCOMP	FLAG	CMPR TRACE FLAG
TROBJRES	FLAG	OBJRES TRACE FLAG
TRIMPLOD	FLAG	IMPL TRACE FLAG
TREXPLOD	FLAG	EXPL TRACE FLAG
TRLASIDE	FLAG	LOOKASID TRACE FLAG
TRENQUE	FLAG	ENQDEQ TRACE FLAG
TRSTATS	FLAG	STATS FLAG
TRRESOL1	FLAG	OJBRES1 TRACE FLAG
TRTCPIP	FLAG	TCP TRACE FLAG
TRADMPRM	FLAG	ADMPROM TRACE FLAG
TRNOTIFY	FLAG	NOTIFY TRACE FLAG
TRSESBLK	FLAG	SESSBLK TRACE FLAG
TRSTORAG	FLAG	STORAGE TRACE FLAG
TRY2K	FLAG	YEAR2000 TRACE FLAG
TRDMA	FLAG	DMA TRACE FLAG
TRVSAPI	FLAG	VSAM TRACE FLAG
TRVSCB	FLAG	VSAMRPLS TRACE FLAG

Variable Name	Variable Type	Description
TRVSDATA	FLAG	VSAMDATA TRACE FLAG
TRREXOFF	FLAG	REXXOFF TRACE FLAG
STBBSENT	FLAG	BB SENT FLAG
STNOSNAP	FLAG	NO SNAP FLAG
STCOWAIT	FLAG	WAIT FOR COMM OP FLAG
STPWDVER	FLAG	EDATS SF ORDERS OK FLAG
STTIMOUT	FLAG	TIMEOUT IN PROGRESS FLAG
STFORTER	FLAG	FORCED TREMINATION FLAG
STPARSES	FLAG	PAR SESS PARTNER FLAG
STSESEST	FLAG	SESSION ESTABLISHED FLAG
STSESTER	FLAG	SESSION TERMINATED FLAG
STSESLST	FLAG	SESSION LOST FLAG
STTSKABN	FLAG	TASK ABENDED FLAG
STSESSND	FLAG	SEND FLAG
STNOPDS	FLAG	NO PARSE R/S DATA STREAM FLAG
STTSKINA	FLAG	TASK BEING INACTIVATED FLAG
STTIMSND	FLAG	TIME SENT FLAG
STNODSCO	FLAG	NO DS COMPRESSION FLAG
STABTRES	FLAG	ABORT OBJECT RESOLUTION FLAG
STABTLEG	FLAG	ABORT OBJECT RESOLUTION FLAG
STSTRLOG	FLAG	IN LOGGER FLAG
STEOT	FLAG	EOT FLAG
STNOSUB	FLAG	NO SUBSTITUTION FLAG
STDRAINS	FLAG	DRAIN FLAG
STCONSOL	FLAG	CONSOLE IS RUNNING
STHRDLCK	FLAG	SYSTEM HARDLOCKED FLAG

Variable Name	**Variable Type**	**Description**
STSSRESO	FLAG	SINGEL SERVICE RESOLUTION FLAG
STUSEMET	FLAG	USER METHOD RUNNING FLAG
STMSGLIM	FLAG	LOG MSG LIMIT REACHED FLAG
STMETMES	FLAG	METHOD MSG LIMIT REACHED FLAG
STREXMET	FLAG	REXX METHOD RUNNING FLAG
TOCOMP	LONG	NUMBER OF TIMES COMPRESSION DONE
TOCOMPI	LONG	COMPRESSION TOTAL BYTES IN
TOCOMPO	LONG	COMPRESSION TOTAL BYTES OUT
TODCOMPI	LONG	DECOMPRESSION TOTAL BYTES IN
TODCOMPO	LONG	DECOMPRESSION TOTAL BYTES OUT
TODCOMP	LONG	NUMBER OF TIME DECOMPRESSION DONE
TODBGETS	LONG	NUMBER OF GETS
TODBPUTS	LONG	NUMBER OF PUTS
TODBADDS	LONG	NUMBER OF ADDS
TODBDELE	LONG	NUMBER OF DELETES
TOFILEIO	LONG	FILE I/O COUNT
TOFALLOC	LONG	FILE ALLOCATION COUNT
TOMTHBIN	LONG	NUMBER OF METHODS RUNA – ASM AND C
TOMTHREX	LONG	NUMBER OF METHOS RUN – REXX
REMIPNAM	STR	IP NAME OF REMOTE CLIENT

Configuration Server Methods

Overview

A method is a program or procedure that can be packaged and exchanged as an object, specifically as an instance of the METHOD (ZMETHOD in EDM) class. By connecting an instance of this class to another class instance, you can specify where and when that procedure will run. You can also run a method from a REXX script, enabling you to execute methods outside of the object resolution process. The following is an example of the format used to execute a method in this way.

```
ADDRESS EDMLINK EDMMCMPR 'ZTEST'
```

EDMLINK is a method that allows you to process other methods. It returns the return code of the invoked method. The format for EDMLINK is:

```
ADDRESS EDMLINK (methodname) '(Parameter associated
with Method)'
```

MVS User's Note

`ADDRESS` must be replaced with **`CALL`**, as in:

`CALL EDMLINK EDMMCMPR 'ZTEST'`

Configuration Server methods allow you to manipulate in-storage objects and database entities (database components) at the system (Configuration Server) level as opposed to the client or workstation objects. Radia database components are the entities (files, domains, classes, instances, and variables) that reside in the Radia database. In-storage objects are used or created during the object resolution process.

Appendix A, Configuration Server Methods describes each method, with parameters, examples, and return codes.

Table 82 lists the Configuration Server methods that affect in-storage objects, database entities, or neither.

Table 82: Methods Affecting In-Storage Objects or Radia Database Entities

Method Name	Affects
EDMMAILQ	In-Storage Objects

Method Name	**Affects**
EDMMALLO	N/A
EDMMCACH	In-Storage Objects
EDMMDALO	N/A
EDMMDB	Radia Database Entities
EDMMGNUG	N/A
EDMMOLOG	N/A
EDMMPUSH	N/A
EDMMPUTD	N/A
EDMMRPRO	Radia Database Entities
EDMMSQLG	N/A
EDMMSQLP	N/A
EDMMULOG	N/A
EDMSIGN	N/A
EDMSIGNR	N/A
ZADMIN	N/A
ZDCLASS	Radia Database Entities
ZDELINS	Radia Database Entities
ZDELOBJS	In-Storage Objects
ZDELPROF	Radia Database Entities
ZEXIST	Radia Database Entities
ZGETPROF	Radia Database Entities
ZNFYT	N/A
ZOBJCMPR	In-Storage Objects
ZOBJCOPY	In-Storage Objects
ZOBJDELI	In-Storage Objects
ZOBJDELV	In-Storage Objects

Method Name	Affects
ZOBJSORT	In-Storage Objects
ZPROMANY	Radia Database Entities
ZPUTHIST	In-Storage Objects
ZPUTPROF	In-Storage Objects
ZSIMRESO	In-Storage Objects
ZTOUCH	Radia Database Entities
ZVARDEL	In-Storage Objects
ZVARGBL	In-Storage Objects
ZVARLOG	In-Storage Objects
ZXREF	In-Storage Objects

Methods are often used in conjunction with one another to achieve a purpose. For example, you can use the EDMMDOBJ method to delete an in-storage object, and then execute EDMMCOPY to copy an object, giving it the original object name.

Methods must be connected to other class instances at an appropriate point to achieve a desired result. For example, you do not want to delete an instance before it is used in object resolution, or create an instance if it will be immediately overwritten.

The default file and domain used by some methods can be specified in the DBASE and DOMAIN values of the MGR_STARTUP setting of the Configuration Server `edmprof` file.

Method Naming Standards

The standard that is used to name the methods is dissectible, enabling you to ascertain the method's use. All method names are structured as detailed in Table 83.

Table 83: Configuration Server Methods Naming Standard

Symbol	Definition
EDM	Identifies the method as an HP method.

Symbol	Definition
M	Identifies the method as a Configuration Server method.
DOBJ	Represents an abbreviation of the function for which the method can be used, in this case – Delete Object.

"Must-Run" Methods

When you configure methods to run during the resolution process, you expect specific outcomes. If one method is intended to work in conjunction with another, or have a direct effect on the correct outcome of the resolution, the entire resolution process might depend on, first the existence of, then the successful launching of, this method.

You can designate a method as "Must Run," which means that, before continuing with the resolution process, the Configuration Server will determine if the method exists and can be run. If it is not found or cannot be launched, the return code for the method will be set to 16 (Abort Resolution). The resolution will then be halted.

If you do not designate a method as "Must Run," the Configuration Server does not recognize it as being essential to the outcome of the resolution and will continue processing based on the resulting return code. The only indications that the method was not processed are the return codes (as shown in messages in the log) and the result of the resolution path.

To configure a method as "Must Run," insert the ZMUSTRUN variable in the METHOD instance and set the value to YES. (The default value for ZMUSTRUN is NO.) You can also establish a specific message to be returned by inserting the MSGONERR variable in the ZMETHOD instance.

If no value is specified for MSGONERR, the following message will appear: "`CONFIGURATION UNCHANGED! UNABLE TO DETERMINE NEW CONFIGURATION.`"

4 Notifying Clients

At the end of this chapter, you will:

- Know about the different ways to invoke the Notify function.
- Know how to configure multiple Notify Managers.
- Know how notification retries are established.

An Overview of Notify

The notify function enables the initiation and execution of programs on a client desktop from another location, and is usually, initiated by the Configuration Server. However, a client that is configured as an administrator can also initiate notify processing requests.

The uses of the notify function vary from initiating client connections via notify to ad-hoc notifies to reboot a single machine. It is a powerful, flexible tool that can be used for starting non-Radia-specific processes (such as, restart, backup, and so forth) on Radia-managed desktops.

In a typical manual connect scenario, the client initiates the client-connect process to receive the resources configured for that desktop. By using notify, the Configuration Server can contact a client and request that it connect or, alternately, accomplish some other task defined for that desktop, at any time.

Generally, this process depends on the Configuration Server having a reliable method by which to identify and contact each client. This method of contact might the IP address that was used at the last client-Configuration Server connect in a static IP address environment, or the host name of the client machine in a DHCP environment. Any of these methods can be used to identify the client as the target of a Configuration Server-initiated notify. It is by this identifier, as well as the communications environment being used, that the Configuration Server knows how to contact the client. Once communication is established with the client, the Configuration Server can initiate processes to perform a variety of functions. The client identifier must be unique and reliable in order to predict which client will receive the notify.

In order for TCP/IP to receive notify messages from the Configuration Server, the receiving client desktops must have a notify receive daemon running. The UNIX, Windows NT, Windows 2000, Windows Server 2003, Windows XP, Windows 9*x*, and Macintosh platforms must run the notify receive daemon and the client notify receive programs in order to receive any incoming messages.

The EDM and Radia client notify receive programs are EDMEXECD and RADEXECD, respectively.

Notify currently supports TCP/IP and e-mail. The sender and receiver must be using the same communications protocol if the program is to execute properly.

The following section describes how the Configuration Server can notify clients to initiate the client-connect process.

Notify and the Client-Connect Process

Software distribution is typically discussed in terms of push and pull. This refers to the concepts of pushing software out from a central location to a client, or the client pulling software in from a central location. The major difference between the push and pull scenarios is the point at which the distribution activity is initiated—the server or the client. HP supports both models by offering numerous options that are used to define how, when, and where the distribution process is initiated. Notify provides users with the means to configure and implement a push scenario.

The HP distribution process—the client-connect process—is part of an entire configuration process during which the client connects to the Configuration Server to determine what its configuration should be. This connection process is comprised of a series of programs that execute on the client to perform comprehensive, continuous configuration management. Many of these programs communicate with the Configuration Server to obtain required information, while other programs perform strictly local processing.

Radia supports three basic connect types, as follows:

- **Manual Connect**
  The user invokes the client connect process by choosing the appropriate icon. This process can be defined as a pull.
- **Timed Connect**
  A timer process that runs on the client executes the client connect process at a predetermined date and time. This operation can also be defined as a pull.

  For more information on these client-initiated connects, refer to the *Software Manager Guide*.
- **Notify Connect**
  The client is notified from a central control point to perform the connection. The notify process is a push.

Notify can be executed only if the “notifier” (usually the Configuration Server) and the “notifyee” support the same communications protocol.

When to Use Notify

In addition to forcing a Radia client to connect to a Configuration Server, the notify function can be used in the following ways:

- Using notification for other purposes
  The notify functionality can be used for emergency distribution of files outside of a client-connect process; for initiating some event on the client (such as switching versions); and for collecting debugging information about a connect failure.
- Instead of EDMTIMER
  EDMTIMER is used to deploy applications at specific time intervals. It is often set to execute during non-peak hours. Initiating a large number of simultaneous connects in large network environments can slow down deployment by overburdening the source. Notify can be used in place of EDMTIMER to force smaller groups of users to receive information at staggered time intervals.

Types of Notify

There are three ways in which to invoke the notify function, as described in the following sections:

- Simple Notify (starting below)
- GUI-Configured Notify (starting on page 169)
- EDMMPUSH (starting on page 443)

Simple Notify

The EDMMNFYT method is for a TCP/IP environment. It enables a remote client notification, instructing the client to initiate the connect. To execute a Simple Notify, RADEXECD must be running on the clients on which a push is being executed.

The EDMMNFYT method stores the results of the notification in the Configuration Server's NOTIFY file, producing an instance for each client notified. This Configuration Server method works on all HP-supported Configuration Server platforms. Table 84 presents the parameters for EDMMNFYT, along with a description of each.

Table 84: EDMMNFYT Parameters

Parameter	Description
domain	Name of the domain where notification results are stored. If this parameter is not specified, the domain name will be automatically generated as a function of date/time.
instance	Instance name containing results of the single notification. If this parameter is not specified, instance name will be automatically generated as an eight-digit number. Default is 00000001.
password	ZNFYPWD for the ZMASTER object of the target terminal.
port	Port number. This should have the same value as the ZMASTER port number. (*MVS only*)
"process to run"	Application you are forcing the client desktop to execute.

Parameter	Description
target IP address	IP address of the client desktop to which you are executing a push.
user ID	Client user ID.

Newer notify features, such as Wake-On-LAN and Scheduling for Notifies are not supported by Simple Notify. To take advantage of these features, you must use the EDMMPUSH form of notify.

Figure 1 presents an overview of the Simple Notify process.

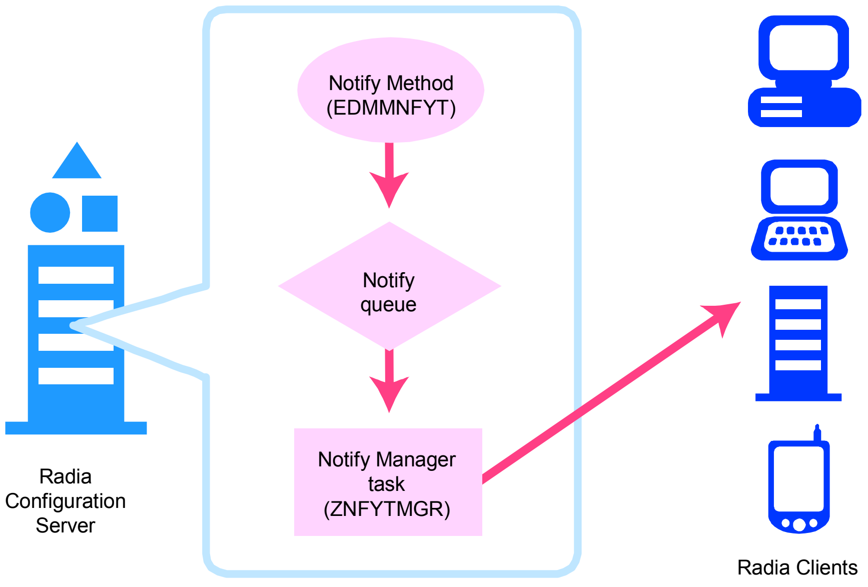


Figure 1: Overview of the Simple Notify process.

For information on how to configure multiple Notify Manager tasks, see Multiple Notify Managers on page 179.

For a look at how to retry failed notifications, see Retrying Failed Notifies on page 181.

For information on how to schedule a client notification, see Scheduling for Notify on page 183.

GUI-Configured Notify

The Configuration Server supports a method for establishing notifies using a standard graphical user-interface. This easy-to-use process is called drag-and-drop notification (**DDN**). The System Explorer provides the support for drag-and-drop notification on the administrator side. The drag-and-drop notification feature was incorporated to make it easier to notify large groups of users. Presently, this feature allows for the notification of all the users of the department, user group, single user, or all the users of the service. It is very important to understand that the destination information is searched for in the PROFILE file—specifically, in `PROFILE.`*`USER_ID`*`.ZMASTER.OBJECT`.

➤ In order for DDN to work, all the necessary information should be written to the PROFILE file during the client resolution process before DDN. If the necessary information is not in the PROFILE file, DDN will not be possible. DDN is not designed to notify new users whose destination information is not yet in the PROFILE file of the Radia database.

There are two aspects to DDN: the source icon (*dropper*) and the destination icon (*droppee*). The source icon is that which is dragged and dropped onto the destination icon.

➤ The source instance must belong to the USER, WORKGRP, DEPT, or ZSERVICE class, found in the PRIMARY.SYSTEMX domain. The destination instance must be a member of the ZCOMMAND class found in PRIMARY.ZSYSTEM domain.

To perform drag-and-drop notification (DDN)

1 In the System Explorer, open the Radia database tree all the way down to the icon that represents the source instance (for example, PRIMARY.SYSTEMX.WORKGRP.PROD_USERS).

2 Click the instance.

3 While holding down the left mouse button, drag the instance to the icon that represents the destination instance (for example, ZSYSTEM.ZCOMMAND.NOTIFY).

An icon, resembling a magic wand will appear, indicating that you are in the COMMAND class.

4 Release the mouse button, thereby associating the source icon (instance) with the destination icon (instance).

The result of this example is that all instances belonging to WORKGRP.PROD_USERS will be notified.

The response message from the Configuration Server will tell you how many users were found in this particular group and how many were scheduled for notification. Again, if the information about the selected users is not in the PROFILE file, no notification will occur.

As usual, all the results of the notification can be found in the Configuration Server log. Additionally, the notification results will be written to the NOTIFY file with a domain name created dynamically for each DDN action. The name of the domain will be returned to the client in the ZADMNHNL attribute of the ZADMIN object. To see the results, right-click on the domain, and from the pop-up menu, select **Status Display**. Another option, **Status Delete**, will delete the domain when you do not need it.

Types of Notifications Supported

It is important to note that a COMMAND class instance is a special type of instance used to define a command to be executed. Table 85 lists the ZCOMMAND class variables used for DDN.

Table 85: ZCOMMAND Class Variables Used for DDN

Variable	Description	Length
ZCMDNAME	Command name (NOTIFY, EMAIL).	8
ZCMDPATH	Location of the command. Used only for EXEs that are not pre-established (NOTIFY, EMAIL).	255
ZCMDPRMS	Parameters passed to the command.	255
ZCMDSEP	Separator used for parameters in user-defined commands.	1
ZCMDSYNC	A synchronization flag that defines whether to wait until the user command executes and ends, or to return control immediately (Y/N).	1

Variable	Description	Length
ZCMDUCLS	User class name. This is the name of the class in which to look for users connected to the droppee. For example, if the value is set to COMPUTERS and the droppee is WORKGROP.ACCOUNTING, instances of the COMPUTERS class that are members of WORKGROP.ACCOUNTING will be the selected audience for the notification. If ZCMDUCLS is not specified, then (using the above example) the audience will be created by instances of the COMPUTERS class that are members of WORKGROP.ACCOUNTING. The default for ZCMDUCLS is USER.	8
ZCMDTYPE	Type of command to be executed (REXX, EXE).	8
ZCMDHNDL	Notify handle is a domain name created/reused in a NOTIFY file to store the information about notification results. The class name is created depending on the type of notification, and an instance name is generated as a sequential number (for first request it's 00000001, for next 00000002, and so on). If the handle is not specified, the name will be generated as a function of date/time to make it unique.	32
ZCMDUINF	User information passed to notify start and end methods. If not specified, ZCMDHNDL in combination with instance name (heap number) will be used. For example, if handle was specified as **`USERS_DEFERRED_NOTIFY.`** For the first request, user info will be `USERS_DEFERRED_NOTIFY_00000001.` For the second, `USERS_DEFERRED_NOTIFY_00000002,` and so forth.	128
ZCMDRMAX	Maximum number of retries in case of notify failure.	3
ZCMDDLAY	In case of failure, the interval (in seconds) to wait before a retry will be scheduled. The default is 300 (5 minutes).	4
ZCMDNFYD	Date when the notify request should be executed the first time. The format is YYYY/MM/DD. The default is the current date.	10

Variable	Description	Length
ZCMDNFYT	Time when the notify request should be executed the first time. The format is HH:MM:SS. The default is the current time.	8

The user information in ZCMDUINF can be used as a key to write the information to an SQL database. The key would be unique. It is recommended that you do not specify the value, rather, let it be generated as described above. However, if the value is used in some other fashion in notify methods, it can be defined and then further processed in any way in notify methods (for instance, combine the handle and IP address for TCP/IP).

Currently, two types of notify operations are supported in the COMMAND class: NOTIFY and EMAIL. This means that you can choose between TCP/IP and e-mail for the notification.

You can also use the ZNFYTSTA REXX in conjunction with all types of Notifies. For more information, see Chapter 3, Managing Configuration Server Processing.

The notification command, specified as the ZCMDPRMS variable, is the text sent to the client either as the command line in a TCP/IP Notify, or as a message and a subject in an e-mail notify. (See Scheduling for Notify on page 183 for information on configuring the COMMAND class for deferred notification.)

The source instance must belong to the USER, WORKGRP, DEPT, or ZSERVICE class, found in the PRIMARY.SYSTEMX domain. The destination instance must be a member of the ZCOMMAND class found in PRIMARY.ZSYSTEM domain.

It is also important to understand that before the COMMAND class instance is taken for processing, the secondary resolution is done, and all the variables of the instance will be substituted. This will allow a partial or complete change of the command line, and/or even the notification type.

Necessary Profile Information

In order for the DDN to work, the following information must be in the source instance's PROFILE.userid.ZMASTER.OBJECT.

Table 86: Drag-and-Drop Profile Information

Variable	Description	e-mail	TCP/IP
ZUSERID	At the time of notification, the target user ID must match the one in the ZMASTER object.	N/A	√
ZNFYPWD	Password. If there is a ZNFYPWD in the ZMASTER object of the PROFILE file, notify will use it. Otherwise, the default, EDMPASS, will be used.	N/A	√
ZNTFPORT	Port number for notify daemon (The default is 512).	N/A	√
ZCIPADDR	IP address of the client machine.	N/A	√
ZIPNAME	Fully qualified host name of the client machine. If ZCIPADDR is not specified, this variable can be used instead.	N/A	√
EMAIL	Fully qualified e-mail address of the destination.	√	N/A

There are two ways of specifying a client IP address:

- **ZCIPADDR**
  This variable can be used to point to a specific host name/address, and is not resolved by the Configuration Server.
- **IPNAME**
  This variable contains the fully qualified symbolic host name of the connecting client and is resolved by the Configuration Server.

The order of precedence for the sources of client IP addresses is ZCIPADDR, then ZIPNAME.

In a case when the dropper is an instance of the SERVICE class, the service instance should be generated for each user of the service and this instance should be written to either the ZSVCSTAT or ZERVICE class. The ZSRCDOMN and ZSRCCLAS variables of this instance should specify the domain and class names of the service. The instance name has to match the name of the service itself.

Programmatically Configuring Notifies

Drag-and-drop was initially designed and implemented to support the System Explorer. However, it can be used separately in a well-established infrastructure for mass notification controlled by a timer on the client. The only input that is necessary for the back end is a ZADMIN object that has all the variables in it defining the source and destination instances.

Table 87 lists and describes the ZADMIN object variable names, and offers a sample value.

Table 87: Variables of the ZADMIN Object

Variable Name	Description	Sample Value
ZADMFILE	Destination file name	PRIMARY
ZADMDOMN	Destination domain name	ZSYSTEM
ZADMCLAS	Destination class name	ZCOMMAND
ZADMINST	Destination instance name	NOTIFY
ZADMDFIL	Source file name	PRIMARY
ZADMDDOM	Source domain name	SYSTEMX
ZADMDCLS	Source class name	WORKGRP
ZADMDINS	Source instance name	PROD_USERS
ZADMFUNC	Function name for DDN	EXECUTE
ZUSERID	Administrator user ID	Vladimir
ZNFYPWD	Password	nvdm123

An operation based on the above specifications would cause all the users of the PRIMARY.SYSTEMX.WORKGRP.PROD_USERS to be notified with the command defined in PRIMARY.ZSYSTEM.ZCOMMAND.NOTIFY. All the source information will be retrieved from the PROFILE file Radia database.

EDMMPUSH

The EDMMPUSH method is another way to implement the existing Configuration Server client-notification procedures. In fact, EDMMPUSH enhances the process, rather than acting as a substitute for any of the existing notify methods; and it might support future notify features that other types of notify do not.

The major advantage of EDMMPUSH is that it is not dependent on any one communications protocol, largely because it does not do the notification. Rather, it:

- receives the input requests,
- gets the required parameters, and
- puts the requests to the correct queues for subsequent processing by the appropriate Notify Manager.

This way, configuring the Radia database is simplified because all notify requests, of all notification types, can be concentrated in a single input object. This object, or even a dynamic object that was created because of object resolution, can be used to deliver notification requests to the EDMMPUSH method. This object might require different types of notification for each heap (request).

The Radia administrator can create a single multi-heap object that will notify all the clients, regardless of notification type, and then send the object to the Configuration Server to accomplish the notification.

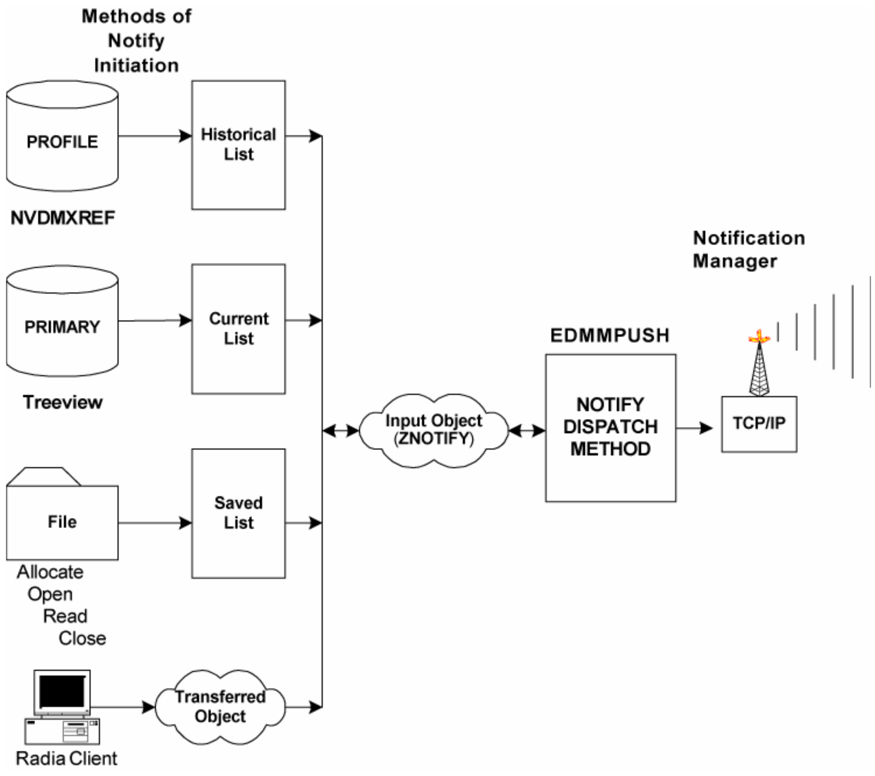


Figure 2: Input-object creation methods for EDMMPUSH.

The left side of Figure 2 shows the various methods of building the client-notification list that are put into an input object for EDMMPUSH. Each method can input various types of requests. The methods write the notify requests into input objects that will later be processed by EDMMPUSH because of object resolution. These various notify requests will be processed by a single method that will route them into the right queues for processing.

Input Object Used by EDMMPUSH

The EDMMPUSH method receives all the information about the notify requests from the input object. The name of the input object is defined in the Radia database, specifically in the PRIMARY.ZSYSTEM.ZMETHOD.EDMMPUSH instance field named "Parameters passed to Method." If this field does not specify an input object name, ZNOTIFY (the default) will be used. Each communications protocol that is used to execute a notification requires a specific set of variables. However, there are control variables that must be specified for every heap of the input object.

The following sections discuss these common control variables and describe specific protocol-dependent variables for each.

Common Control Variables

Common Input Variables

You must specify the following input variables.

Table 88: EDMMPUSH Input Variables

Variable	Description
NFYDELAY	Specifies the delay interval for trying to re-notify a client. If no value is entered, the default value is the value specified in the NFYT_TIMEOUT setting of the MGR_NOTIFY section of the `edmprof` file.
NFYHNDL	Specifies the domain name of the NOTIFY file where the results of notifications will be stored. The heap number of the request object will become the instance name.
NFYMRTRY	Specifies the maximum number of retries. If no value is entered, the default is the value specified in NFY_RETRY of the MGR_NOTIFY section of the `edmprof` file.

Variable	Description
NTFYRTIM	HP timestamp defining the time after which the notification should occur.
NFYPROC	Controls processing of the current heap request. If Y, the heap will be processed. If N, the request for the current heap will be ignored. The default is Y.
NFYTYPE	Defines the type of notify requested. The following values are allowed: • TCP • EMAIL The first three bytes of the type are used for the identification, so EMAIL and EMA are treated the same. There is no default value for this variable. If it is not defined, the current heap of the object will be ignored.
NFYUINFO	Allows you to enter user information.

Common Variables Set by EDMMPUSH

Due to the input request processing, the following variables are set in the input object.

Table 89: Variables Set by EDMMPUSH

Variable	Description or Setting
ZMMSG	Message regarding success status of required notification scheduling.
ZMRC	Return code (0 – success, 4 – warning, 16 – failure).
ZOBJCDEL	Set to Y in order to enforce control object deletion after object transfer is done.
ZOBJRDEL	Set to Y in order to enforce response object deletion after object transfer is done.

Protocol Dependent Input Variables

TCP/IP

The TCP/IP Notify uses REXEC protocol to deliver notification to the client. Therefore, a user ID and password are required for this type of notification. After the connection is established and the user ID-password combination is verified, the command line sent with the request will be executed on the remote (destination) machine. This command line should initiate the client-connect process. It is the administrator's responsibility to make sure that the command line contains the call that will be executed on the client and will initiate the client connect to the Configuration Server.

Table 90: TCP/IP Variables and Descriptions

Variable	Description
NFYCMD	Command line to be executed on the destination machine. This command line should initiate the client connect on the client.
NFYIPADR	TCP/IP address of the destination client.
NFYIPORT	Listening port number on the destination machine.
NFYPASSW	Password acceptable with user ID specified in NFYUSER.
NFYUSER	User ID acceptable on the destination system (used by native operating system security system).
NFYMAC	Mac address of the destination machine will be used for Wake-On-LAN. If NFYMAC is not specified, Wake-On-LAN cannot be used.

E-mail

Table 91: EMAIL Variables and Descriptions

Variable	Description
EMAILATT	Attachment to send in an e-mail (optional) message. User can specify multiple attachments by separating them with semi-colons (;).
EMAILFRM	E-mail address of the sender (mandatory).
EMAILMFN	Message file name, in case message is greater than 255 characters (mandatory, if EMAILMSG is not used).
EMAILMSG	Message that is restricted to 255 characters (mandatory, if EMAILMFN is not used). To send a message with spaces, follow the directions as in EMAILATT.

Variable	Description
EMAILSUB	The subject of the e-mail (optional) message. The subject must be enclosed in quotation marks if a space is used (as in “Hello Test”). If the quotation mark is not inserted at the end of the message, the text up to first space will be sent as a subject (“Hello” in this case).
EMAILTO	E-mail address to which the message is being sent (mandatory).

For information on how to configure multiple Notify Manager tasks, see the following section, Multiple Notify Managers.

For a look at how to retry failed notifications, see Retrying Failed Notifies, starting on page 181.

For information on how to schedule a client notification, see Scheduling for Notify, starting on page 183.

Multiple Notify Managers

In order to speed up the processing of a large number of notification requests, multiple Notify Manager tasks (of the same communications type) can be configured. When multiple Notify Managers are used, a single notification pipeline is substituted by as many lines as there are Notify Managers configured. This approach is based on a single request queue, resource-protected by a read mutex semaphore, with multiple Notify Managers reading from it.

Each Notify Manager waits for the semaphore. The Notify Manager that owns the semaphore reads it from the queue, and then immediately releases it and continues the notification process for the request it currently has. Once a Notify Manager releases the semaphore, other Notify Managers can start processing their queued requests.

Configuring Multiple Notify Managers

Multiple Notify Managers should be started in the same way a single Notify Manager is started—specified as a Configuration Server task in the MGR_ATTACH_LIST section of the `edmprof` file. Sequentially specify as many Notify Managers as necessary. Additional parameters and values (such as NAME=ZNFYTM*nnn*) can be added to the CMD_LINE settings in order to uniquely identify each of the Notify Managers. For example:

```
CMD_LINE=(znfytmgr NAME=ZNFYTM001)…
CMD_LINE=(znfytmgr NAME=ZNFYTM002)…
CMD_LINE=(znfytmgr NAME=ZNFYTM003)…
```

This will cause the Task Manager to start and maintain three tasks: ZNFYTM001, ZNFYTM002, and ZNFYTM003.

An overview of the notify process with multiple Notify Managers is illustrated in Figure 3 below.

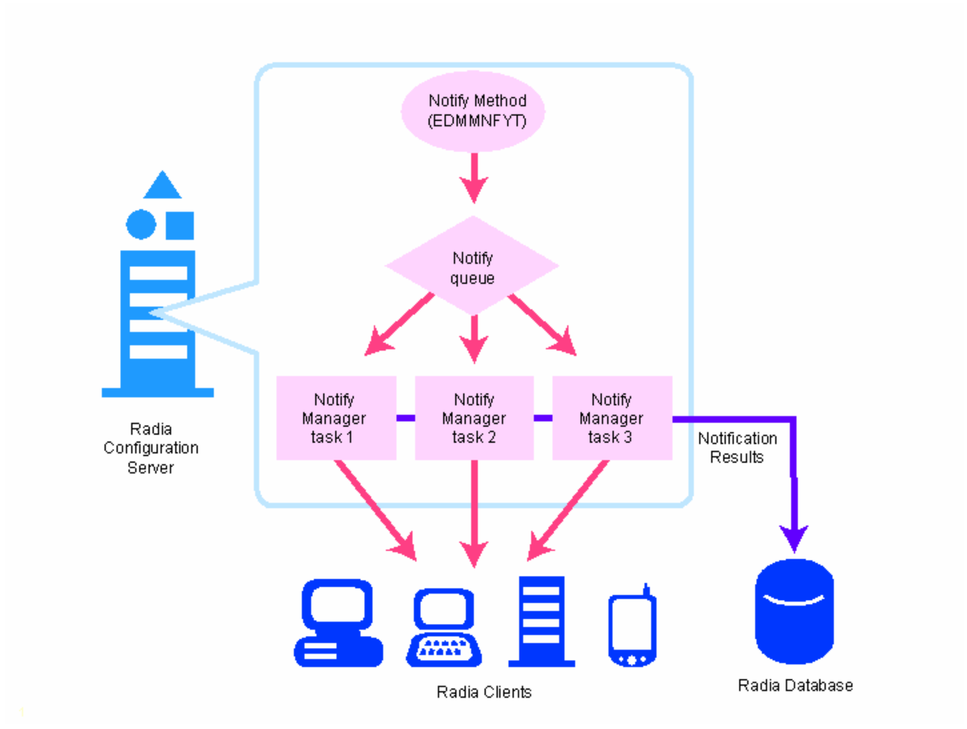


Figure 3: Notify process with multiple Notify Managers.

For information on how to retry failed notifications, see the following section, Retrying Failed Notifies.

For information on how to schedule a client notification, see Scheduling for Notify, starting on page 183.

Retrying Failed Notifies

Prior to the release of the version 4.1 Configuration Server, the notification process included the three ways of invoking client notification that have been previously described. These types of notification put notification requests into a notify queue. The Notify Manager task would take one request from the queue, process it, and write the results to the Radia database's NOTIFY file. The Notify Manager would then sequentially process the remaining requests, until all requests were processed.

If, for some reason, one of the notifications takes an inordinate amount of time, all queued requests would be delayed. In the case of an error, the Notify Manager does not retry the notification.

To enable the retrying of notifications, the Notify Manager stores failed-request information in the RETRY domain of the NOTIFY file. The suitable time for retrying the notification is set in the request. The Retry Manager wakes up every minute and checks all instances of all classes in the RETRY domain. If failed requests exist, the Retry Manager compares the scheduled re-notification time with the current time and, if the time is right, re-queues the request. The Retry Manager processes failed notifications for all types of communications Managers and re-queues them accordingly.

To configure for the Retry Manager, add zrtrymgr to the MGR_ATTACH_LIST section of the `edmprof` file.

Figure 4 on page 182 illustrates the Notify Retry process.

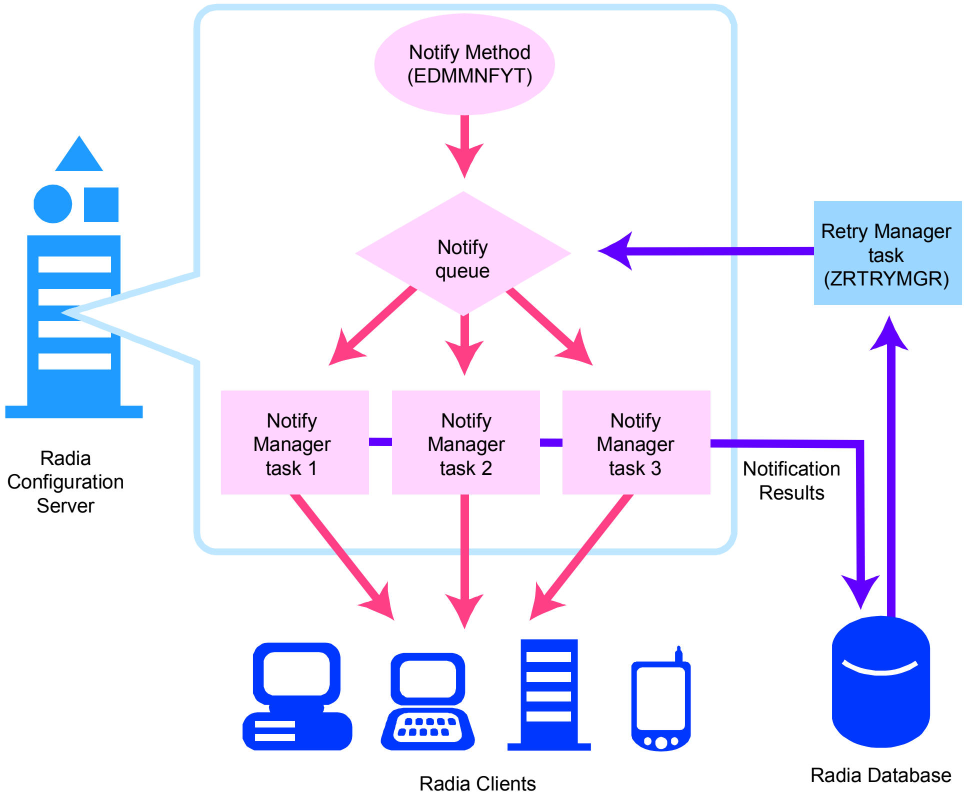


Figure 4: The Notify Retry process.

Scheduling a client notification is covered in the next section, Scheduling for Notify.

Scheduling for Notify

You can use, in combination, the EDMMPUSH method and the Retry Manager to schedule delayed executions of notify. To configure this scheduling feature, add an NTFYRTIM variable to the in-bound EDMMPUSH object, and have the zrtrymgr task included in the MGR_ATTACH_LIST section of the `edmprof` file.

NTFYRTIM

The NTFYRTIM variable is used to schedule the time at which a notify event should occur. If NTFYRTIM is specified, EDMMPUSH does not put the request in the notify queue. Rather, the request is written to the RETRY domain of the NOTIFY file. The zrtrymgr task checks the RETRY domain every minute, and when the date and time specified by NTFYRTIM is reached, it puts the notify request into the queue for the Notify Manager to process. The scheduling function allows you to retry failed notifications also. Additionally, you can recover those notifications that were scheduled, but where the Configuration Server was stopped and restarted.

Table 92: NTFYRTIM Settings

Variable	Description		
NTFYRTIM	Time (in the format of EDM_TIMESTAMP) at which the notification should execute. If this variable is absent or blank, EDMMPUSH will presume that the request should be executed immediately. This value must be exactly 22 characters and cannot contain commas or spaces. The format of the time stamp is:		
	Parameter	**Length**	**Options**
	Year	4	*YYYY*
	Month	2	01-12 (where 01=January)
	Weekday	1	0-6 (where 0=Sunday)
	Date	2	01-31
	Hour	2	00-23
	Minute	2	00-59

Variable	Description		
	Second	2	00-59
	Millisecond	3	000-999
	Time zone adjustment		This is the adjustment according to the location of the Configuration Server. This setting must start with **+** or **–**, followed by a three-digit (number of minutes) adjustment.
	NTFYRTIM Time Zone Adjustments details how to configure NTFYRTIM.		

NTFYRTIM Time Zone Adjustments

In order for NTFYRTIM to function properly, time-zone adjustments must be configured correctly. Since the Configuration Server uses the operating system's clock (which might have an automatic Daylight Saving Time (**DST**) adjustment feature), it is important that the NTFYRTIM setting be properly set, using Greenwich Mean Time (**GMT**), with DST accounted for, when necessary. Additionally, when configuring this setting, 24-hour-clock (a.k.a. military) time must be used.

GMT is a constant and *does not* adjust for Daylight Saving Time.

The first eight values in the table are the date and time (in GMT) that the notify event is scheduled to execute. The last setting, time zone adjustment, represents the adjustment (to the physical time of the Configuration Server machine) necessary to synchronize it with GMT.

Therefore, a Configuration Server in NY, USA, which is 5 hours behind GMT during standard time (and 4 hours behind during DST), would need the proper number of adjustment minutes (300) added, to be synchronized with GMT. The follow examples offer several sample NTFYRTIM settings.

Example A

To schedule a notify event for Wednesday July 09, 2001 at 2:35:15:000 (P.M.) GMT, specify:

- on a Configuration Server in New York, USA (GMT -4 hours, since DST is in effect)

  `2001 07 3 09 14 35 15 000 +240 = 2001073091435 15000+240.`

- on a Configuration Server in Paris, France (GMT +2 hour, since DST is in effect)

  2001 07 3 09 14 35 15 000 -120 = 200107309143515000-120.

Example B

To schedule a notify event for Friday November 09, 2001 at 2:35:15:000 (A.M.) GMT, specify:

- on a Configuration Server in Seattle, USA (GMT -8 hours, since DST is not in effect)

  2001 11 5 09 02 35 15 000 +480 = 200111509023515000+480.

- on a Configuration Server in Tokyo, Japan (GMT +9 hours, since DST is not in effect)

  2001 11 5 09 02 35 15 000 -540 = 200111509023515000-540.

In Example B, a Configuration Server located in Seattle, WA, USA would actually be on a different date (the previous day), November 08, 2001. This is irrelevant because the task is scheduled using GMT.

Another way to specify this, is to make the adjustment in the four time values (hour, minute, second, and millisecond) of NTFYRTIM, and specify (+ / –) 000 for the time zone adjustment. Re-using the parameters from Example A, Example C makes the adjustment in the time values.

Example C

To schedule a notify event for Wednesday July 09, 2001 at 2:35:15:000 (P.M.) GMT, specify:

- on a Configuration Server in New York, USA (GMT -4 hours, since DST is in effect)

  2001 07 3 09 10 35 15 000 +000 = 200107309103515000+000.

- on a Configuration Server in Paris, France (GMT +2 hour, since DST is in effect)

  2001 07 3 09 16 35 15 000 -000 = 200107309163515000-000.

In Example C, the time zone adjustment value must still be specified, but the offset symbol (+ / -) preceding 000 is irrelevant.

Time Zone Offsets

Figure 5 has been included in order to assist in remembering whether to adjust forward or back for the various time zones, in relation to GMT.

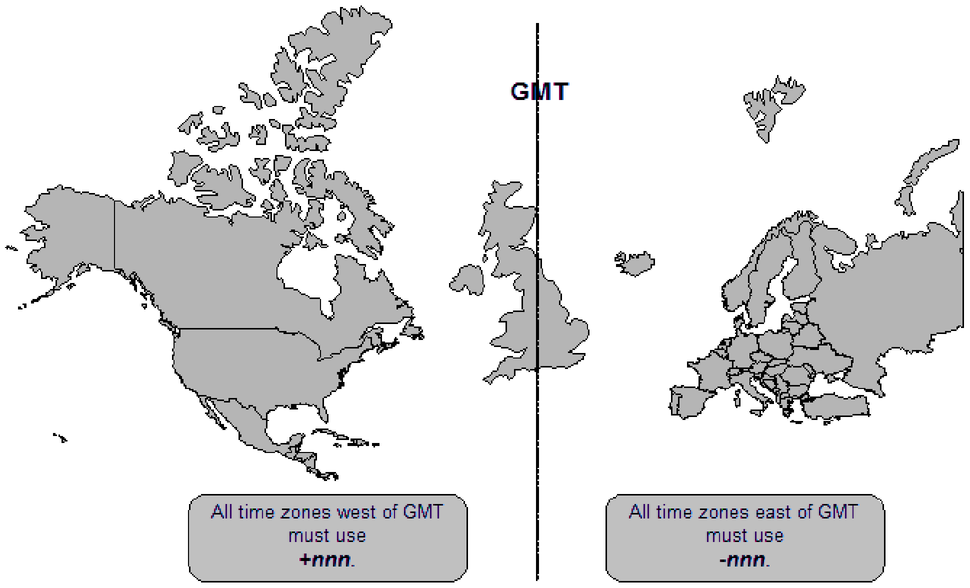

GMT
All time zones west of GMT must use +nnn.
All time zones east of GMT must use -nnn.

Figure 5: The vertical line represents Greenwich Mean Time (GMT).

Automatic Adjustments for Daylight Saving Time

If the Configuration Server machine offers the ability to have its clock automatically adjust for Daylight Saving Time, we recommend that you activate this feature.

- On a Windows machine, this is accomplished in the **Control Panel** area.
- On a UNIX machine, this is configured during installation. If you need further information, consult the documentation for the operating system.

Drag-and-Drop Notify

Scheduling for notify can also be configured for Drag-and-Drop Notify (DDN). To enable the scheduling function, you must add the following variables to the ZCOMMAND class:

Table 93: ZCOMMAND Variables Required for Notify

Variable Name	Description	Length
ZCMDHNDL	Domain name created/reused in the NOTIFY file to store the information about notification results. The class name will be created depending on the type of notification. Individual instance names are generated as sequential numbers (for example, 000000001 for the first request, 000000002 for the second, etc.). If ZCMNDHNDL is not specified, it will be uniquely generated as a function of date and time.	32
ZCMDUINF	User information that is passed to notify start and stop methods. If ZCMDUINF is not specified, it will be uniquely generated as a combination of ZCMNDHNDL and instance name (heap number).	128
ZCMDRMAX	Maximum number of retries in case of notify failure. The default is 7.	3
ZCMDDLAY	Delay interval (in seconds) before the retry will be scheduled. The default is 300 (5 minutes).	4
ZCMDNFYD	Date when the notify request should be executed for the first time. The format is YYYY/MM/DD. The default is the current date.	10
ZCMDNFYT	Time when the notify request should be executed for the first time. The format is HH:MM:SS. The default is the current time.	8

Wake-On-LAN

Another notify feature that takes advantage of the Retry Manager is Wake-On-LAN (WOL). Wake-On-LAN is a management tool that enables a system to remotely power-on other systems that support WOL, by simply sending a "wake-up" packet. Wake-On-LAN allows the workstation to go into a sleep mode and to then wake when it is sent a specially formatted packet.

Wake-On-LAN enables an administrator to remotely upload/download data to/from systems, as well as schedule client maintenance for off-peak hours.

The Benefits of Wake-On-LAN

Some of the advantages of Wake-On-LAN are:

- Increased flexibility for the system administrator,
- A reduction in operating costs, and
- Extended ability to perform distribution during off-peak time windows.

Components Required to Enable Wake-On-LAN

To enable the Wake-On-LAN function, your system requires:

- An Ethernet LAN-adapter card (such as the ASUS PCI-L101) that supports Wake-On-LAN,
- A motherboard that supports Wake-On-LAN,
- A jumper cable installed from the LAN adapter to the motherboard.

Configuring Wake-On-LAN

In order to enable WOL, and to have it function properly, some configuration is required on the Configuration Server, and network routers must be enabled for sub-network broadcasts.

EDMWAKE

EDMWAKE is not a part of the standard Configuration Server product and is currently available as optional material on selected platforms. Support for EDMWAKE is now limited only to notify requests that are initiated using the EDMMPUSH method.

EDMWAKE on the Command Line

When running EDMWAKE on the command line, it requires two address parameters, separated by a blank character, and an optional parameter, TTL.

- The broadcast address of the destination machine (herein, destination broadcast address). This is required to ensure the broadcast packet traverses intermediate network routers, if any exist.
- The Media Access Control address of the destination machine (herein, MAC address). EDMWAKE issues an asynchronous data-flow from which

no response is expected. The return codes that are issued are indicative of program execution only; they do not reflect the success of contacting the target machine.

- The optional TTL (time to live) is the maximum number of routers to pass.

The MAC and destination broadcast addresses can be found on the destination machine by typing:

```
IPCONFIG /all
```

This will generate output to the screen. The output of a sample IPCONFIG /all command is shown in Table 94.

Table 94: Sample IPCONFIG /all Results

Parameter	Value
Physical Address	00-06-5B-2F-99-23 Note: This is also the MAC address.
DHCP Enabled	Yes
Auto-configuration Enabled	Yes
IP Address	192.168.102.191 Notes: This is a Class C type address. In an enterprise with the network address, 192.168.102, this machine is identified as 191.
Subnet Mask	255.255.255.0 Note: This is the Class C type address default.
Default Gateway	192.168.102.1
DHCP Server	192.168.102.70
DNS Servers	192.168.110.4
	192.168.110.5
Primary WINS Server	208.244.225.122

The Physical Address that is displayed is the MAC address also.

Network Addresses

The destination broadcast address is generated by combining the network and host portions of the target machine's IP address; after the host portion has been replaced by the generic broadcast address, 255.

Therefore, using the information in Table 94, the destination IP address, 192.168.102.191, is used to create a destination broadcast address of 192.168.102.255. (The Configuration Server does this transparently.) This results in the data packets being sent to all of the machines on the (192.168.102.0) network.

Again, using the sample addresses in Table 94, with the subnet mask being 255.255.255.0 (the class C default), the network address is 192.168.102.0.

For a comprehensive look at IP addresses, visit: **http://www.networkcomputing.com/netdesign/ip101.html**.

HP recommends running EDMWAKE from the command line first to make sure it works, and then configure the Configuration Server for usage via notify.

In EDMMPUSH, use the NFYMAC variable to specify the physical address of the machine and all other parameters (as specified in HP documentation).

EDMWAKE issues an asynchronous data flow from which no response is expected. The return codes issued are indicative of program execution only and do not reflect the success or failure of contacting the intended target machine.

A log, `EDMWAKE.LOG`, s generated in the current directory, with the following possible return codes generated:

0	successful
32	parms invalid, or some other error encountered

Network Requirements

EDMWAKE issues a broadcast packet that traverses an IP network. In order to operate correctly, it is necessary that the IP routers and gateways be configured to allow such broadcast traffic to pass through; otherwise, the data-gram will not have the intended effect.

It might be necessary to involve the network management staff within your enterprise during the testing and extended use of this component.

Configuration Server Requirements

Add the following settings and values to the MGR_NOTIFY section of the Configuration Server `edmprof` file to enable support for WOL:

```
ISSUE_WAKE_ON_LAN=YES
```

The default is NO.

Wake-On-LAN can only wake up machines that have been gracefully shut down. If power has been turned off, a machine cannot be contacted.

```
SUBNET_MASK = 255.255.0.0
```

For more information on SUBNET_MASK, see MGR_NOTIFY on page 76.

Also, edit the Configuration Server retry variable in the MGR_ATTACH_LIST section of the Configuration Server `edmprof` file as follows:

```
[MGR_ATTACH_LIST]
CMD_LINE    =(ztcpmgr,addr=joe,port=1955,name=TCP_Mgr_1955) RESTART  =YES
CMD_LINE    =(zrtrymgr) RESTART=YES
CMD_LINE    =(znfytmgr,NAME=NFYTMGR1) RESTART=YES
CMD_LINE    =(znfytmgr,NAME=NFYTMGR2) RESTART=YES
```

The Configuration Server attempts to issue a notify. If the notify fails on the Connect stage with an error other than "destination port is not active," the machine might need to be powered-up. The WAKE_ON_LAN is issued on the first failure, and must be requested in the Configuration Server `edmprof` file. The retry of the notify will be 300 seconds after Wake-On-LAN (to allow enough time to boot). For the Retry Manager, which has no knowledge of WOL, it is a normal retry operation.

Client/PC Requirements

EDMWAKE implements a Wake-On-LAN functionality that is part of the Wired-for-Management (WfM) initiative. Only client machines that are properly configured with appropriate motherboards, NICs, and the correct jumpers connecting the two, will work with this data-flow. Additionally, there might be BIOS settings that need to be enabled in order to allow the client machine to be responsive to this data-flow. Check with the hardware vendor to find out if your computer is enabled for WOL.

Wake-On-LAN Supporting Remote Broadcast

In order to have the destination broadcast address included in the wake command when it is issued, make sure that the sub-network broadcast address is specified as a parameter of EDMWAKE (see Network Addresses, on page 190). The Configuration Server formulates the sub-network broadcast address based on the destination IP address. The destination broadcast address will be adjusted according to the type of IP address (A, B, C, D, or E). This is required to allow packets to traverse any intermediate routers.

Example

```
208.107.6.5 (subnet 208.107.6.255)
```

A specific SUBNET_MASK can be used if defined in the Configuration Server `edmprof` file. For more information on SUBNET_MASK, see MGR_NOTIFY on page 76.

5 HP SQL Methods

At the end of this chapter, you will:

- Have a better understanding of the HP Structured Query Language (**SQL**) methods and Open Database Connectivity (**ODBC**) data sources.

This chapter is divided into two primary sections:

- Data Exchange with ODBC-Compliant Databases below
- Using HP SQL Methods on page 220.

Data Exchange with ODBC-Compliant Databases discusses an ODBC data source, and details why and how to define, obtain, and configure an ODBC data source. Also covered is how to configure an ODBC connection to a SQL Server (on Windows and UNIX).

Using HP SQL Methods details the HP SQL methods (EDMMSQLG and EDMMSQLP), including:

- How to invoke HP SQL methods,
- The WHERE clause (which identifies the rows in the SQL database table that are to be replaced), and
- Usage considerations and examples.

The HP SQL methods support only the following ODBC-compliant databases:

- MS SQL
- Oracle
- Sybase

Data Exchange with ODBC-Compliant Databases

Introduction

Before using the HP SQL methods, configure an ODBC data source. ODBC is a platform-independent Application Program Interface (**API**) specification that allows SQL statements to be submitted from programs external to the database system, and then be processed by the database system over the ODBC connection. A database system exposes its ODBC interface to external programs through ODBC data-source definitions.

An ODBC Data Source: Prerequisites

The HP SQL methods are client programs that are external to the back-end database, which acts as a server.

The ODBC data-source definition identifies the location of the ODBC-compliant database's tables for EDMMSQLG and EDMMSQLP. It also specifies any options regarding how the back-end database system services the ODBC connection. Any such options will vary from one back-end database to another, and familiarity with the back-end databases is needed in order to set them properly.

Many database systems are ODBC-compliant, and the specifics of configuring an ODBC data source for a particular database system are described in that database system's documentation. In this section, several examples are presented.

Defining an ODBC Data Source

The ODBC data source must be defined on the computer that is running the Configuration Server, but the database tables can be located on any machine accessible to the Configuration Server machine.

On Windows NT and Windows 2000 computers, a data source can be defined as either a **User DSN** or a **System DSN** (Data Source Name). A User DSN is visible only to the user that defines it. A System DSN is visible to any user on the computer. Since the Configuration Server normally runs as a service in the system context, define the data source as a System DSN so that the Configuration Server can use it.

Obtaining an ODBC Data Source

To configure an ODBC data source for a particular database system, the ODBC driver for the database system must be installed on the computer on which the ODBC data source will be defined. This driver typically ships with the database system, or can be obtained separately from the database system vendor. Windows ships with a set of ODBC drivers. There are also third-party ODBC drivers available for most popular database systems. The major third-party source for ODBC drivers is Merant Plc. (formerly Intersolv, Inc.).

In some cases, additional software will be needed to completely configure an ODBC connection. This is true, for example, when the Configuration Server is running on a UNIX platform, and the ODBC-compliant database is Microsoft SQL Server running on Windows. This configuration requires Merant SequeLink ODBC Edition software to be installed on the Windows server. An example is provided below.

Configuring an ODBC Data Source

For many ODBC-compliant, desktop-computer database systems (such as Access and FoxPro), configuring an ODBC data source is no more complicated than making up a name for the data source, then specifying a path to the folder that contains the database tables, and perhaps specifying a small number of database-specific settings. The following examples show this process.

To configure an ODBC data source with Microsoft FoxPro 2.6

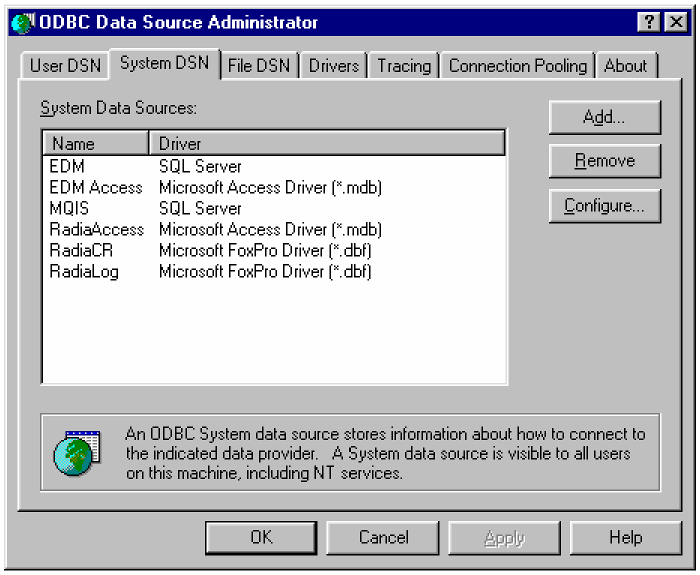
This example illustrates configuring the ODBC data source that is used in some of the examples in this section. The ODBC-compliant database system is Microsoft FoxPro 2.6 for Windows. The Configuration Server is running under Windows NT 4.0.

1 Go to **Start→Settings→Control Panel** to open the Control Panel folder.

2 Double-click the **ODBC** icon to launch the ODBC Data Source Administrator.

3 Click the **System DSN** tab.

A dialog box similar to the following opens:



System Data Sources:

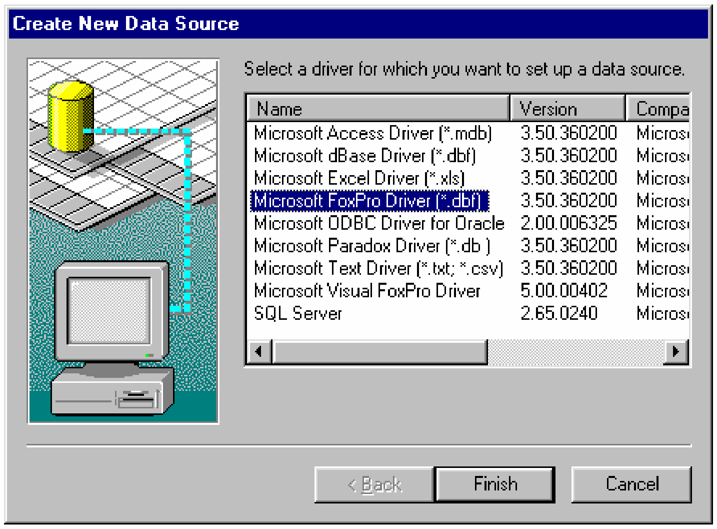
Name	Driver
EDM	SQL Server
EDM Access	Microsoft Access Driver (*.mdb)
MQIS	SQL Server
RadiaAccess	Microsoft Access Driver (*.mdb)
RadiaCR	Microsoft FoxPro Driver (*.dbf)
RadiaLog	Microsoft FoxPro Driver (*.dbf)

An ODBC System data source stores information about how to connect to the indicated data provider. A System data source is visible to all users on this machine, including NT services.

When the Configuration Server is running as an NT service, be sure to configure a System DSN rather than a User DSN. Be sure to click on the **System DSN** tab to open the System Data Sources panel.

4 Click **Add** to configure a new ODBC data source.

A dialog box similar to the following opens:

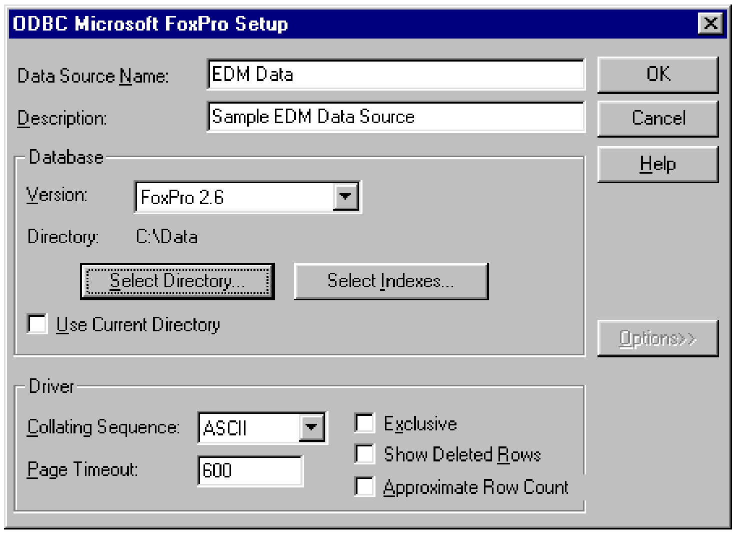

Create New Data Source

Select a driver for which you want to set up a data source.

Name	Version	Compa
Microsoft Access Driver (*.mdb)	3.50.360200	Micros
Microsoft dBase Driver (*.dbf)	3.50.360200	Micros
Microsoft Excel Driver (*.xls)	3.50.360200	Micros
Microsoft FoxPro Driver (*.dbf)	3.50.360200	Micros
Microsoft ODBC Driver for Oracle	2.00.006325	Micros
Microsoft Paradox Driver (*.db)	3.50.360200	Micros
Microsoft Text Driver (*.txt; *.csv)	3.50.360200	Micros
Microsoft Visual FoxPro Driver	5.00.00402	Micros
SQL Server	2.65.0240	Micros

5 Click on the driver for the database you intend to use, and click **Finish**. In this example, we selected **Microsoft FoxPro Driver**.

The following dialog box opens (shown here after clicking **Options**, and with the required information specified).


ODBC Microsoft FoxPro Setup

Data Source Name: EDM Data

Description: Sample EDM Data Source

Database

Version: FoxPro 2.6

Directory: C:\Data

Use Current Directory

Driver

Collating Sequence: ASCII

Page Timeout: 600

Exclusive

Show Deleted Rows

Approximate Row Count

Table 95: Microsoft FoxPro 2.6 ODBC Specifications

Setting	Description
Data Source Name	A name that you make up to identify this ODBC data source to external programs, such as EDMMSQLG and EDMMSQLP. You will supply this DSN as a parameter to EDMMSQLG/EDMMSQLP, as described later in this document.
Description	This is also a free text field. Make up a description that denotes the purpose or use of the data source definition. It identifies this data source when navigating the Control Panel ODBC applet.
Database	• Version – This identifies the version of FoxPro. This setting pertains only to FoxPro, you will not see this in dialog boxes that configure ODBC data sources for other back-end databases. • Directory – This identifies a folder where the ODBC-accessible FoxPro data tables are located. • Select Directory – This setting defines a directory for the ODBC data source. Note: To enable the **Select Directory** button, clear the **Use Current Directory** check box. Then click **Select Directory**, and use the resulting dialog box to select the correct folder. Click **OK**, and the ODBC data source definition is compete, and added to the system. • Select Indexes – This option isn't applicable.
Driver	This setting is specific to FoxPro. It appears only when you click **Options**. When you first open this dialog box, the **Options** button is enabled, and its settings are hidden. *Do not alter these settings.*

To configure an ODBC data source with Microsoft Visual FoxPro

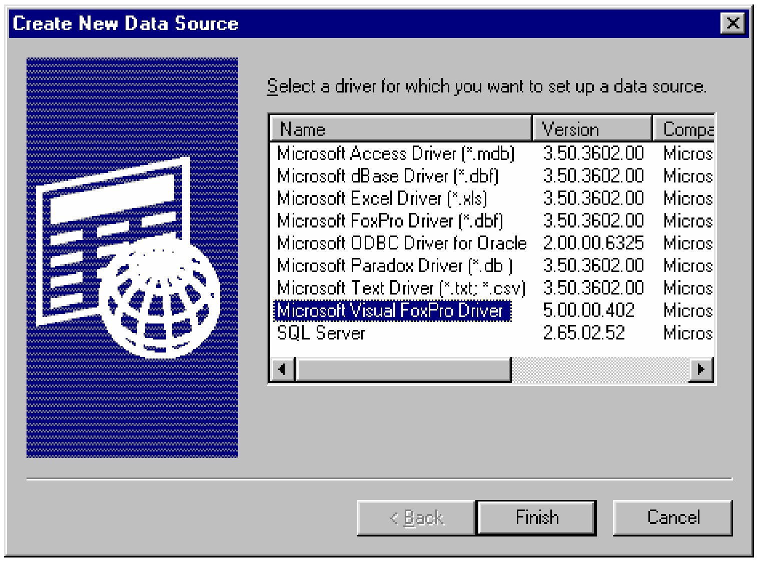
The following example illustrates how to configure the ODBC data source used in some of the examples later in this document. The back-end database system is Microsoft Visual FoxPro. The Configuration Server is running under Windows NT.

1 Go to **Start→Settings→Control Panel** to open the Control Panel folder.

2 Double-click the **ODBC** icon to launch the ODBC Data Source Administrator.

3 Click the **System DSN** tab, and click **Add**.

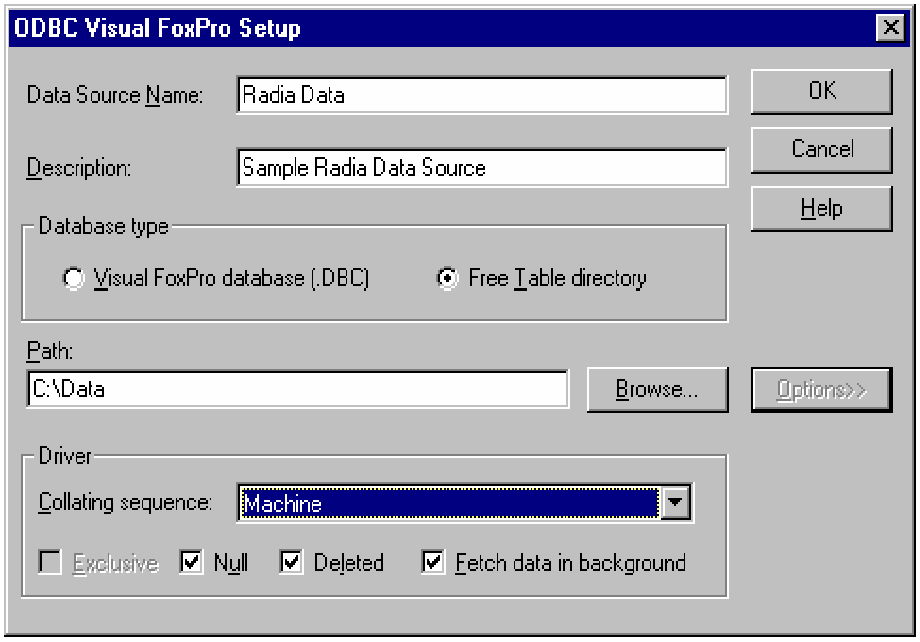
A dialog box similar to the following opens.



Name	Version	Compa
Microsoft Access Driver (*.mdb)	3.50.3602.00	Micros
Microsoft dBase Driver (*.dbf)	3.50.3602.00	Micros
Microsoft Excel Driver (*.xls)	3.50.3602.00	Micros
Microsoft FoxPro Driver (*.dbf)	3.50.3602.00	Micros
Microsoft ODBC Driver for Oracle	2.00.00.6325	Micros
Microsoft Paradox Driver (*.db)	3.50.3602.00	Micros
Microsoft Text Driver (*.txt; *.csv)	3.50.3602.00	Micros
Microsoft Visual FoxPro Driver	5.00.00.402	Micros
SQL Server	2.65.02.52	Micros

4 Click on the driver for the database you intend to use, and click **Finish**. In this example, we clicked **Microsoft Visual FoxPro Driver**.

The following dialog box opens (shown here with the required information specified).



Data Source Name: Radia Data

Description: Sample Radia Data Source

Database type: Visual FoxPro database (.DBC) [✓ Free Table directory]

Path: C:\Data

Driver

Collating sequence: Machine

Exclusive [✓ Null] [✓ Deleted] [✓ Fetch data in background]

Table 96: Microsoft Visual FoxPro ODBC Specifications

Data Source Name	Type a name to identify this ODBC data source to external programs, such as EDMMSQLG and EDMMSQLP. You will supply this DSN as a parameter to EDMMSQLG/EDMMSQLP, as described later in this document.
Description	Type a description that denotes the purpose or use of the data source definition. It identifies this data source when navigating the Control Panel ODBC applet.
Database type	This setting pertains to Visual FoxPro only; you will not see this in dialog boxes to configure ODBC data sources for other back-end databases.
Path	This setting identifies a folder where the ODBC-accessible Visual FoxPro data tables are located. You can type the path to this folder into the text box, or click **Browse** to open a dialog box that enables you to select the path from a list.
Driver	This setting is specific to FoxPro. It appears only when you click **Options**. When you first open this dialog box, the **Options** button is enabled, and its settings are hidden. In this area, select the following: **Null**, **Deleted**, and **Fetch data in background**. From the **Collating sequence** drop-down list-box, select **Machine**.

5 Click **OK** to save the ODBC data source definition, and add it to the system.

To configure an ODBC data source with Microsoft Access

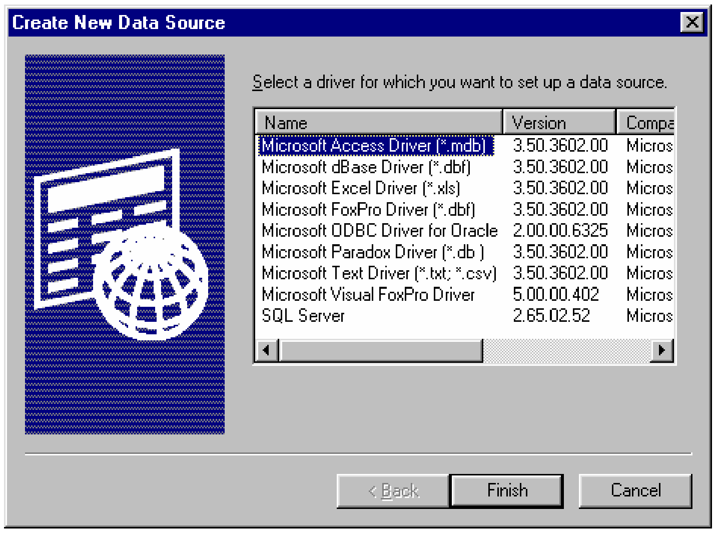
The following example illustrates configuring an ODBC data source where Microsoft Access is the back-end database. The Configuration Server is running under Windows NT.

1 Go to **Start→Settings→Control Panel** to open the Control Panel folder.

2 Double-click the **ODBC** icon to launch the ODBC Data Source Administrator.

3 Click the **System DSN** tab, and click **Add**.

A dialog box similar to the following opens.



Select a driver for which you want to set up a data source.

Name	Version	Compa
Microsoft Access Driver (*.mdb)	3.50.3602.00	Micros
Microsoft dBase Driver (*.dbf)	3.50.3602.00	Micros
Microsoft Excel Driver (*.xls)	3.50.3602.00	Micros
Microsoft FoxPro Driver (*.dbf)	3.50.3602.00	Micros
Microsoft ODBC Driver for Oracle	2.00.00.6325	Micros
Microsoft Paradox Driver (*.db)	3.50.3602.00	Micros
Microsoft Text Driver (*.txt; *.csv)	3.50.3602.00	Micros
Microsoft Visual FoxPro Driver	5.00.00.402	Micros
SQL Server	2.65.02.52	Micros

4 Click on the driver for the database you intend to use, and click **Finish**. In this example, we've selected **Microsoft Access Driver**.

The following dialog box opens (shown after the required information has been specified).

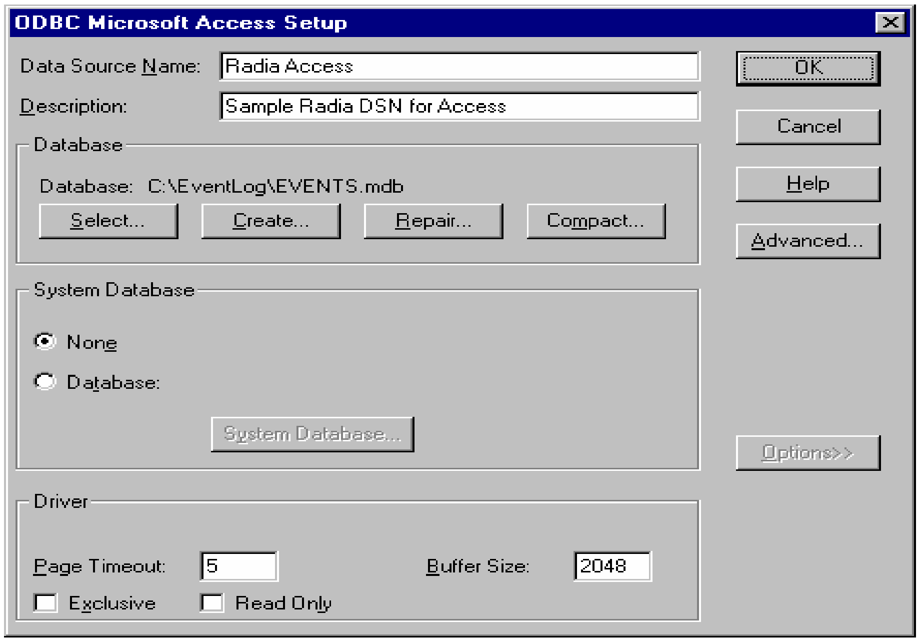


Table 97: Microsoft Access ODBC Specifications

Setting	Description
Data Source Name	Type a name to identify this ODBC data source to external programs, such as EDMMSQLG and EDMMSQLP. You will supply this DSN as a parameter to EDMMSQLG/EDMMSQLP, as described later in this document.
Description	Type a description that denotes the purpose or use of the data source definition. It identifies this data source when navigating the Control Panel ODBC applet.
Database	This setting pertains to Microsoft Access only; you will not see this in dialog boxes to configure ODBC data sources for other back-end databases. These settings enable you to Select or Create a database, or perform maintenance functions (Repair and Compact). Typically, you would click **Select** and use the resulting file-selection dialog box to locate and choose the Microsoft Access database with which you want to exchange data.
System Database	This setting identifies a folder where the ODBC-accessible Visual FoxPro data tables are located. You can type the path to this folder into the text box, or click **Browse** to open a dialog box that enables you to select the path from a list.

Setting	Description
Driver	This setting is specific to FoxPro. It appears only when you click **Options**. When you first open this dialog box, the **Options** button is enabled, and its settings are hidden. In this area, specify the settings as shown in the previous figure.

5 Click **OK** to save your ODBC data source definition and add it to the system.

SQL Servers

ODBC data sources for server-based database systems such as Microsoft SQL Server, Oracle, and Sybase, are more complex to configure. Since these are generally password-protected, authorization rights need to be configured in the back-end database. Also, when the database is running on a machine other than that which houses the Configuration Server, the ODBC connection will operate over a communications link, which will require some configuration. Typically, your organization's database systems administrator and/or network administrator handles these jobs.

Additional software might need to be installed and configured. You will need to consult your database systems administrator to have the ODBC data source properly established in these cases. For additional information, see the following examples for setting up an ODBC Data Source for Microsoft SQL Server under Windows and UNIX.

Microsoft SQL Server with a Windows Configuration Server

Configuring an ODBC connection with Microsoft SQL Server is more complex than setting up an ODBC connection to desktop databases such as Microsoft Access and FoxPro.

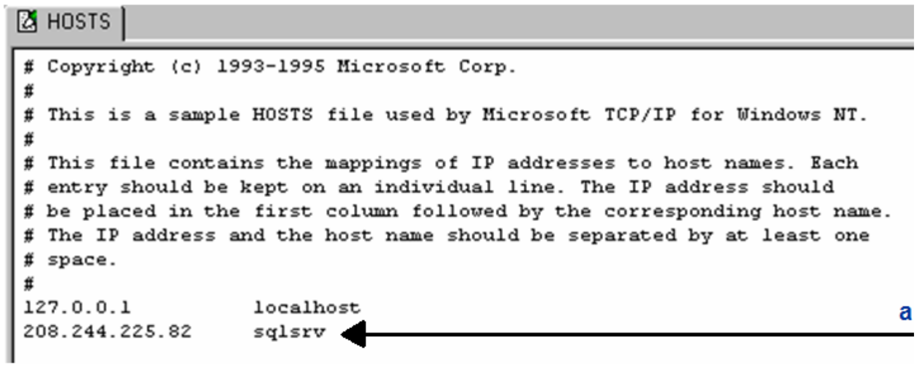
Gather Information

First, the administrator of the SQL Server database must provide a user ID and password for the ODBC connection to use when it logs on to SQL Server.

You will provide these to EDMMSQLG/EDMMSQLP as parameters at run time.

Second, since SQL Server is server-based, you are required to specify the communication link for accessing the data. For desktop databases, the data tables are located within the file system space of the Configuration Server, either on the same machine, or via a mapped drive on a LAN. You can communicate with a machine running SQL Server using one of a number of communications protocols. In this example, we used a TCP/IP connection. You must know the SQL Server machine's IP address and port number for SQL Server client communications.

To facilitate any future change in the IP address of the SQL Server machine, define a name for the SQL Server machine's IP address in the Configuration Server machine's HOSTS file, `C:\WINNT\SYSTEM32\DRIVERS\ETC\HOSTS`, as in the following:



```
# Copyright (c) 1993-1995 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows NT.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
127.0.0.1          localhost
208.244.225.82     sqlsrv
```

Figure 6: Sample Configuration Server HOSTS file.

a SQL Server machine's entry in the HOSTS file

Third, you will need to have the name of the SQL Server database with which the Configuration Server will exchange data, the tables within that database to be used, and within those tables, the names of the fields that will participate in the data exchange.

Install Necessary Software

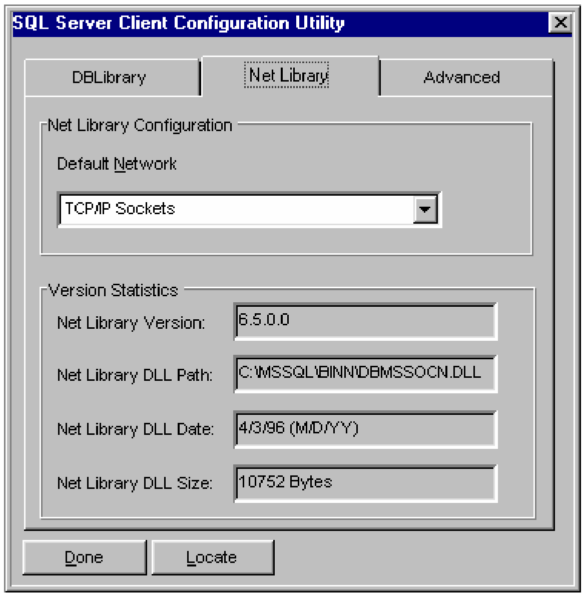
In order to configure a client machine (in this case, the machine that is running the Configuration Server) for connection to the machine running

SQL Server, the SQL Server Client Utilities must be installed on the Configuration Server machine.

To configure the Configuration Server as an SQL server client

After installing the SQL Server Client Utilities, run the SQL Server Client Configuration Utility.

1 Select the **Net Library** tab.



SQL Server Client Configuration Utility

DBLibrary | Net Library | Advanced

Net Library Configuration

Default Network: TCP/IP Sockets

Version Statistics

Net Library Version: 6.5.0.0

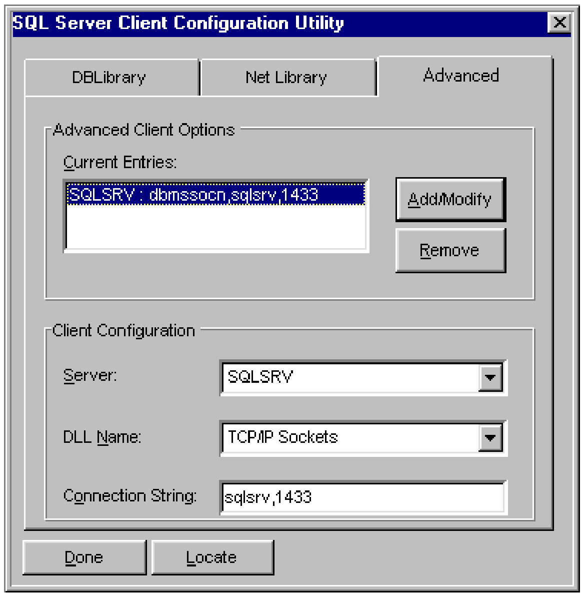
Net Library DLL Path: C:\MSSQL\BINN\DBMSSOCN.DLL

Net Library DLL Date: 4/3/96 (M/D/YY)

Net Library DLL Size: 10752 Bytes

2 Set **Default Network** to **TCP/IP Sockets**, as shown above.

3 Select the **Advanced** tab.



SQLSRV : dbmssocn,sqlsrv,1433

Server: SQLSRV

DLL Name: TCP/IP Sockets

Connection String: sqlsrv,1433

4 In the **Client Configuration** area:

— From the **Server** drop-down list, select **SQLSRV**.

— From the **DLL Name** drop-down list, select **TCP/IP Sockets**.

— Enter the SQL Server machine's IP address and port number (separated by a comma) in the **Connection String** text box. Since we have designated **sqlsrv** as the IP address of the Configuration Server (in the HOSTS file), we can refer to the IP address by that name.

5 Click **Add/Modify** to save the settings in the **Current Entries** list.

6 Click **Done** to exit the utility.

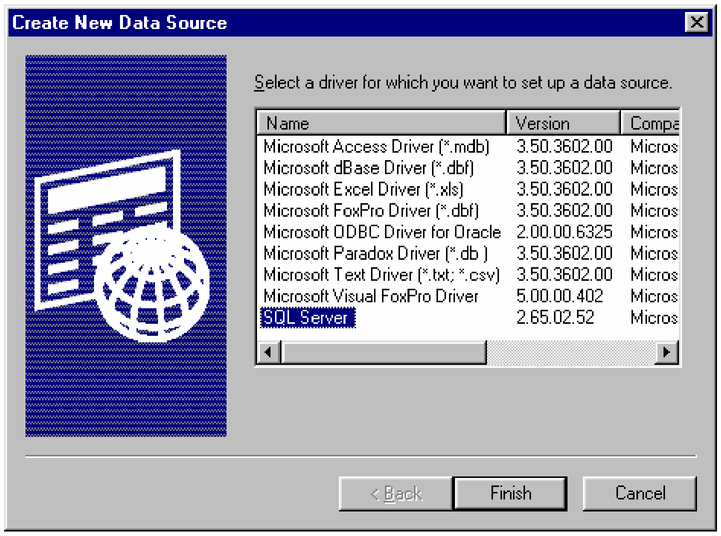
Create the SQL Server ODBC Data Source

1 Go to **Start→Settings→Control Panel** to open the **Control Panel** folder.

2 Double-click the **ODBC** icon to launch the **ODBC Data Source Administrator**.

3 If the Configuration Server is running as a Windows service, click the **System DSN** tab, and click **Add**.

A dialog box similar to the following opens.


Create New Data Source

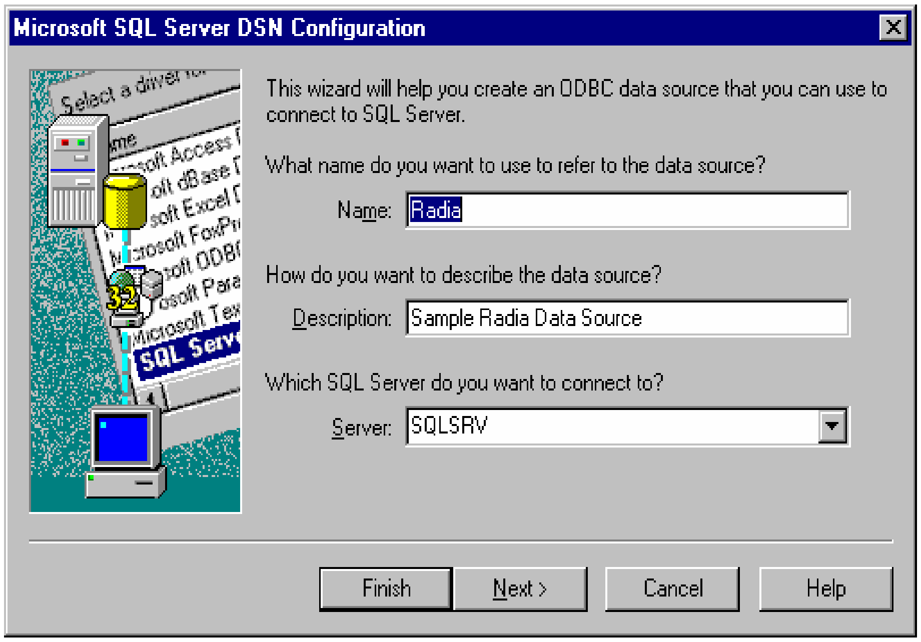
Select a driver for which you want to set up a data source.

Name	Version	Compa
Microsoft Access Driver (*.mdb)	3.50.3602.00	Micros
Microsoft dBase Driver (*.dbf)	3.50.3602.00	Micros
Microsoft Excel Driver (*.xls)	3.50.3602.00	Micros
Microsoft FoxPro Driver (*.dbf)	3.50.3602.00	Micros
Microsoft ODBC Driver for Oracle	2.00.00.6325	Micros
Microsoft Paradox Driver (*.db)	3.50.3602.00	Micros
Microsoft Text Driver (*.txt; *.csv)	3.50.3602.00	Micros
Microsoft Visual FoxPro Driver	5.00.00.402	Micros
SQL Server	2.65.02.52	Micros

4 Select the driver for the database you intend to use, and click **Finish**.

This invokes a wizard that leads you through the process of defining the ODBC data source.

The first dialog box of the wizard is similar to the following:


Microsoft SQL Server DSN Configuration

This wizard will help you create an ODBC data source that you can use to connect to SQL Server.

What name do you want to use to refer to the data source?

Name: Radia

How do you want to describe the data source?

Description: Sample Radia Data Source

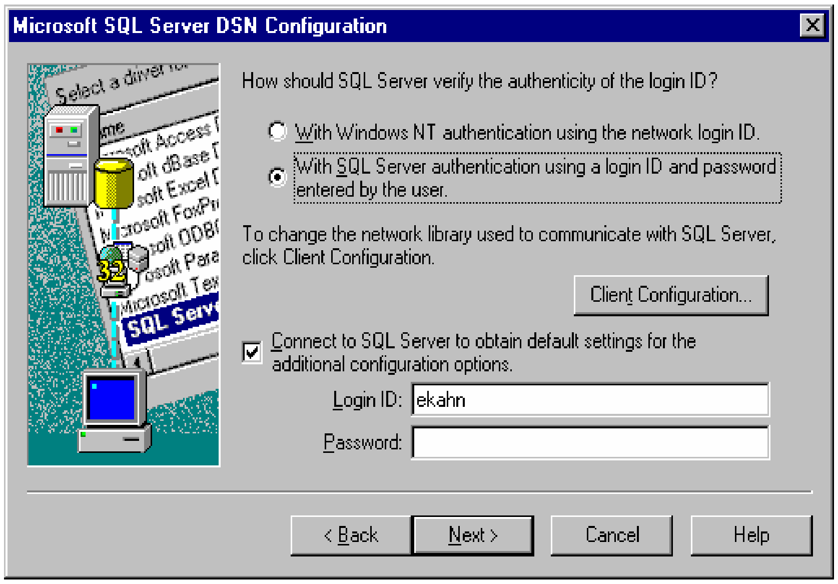
Which SQL Server do you want to connect to?

Server: SQLSRV

Table 98: **Microsoft SQL Server DSN Specifications**

Setting	Description
Data Source Name	Type a name to identify this ODBC data source to external programs, such as EDMMSQLG and EDMMSQLP. You will supply this **DSN** as a parameter to EDMMSQLG/EDMMSQLP, as described later in this document.
Description	Type a description that denotes the purpose or use of the data source definition. It identifies this data source when navigating the Control Panel ODBC applet.
Server	From the drop-down list, select the SQL Server with which the Configuration Server will exchange data.

5 Click **Next** to display the next dialog box of the wizard.



How should SQL Server verify the authenticity of the login ID?

- ○ With Windows NT authentication using the network login ID.
- ● With SQL Server authentication using a login ID and password entered by the user.

To change the network library used to communicate with SQL Server, click Client Configuration.

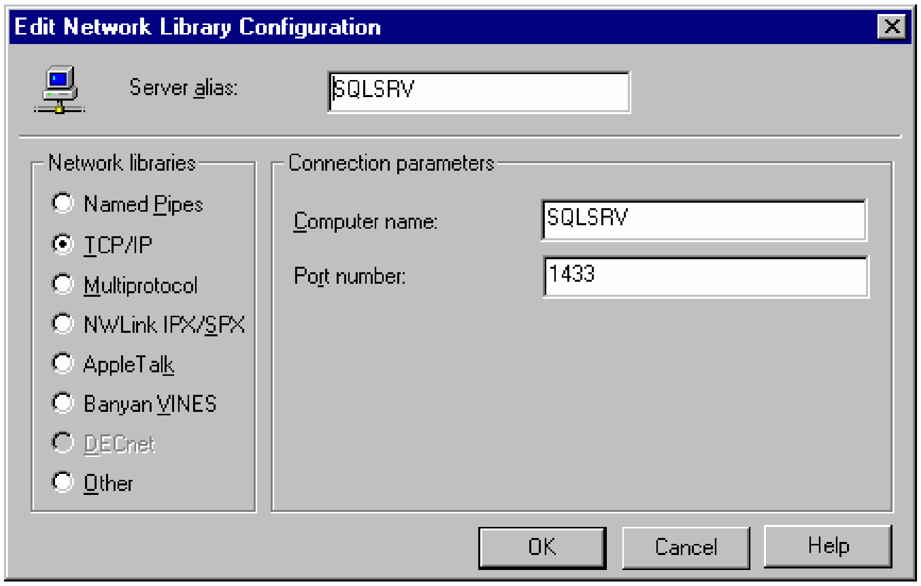
☑ Connect to SQL Server to obtain default settings for the additional configuration options.

Login ID: ekahn

Password:

6 Specify your **Login ID** and **Password** to connect to the SQL Server database.

To view or modify the network library used to communicate with SQL Server, click **Client Configuration**.


Edit Network Library Configuration

Server alias: SQLSRV

Network libraries: Named Pipes, [✓ TCP/IP], Multiprotocol, NWLink IPX/SPX, AppleTalk, Banyan VINES, DECnet, Other

Connection parameters

Computer name: SQLSRV

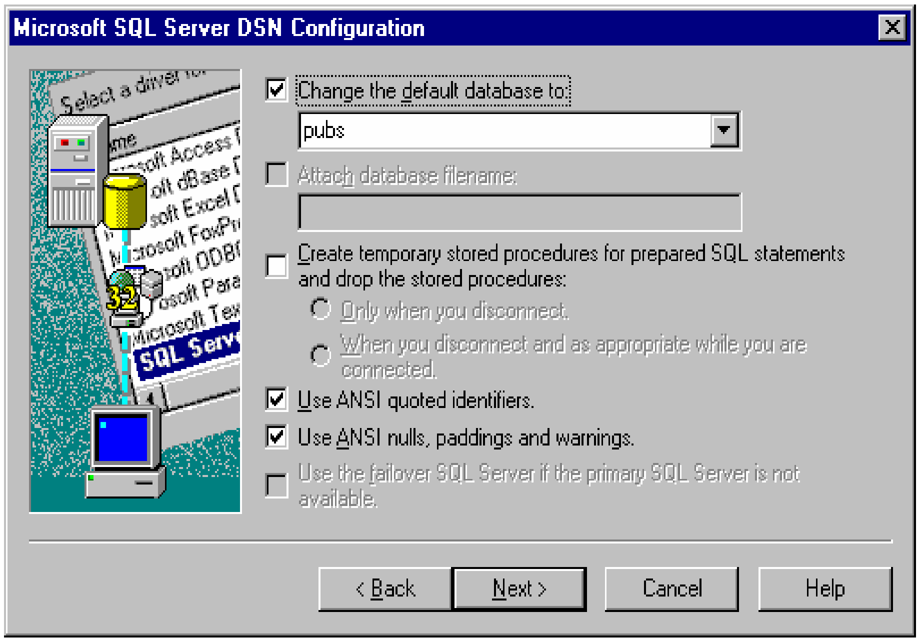
Port number: 1433

— In the **Connection parameters** group box, enter the SQL Server's IP address (**Computer name**) and **Port number**. Since we designated **SQLSRV** as the IP address of the Configuration Server (in the HOSTS file), we can refer to the IP address by that name.

— Click **OK** to save and return to the previous panel.

7 In the Microsoft SQL Server DSN Configuration dialog box, click **Next** to proceed.

The following dialog box opens.



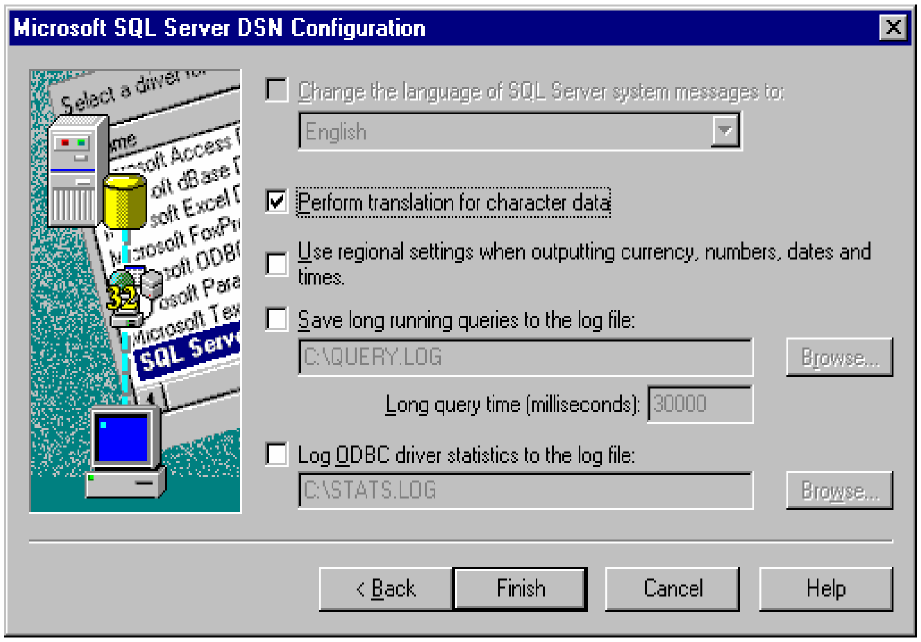
[✓] Change the default database to: pubs
[ ] Attach database filename:
[ ] Create temporary stored procedures for prepared SQL statements and drop the stored procedures:
( ) Only when you disconnect.
( ) When you disconnect and as appropriate while you are connected.
[✓] Use ANSI quoted identifiers.
[✓] Use ANSI nulls, paddings and warnings.
[ ] Use the failover SQL Server if the primary SQL Server is not available.

8 In the **Change the default database to** combo box, select or type the name of the SQL Server database.

Set the remaining controls in this dialog box based on the requirements of the selected database (see your SQL Server administrator for the necessary information).

9 Click **Next** to proceed.

The following dialog box opens.


Microsoft SQL Server DSN Configuration

☐ Change the language of SQL Server system messages to: English

☑ Perform translation for character data

☐ Use regional settings when outputting currency, numbers, dates and times.

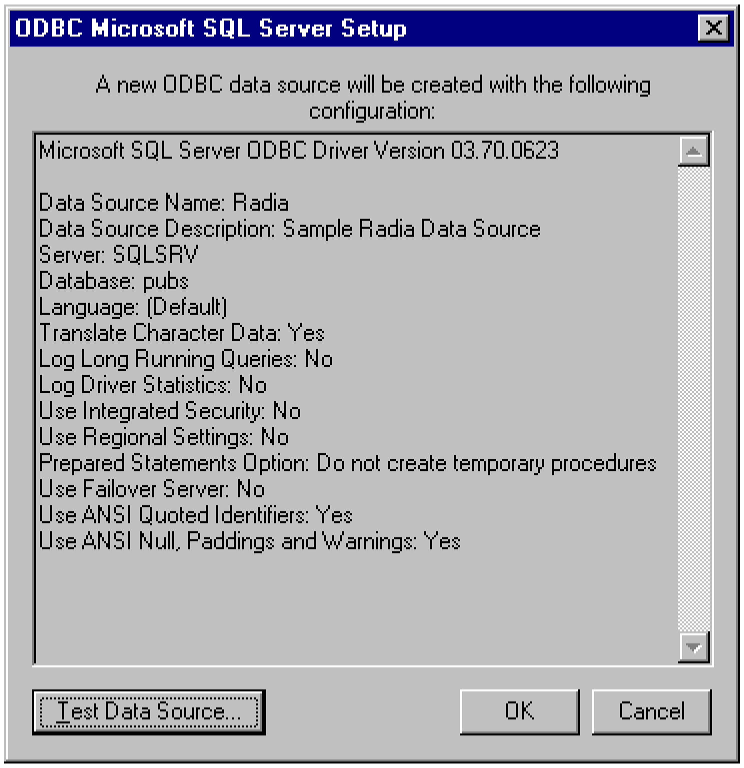
☐ Save long running queries to the log file: C:\QUERY.LOG

Long query time (milliseconds): 30000

☐ Log ODBC driver statistics to the log file: C:\STATS.LOG

10 Set the controls in this dialog box per the requirements of the selected database (see your SQL Server administrator for the necessary information).

11 Click **Finish** to proceed to the next dialog box, which summarizes the settings that you have specified.



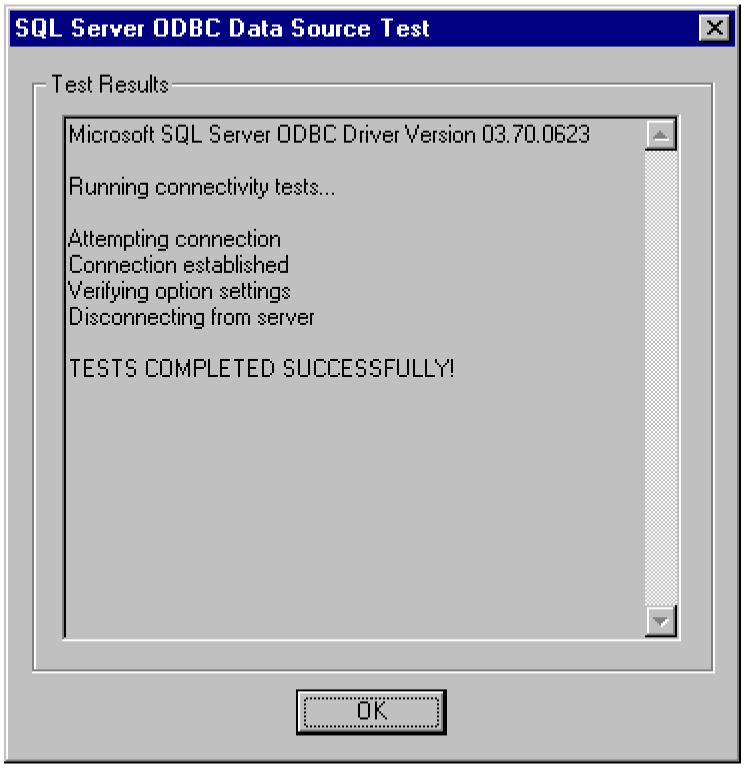
A new ODBC data source will be created with the following configuration:

```
Microsoft SQL Server ODBC Driver Version 03.70.0623

Data Source Name: Radia
Data Source Description: Sample Radia Data Source
Server: SQLSRV
Database: pubs
Language: (Default)
Translate Character Data: Yes
Log Long Running Queries: No
Log Driver Statistics: No
Use Integrated Security: No
Use Regional Settings: No
Prepared Statements Option: Do not create temporary procedures
Use Failover Server: No
Use ANSI Quoted Identifiers: Yes
Use ANSI Null, Paddings and Warnings: Yes
```

12 Review the settings and click **Test Data Source** to perform a test of your ODBC data source definition.

If you have correctly entered the information necessary to define the ODBC data source, the system will display a message similar to the following:



```
Microsoft SQL Server ODBC Driver Version 03.70.0623

Running connectivity tests...

Attempting connection
Connection established
Verifying option settings
Disconnecting from server

TESTS COMPLETED SUCCESSFULLY!
```

13 Click **OK** to save the data source definition and close the dialog box.

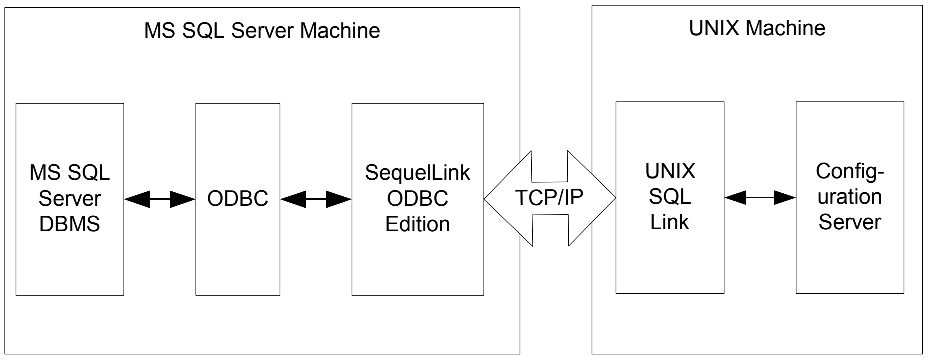
Microsoft SQL Server with UNIX Configuration Server

This section provides information about setting up ODBC on UNIX. Note that as installation directories vary from system to system, you must substitute the name of the installation directory on your system, where noted below.

Install Necessary Software

To build an interface to a Microsoft SQL Server database from the UNIX machine running the Configuration Server, you will need to acquire a piece of middleware, SequeLink ODBC Edition, and install it on the SQL Server host. This module is the interface between UNIX and the ODBC interface of the Microsoft SQL Server machine. SequeLink ODBC Edition is a product of Merant Plc.

Specifically:



Configure the Configuration Server machine as an SQL Server client, by following the steps outlined below.

1 On the UNIX host, a shell script has to be run prior to using the SQL Link. This shell script is located in the installation directory where SQL Link is installed.

For the Bourne or Korn Shell, issue the following command:

```
prompt> . <installation directory>/.sqlnk.sh
```

There is a <space><period><space> before `<installation directory>/.sqlnk.sh`.

For the C-Shell, issue the following command:

```
prompt> source <installation directory>/.sqlnk.csh
```

This shell script will set several environmental variables that are necessary to run the SQL Link interface.

2 Use the sqlnkcau utility to create a Data Source definition on the UNIX machine. This is required in order to access a database.

3 Change to the bin directory under the installation directory and enter the following command sequence to create a new definition.

```
Prompt> sqlnkcau
SequeLink Connect Administration Tool on HP-UX (ANSI)
(c)Copyright 1995-1998 INTERSOLV, Inc., All rights reserved
The following Data Source is selected:
        [1] Select a Data Source
        [2] New
        [7] About
        [0] Cancel
        Select an action [2]:           ←Select: New (2)
Name[]:                                 ←Enter the Data Source Name (e.g., Radia)
```

```
*Description[]:                       ←Enter a description (e.g., Sample Radia Data Source)
*Transliteration[]:                   ←Leave blank
The following network types are available:
        [1] TCP/IP
        Select a network []:          ←Select: TCP/IP (1)
Host[]:                               ←Enter host name or IP Address (SQLSRV)
The following server types are available:
        [1] AS/400
        [2] OS/390
        [3] UNIX
        [4] Windows NT
        Select a server []:           ←Select: Windows NT (4)
*User[]:                              ←Enter the Windows NT user ID
*Password[*****]:                     ←Enter the Windows NT Password for the user ID
The following service types are available:
        [1] DB2 on AS/400
        [2] DB2 on OS/390
        [3] DB2 on NT
        [4] DB2 on UNIX
        [5] INFORMIX on NT
        [6] INFORMIX on UNIX
        [7] Microsoft SQL Server
        [8] ODBC Btrieve
        [9] ODBC dBase
        [10] ODBC Excel
        [11] ODBC FoxPro
        [12] ODBC MS Access
        [13] ODBC Paradox
        [14] ODBC Socket
        [15] ODBC Text
        [16] OpenIngres
        [17] ORACLE
        [18] Progress
        [19] Sybase
        Select a service []:          ←Select: Microsoft SQL Server (7)
Name[]:                               ←Enter the default port number (4006) Or the name 'SLSQLServer' can be used
*Database[]:                          ←Enter the Database name (e.g., pubs)
User[]:                               ←Enter the SQL Server user ID
Password[*****]:                      ←Enter the SQL Server user ID password
```

4 Once the Data Source has been defined, test the access by using the Test command. The Data Source Radia is used in this example:

```
The following Data Source is selected:
        [1] Select a Data Source
        [2] New
        [7] About
        [0] Cancel
        Select an action [1]:                     ←Select: Select a Data Source (1)
The following SequeLink Data Sources are available:
        [1] Radia
        [0] Cancel
        Select a SequeLink Data Source [1]        ←Select: Radia (1)
```

```
The following Data Source is selected: Radia.
        [1] Select a Data Source
        [2] New
        [3] Duplicate
        [4] Edit
        [5] Delete
        [6] Test
        [7] About
        [0] Cancel
        Select an action [0]:                ←Select: Test (6)
Test passed: connection to 'Radia' made.
```

If all the parameters are set properly and a connection can be made to the database, the response shown above will be received. If there is a problem, an error code and message will be displayed. Refer to the Microsoft SQL Server and/or Merant (Intersolv) SequeLink documentation for an explanation of the error codes. If another database server is used, refer to the respective documentation.

5 Once the connection to the database has been established, it is necessary to define an ODBC source that points to the Data Source. This definition has to made directly to the ini file for the interface. This ini file is located in:

`<`*`installation directory`*`>/ini/.odbc.ini`.

This is a hidden file. To see a directory listing that includes this file, you must issue the following command:

```
ls -a
```

The following shows the changes you need to apply to the ini file. Copy the template entry, and edit it to reflect your data source.

```
[ODBC Data Sources]
        DataSourceName=INTERSOLV 3.10 SequeLink
        VFHSQL=INTERSOLV 3.10 SequeLink
[DataSourceName]
        Driver=/work/sqlnk/4_51_00/lib/ivslk13.sl
        Description=INTERSOLV 3.10 SequeLink
        SqlnkDSN=RADIA_DATA_SOURCE
        LogonID=
        Database=
        AllowBatchStatements=0
        UidPwdMapping=0                                  Template entry
        PreFetchRows=30
        EnableWarnings=0
        EnableScrollableCursors=0
        DataDictionary=(Default)
        DataDictionaryCatalog=
        DataDictionarySchema=
[RADIA]                                          ←ODBC Name
        Driver=/work/sqlnk/4_51_00/lib/ivslk13.sl ←Leave as default
        Description=Sample Radia Data Source      ←Database Description
        SqlnkDSN=Radia                            ←Data Source Name
        LogonID=                                  ←Leave as default
        Database=pubs                             ←Database name
        AllowBatchStatements=0                    ←Leave as default
        UidPwdMapping=0                           ←Leave as default
        PreFetchRows=30                           ←Leave as default
        EnableWarnings=0                          ←Leave as default
        EnableScrollableCursors=0                 ←Leave as default
        DataDictionary=(Default)                  ←Leave as default
        DataDictionaryCatalog=                    ←Leave as default
        DataDictionarySchema=                     ←Leave as default
[ODBC]
        Trace=0
        TraceFile=odbctrace.out
        TraceDll=/work/sqlnk/4_51_00/lib/odbctrac.sl
        InstallDir=/work/sqlnk/4_51_00
```

ODBC Reserved Words

ODBC reserved words are part of the ODBC syntax, and are poor choices for column names in the back-end databases tables. Table 99 below lists ODBC reserved words.

Table 99: ODBC Reserved Words

ABSOLUTE	ADA	ADA	ALL
ALLOCATE	ALTER	AND	ANY
ARE	AS	ASC	ASSERTION

AT	AUTHORIZATION	AVG	BEGIN
BETWEEN	BIT	BIT_LENGTH	BY
CASCADE	CASCADED	CASE	CAST
CATALOG	CHAR	CHAR_LENGTH	CHARACTER
CHARACTER_LENGTH	CHECK	CLOSE	COALESCE
COBOL	COLLATE	COLLATION	COLUMN
COMMIT	CONNECT	CONNECTION	CONSTRAINT
CONSTRAINTS	CONTINUE	CONVERT	CORRESPONDING
COUNT	CREATE	CURRENT	CURRENT_DATE
CURRENT_TIME	CURRENT_TIMESTAMP	CURSOR	DATE
DAY	DEALLOCATE	DEC	DECIMAL
DECLARE	DEFERRABLE	DEFERRED	DELETE
DESC	DESCRIBE	DESCRIPTOR	DIAGNOSTICS
DICTIONARY	DISCONNECT	DISPLACEMENT	DISTINCT
DOMAIN	DOUBLE	DROP	ELSE
END	END-EXEC	ESCAPE	EXCEPT
EXCEPTION	EXEC	EXECUTE	EXISTS
EXTERNAL	EXTRACT	FALSE	FETCH
FIRST	FLOAT	FOR	FOREIGN
FORTRAN	FOUND	FROM	FULL
GET	GLOBAL	GO	GOTO
GRANT	GROUP	HAVING	HOUR
IDENTITY	IGNORE	IMMEDIATE	IN
INCLUDE	INDEX	INDICATOR	INITIALLY
INNER INPUT	INSENSITIVE	INSERT	INTEGER
INTERSECT	INTERVAL	INTO	IS
ISOLATION	JOIN	KEY	LANGUAGE
LAST	LEFT	LEVEL	LIKE
LOCAL	LOWER	MATCH	MAX
MIN	MINUTE	MODULE	MONTH

MUMPS	NAMES	NATIONAL	NCHAR
NEXT	NONE	NOT	NULL
NULLIF	NUMERIC	OCTET_LENGTH	OF
OFF	ON	ONLY	OPEN
OPTION	OR	ORDER	OUTER
OUTPUT	OVERLAPS	PARTIAL	PASCAL
PLI	POSITION	PRECISION	PREPARE
PRESERVE	PRIMARY	PRIOR	PRIVILEGES
PROCEDURE	PUBLIC	RESTRICT	REVOKE
RIGHT	ROLLBACK	ROWS	SCHEMA
SCROLL	SECOND	SECTION	SELECT
SEQUENCE	SET	SIZE	SMALLINT
SOME	SQL	SQLCA	SQLCODE
SQLERROR	SQLSTATE	SQLWARNING	SUB-STRING
SUM	SYSTEM	TABLE	TEMPORARY
THEN	TIME	TIMESTAMP	TIMEZONE_HOUR
TIMEZONE_MINUTE	TO	TRANSACTION	TRANSLATE
TRANSLATION	TRUE	UNION	UNIQUE
UNKNOWN	UPDATE	UPPER	USAGE
USER	USING	VALUE	VALUES
VARCHAR	VARYING	VIEW	WHEN
WHENEVER	WHERE	WITH	WORK
YEAR			

Using HP SQL Methods

Overview

This section provides the information needed to enable the Configuration Server to exchange data with an ODBC-compliant foreign SQL database, using HP SQL methods. A foreign SQL database is one that is used by the HP SQL methods, but is neither created, supported, nor maintained by HP.

The HP SQL methods support only the following ODBC-compliant databases:

- MS SQL
- Oracle
- Sybase

The HP SQL Methods

- Are tools with which you can add, update, and retrieve SQL database information.
- Work with single- and multi-heap objects, updating and inserting an equivalent number of rows for as many heaps exist in the source object.
- Recognize and sort character strings, integers, decimals, and date/time input.
- Are sensitive to column data-types in an SQL database.

The EDMMSQL**G** (get) method imports data from an external database to an in-storage object, and is useful for influencing the Configuration Server's resolution process with data from an external source. See Figure 7 on page 222.

The EDMMSQL**P** (put) method exports data to an external database, and is useful for delivering data to an external sub-system for reporting and other purposes. See Figure 7 on page 222.

Additionally, this section details the proper use of the WHERE clause within control objects.

EDMMSQLG Method

The EDMMSQLG method provides users with a tool to extract data from a customer-specified SQL database table, and import it, via an ODBC connection, to an in-storage object, at a point in the Configuration Server resolution process that has been defined by the administrator. The data to be imported can be contained in any ODBC-compliant database.

EDMMSQLG is a Configuration Server method. Therefore, by establishing a connection to this method in the distribution model for one or more desktop computers under management, an administrator can have it invoked during the client connect process.

Each invocation of EDMMSQLG executes an SQL SELECT statement that retrieves data from the ODBC-compliant database, and stores the result in an in-storage object. There will be one heap in the resulting in-storage object for each row in the resulting set.

EDMMSQLP Method

The EDMMSQLP method provides users with a tool to extract data from the Radia database and store it in an external database table, at a point in the resolution process that has been defined by the administrator. The exported data can be stored in any ODBC-compliant database.

EDMMSQLP is a Configuration Server method. Therefore, by establishing a connection to this method in the distribution model for one or more desktop computers under management, an administrator can have it invoked during the client connect process.

Each invocation of EDMMSQLP executes an SQL INSERT or UPDATE statement to insert (or replace) data in the ODBC-compliant database. The data is taken from an in-storage object. In the external database table, there will be one row inserted (or replaced) for each heap in the in-storage data source object.

Figure 7 on page 222, presents a graphical overview of the HP SQL methods processes.

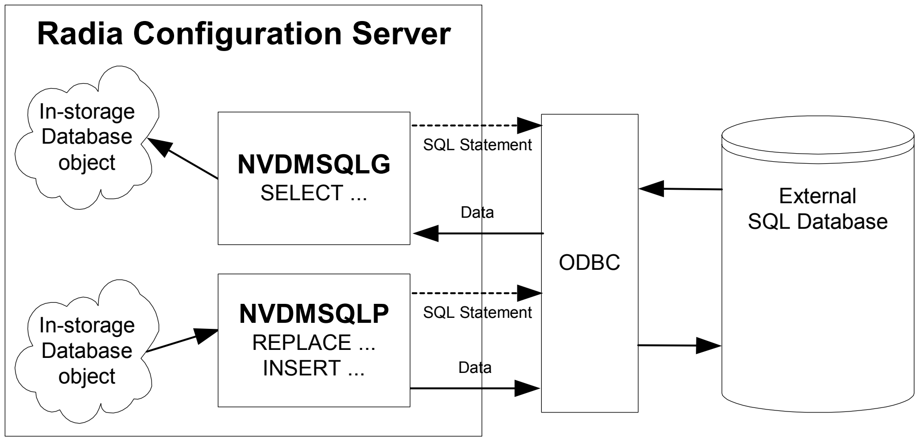


Figure 7: An overview of the EDMMSQLG and EDMMSQLP methods processes.

EDMMSQLG takes (gets) data from the SQL table and writes it to variables of the destination object. The mapping relationships between the columns of the table and the variables of the destination object are defined in the control information.

EDMMSQLP takes specified variables from the source object and writes them into the SQL database. It works with the keywords REPLACE and INSERT, as described in the parameter PUTTYPE in Table 100 below.

Because EDMMSQLG and EDMMSQLP are generic tools for transferring data between an in-storage object and a back-end database, you can make creative use of their capabilities in order to meet your organization's unique requirements.

The following table defines the keywords that are accepted by the HP SQL methods.

Table 100: Keywords Accepted by the HP SQL Methods

Keyword	**Description**	**Method Put / Get**
CTRLFILE	Name of the file that contains the control information. If this parameter is found, the parsing of the parameter string stops, and all the control information will be read from the specified file.	Put and Get

Keyword	Description	Method Put / Get
CTRLOBJ	Name of the HP object that contains the control information. If this parameter is found, the parsing of the parameter string stops, and all the control information will be received from the specified object.	Put and Get
SRCOBJ	Name of the HP source object.	Put
DESTOBJ	Name of the HP destination object. This object is used in read type methods.	Get
SQLDSN	*Data Source Name* (DSN) used on the Configuration Server to connect to the user database.	Put and Get
SQLTABLE	The name of the SQL table to deal with in the method.	Put and Get
SQLUSER	The user ID to use in the database connect process.	Put and Get
SQLPASSW	The password to use in the database connect process.	Put and Get
SQLTOUT	Time-out value for the SQL connect operation.	Put and Get
VC	Defines one VARIABLE-COLUMN (VC) pair. There might be more than one VC keyword in the parameter string. One VC value must be specified for each VARIABLE-COLUMN pair participating in the operation. For more information, refer to the section, VARIABLE-COLUMN Pairs, on page 256.	Put and Get
WHERE	This defines the search criteria for the WHERE clause. The format is COLUMN_NAME=value, COLUMN_NAME=value, etc. For more information, refer to the section, The WHERE Clause, on page 264.	Put and Get
PUTTYPE	{R, I} Type of Put operation requested, REPLACE or INSERT. When REPLACE is specified, the method will try to update the existing row first. If the row does not exist, the method will attempt to INSERT it. When INSERT is specified, the method will try to insert the row. If this operation fails, no other action is taken. The default is R.	Put

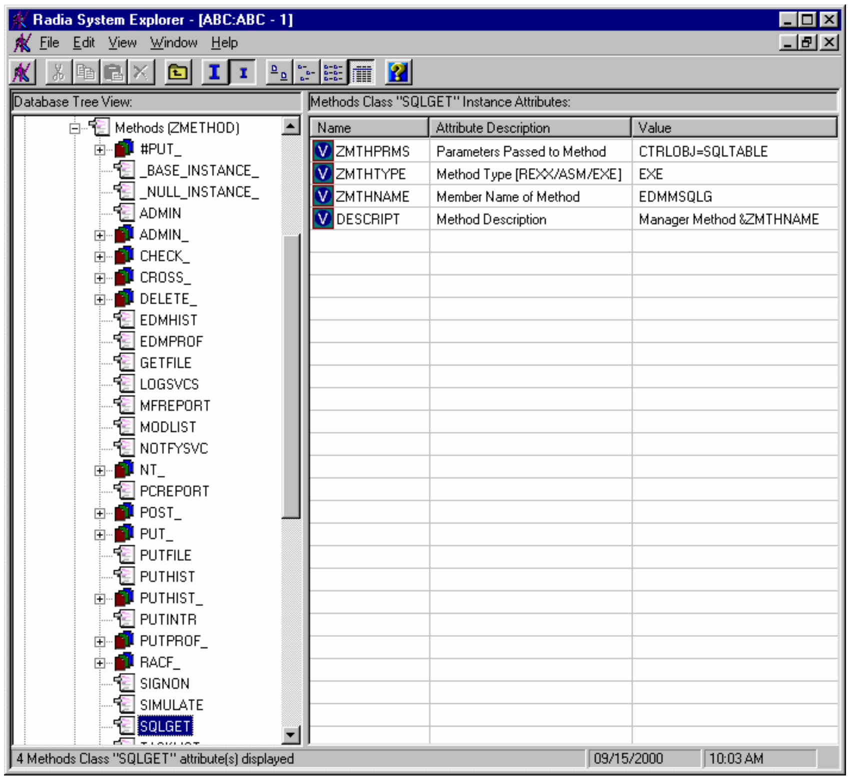
In order for the HP SQL methods to work, they must be configured for execution in the Radia database. During the resolution process, the method is executed and the control information is passed as a parameter string. Generally, the HP SQL methods deals with three types of information:

- control information,
- source (or destination) object information, and
- SQL database information.

Defining EDMMSQLG and EDMMSQLP as Configuration Server Methods

Before you can place a connection to the EDMMSQLG and EDMMSQLP methods in your distribution model, you must define an instance in the ZMETHOD class of the SYSTEM domain for each method. The instance contains the information needed to execute the SQL method.

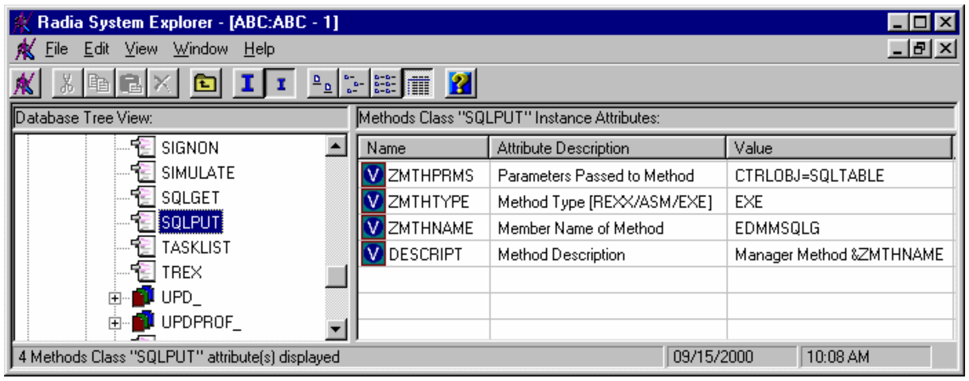
Figure 8 on page 225 shows an example (as seen in Radia System Explorer) of a ZMETHOD instance properly configured to executing the EDMMSQLG method.



Name	Attribute Description	Value
ZMTHPRMS	Parameters Passed to Method	CTRLOBJ=SQLTABLE
ZMTHTYPE	Method Type [REXX/ASM/EXE]	EXE
ZMTHNAME	Member Name of Method	EDMMSQLG
DESCRIPT	Method Description	Manager Method &ZMTHNAME

Figure 8: SQLGET instance attributes.

Figure 9 on page 226 shows an example (as seen in System Explorer) of a ZMETHOD instance properly configured to executing the EDMMSQLP method.


Radia System Explorer - [ABC:ABC - 1]
File Edit View Window Help
Database Tree View:
SIGNON
SIMULATE
SQLGET
SQLPUT
TASKLIST
TREX
UPD_
UPDPROF_
Methods Class "SQLPUT" Instance Attributes:

Name	Attribute Description	Value
ZMTHPRMS	Parameters Passed to Method	CTRLOBJ=SQLTABLE
ZMTHTYPE	Method Type [REXX/ASM/EXE]	EXE
ZMTHNAME	Member Name of Method	EDMMSQLG
DESCRIPT	Method Description	Manager Method &ZMTHNAME

4 Methods Class "SQLPUT" attribute(s) displayed 09/15/2000 10:08 AM

Figure 9: SQLPUT instance attributes.

You can name the instances whatever you wish, consistent with the instance naming conventions used by your organization (in our examples, SQLGET and SQLPUT).

All of the variables in the instance should appear as shown here, except for ZMTHPRMS, the variable that contains or identifies the control information passed to the EDMMSQLG/EDMMSQLP method when it executes. There are a number of ways to pass control information to EDMMSQLG/EDMMSQLP. They are described in the next section.

How to Invoke EDMMSQLG

This section provides information needed to invoke EDMMSQLG, and provides some examples.

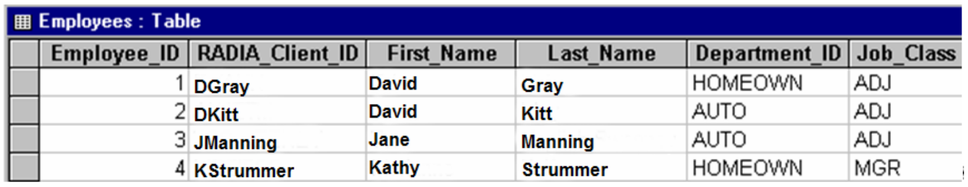
Refer to the instructions for creating an instance in the *System Explorer Guide*. Before invoking the method, at least one instance must be defined in the Radia database. You can define multiple instances to invoke EDMMSQLG, where each instance (with its unique name) refers to a different set of control information in its ZMTHPRMS variable.

To provide policy data from an external database

In this example, an external database contains information defining what type of user the client is, and therefore, what set of applications the user should receive. For example, an insurance company might have hundreds of claims adjusters for whom Radia manages a suite of identical applications. Rather than define an instance for each claims adjuster in the Radia database USER class, a generic USER instance will serve to link all adjusters to their appropriate application suite, based on the type of claims they adjust. An external database is used to look up the client's identity, and return an identifier that is then used to select the appropriate generic USER instance for the client.

The back-end database is Microsoft Access, and the Configuration Server is running under Windows NT.

The Microsoft Access database is named PERSONNEL.MDB. It contains a table (EMPLOYEES) that contains the data to be extracted by EDMMSQLG. The EMPLOYEES table looks like this:


Employees : Table

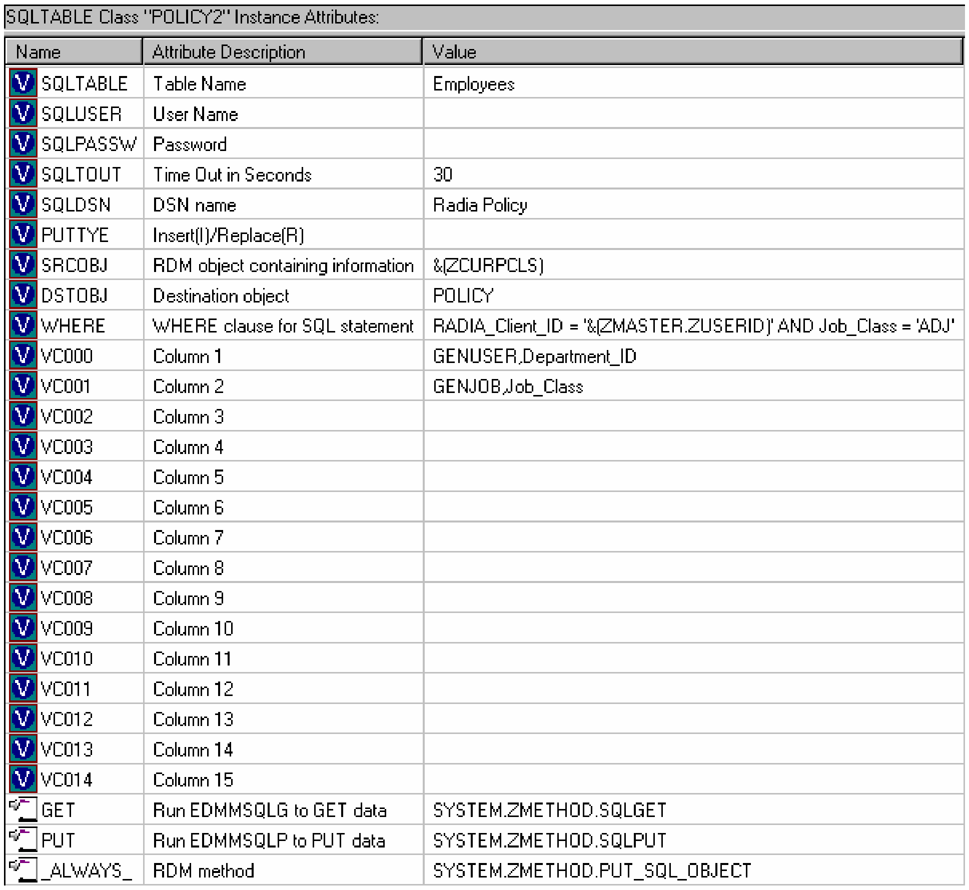
Employee_ID	RADIA_Client_ID	First_Name	Last_Name	Department_ID	Job_Class
1	DGray	David	Gray	HOMEOWN	ADJ
2	DKitt	David	Kitt	AUTO	ADJ
3	JManning	Jane	Manning	AUTO	ADJ
4	KStrummer	Kathy	Strummer	HOMEOWN	MGR

Figure 10: The Employees table of the Microsoft Access database.

We'll use EDMMSQLG to look up, in the EMPLOYEES table's **RADIA_Client_ID** column, the user ID provided by the user in the client-connect login process. If it exists in a record who's **Job_Class** is equal to ADJ, we'll extract the **Department_ID** field. **Department_ID** will then be used in the Configuration Server resolution process to select a generic USER instance to provide the appropriate set of applications for the end user.

The generic USER instance will be named **AUTO** for automobile insurance adjusters and **HOMEOWN** for homeowner's insurance adjusters.

We've defined an ODBC Data Source named Radia Policy to specify an ODBC connection to this database. Once the ODBC Data Source exists, create an instance of the SQLTABLE class to provide a control object for EDMMSQLG, as in Figure 11 on page 228.


SQLTABLE Class "POLICY2" Instance Attributes:

Name	Attribute Description	Value
SQLTABLE	Table Name	Employees
SQLUSER	User Name	
SQLPASSW	Password	
SQLTOUT	Time Out in Seconds	30
SQLDSN	DSN name	Radia Policy
PUTTYE	Insert(I)/Replace(R)	
SRCOBJ	RDM object containing information	&(ZCURPCLS)
DSTOBJ	Destination object	POLICY
WHERE	WHERE clause for SQL statement	RADIA_Client_ID = '&(ZMASTER.ZUSERID)' AND Job_Class = 'ADJ'
VC000	Column 1	GENUSER,Department_ID
VC001	Column 2	GENJOB,Job_Class
VC002	Column 3	
VC003	Column 4	
VC004	Column 5	
VC005	Column 6	
VC006	Column 7	
VC007	Column 8	
VC008	Column 9	
VC009	Column 10	
VC010	Column 11	
VC011	Column 12	
VC012	Column 13	
VC013	Column 14	
VC014	Column 15	
GET	Run EDMMSQLG to GET data	SYSTEM.ZMETHOD.SQLGET
PUT	Run EDMMSQLP to PUT data	SYSTEM.ZMETHOD.SQLPUT
ALWAYS	RDM method	SYSTEM.ZMETHOD.PUT_SQL_OBJECT

Figure 11: SQLTABLE Class POLICY2 Instance Attributes.

We've named this instance POLICY2. This instance should be named in accordance with your organization's convention for naming Radia instances.

- The SQLTABLE variable identifies the database table that EDMMSQLG will access (in this example, Employees).
- The SQLDSN variable specifies which ODBC Data Source to use (in this example, Radia Policy).
- As a result of its query, EDMMSQLG will produce an in-storage object (in this example, POLICY) as specified in the DESTOBJ variable.

Examine the WHERE variable. Note the symbolic substitution using &(ZMASTER.ZUSERID). At logon, the end user provides a user ID (and, optionally, a password) in the Radia logon dialog box. The user ID is transmitted to the Configuration Server in the ZMASTER object ZUSERID attribute at the beginning of the client-connect process.

This example uses this user ID to retrieve a record from the Microsoft Access Personnel database EMPLOYEES table, identifying the type of user. The WHERE clause retrieves any record whose RADIA_Client_ID is equal to the user ID supplied by the end user, and whose Job_Class is ADJ.

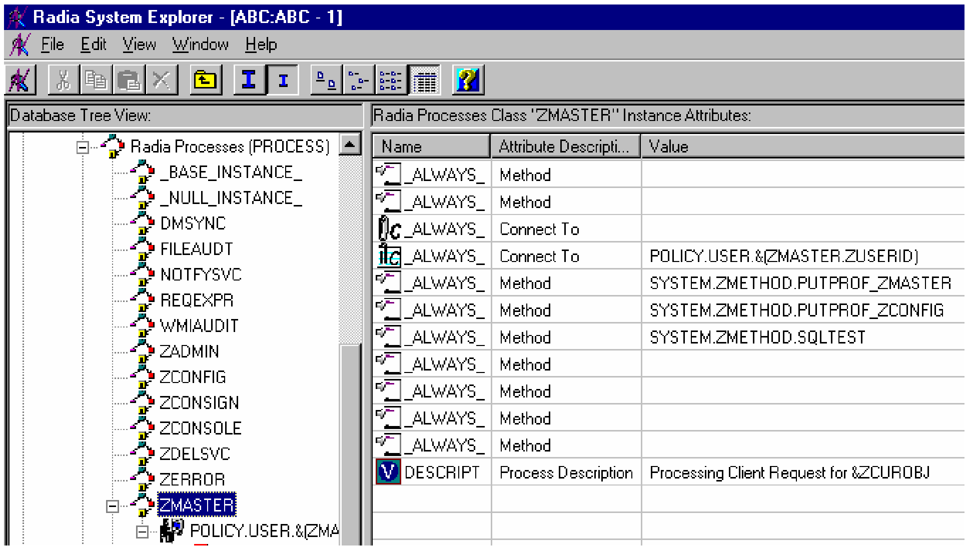
The specifications for RADIA_Client_ID and Job_Class must be enclosed in single quotes (‘ ’). This is an ODBC requirement.

Note: Quotation marks will not work.

Some databases do not permit embedded spaces in column names, while others (like Microsoft Access) do. Notice (in Figure 11 on page 228) that the names of the columns from which EDMMSQLG retrieves data have no embedded spaces.

EDMMSQLG is limited to retrieving data from columns with a name that does not contain embedded spaces.

Connect the SYSTEM.PROCESS.ZMASTER instance to this SQLTABLE instance as shown in Figure 27 on page 243.


Radia System Explorer - [ABC:ABC - 1]
File Edit View Window Help
Database Tree View:
Radia Processes Class "ZMASTER" Instance Attributes:

Radia Processes (PROCESS)
- _BASE_INSTANCE_
- _NULL_INSTANCE_
- DMSYNC
- FILEAUDT
- NOTFYSVC
- REQEXPR
- WMIAUDIT
- ZADMIN
- ZCONFIG
- ZCONSIGN
- ZCONSOLE
- ZDELSVC
- ZERROR
- ZMASTER
  - POLICY.USER.&(ZMA

Name	Attribute Descripti...	Value
ALWAYS	Method	
ALWAYS	Method	
ALWAYS	Connect To	
ALWAYS	Connect To	POLICY.USER.&(ZMASTER.ZUSERID)
ALWAYS	Method	SYSTEM.ZMETHOD.PUTPROF_ZMASTER
ALWAYS	Method	SYSTEM.ZMETHOD.PUTPROF_ZCONFIG
ALWAYS	Method	SYSTEM.ZMETHOD.SQLTEST
ALWAYS	Method	
ALWAYS	Method	
ALWAYS	Method	
ALWAYS	Method	
DESCRIPT	Process Description	Processing Client Request for &ZCUROBJ

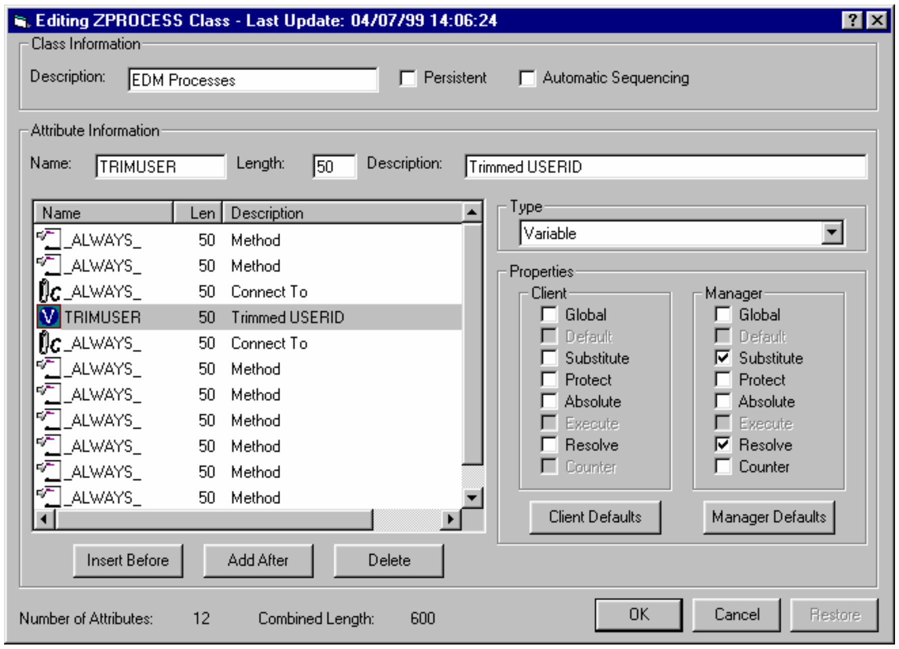
Figure 12: Radia Processes Class ZMASTER Instance Attributes.

The connection is SYSTEM.SQLTABLE.POLICY2(GET). It sets the system message value to GET, so that the POLICY2 instance will execute EDMMSQLG, not EDMMSQLP.

The POLICY object GENUSER attribute contains the value that EDMMSQLG retrieved from the **Department_ID** column for the user who signed on to the client connect: either **AUTO** or **HOMEOWN.** This value is then substituted into the USER class connection that immediately follows, connecting either to USER.AUTO or USER.HOMEOWN, depending on what type of adjuster the end user is.

This example would require only two USER class instances in order to service all (auto and homeowner's) adjusters. Since ZMASTER.ZUSERID is not affected by the design of this example, individual Profile file domains are stored for each user who connects, despite the fact that a generic set of applications is being supplied him.

We've modified the ZPROCESS (PROCESS) class to include a TRIMUSER variable, as shown in the ZPROCESS class template in Figure 13 on page 231.


Editing ZPROCESS Class - Last Update: 04/07/99 14:06:24

Class Information

Description: EDM Processes   Persistent [ ]   Automatic Sequencing [ ]

Attribute Information

Name: TRIMUSER   Length: 50   Description: Trimmed USERID

Name	Len	Description
ALWAYS	50	Method
ALWAYS	50	Method
ALWAYS	50	Connect To
TRIMUSER	50	Trimmed USERID
ALWAYS	50	Connect To
ALWAYS	50	Method
ALWAYS	50	Method
ALWAYS	50	Method
ALWAYS	50	Method
ALWAYS	50	Method
ALWAYS	50	Method

Type: Variable

Properties

Client	Manager
Global [ ]	Global [ ]
Default [ ]	Default [ ]
Substitute [ ]	Substitute [✓]
Protect [ ]	Protect [ ]
Absolute [ ]	Absolute [ ]
Execute [ ]	Execute [ ]
Resolve [ ]	Resolve [✓]
Counter [ ]	Counter [ ]

Number of Attributes: 12   Combined Length: 600

Figure 13: The Editing ZPROCESS Class dialog box.

We need to do this because we intend to use symbolic substitution to refer to a variable in the object created by EDMMSQLG. EDMMSQLG stores data retrieved from a text field in a back-end database, in an object variable whose length is 255, regardless of the size defined for the field in the back-end database. If we tried to symbolically substitute a 255-character field into part of another attribute, symbolic substitution would fail with a buffer overflow. Thus, the purpose of the TRIMUSER attribute is to reduce the length of the data retrieved from the back-end database to a size that can be successfully symbolically substituted in the following attribute.

Two examples from the Configuration Server log illustrate this. First, the sample code below contains an excerpt from the log showing the buffer overflow that occurs when we try to symbolically substitute the object value created from the back-end database.

```
Radia Client     ---RESOLUTION ENDS:  SQLTABLE.POLICY   CRC:00000000
Radia Client     ---Substituting SYSTEMX.USER.&(POLICY.GENUSER)(EDMSETUP)
Radia Client     ---Passing to Substitution ...: [&(POLICY.GENUSER)]
Radia Client     ---PASSED TO SUBSTITUTION..: &(POLICY.GENUSER)
Radia Client     ---GET POLICY  .GENUSER  (1)  (255)  'AUTO'
Radia Client     ---BACK FROM SUBSTITUTION...: 255 [AUTO]
Radia Client     --! Substitution buffer overflow
Radia Client    --! SUBSTITUION FAILURE
Radia Client    ---Substitution Failed [SYSTEMX.USER.&(POLICY.GENUSER)(EDMSETUP)
```

The sample of code below presents an excerpt from the Configuration Server log showing successful symbolic substitution of the data obtained from the Microsoft Access database, when we trim its length first in the TRIMUSER variable:

```
Radia Client    --- RESOLUTION ENDS: SQLTABLE.POLICY
Radia Client    --- Passing to Substitution ...: [&(POLICY.GENUSER)]
Radia Client    --- PASSED TO SUBSTITUTION..: &(POLICY.GENUSER)]
Radia Client    --- GET POLICY  .GENUSER  (1) (255) 'AUTO'
Radia Client    --- BACK FROM SUBSTITUTION...: 255 [AUTO]
Radia Client    --- AFTER SUBSTITUTION [AUTO]
Radia Client    --! Subst. Value Truncated ZPROCESS.TRIMUSER (1) Actual (255) Allocated (50)
                    'AUTO'
Radia Client    --- ADD ZPROCESS.TRIMUSER  (1) (50) 'AUTO'
Radia Client    --- Substituting SYSTEMX.USER.&TRIMUSER(EDMSETUP)
Radia Client    --- Passing to Substitution ...: [&TRIMUSER]
Radia Client    --- PASSED TO SUBSTITUTION..: &TRIMUSER
Radia Client    --- GET ZPROCESS.TRIMUSER (1) (50) 'AUTO'
Radia Client    --- BACK FROM SUBSTITUTION...: 50 [AUTO]
Radia Client    --- AFTER SUBSTITUTION [SYSTEMX.USER.AUTO]
Radia Client    --- SUBSTITUTION VALUE [SYSTEMX.USER.AUTO]
Radia Client    --- Substituted value SYSTEMX.USER.AUTO
Radia Client    --- MESSAGE CHANGES USER     AUTO (        ) (EDMSETPU)
Radia Client    --- RESOLUTION BEGINS USER    .AUTO (EDMSETUP)
```

In the ZPROCESS class template, the Manager, Global property is not selected for the TRIMUSER variable (see Figure 13 on page 231). There is no need to preserve the TRIMUSER variable in a parent persistent object, because it is used only as temporary storage to reduce the length of the data retrieved from the back-end database.

Also note that the Manager, Resolve property is selected for the TRIMUSER variable. This assures that symbolic substitution will occur.

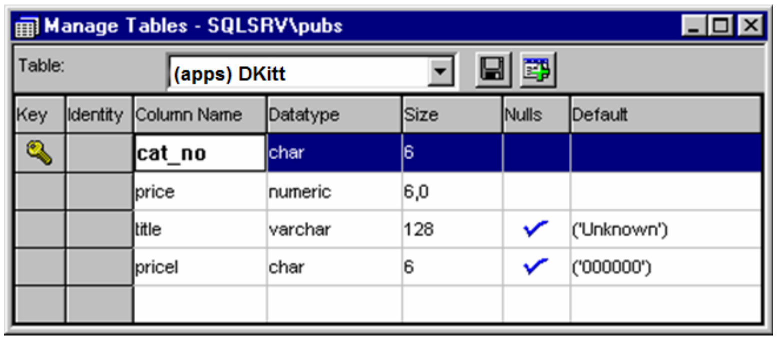
To extract pricing data from an external database

This example illustrates pricing content distributed by Radia according to pricing records maintained in an external database. EDMMSQLG is used to price each unit of content that Radia distributes. Radia totals the prices for all content delivered during a client connect, and EDMMSQLP reports the results to an external billing system.

The pricing data are kept in a Microsoft SQL Server database table, the Configuration Server is running Windows NT, and Radia stores the billing data in a Microsoft FoxPro table.

For this example, we will use the Radia data source for Microsoft SQL Server as described beginning on page 203, and the Radia data source for Microsoft FoxPro as described beginning on page 198.

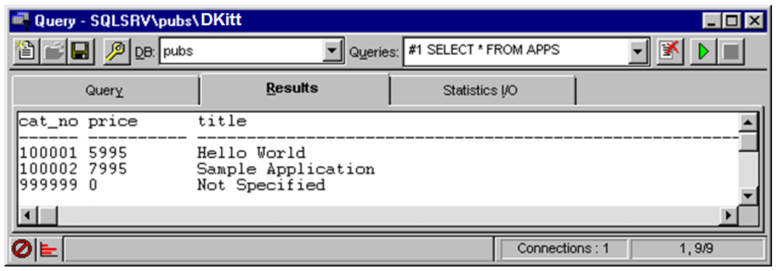
The Microsoft SQL Server pubs database table (apps) holds the pricing data. The format of this table is shown in Figure 14 below.



Key	Identity	Column Name	Datatype	Size	Nulls	Default
🔑		cat_no	char	6		
		price	numeric	6,0		
		title	varchar	128	✓	('Unknown')
		pricel	char	6	✓	('000000')

Figure 14: Microsoft SQL Server – SQLSRV\pubs.

This table contains the following data:

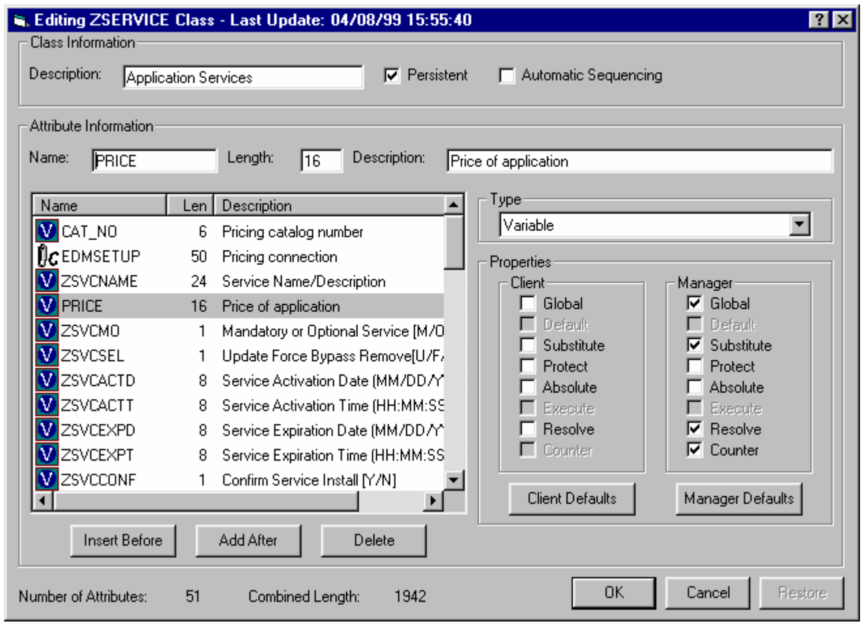


```
cat_no price      title
------ ---------- ------------------------------------------------------------
100001 5995       Hello World
100002 7995       Sample Application
999999 0          Not Specified
```

Figure 15: Microsoft SQL Server – SQLSRV\pubs\dkitt.

For this example, each service that Radia manages has a catalog number (cat_no). We will use EDMMSQLG to look up the service's catalog number in the apps table, and extract the price into a Radia object.

To implement this design, we added three attributes to the ZSERVICE class template:



Editing ZSERVICE Class - Last Update: 04/08/99 15:55:40

Class Information

Description: Application Services ☑ Persistent ☐ Automatic Sequencing

Attribute Information

Name: PRICE Length: 16 Description: Price of application

Name	Len	Description
CAT_NO	6	Pricing catalog number
EDMSETUP	50	Pricing connection
ZSVCNAME	24	Service Name/Description
PRICE	16	Price of application
ZSVCMO	1	Mandatory or Optional Service [M/O
ZSVCSEL	1	Update Force Bypass Remove[U/F/
ZSVCACTD	8	Service Activation Date (MM/DD/Y
ZSVCACTT	8	Service Activation Time (HH:MM:SS
ZSVCEXPD	8	Service Expiration Date (MM/DD/Y
ZSVCEXPT	8	Service Expiration Time (HH:MM:SS
ZSVCCONF	1	Confirm Service Install [Y/N]

Type: Variable

Properties

Client	Manager
☐ Global	☑ Global
☐ Default	☐ Default
☐ Substitute	☑ Substitute
☐ Protect	☐ Protect
☐ Absolute	☐ Absolute
☐ Execute	☐ Execute
☐ Resolve	☑ Resolve
☐ Counter	☑ Counter

Number of Attributes: 51 Combined Length: 1942

Figure 16: The Editing ZSERVICE Class dialog box.

- The CAT_NO attribute holds the catalog number for the service. EDMMSQLG will look up this value in the SQL Server database.
- The EDMSETUP connection attribute (Pricing connection) holds a connection to the SQLTABLE instance that invokes EDMMSQLG.
- The PRICE attribute stores, for this service, the price value that EDMMSQLG extracts from the SQL Server database.

  Note that the PRICE attribute has the **Global**, **Manager**, **Resolve**, and **Counter** properties selected.

  — **Resolve** enables symbolic substitution of the price, by reference to the object that EDMMSQLG creates to contain the price value extracted from the SQL Server database.

  — **Counter** indicates that the PRICE attribute's value will be accumulated in an attribute named PRICE in all parent persistent objects. This accomplishes summation of the price of all services managed for each user into a PRICE variable in each user's

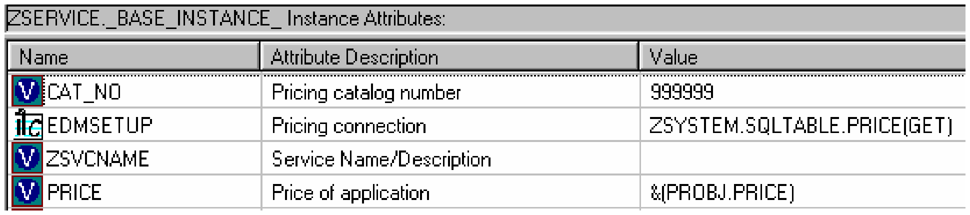
ZMASTER object. Radia automatically creates the PRICE attribute in ZMASTER (parent persistent) object when a child object (in this case, ZSERVICE) contains a PRICE attribute with the Counter property.

Radia counter fields are treated as integers. Therefore, the price in the SQL database must be expressed in cents.

The **Pricing connection** EDMSETUP attribute connects to SQLTABLE.PRICE.

EDMMSQLG looks up the ZSERVICE instance CAT_NO value in the SQL Server database, and creates a PROBJ object containing a PRICE variable that stores the price of the service retrieved from the SQL Server database.

To implement this design, we modify the base instance of the ZSERVICE class as follows:



ZSERVICE._BASE_INSTANCE_ Instance Attributes:

Name	Attribute Description	Value
CAT_NO	Pricing catalog number	999999
EDMSETUP	Pricing connection	ZSYSTEM.SQLTABLE.PRICE(GET)
ZSVCNAME	Service Name/Description	
PRICE	Price of application	&(PROBJ.PRICE)

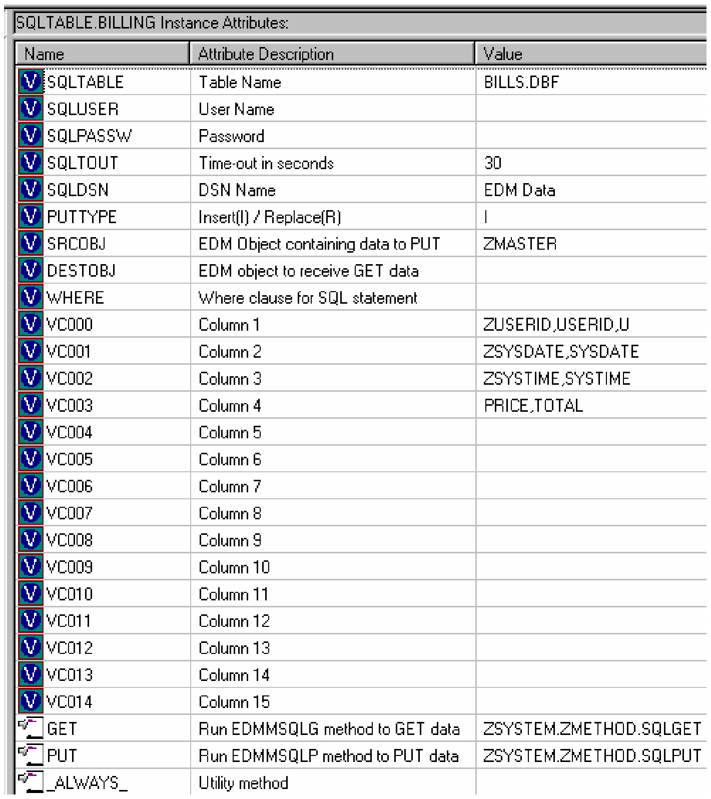
Figure 17: ZSERVICE._BASE_INSTANCE_ instance attributes.

If no value is provided for CAT_NO in a ZSERVICE instance, the base instance CAT_NO will default to 999999.

The EDMSETUP attribute **Pricing connection** is set to connect to the SQLTABLE.PRICE instance, providing a control object for EDMMSQLG, and invoking the method by setting the system message to GET.

The PRICE attribute retrieves the service's price from the PROBJ object by symbolic substitution.

During a client connect, as each of the user's services is resolved, EDMMSQLG is invoked to retrieve the price from the SQL Server database. Radia accumulates the prices of all of the user's services in the user's ZMASTER object. To write the totaled price for the user to the FoxPro database, we invoke EDMMSQLP using the following SQLTABLE instance:

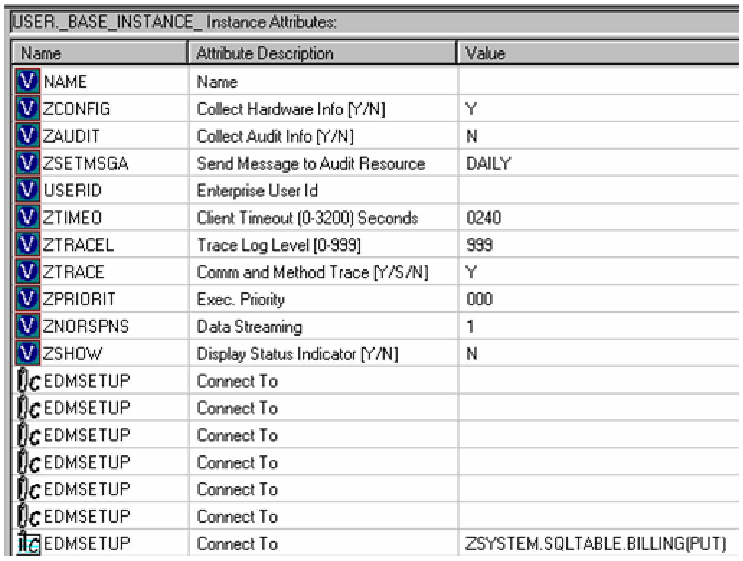

SQLTABLE.BILLING Instance Attributes:

Name	Attribute Description	Value
SQLTABLE	Table Name	BILLS.DBF
SQLUSER	User Name	
SQLPASSW	Password	
SQLTOUT	Time-out in seconds	30
SQLDSN	DSN Name	EDM Data
PUTTYPE	Insert(I) / Replace(R)	I
SRCOBJ	EDM Object containing data to PUT	ZMASTER
DESTOBJ	EDM object to receive GET data	
WHERE	Where clause for SQL statement	
VC000	Column 1	ZUSERID,USERID,U
VC001	Column 2	ZSYSDATE,SYSDATE
VC002	Column 3	ZSYSTIME,SYSTIME
VC003	Column 4	PRICE,TOTAL
VC004	Column 5	
VC005	Column 6	
VC006	Column 7	
VC007	Column 8	
VC008	Column 9	
VC009	Column 10	
VC010	Column 11	
VC011	Column 12	
VC012	Column 13	
VC013	Column 14	
VC014	Column 15	
GET	Run EDMMSQLG method to GET data	ZSYSTEM.ZMETHOD.SQLGET
PUT	Run EDMMSQLP method to PUT data	ZSYSTEM.ZMETHOD.SQLPUT
ALWAYS	Utility method	

Figure 18: SQLTABLE.BILLING Instance Attributes.

The user ID, connection date and time, and totaled price will be written from the ZMASTER object to the `BILLS.DBF` FoxPro table.

To invoke the EDMMSQLP method, we modify the base instance of the USER class, as follows:



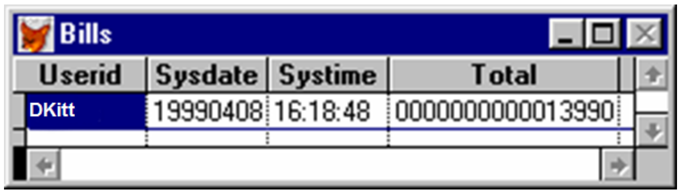
USER._BASE_INSTANCE_ Instance Attributes:

Name	Attribute Description	Value
NAME	Name	
ZCONFIG	Collect Hardware Info [Y/N]	Y
ZAUDIT	Collect Audit Info [Y/N]	N
ZSETMSGA	Send Message to Audit Resource	DAILY
USERID	Enterprise User Id	
ZTIMEO	Client Timeout (0-3200) Seconds	0240
ZTRACEL	Trace Log Level (0-999)	999
ZTRACE	Comm and Method Trace [Y/S/N]	Y
ZPRIORIT	Exec. Priority	000
ZNORSPNS	Data Streaming	1
ZSHOW	Display Status Indicator [Y/N]	N
EDMSETUP	Connect To	
EDMSETUP	Connect To	
EDMSETUP	Connect To	
EDMSETUP	Connect To	
EDMSETUP	Connect To	
EDMSETUP	Connect To	
EDMSETUP	Connect To	ZSYSTEM.SQLTABLE.BILLING(PUT)

Figure 19: USER._BASE_INSTANCE_ instance attributes.

The connection to SYSTEM.SQLTABLE.BILLING provides a control object for EDMMSQLP, and invokes the method by setting the system message to PUT. By making this connection the last step in the resolution of the USER instance, we assure that all services have been resolved and their prices totaled in the ZMASTER object at the point where we invoke EDMMSQLP.

As a result of the client connect for user dkitt, the following record is inserted in the `FoxPro BILLS.DBF` table:



Bills

Userid	Sysdate	Systime	Total
DKitt	19990408	16:18:48	0000000000013990

Figure 20: Microsoft FoxPro BILLS.DBF table.

The Total (13990) is the sum of the prices of the two services that Radia manages for this user (DKitt).

Destination Object (DESTOBJ Parameter) Considerations

EDMMSQLG creates the object identified in the DESTOBJ parameter. The attributes of the object appear in the same order in which they are defined in the VC pairs of the control information. One heap is created in the destination object for each row retrieved from the back-end database. To limit the number of rows retrieved from the back-end database, code an appropriate WHERE clause in the control information.

How to Invoke EDMMSQLP

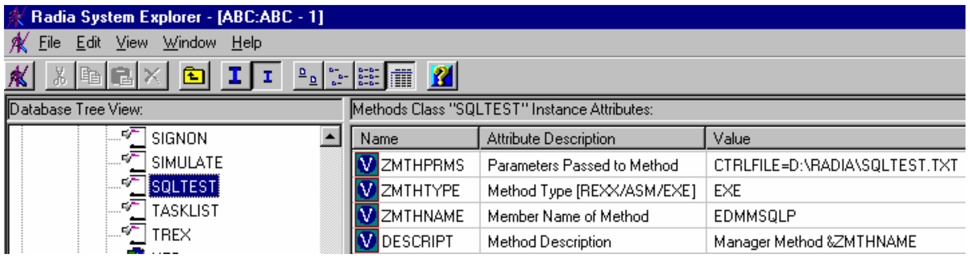
This section provides information needed to invoke EDMMSQLP, and some examples.

Refer to the instructions for creating an instance in the *System Explorer Guide*. Before invoking the method, at least one instance must be defined in the Radia database. You can define multiple instances to invoke EDMMSQLP, where each instance (with its unique name) refers to a different set of control information in its ZMTHPRMS variable.

Mimicking the PROFILE File

In this example, we'll have the Radia client connect store information about the client desktop's hardware configuration in the back-end database. The information will be extracted from the ZCONFIG object, and transferred to a back-end Visual FoxPro table according to control information contained in a text file.

Here is the PRIMARY.SYSTEM.ZMETHOD instance created to invoke EDMMSQLP:

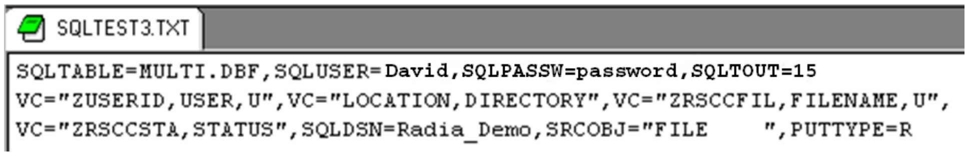

Radia System Explorer - [ABC:ABC - 1]
File Edit View Window Help
Database Tree View:
SIGNON
SIMULATE
SQLTEST
TASKLIST
TREX

Methods Class "SQLTEST" Instance Attributes:

Name	Attribute Description	Value
ZMTHPRMS	Parameters Passed to Method	CTRLFILE=D:\RADIA\SQLTEST.TXT
ZMTHTYPE	Method Type [REXX/ASM/EXE]	EXE
ZMTHNAME	Member Name of Method	EDMMSQLP
DESCRIPT	Method Description	Manager Method &ZMTHNAME

Figure 21: Methods Class SQLTEST Instance Attributes.

Figure 22 on page 239 presents the control information text file.

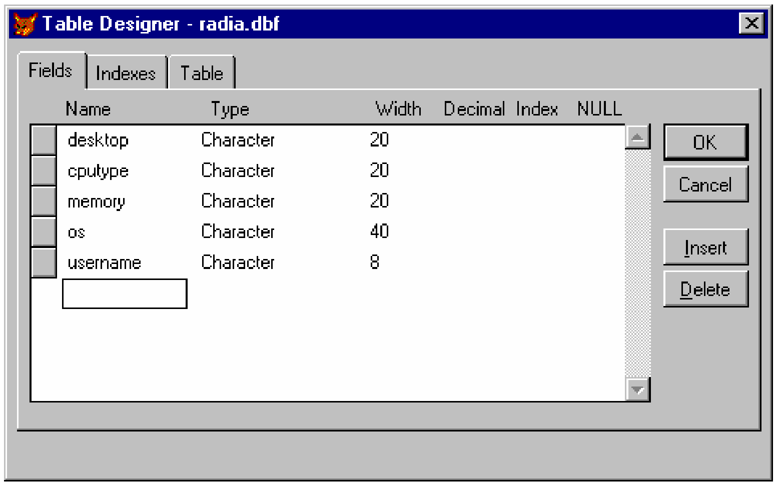


```
SQLTABLE=MULTI.DBF,SQLUSER=David,SQLPASSW=password,SQLTOUT=15
VC="ZUSERID,USER,U",VC="LOCATION,DIRECTORY",VC="ZRSCCFIL,FILENAME,U",
VC="ZRSCCSTA,STATUS",SQLDSN=Radia_Demo,SRCOBJ="FILE     ",PUTTYPE=R
```

Figure 22: SQLTEST control information file.

The third sub-parameter, U, is coded on the DESKTOP, CPUTYPE, and USERNAME VC parameters, because the concatenation of these fields is required to uniquely specify a client desktop in the back-end database.

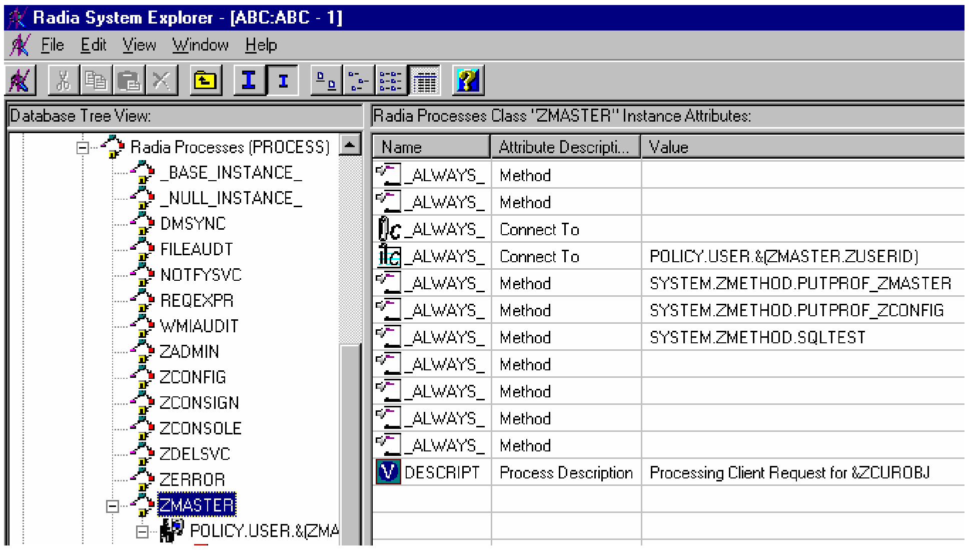
The structure of the Visual FoxPro `radia.dbf` table is as follows.



Name	Type	Width	Decimal	Index	NULL
desktop	Character	20			
cputype	Character	20			
memory	Character	20			
os	Character	40			
username	Character	8			

Figure 23: The Visual FoxPro radia.dbf table.

To invoke the EDMMSQLP method, a connection to the PRIMARY.SYSTEM.ZMETHOD instance named SQLTEST is added to the PRIMARY.SYSTEM.PROCESS.ZMASTER instance, which is processed when the client connect sends the desktop ZMASTER object to the Configuration Server.



Radia System Explorer - [ABC:ABC - 1]

Database Tree View:

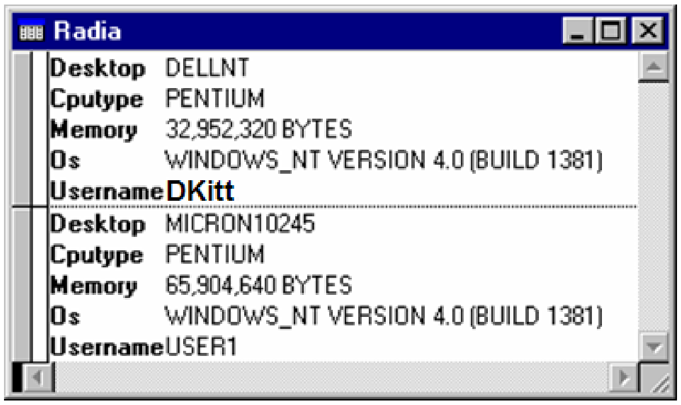
- Radia Processes (PROCESS)
  - _BASE_INSTANCE_
  - _NULL_INSTANCE_
  - DMSYNC
  - FILEAUDT
  - NOTFYSVC
  - REQEXPR
  - WMIAUDIT
  - ZADMIN
  - ZCONFIG
  - ZCONSIGN
  - ZCONSOLE
  - ZDELSVC
  - ZERROR
  - ZMASTER
    - POLICY.USER.&(ZMA

Radia Processes Class "ZMASTER" Instance Attributes:

Name	Attribute Descripti...	Value
ALWAYS	Method	
ALWAYS	Method	
ALWAYS	Connect To	
ALWAYS	Connect To	POLICY.USER.&(ZMASTER.ZUSERID)
ALWAYS	Method	SYSTEM.ZMETHOD.PUTPROF_ZMASTER
ALWAYS	Method	SYSTEM.ZMETHOD.PUTPROF_ZCONFIG
ALWAYS	Method	SYSTEM.ZMETHOD.SQLTEST
ALWAYS	Method	
ALWAYS	Method	
ALWAYS	Method	
ALWAYS	Method	
DESCRIPT	Process Description	Processing Client Request for &ZCUROBJ

Figure 24: Radia Processes Class ZMASTER Instance Attributes.

As each client connects to the Configuration Server, the pertinent fields are extracted from the client's ZCONFIG object and stored in the Visual FoxPro `radia.dbf` table.



Radia

Desktop	DELLNT
Cputype	PENTIUM
Memory	32,952,320 BYTES
Os	WINDOWS_NT VERSION 4.0 (BUILD 1381)
Username	DKitt
Desktop	MICRON10245
Cputype	PENTIUM
Memory	65,904,640 BYTES
Os	WINDOWS_NT VERSION 4.0 (BUILD 1381)
Username	USER1

Figure 25: The Visual FoxPro radia.dbf table.

Examine how this example mimics Radia's saving of the ZCONFIG object in the PROFILE file, which is accomplished via a connection to SYSTEM.ZMETHOD.PUTPROF_ZCONFIG in the ZMASTER instance, above.

Extracting from Multiple Objects

By design, EDMMSQLP extracts data from a single database object each time it is invoked. If you need to extract data from multiple objects, you must either:

- invoke EDMMSQLP multiple times (once for each object), or
- write a custom Configuration Server method to compile data from multiple objects into a single object prior to invoking EDMMSQLP.

The following example demonstrates the latter method with a custom REXX method named SQLPHDW.

```
********************Put   SQL************************/
/* will format and PUT a ZCONFIG object into an     */
/* ODBC compliant database                          */
/****************************************************/

/****************************************************/
/*                                                  */
/* COPYRIGHT HP INC. 2000                           */
/* LICENSED MATERIAL PROPERTY OF HP          .      */
/* HP Radia(tm)                                     */
/*                                                  */
/****************************************************/
/**********************************/
/* get the ZCONFIG object         */
/**********************************/
address cmd
RC = EDMGET('ZCONFIG',1);                         /* Get the ZCONFIG object  */
if RC <> 0 then exit
RC = EDMGET('ZMASTER',1);
ZCONFIG.ZOS = ZMASTER.ZOS || ' ' || ZCONFIG.ZHDWOSDB;
ZCONFIG.ZUSERID = ZMASTER.ZUSERID;
RC = EDMSET('ZCONFIG',1);

/**********************************/
/* the main function              */
/**********************************/
RC = EDMGET('SQLCNTRL',1);
SQLCNTRL.PUTTYPE = 'R';
SQLCNTRL.SQLDSN = 'Radia_Demo';
SQLCNTRL.SQLTABLE = 'RADIA.DBF';
SQLCNTRL.SQLTOUT = '10';
SQLCNTRL.SQLUSER = 'David';
SQLCNTRL.SQLPASSW = 'password';
SQLCNTRL.SRCOBJ = 'ZCONFIG';
SQLCNTRL.VC000 = 'ZHDWCOMP,DESKTOP,U';
SQLCNTRL.VC001 = 'ZHDWCPU,CPUTYPE';
SQLCNTRL.VC002 = 'ZHDWMEM,MEMORY';
SQLCNTRL.VC003 = 'ZOS,OS';
SQLCNTRL.VC004 = 'ZUSERID,USERNAME,U';
RC = EDMSET('SQLCNTRL',1);
params =  " CTRLOBJ=SQLCNTRL"
address edmlink EDMMSQLP params
/return;
```

On MVS, ADDRESS must be replaced with CALL, as in:

```
CALL EDMLINK EDMMSQLP params
```

This method executes the following process:

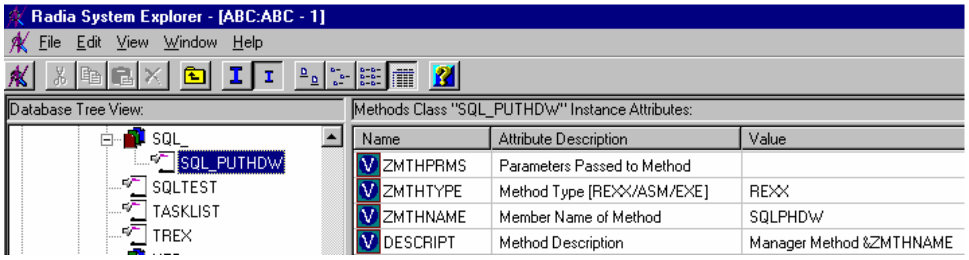
1 It combines variables (ZOS, ZUSERID) from the ZMASTER object with the variables in the ZCONFIG object.

2 It then builds an object (SQCNTRL) to contain the control information for a call to EDMMSQLP.

3 Lastly, it invokes EDMMSQLP, and passes the control object via the CTRLOBJ=SQLCNTRL parameter.

For further information on constructing custom methods in the REXX programming language, refer to the *REXX Programming Guide for the HP OpenView Application Manager and Configuration Server Using Radia.*

To invoke the SQLPHDW method, you must create an instance (for example, SQL_PUTHDW) of the PRIMARY.SYSTEM.ZMETHOD class, which, in this example, looks like:


Radia System Explorer - [ABC:ABC - 1]
File Edit View Window Help
Database Tree View:
SQL_
SQL_PUTHDW
SQLTEST
TASKLIST
TREX

Methods Class "SQL_PUTHDW" Instance Attributes:

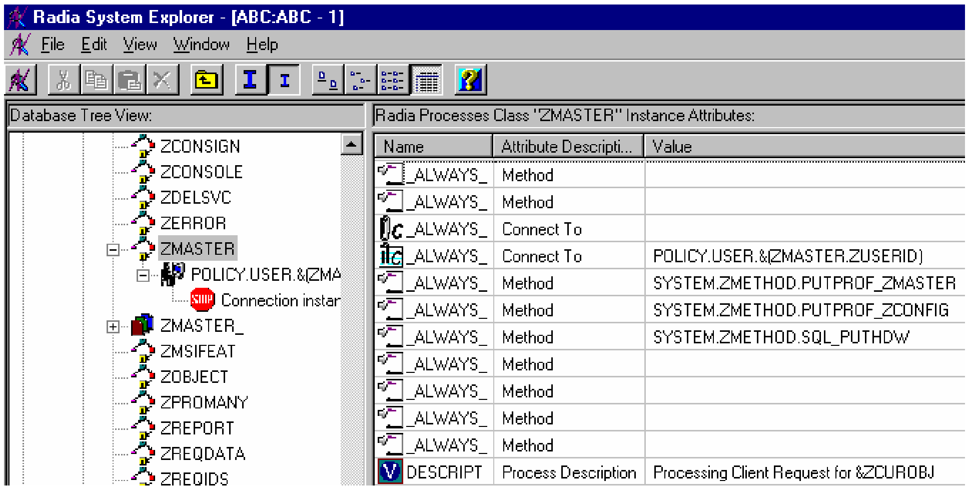
Name	Attribute Description	Value
V ZMTHPRMS	Parameters Passed to Method	
V ZMTHTYPE	Method Type [REXX/ASM/EXE]	REXX
V ZMTHNAME	Member Name of Method	SQLPHDW
V DESCRIPT	Method Description	Manager Method &ZMTHNAME

Figure 26: Methods Class SQL_PUTHDW Instance Attributes.

Notice that the ZMTHPRMS variable is empty. The control information for EDMMSQLP is built within the SQLPHDW method and will be passed to the EDMMSQLP method in a control object.

- The SQLPHDW file (without a file extension) must be located in the `Manager\Rexx` directory.
- For methods written in the REXX programming language, the ZMTHTYPE variable must be **REXX**.

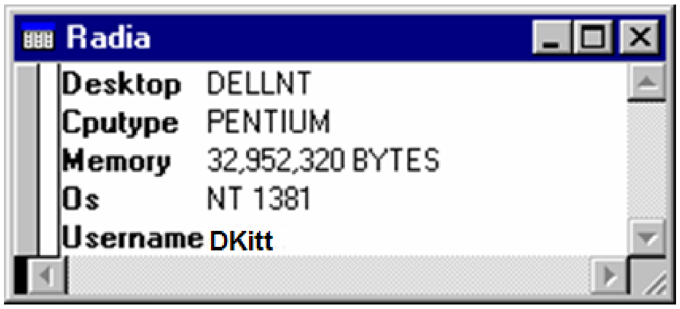
To have the client connect invoke the SQLPHDW method, the SYSTEM.PROCESS.ZMASTER instance has been changed from the previous example to include an _ALWAYS_ connection to SYSTEM.ZMETHOD.SQL_PUTHDW (Figure 27 below).



Name	Attribute Descripti...	Value
ALWAYS	Method	
ALWAYS	Method	
ALWAYS	Connect To	
ALWAYS	Connect To	POLICY.USER.&(ZMASTER.ZUSERID)
ALWAYS	Method	SYSTEM.ZMETHOD.PUTPROF_ZMASTER
ALWAYS	Method	SYSTEM.ZMETHOD.PUTPROF_ZCONFIG
ALWAYS	Method	SYSTEM.ZMETHOD.SQL_PUTHDW
ALWAYS	Method	
ALWAYS	Method	
ALWAYS	Method	
ALWAYS	Method	
DESCRIPT	Process Description	Processing Client Request for &ZCUROBJ

Figure 27: Radia Processes Class ZMASTER Instance Attributes.

As a result of the client connect, the data is transferred to the back-end database for the connecting desktop.



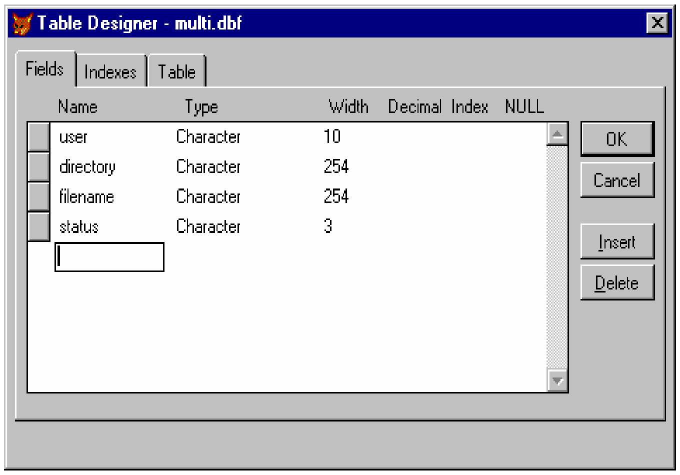
Desktop	DELLNT
Cputype	PENTIUM
Memory	32,952,320 BYTES
Os	NT 1381
Username	DKitt

Figure 28: The Visual FoxPro radia.dbf table.

To transfer data from a multiple heap object

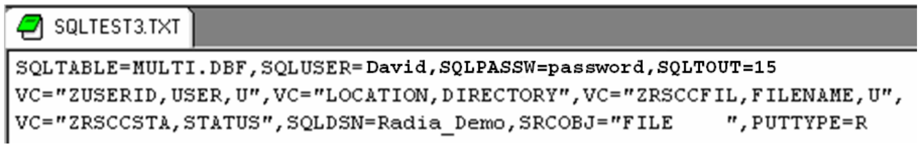
In this example, data is transferred from a FILE object. The FILE object has a heap for each file transferred to the client desktop during the deployment of an application.

1 Create a Visual FoxPro table to receive the data.



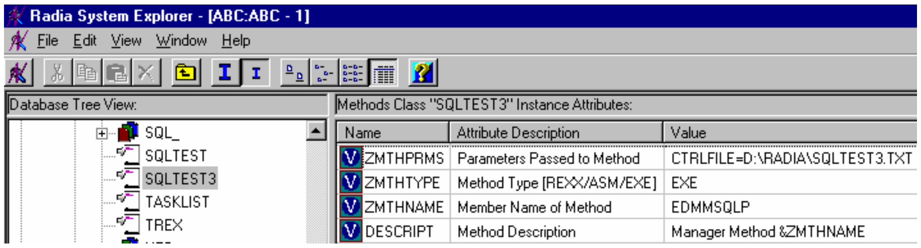
Name	Type	Width	Decimal	Index	NULL
user	Character	10			
directory	Character	254			
filename	Character	254			
status	Character	3			

2 Create `SQLTEST3.TXT`, a control file for EDMMSQLP.



```
SQLTABLE=MULTI.DBF,SQLUSER=David,SQLPASSW=password,SQLTOUT=15
VC="ZUSERID,USER,U",VC="LOCATION,DIRECTORY",VC="ZRSCCFIL,FILENAME,U",
VC="ZRSCCSTA,STATUS",SQLDSN=Radia_Demo,SRCOBJ="FILE     ",PUTTYPE=R
```

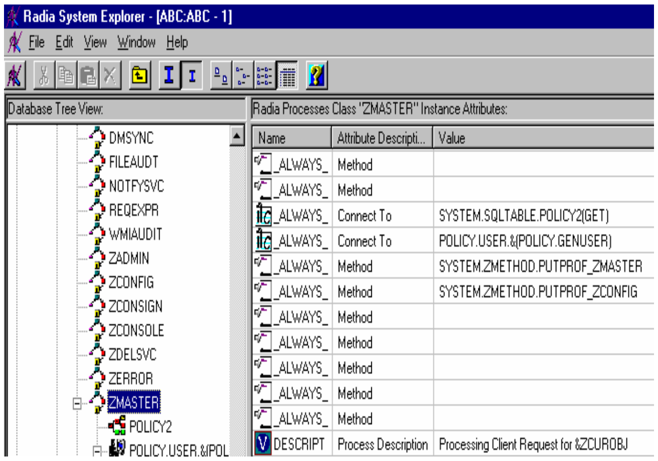
3 Create a SYSTEM.ZMETHOD instance named SQLTEST3 to invoke the EDMMSQLP method with the SQLTEST3.TXT control file.


Radia System Explorer - [ABC:ABC - 1]
File Edit View Window Help
Database Tree View:
SQL_
SQLTEST
SQLTEST3
TASKLIST
TREX

Methods Class "SQLTEST3" Instance Attributes:

Name	Attribute Description	Value
ZMTHPRMS	Parameters Passed to Method	CTRLFILE=D:\RADIA\SQLTEST3.TXT
ZMTHTYPE	Method Type [REXX/ASM/EXE]	EXE
ZMTHNAME	Member Name of Method	EDMMSQLP
DESCRIPT	Method Description	Manager Method &ZMTHNAME

4 Modify SYSTEM.PROCESS.ZMASTER to connect to SYSTEM.ZMETHOD.SQLTEST3.

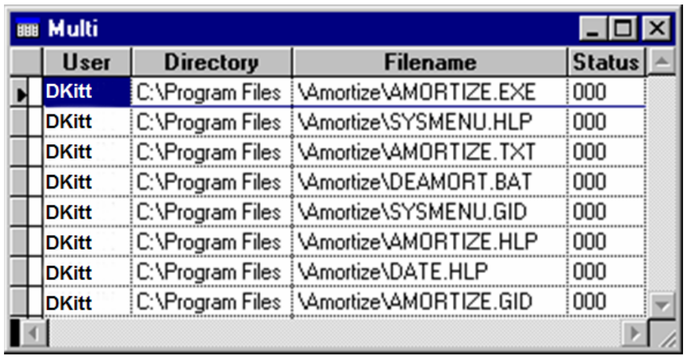


Radia Processes Class "ZMASTER" Instance Attributes:

Name	Attribute Descripti...	Value
ALWAYS	Method	
ALWAYS	Method	
ALWAYS	Connect To	SYSTEM.SQLTABLE.POLICY2(GET)
ALWAYS	Connect To	POLICY.USER.&(POLICY.GENUSER)
ALWAYS	Method	SYSTEM.ZMETHOD.PUTPROF_ZMASTER
ALWAYS	Method	SYSTEM.ZMETHOD.PUTPROF_ZCONFIG
ALWAYS	Method	
ALWAYS	Method	
ALWAYS	Method	
ALWAYS	Method	
ALWAYS	Method	
DESCRIPT	Process Description	Processing Client Request for &ZCUROBJ

5 Run the client connect.

Here is the result in the Visual FoxPro table for one user with a ZRSOURCE object containing eight heaps.



User	Directory	Filename	Status
DKitt	C:\Program Files	\Amortize\AMORTIZE.EXE	000
DKitt	C:\Program Files	\Amortize\SYSMENU.HLP	000
DKitt	C:\Program Files	\Amortize\AMORTIZE.TXT	000
DKitt	C:\Program Files	\Amortize\DEAMORT.BAT	000
DKitt	C:\Program Files	\Amortize\SYSMENU.GID	000
DKitt	C:\Program Files	\Amortize\AMORTIZE.HLP	000
DKitt	C:\Program Files	\Amortize\DATE.HLP	000
DKitt	C:\Program Files	\Amortize\AMORTIZE.GID	000

Source Object (SRCOBJ Parameter) Considerations

Any Radia database object can be used as a source for EDMMSQLP. In the source object, EDMMSQLP expects to find all the variables as defined in VC keywords in the control information parameter string, or in VCnnn variables in the control object.

Then, for each heap of the source object, EDMMSQLP reads the values of the requested variables and writes them into the SQL table's columns that are defined in the corresponding VC pair. All other variables in the source object that are not defined in any of the VC pairs of the control information are ignored. All the requested variables from a single source object heap will be put in one row of the SQL table. The next heap of the source object will provide values for the next row of the SQL table, and the process will continue until all the heaps of the source object are processed.

Passing Control Information to EDMMSQLG and EDMMSQLP

The EDMMSQLG and EDMMSQLP methods require a set of control information to perform their function. Control information is passed to EDMMSQLG and EDMMSQLP as a parameter (or set of parameters) at execution time in one of the following ways:

- As a parameter string passed on the command line (such as, in the ZMTHPRMS variable of the PRIMARY.SYSTEM.ZMETHOD instance used to invoke EDMMSQLG/EDMMSQLP). The maximum length of the command-line parameter string is 255 characters. If the control information you need to pass to the method is longer than 255 characters, you must use one of the other options.
- In a text file (identified by the CTRLFILE=<file_name> parameter).

- In a control object (identified by the CTRLOBJ=<object_name> parameter)

Control Parameters

Table 101 below identifies the control information required by EDMMSQLG and EDMMSQLP.

Table 101: EDMMSQLG/EDMMSQLP Control Information

Keyword	**Description**
CTRLFILE	The fully qualified name of a text file that contains the control information. If this parameter is present on the command line, the parsing of the parameter string stops, and all the control information will be taken from the specified file. Required only if the control information is supplied in the specified file.
CTRLOBJ	The name of the object that contains the control information. If this parameter is present in the command line, the parsing of the parameter string stops, and all the control information will be taken from the specified object. Required only if the control information is supplied in the specified object.
SRCOBJ	EDMMSQLP only. The name of the source object (right-padded with blanks to eight characters and enclosed in quotation marks when specified in a text file or on the command line). The source object contains the data to be transferred to the back-end SQL database.
DESTOBJ	EDMMSQLG only. The name of the destination object (right-padded with blanks to eight characters and enclosed in quotation marks when specified in a text file or on the command line). The destination object contains the data to be received from the back-end SQL database. EDMMSQLG will create this object when it is executed.
SQLDSN	The ODBC data source name (DSN) to be used to connect to the back-end SQL database. See Configuring an ODBC Data Source on page 195, for details.

SQLTABLE	The fully qualified file name of the file containing the SQL table, or simply the name of the table (depending on which the back-end database requires the ODBC data source to supply). EDMMSQLP will store the data into this table. EDMMSQLG will extract the data from this table.
SQLUSER	User ID to use in the Radia database connect process. Some back-end databases ignore this information; others require and verify it. See the database administrator of the back-end database in your organization for details.
SQLPASSW	The password to use in the Radia database connect process. Some back-end databases ignore this information; others require and verify it. See the database administrator of the back-end database in your organization for details.
SQLTOUT	Time-out value (in seconds) for the SQL connect operation. If a connection to the back-end database cannot be established within this time, the connection attempt will be terminated and an error logged in the Configuration Server log.
VC, or VCnnn (where nnn is a sequential 3-digit number from 000 to the total number of variables to be transferred to the back-end database; used when control information is passed in an object)	Defines the correspondence between a variable in the source or destination object and the column in the back-end database table where it will be stored (EDMMSQLP), or from which it will be extracted (EDMMSQLG). One VC value must be specified for each variable-column pair participating in the operation. Specify: **`"VARNAME [,COLUMN_NAME ][,U]"`** (include the quotation marks in a text file or on a command line, omit them in an object) • VARNAME is the name of the variable in the object whose value will be set (EDMMSQLG), or transferred to (EDMMSQLP), the back-end database. • COLUMN_NAME is the name of the column in the back-end database table that will supply (EDMMSQLG), or receive (EDMMSQLP), the data. If COLUMN_NAME is omitted, VARNAME will be used; this assumes that the back-end database table's column name is the same as the object variable receiving (EDMMSQLG), or supplying (EDMMSQLP), its data. • The third sub-parameter, U, identifies the key fields used to locate the row to be replaced in the back-end database. If PUTTYPE=R, at least one VC value must have the third sub-parameter coded.

PUTTYPE	EDMMSQLP only. Indicator for type of operation to be performed on the back-end database, either R (Replace) or I (Insert). When R is specified, EDMMSQLP will try to update the identified row. If the row does not exist, EDMMSQLP will try to insert it. When I is specified, EDMMSQLP will try to insert the row. If this operation fails, no other action is taken. The default is R.
WHERE	Contains a WHERE clause for the SQL statement that EDMMSQLG builds to extract data from the back-end database, or which EDMMSQLP constructs to update data in the back-end database. Use this to limit the result set to records that meet a specific condition. Do not code the word WHERE in this parameter; only code the clause that would follow the WHERE keyword in the SQL SELECT statement that retrieves the data from the back-end database, or the SQL UPDATE statement that stores data in the back-end database.

PUTTYPE

If, during an execution of EDMMSQLP, PUTTYPE=R, the following are applicable:

- It generates a REPLACE statement using VC groups specified in the control information provided. Typically, this will do a direct update for all the rows that were found according to the search criteria in the WHERE clause. The WHERE clause uses the WHERE variable content from the control information (version 4.4 Configuration Server), or built as a combination of unique fields, marked as such in VC variables (pre-V.4.4 Configuration Servers).

  In a case where the WHERE variable exists, EDMMSQLP will ignore all unique keys specified in VC variables. However, since various Database Management Systems (**DBMS**) can use different implementations for this operation, consult the documentation for your particular DBMS for implementation issues.

- If the REPLACE operation fails due to SQL errors, nothing else is done and an error is reported in the Configuration Server log.
- If the REPLACE does not fail, but the number_of_affected_rows=0 (meaning there were no rows found that meet the criteria), EDMMSQLP will generate an INSERT and attempt to create a new row in the database table.

Generally, PUTTYPE=I should be used only when the key part of the row is always different (for example, date and time plus user ID are used as a combined key). For all other cases, PUTTYPE=R will handle initial inserts as well as updates.

The U Sub-parameter

The U sub-parameter is used to identify one or more fields which EDMMSQLP builds into a WHERE clause for the REPLACE SQL statement it generates. The U sub-parameter is recognized by EDMMSQLP only, and only when PUTTYPE=R (replace). If these two conditions are not met, it is ignored.

- If one VC pair has the U sub-parameter coded, the WHERE clause generated will resemble:

  ```
  WHERE fieldname = current source object corresponding
  variable value
  ```

  The current source object corresponding variable value is the value currently (at the time of the call to EDMMSQLP) held in the source object variable, coded in the control information's VC pair that links the variable with the back-end database.

- If more than one VC pair has the U sub-parameter coded, the WHERE clause 'ANDS' them together, as in:

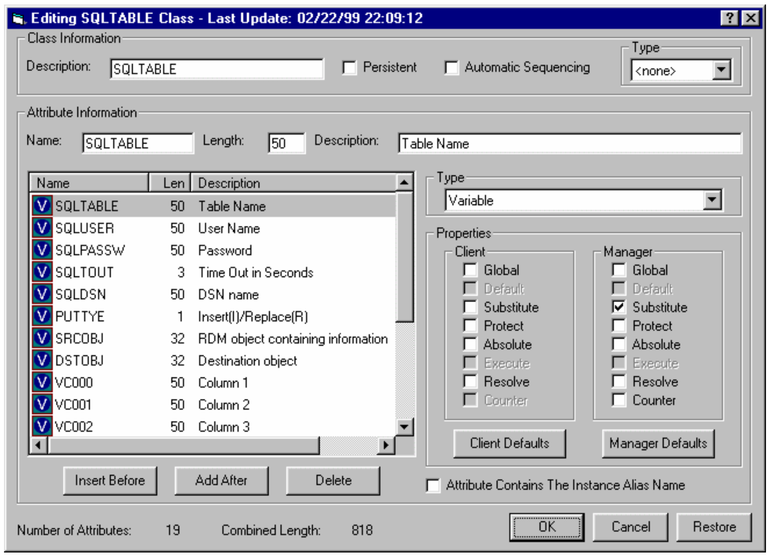
  ```
  WHERE fieldname1 = value1 AND fieldname2 = value2…
  ```

- If the result of executing the REPLACE statement with the WHERE clause yields zero matching records, EDMMSQLP regenerates the SQL statement as an insert, and inserts the record.
- If the result of executing the REPLACE statement with the WHERE clause finds exactly one record in the database, the record is replaced with the information contained in the source object.
- If the result of executing the REPLACE statement with the WHERE clause finds more than one record in the database, all the records are replaced with the information contained in the source object.
- If PUTTYPE=R and no VC pairs in the control input have the U sub-parameter coded, EDMMSQLP terminates and logs the following error message:

  ```
  NO UNIQUE COLUMNS WERE FOUND FOR WHERE CLAUSE. TERMINATING.
  ```

Configuring the Radia Database SQLTABLE Class

The easiest way to provide the necessary control information to EDMMSQLG and EDMMSQLP is to use a control object instantiated from a class in the Radia database. This can be provided by an instance of a Radia database class, such as the SQLTABLE class in the SYSTEM domain of the Primary file. If your Radia database does not contain this class, use the Radia System Explorer to add it. Figure 29 below shows the class template:


Editing SQLTABLE Class - Last Update: 02/22/99 22:09:12

Class Information

Description: SQLTABLE ☐ Persistent ☐ Automatic Sequencing Type: <none>

Attribute Information

Name: SQLTABLE Length: 50 Description: Table Name

Name	Len	Description
SQLTABLE	50	Table Name
SQLUSER	50	User Name
SQLPASSW	50	Password
SQLTOUT	3	Time Out in Seconds
SQLDSN	50	DSN name
PUTTYE	1	Insert(I)/Replace(R)
SRCOBJ	32	RDM object containing information
DSTOBJ	32	Destination object
VC000	50	Column 1
VC001	50	Column 2
VC002	50	Column 3

Type: Variable

Properties

Client	Manager
☐ Global	☐ Global
☐ Default	☐ Default
☐ Substitute	☑ Substitute
☐ Protect	☐ Protect
☐ Absolute	☐ Absolute
☐ Execute	☐ Execute
☐ Resolve	☐ Resolve
☐ Counter	☐ Counter

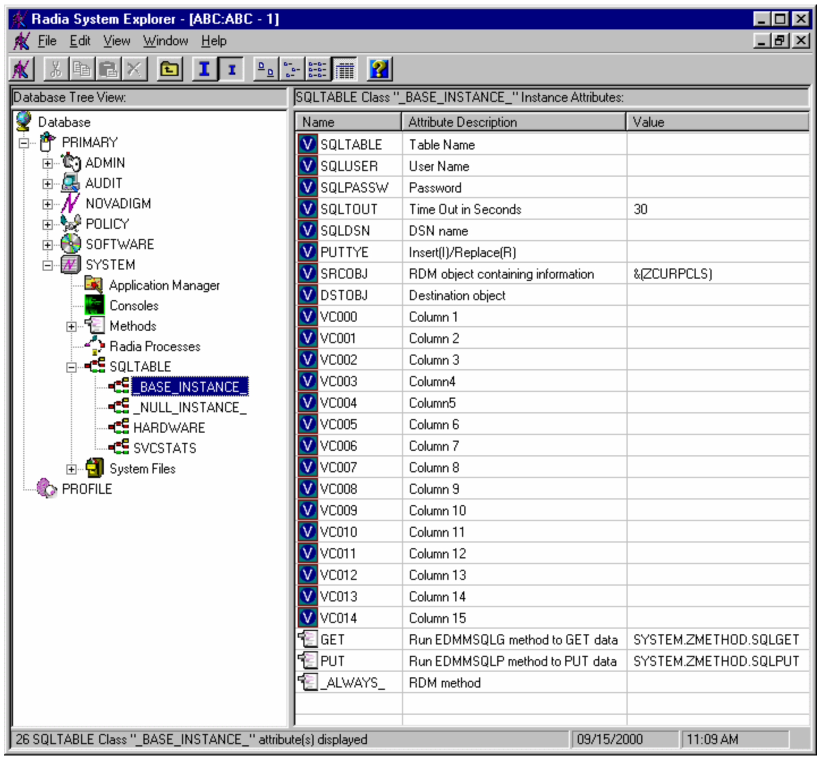
☐ Attribute Contains The Instance Alias Name

Number of Attributes: 19 Combined Length: 818

Figure 29: The Editing SQLTABLE Class dialog box.

Note that in the **Properties** area, under **Manager**, the **Global** check box is cleared for all variables in the SQLTABLE class template. This prevents unnecessary storage of the variables of the control object instantiated from an instance of the SQLTABLE class from being stored in parent persistent objects, such as ZSERVICE or ZMASTER (depending on where in the resolution process the EDMMSQLP or EDMMSQLG method is invoked).

There are 15 VCnnn fields (VC000 – VC014) in the sample class template. You can define as many as you need in your template. The recommended configuration for this class's base instance is shown in Figure 30 on page 252.



Radia System Explorer - [ABC:ABC - 1]

Database Tree View:

```
Database
  PRIMARY
    ADMIN
    AUDIT
    NOVADIGM
    POLICY
    SOFTWARE
    SYSTEM
      Application Manager
      Consoles
      Methods
      Radia Processes
      SQLTABLE
        _BASE_INSTANCE_
        _NULL_INSTANCE_
        HARDWARE
        SVCSTATS
      System Files
  PROFILE
```

SQLTABLE Class "_BASE_INSTANCE_" Instance Attributes:

Name	Attribute Description	Value
SQLTABLE	Table Name	
SQLUSER	User Name	
SQLPASSW	Password	
SQLTOUT	Time Out in Seconds	30
SQLDSN	DSN name	
PUTTYE	Insert(I)/Replace(R)	
SRCOBJ	RDM object containing information	&(ZCURPCLS)
DSTOBJ	Destination object	
VC000	Column 1	
VC001	Column 2	
VC002	Column 3	
VC003	Column4	
VC004	Column5	
VC005	Column 6	
VC006	Column 7	
VC007	Column 8	
VC008	Column 9	
VC009	Column 10	
VC010	Column 11	
VC011	Column 12	
VC012	Column 13	
VC013	Column 14	
VC014	Column 15	
GET	Run EDMMSQLG method to GET data	SYSTEM.ZMETHOD.SQLGET
PUT	Run EDMMSQLP method to PUT data	SYSTEM.ZMETHOD.SQLPUT
ALWAYS	RDM method	

26 SQLTABLE Class "_BASE_INSTANCE_" attribute(s) displayed | 09/15/2000 | 11:09 AM

Figure 30: SQLTABLE Class _BASE_INSTANCE_ Instance Attributes.

Note the GET and PUT method variables. These connect to ZMETHOD instances to run the EDMMSQLG and EDMMSQLP methods. If the current value of the system message is GET, EDMMSQLG is run; if the value of the system message is PUT, EDMMSQLP is run. If the value of the system message is something other than GET or PUT, neither method is run.

Control Information

Control information includes:

- how the method knows which database object to use as a source,
- which variables of that object to write to which columns of the destination table,
- how to connect to the database,
- which table of the database to use in the operation, and

- what the user ID, password, and timeout are.

All this control information must be passed to the method.

Content

The following control information is required:

- The name of the object that contains the source data (SRCOBJ) for the SQL put request. Or the name of the object in which to store the information (DESTOBJ) retrieved from the SQL database.
- Data Source Name - the logical name used to connect to the specific SQL database.
- The fully qualified name of the table to access in the <SQLDSN> database.
- The user ID and password to use for connecting to the database.
- VARNAME [,COLUMN_NAME][,U], which describes the relationship between a variable of the (source or destination) object and the corresponding column of the database table. For more information on this, refer to the section, VARIABLE-COLUMN Pairs, on page 256.
- REPLACE (UPDATE) or INSERT as the type of Put operation requested. (put method only)
- Number of seconds to wait on the SQL-connect operation.
- (Optional) The WHERE clause to use in the selected statement. The method will substitute the WHERE, so that if USER=JANE is specified, then in the select statement it will be WHERE (USER=JANE). The variable WHERE is optional, and in cases where it is omitted in the control object, it will be generated by the method, with the help of variables with the U suffix. For more information on the WHERE clause, refer to The WHERE Clause, on page 264.

Delivery

The control information can be delivered to the HP SQL method in one of the following ways:

- via a parameter string,
- via a text file (CTRLFILE=<file_name>), or
- via a control object (CTRLOBJ=<object_name>).

Regardless of the selected delivery method, the parameter string must be passed to the HP SQL method. If the sub-string CTRLFILE=<file_name> is found in the parameter string, the control information will be retrieved from file file_name. If the sub-string CTRLOBJ=<object_name> is found in the parameter string, the control information will be retrieved from the object_name. If neither sub-string is found, the method assumes that all the control information is passed in the parameter string. Your system administrator will determine which method will be used to pass the control information to the HP SQL method.

CTRLFILE always takes precedence over CTRLOBJ. Therefore, if both are specified, regardless of their order, the control information will be retrieved from the file that is specified by CTRLFILE.

Control Information Passed via a Parameter String

The parameter string has the comma-separated key-value format. The maximum length of the parameter string is 255 bytes. If the control information you need to pass to the method exceeds the maximum, use either the CTRLFILE or the CTRLOBJ option and put all the control information in the control file (or the control object).

Examples of the parameter string:

Example 1 – Text File

In the following example, all the control information will be read from the file `C:\Radia\SQLCNTL.TXT`. All other information specified in the parameter string will be ignored.

```
CTRLFILE=C:\Radia\SQLCNTL.TXT, SQLDSN=CUST_DB,
SQLTABLE=joe.USERS, SQLUSER=smith, SQLPASSW=Rabbit,
SQLTOUT=15, VC="ZUSERID,USER", VC="ZBIOS,BIOS", VC="ZOS,OS",
VC="ZOSVER,OS_VERSION"
```

If you remove the CTRLFILE parameter, the remaining parameters on the command line would control the execution of the method.

Example 2 – Control Object

In the following example, all the control information will be retrieved from the SQLCNTL object, while all the other command-line information will be ignored.

```
CTRLOBJ=SQLCNTL, SQLDSN=CUST_DB, SQLTABLE=joe.USERS,
SQLUSER=smith, SQLPASSW=Rabbit, SQLTOUT=15, VC="ZUSERID,USER",
VC="ZBIOS,BIOS", VC="ZOS,OS", VC="ZOSVER,OS_VER"
```

If you remove the CTRLOBJ parameter, the remaining parameters on the command line would control the execution of the method.

Example 3 – Precedence of CTRLOBJ over CTRLFILE

In the following example, all the control information will be retrieved from the `C:\Radia\SQLCNTL.TXT` file, and the information in the SQLCNTL object will be ignored. This does not allow combinations of sources of the control information.

```
CTRLOBJ=SQLCNTL,CTRLFILE=C:\Radia\SQLCNTL.TXT
```

As previously noted, CTRLFILE always supercedes CTRLOBJ, regardless of the order in which they are specified on the command line.

Example 4 – Command-Line Control String

In the following example, all the control information is received from the parameter strings on the command line.

```
SQLTABLE=RADIA.DBF, SQLUSER=joedoe, SQLPASSW=password,
SQLTOUT=15, VC="ZHDWCOMP,DESKTOP,U", VC="ZHDWCPU,CPUTYPE",
VC="ZHDWMEM,MEMORY", VC="ZHDWOS,OS", VC="ZUSERID,USERNAME,U",
SRCOBJ=ZCONFIG, SQLDSN=Radia_Demo, PUTTYPE=R
```

Control Information Passed via a Text File

All control information can be optionally passed to EDMMSQLG and EDMMSQLP in a text file. In general, use a text file to pass the control information when the length of the parameter string exceeds 255 characters, or if you need to make multiple references to the same set of control information.

The format of the control information passed in a text file is identical to that of the parameter string passed on the command line. However, CTRLOBJ and CTRLFILE parameters will be ignored if found in the text file.

Control Information Passed via a Control Object

All control information can be optionally passed to EDMMSQLG and EDMMSQLP in a control object. In this case, keywords that were defined in previous sections will become variable names in the object. Additionally, the group of non-unique VC variables must be converted to unique variable names. In order to create unique variable names, a three-digit index is appended to VC, so that variable names will be VC000, VC001, … VCnnn.

The three-digit indexes that are appended to VC in forming the variable name *must* start with 000, and subsequent variable names must be created from an index value one greater than the previous one. No numbers can be skipped.

The command-line control string that was presented in Example 4 can be implemented in a control object, SQLPARMS, with the variables that are shown in Table 102.

Table 102: SQLPARMS Values

Variable Name	Variable Value
SRCOBJ	ZCONFIG
SQLDSN	Radia_Demo
SQLTABLE	RADIA.DBF
SQLUSER	joedoe
SQLPASSW	password
SQLTOUT	15
VC000	ZHDWCOMP,DESKTOP,U
VC001	ZHDWCPU,CPUTYPE
VC002	ZHDWMEM,MEMORY
VC003	ZHDWOS,OS
VC004	ZUSERID,USERNAME,U
PUTTYPE	R

In this case CTRLOBJ=SQLPARMS is the only parameter passed to the EDMMSQLP method on the command line.

VARIABLE-COLUMN Pairs

A VARIABLE-COLUMN (VC) pair is the designation of one set of information-location data participating in a get or put operation. The VARIABLE is the database object that is being received or transferred in the method. The COLUMN is the category in the SQL database table that is being accessed to receive or supply the data. The following rules apply to the use and execution of VC pairings.

- There might be more than one VC keyword in a parameter string. However, one VC value must be specified for each VARIABLE-COLUMN pair participating in the operation.
- The VC can be specified as VCnnn, where nnn is a sequential, three-digit number from 000 to the total number of variables to be transferred to the back-end database. This is used when control information that is passed in an object defines the correspondence between a variable in the HP (source or destination) object, and the column in the back-end database table. That is, to where it will be stored (EDMMSQLP), or from where it will be extracted (EDMMSQLG).
- Specify as "VARNAME [,COLUMN_NAME][,U]" (include the quotation marks when using a text file or on the command line, but omit them in an object). Here, VARNAME is the name of a variable in the source object.
- In order to be transferable, the value of a source object variable should correspond to that of an SQL-table date-type. Therefore,

  VARNAME, "number"

  should be in the source object.

  "Transferable" means having the following information in the control object: integers for number-type columns, date information for date-type columns, and COLUMN_NAME for the name of the column, in the SQL table.

  If, in the source object, there is not a variable-type that corresponds to an SQL column-type, the method might fail or produce unexpected results in the database.

  Table 103 on page 262 contains a complete list of the SQL column data-types that HP supports.

- VARNAME is the name of the variable in the object whose value will be set from (EDMMSQLG), or transferred to (EDMMSQLP), the back-end database.
- COLUMN_NAME is the name of the column in the back-end database table that will supply (EDMMSQLG) or receive (EDMMSQLP) the data.

If COLUMN_NAME is omitted, VARNAME will be used (provided the column of the back-end database table has the same name as the object variable supplying or receiving the data).

- The third sub-parameter, U, means that this COLUMN_NAME is the Unique Primary Key (PK) for the table; and therefore, identifies it as the key fields to use to locate the back-end database column to be replaced.

  The U sub-parameter is recognized by EDMMSQLP only, and only when PUTTYPE=R. If coded in other situations, it is ignored.

  The U sub-parameter is used to identify one or more fields that EDMMSQLP builds into a WHERE clause for the REPLACE statement that SQL generates. For more information on the impact of the U sub-parameter in the WHERE clause, see The WHERE Clause on page 264.

- If PUTTYPE=R (see Table 100 on page 222), at least one VC value must have the third sub-parameter coded.

HP Object Information

Any database object can be used as a source or destination object for an SQL method. For the source object, the method expects to find all the variables defined in VC keywords in the object. For each heap of the object, the requested variables are read and then written into the SQL table's columns as defined in the corresponding VC pair. Any variables that are not defined in a VC pair are ignored. Generally, all the requested variables from a heap will be put in one row of the SQL table. The next heap will be a source for the next row of the SQL table, and so on until all the heaps of the source object are processed.

By having the value in the WHERE clause substituted from the source object variable, the substitution capability of EDMMSQLP can be used to make the various heaps responsible for updating the rows of the SQL table.

SQL Database Information

To connect the user to the SQL database, follow the steps in To configure DSN.

Data Source Name

The EDMMSQL methods use Open Database Connectivity (ODBC) to connect to the database. Therefore, you must configure a Data Source Name (DSN) for your system.

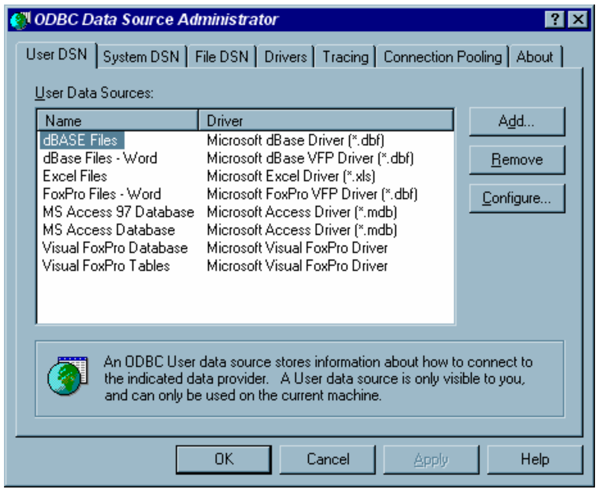
Using the steps outlined in the following example, you can create or choose any DSN as long as it is available in your environment and is supported by ODBC.

To configure DSN

1 Go to **Start→Settings→Control Panel**.

2 Double-click **ODBC Data Sources**.

The ODBC Data Source Administrator dialog box opens.



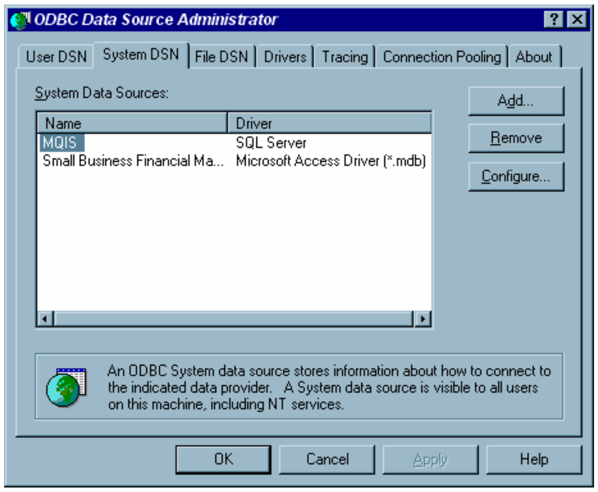
Name	Driver
dBASE Files	Microsoft dBase Driver (*.dbf)
dBase Files - Word	Microsoft dBase VFP Driver (*.dbf)
Excel Files	Microsoft Excel Driver (*.xls)
FoxPro Files - Word	Microsoft FoxPro VFP Driver (*.dbf)
MS Access 97 Database	Microsoft Access Driver (*.mdb)
MS Access Database	Microsoft Access Driver (*.mdb)
Visual FoxPro Database	Microsoft Visual FoxPro Driver
Visual FoxPro Tables	Microsoft Visual FoxPro Driver

An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you, and can only be used on the current machine.

3 Select the **System DSN** tab.

You have the option to select **User DSN**, however, if the Configuration Server runs as a service, User DSNs are not accessible to it and database connects will fail.

The System DSN window opens.



System Data Sources:

Name	Driver
MQIS	SQL Server
Small Business Financial Ma...	Microsoft Access Driver (*.mdb)

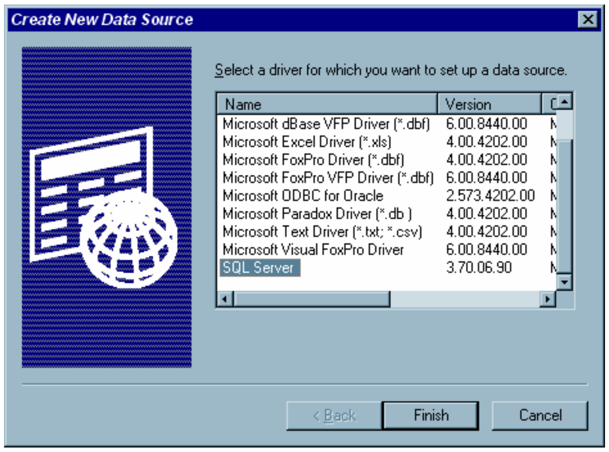
An ODBC System data source stores information about how to connect to the indicated data provider. A System data source is visible to all users on this machine, including NT services.

4 Choose Data Source Name and use it in the SQLDSN keyword.

5 If Data Source Name does not exist, as in the figure above, click **Add** to create a new DSN.

You might want to configure the new DSN exclusively for HP. Generally, any DBMS supported by ODBC is supported by HP.

The Create New Data Source dialog box opens.

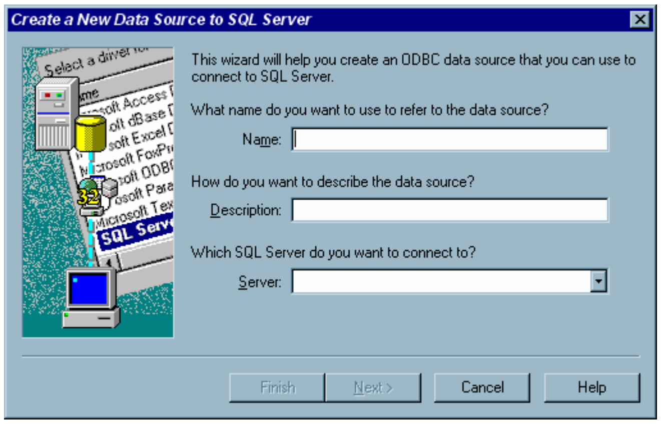


Name	Version
Microsoft dBase VFP Driver (*.dbf)	6.00.8440.00
Microsoft Excel Driver (*.xls)	4.00.4202.00
Microsoft FoxPro Driver (*.dbf)	4.00.4202.00
Microsoft FoxPro VFP Driver (*.dbf)	6.00.8440.00
Microsoft ODBC for Oracle	2.573.4202.00
Microsoft Paradox Driver (*.db)	4.00.4202.00
Microsoft Text Driver (*.txt; *.csv)	4.00.4202.00
Microsoft Visual FoxPro Driver	6.00.8440.00
SQL Server	3.70.06.90

6 Select the driver for which you want to set up a data source, and click **Finish**.

(The figure above shows our choice of Microsoft SQL Server, but this can be any database that supports the SQL language standard, through ODBC, on Windows and UNIX platforms.)

The Create New Data Source to SQL Server dialog box opens.



This wizard will help you create an ODBC data source that you can use to connect to SQL Server.

What name do you want to use to refer to the data source?

Name:

How do you want to describe the data source?

Description:

Which SQL Server do you want to connect to?

Server:

7 From this point, continue with the wizard, specifying information that pertains to your environment.

Table Name, User ID, and Password

The table name is the fully qualified name of the existing table in the database defined by the DSN. The table name, as well as the names of its columns, the user ID, and the password should be obtained from the administrator responsible for the database.

SQL Column Data Types

The EDMMSQLP and EDMMSQLG methods support the following SQL-column data types:

CHAR	VARCHAR	LONGVARCHAR	SMALLINT
INTEGER	TINYINT	REAL	DOUBLE
FLOAT	DECIMAL	NUMERIC	DATE/TIME (TIMESTAMP)

All numeric types go into the EDMMSQLP method through the variable's character strings. They are handled by the methods in the same way that character strings are. The column name that is provided by the input object (control object or control file) should have the appropriate SQL type in order to ensure that the data is inserted properly.

Table 103: SQL Column Data Types and their Definitions

SQL Column Data Type	Description
CHAR	array of character
VARCHAR	array of character
LONGVARCHAR	array of character
SMALLINT	short integer
INTEGER	long integer
TINYINT	signed INT8
REAL	float
DOUBLE	double
FLOAT	double

SQL Column Data Type	Description
DECIMAL	long float
NUMERIC	long float
DATE/TIME (TIMESTAMP)	this data type is dependent on the database being used

This setting has different (short and long) date/time-stamp formats in different databases. For example, in Microsoft Access, it has the Date/Time name, but has "General Date" "Long Date" in Microsoft SQL Server database.

The EDMMSQLP and EDMMSQLG methods use only full timestamps to store the date and time in one column variable.

Depending on the default settings of the database in use, the date and/or time will be stamped by the database when the user fails to provide the full DATE/TIME value for the EDMMSQLG and/or EDMMSQLP methods.

For *decimal* and *numeric* SQL column types, the ODBC standard uses a default placeholder (of an array of characters) to transfer data back and forth, though the suitable program data type is "double".

Since Radia objects used to have character variables as placeholders of these types also, the variables naturally fit ODBC. Extracting the numeric and decimal fields into the program variable from the object is beyond the capability of these methods. Therefore, we recommend that you have the "double" type in the program in order to store the real value, in case the program needs it for arithmetic manipulation.

The lengths of variables provided for the EDMMSQLP method depend on the conversion, like the ftoa() function for the FLOAT type in the calling program. For the DATE/TIME (TIMESTAMP) type in the SQL database, we use the variable, EDM_TIMESTAMP for the put and get methods. This means the source object should have a value specified for this variable type for the put method, and should have a variable placeholder of the same type for the get method.

➤ The same EDM_TIMESTAMP variable in the object can be used to save the date and time in CHARACTER-type columns of the SQL database. However, it will be a simple string, without some specific helpful features provided for SQL-DATE/TIME-timestamps by the database.

The WHERE Clause

The WHERE clause in the control object for EDMMSQLP identifies the rows of the SQL table that are to be replaced by using the literal string of the WHERE variable value as the body of the WHERE clause and ignoring any variables with a U specified. If the optional WHERE variable is not specified in the control object, and some of the variables are specified with U, the method will work transparently (as it is fully compatible with previous implementations of the U sub-parameter), and generate the WHERE clause using variables specified with U.

Having the WHERE clause in place in the control object, the method does not generate the WHERE, but rather, uses the provided string from the WHERE variable of the control object.

The WHERE clause, if defined, will be used and is taken as the only one having priority, and the internal WHERE is not generated as described in the VC keyword. The clause can have a substitution in standard notation.

Considerations

- If one VC pair has the U sub-parameter coded, the WHERE clause that is generated will be in the form:

  WHERE fieldname = current source object corresponding variable value

  The current source object corresponding variable value is the value currently held (at the time of the call to EDMMSQLP) in the source object variable that is coded in the control information's VC pair (that links the variable with the back-end database fieldname).

- If more than one VC pair has the U sub-parameter coded, the WHERE clause will join them with an "AND", as below:

  WHERE *`fieldname1 = value1`* AND *`fieldname2 = value2`*

- If executing the REPLACE statement (PUTTYPE=R) with the WHERE clause yields zero matching records, EDMMSQLP will regenerate the SQL statement as an INSERT, and insert the record.
- If executing the REPLACE statement (PUTTYPE=R) with the WHERE clause finds exactly one record in the database, it is replaced with the source-object information.
- If executing the REPLACE statement (PUTTYPE=R) with the WHERE clause finds more than one record in the database, all database records are replaced with the source-object information.
- If executing the REPLACE statement (PUTTYPE=R) and no VC pairs in the control input have the U sub-parameter coded, EDMMSQLP terminates and logs the following error message:

```
NO UNIQUE COLUMNS FOUND FOR WHERE CLAUSE. TERMINATING.
```

Usage

There are two ways to use the WHERE parameter in the control object.

- Specify the WHERE clause in the control object, as in the example in Table 104.

Table 104: Simple WHERE Clause Usage

Variable Name	Variable Value
SQLTABLE	joe.USERS
SQLUSER	Vladimir
SQLPASSW	Rabbit
SQLTOUT	15
VC000	ZUSERID,USER,U
VC001	ZBIOS,BIOS
VC002	ZOS,OS
VC003	ZOSVER,OS_VERSION
WHERE	BIOS='V2.0'
SRCOBJ	SQLSRC
SQLDSN	SQLSVR01

The generated UPDATE statement for the EDMMSQLP method will have the WHERE clause as in the following:

```
UPDATE joe.USERS  SET joe.USERS.USER
='Value_For_USER_From_SRCOBJ',

joe.USERS.BIOS ='Value_For_BIOS_From_SRCOBJ',…… WHERE
BIOS='V2.0'
```

Here, the first variable (VC000) is not used as a key in the WHERE clause, even though it has a U. This U is ignored due to the presence of the WHERE variable. On the contrary, in the example in Table 105, we see the same control object, without the WHERE clause, and the subsequent UPDATE statement that is generated.

Table 105: Control Object without WHERE Clause

Variable Name	Variable Value
SQLTABLE	joe.USERS
SQLUSER	Vladimir
SQLPASSW	Rabbit
SQLTOUT	15
VC000	ZUSERID,USER,U
VC001	ZBIOS,BIOS
VC002	ZOS,OS
VC003	ZOSVER,OS_VERSION
SRCOBJ	SQLSRC
SQLDSN	SQLSVR01

The generated UPDATE statement for the EDMMSQLP method will have the WHERE clause as in the following:

```
UPDATE joe.USERS  SET joe.USERS.USER
='Value_For_USER_From_SRCOBJ',

joe.USERS.BIOS ='Value_For_BIOS_From_SRCOBJ',

joe.USERS.OS ='Value_For_OS_From_SRCOBJ',

WHERE USER='Value_For_USERS_From_SRCOBJ'
```

- This method of WHERE clause usage actually requires a substitution (in the clause) with the help of the source object variables. Therefore, assuming the source object has the variable ZSOURCE1, the WHERE parameter can be present in the control object, and specified as shown in Table 106.

Table 106: Substitution use with WHERE Clause

Variable Name	Variable Value
SQLTABLE	joe.USERS
SQLUSER	Vladimir
SQLPASSW	Rabbit
SQLTOUT	15
VC000	ZUSERID,USER,U
VC001	ZBIOS,BIOS
VC002	ZOS,OS
VC003	ZOSVER,OS_VERSION
WHERE	BIOS='&(ZSOURCE1)'
SRCOBJ	SQLSRC
SQLDSN	SQLSVR01

The generated UPDATE statement for the EDMMSQLP method will have the WHERE clause as in the following:

```
UPDATE joe.USERS SET joe.USERS.USER
='Value_For_USER_From_SRCOBJ',

joe.USERS.BIOS ='Value_For_BIOS_From_SRCOBJ',……

WHERE BIOS='Value_Substituted_From_SRCOBJ_For_ZSOURCE1'
```

Additionally, the WHERE clause simplifies multi-heap substitution. The processing is the same, but there are multiple updates to the SQL table as the result of a single EDMMSQLP occurrence. Each heap from the source object is transferred to the rows of the SQL database table, thereby satisfying the WHERE condition.

This substitution takes the values from the source object provided for this operation only.

For the following example, we will re-use the control object from the previous example, and assume that the source object is a two-heap object. The following tables show the source object heaps.

Table 107: Multi-Heap Source Object: HEAP1

Variable Name	Variable Value
ZBIOS	"R5"
ZOS	"NT"
ZSOURCE1	"R4"

Table 108: Multi-Heap Source Object: HEAP2

Variable Name	Variable Value
ZOSVER	7
ZSOURCE1	"R5"

The UPDATE statements that are generated for the EDMMSQLP method will have the WHERE clauses specified as in the following:

HEAP1:

```
UPDATE joe.USERS  SET BIOS = "R5", OS = "NT"
WHERE BIOS = "R4"
```

HEAP2:

```
UPDATE joe.USERS  SET OS_VERSION = 7
WHERE BIOS = "R5"
```

Design Considerations

Keep in mind these points when designing data exchange solutions using EDMMSQLG and EDMMSQLP.

1 EDMMSQLG and EDMMSQLP do not process back-end database column names that contain embedded spaces.

2 EDMMSQLG creates an object to receive data from the back-end database. Variables created in this object to hold text fields from the back-end database will be 255 characters in length, regardless of the length defined for the field in the back-end database. You might need to trim these variables to a shorter length before using them in symbolic substitution. There is an example of how to do this in How to Invoke EDMMSQLG on page 226.

Troubleshooting

If data does not transfer successfully to the back-end database tables or to the in-storage object as expected, the first place to look is in the Configuration Server log.

The Configuration Server Log

Here is an SQL example from the Configuration Server log:

```
EDM0532I 16:28 [208.244.225.133 /278]  EDMV4 Client   ---EDMGET 0[SQLTABLE] VN[VC015     ]
        (L)255 V[]
EDM0999I 16:28 [208.244.225.133 /278]  EDMV4 Client   ---GET ZMASTER .ZUSERID  (1) (5) 'jsmith'
EDM0532I 16:28 [208.244.225.133 /278]  EDMV4 Client   ---EDMGET 0[ZMASTER ] VN[ZUSERID ] (L)5 V
        [jsmith]
EDM0999I 16:28 [208.244.225.133 /278]  EDMV4 Client   ---GET ZMASTER .ZSYSDATE (1) (8)
        '19990408'
EDM0532I 16:28 [208.244.225.133 /278]  EDMV4 Client   ---EDMGET 0[ZMASTER ] VN[ZSYSDATE] (L)8
        V[19990408]
EDM0999I 16:28 [208.244.225.133 /278]  EDMV4 Client   ---GET ZMASTER .ZSYSTIME (1) (8)
        '16:20:54'
EDM0532I 16:28 [208.244.225.133 /278]  EDMV4 Client   ---EDMGET 0[ZMASTER ] VN[ZSYSTIME] (L)8
        V[16:20:54]
EDM0999I 16:28 [208.244.225.133 /278]  EDMV4 Client   ---GET ZMASTER .PRICE    (1) (16)
        '0000000000013990'
EDM0532I 16:28 [208.244.225.133 /278]  EDMV4 Client   ---EDMGET 0[ZMASTER ] VN[PRICE   ] (L)16
        V[0000000000013990]
EDM0000E 16:28 [208.244.225.133 /278]  EDMV4 Client   --![Microsoft][ODBC FoxPro Driver] The
        Microsoft Jet database engine cannot open the file 'C:\Data\BILLS.DBF'. It is already
        opened exclusively by another user, or you need permission to view its data.
EDM0000E 16:28 [208.244.225.133 /278]  EDMV4 Client   --!ODBC execute error, DSN=[EDM Data],
        RC=[-8], SQLState=[S1000]
EDM2500E 16:25 [208.244.225.133 /278]  EDMV4 Client   --!Failed to execute INSERT statement for
        object [ZMASTER ] heap [1]
```

In this example, the ODBC FoxPro driver was unable to store the APPEVENT object in the FoxPro table because another user concurrently opened the table. When designing solutions using ODBC-compliant back-end databases, you must take into account when, and how the data will be shared.

In this case, none of the VC pairs in the control information had the third sub-parameter (U) coded, so EDMMSQLP was unable to identify a key field to use in a WHERE clause, which specifies which record in the back-end data table to update.

The data extraction process is sensitive to typing errors. Check all typing carefully, and trace through all connections. Most of the time, your problem will be typographical.

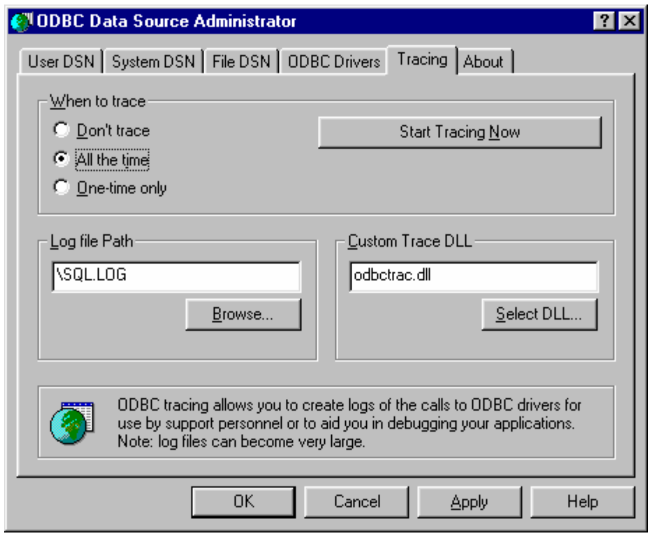
The WHERE clause you provide to EDMMSQLG must comply syntactically with ODBC requirements for quotation-mark usage when specifying literals, and any syntactical requirements imposed by your back-end database.

ODBC Tracing

If the error is not immediately apparent from the Configuration Server log, it is often helpful to repeat the operation with ODBC tracing enabled. ODBC tracing is very detailed and can generate a large log very quickly, so only enable ODBC tracing while you are actively debugging a problem.

One benefit of ODBC tracing is that the log shows you the SQL statement generated by EDMMSQLP or EDMMSQLG, and what ODBC and the back-end database driver did with it.

To enable ODBC tracing, open the Control Panel ODBC applet, and choose the **Tracing** tab. The resulting dialog box will resemble the following:



When to trace: Don't trace [✓ All the time] One-time only

Log file Path: \SQL.LOG

Custom Trace DLL: odbctrac.dll

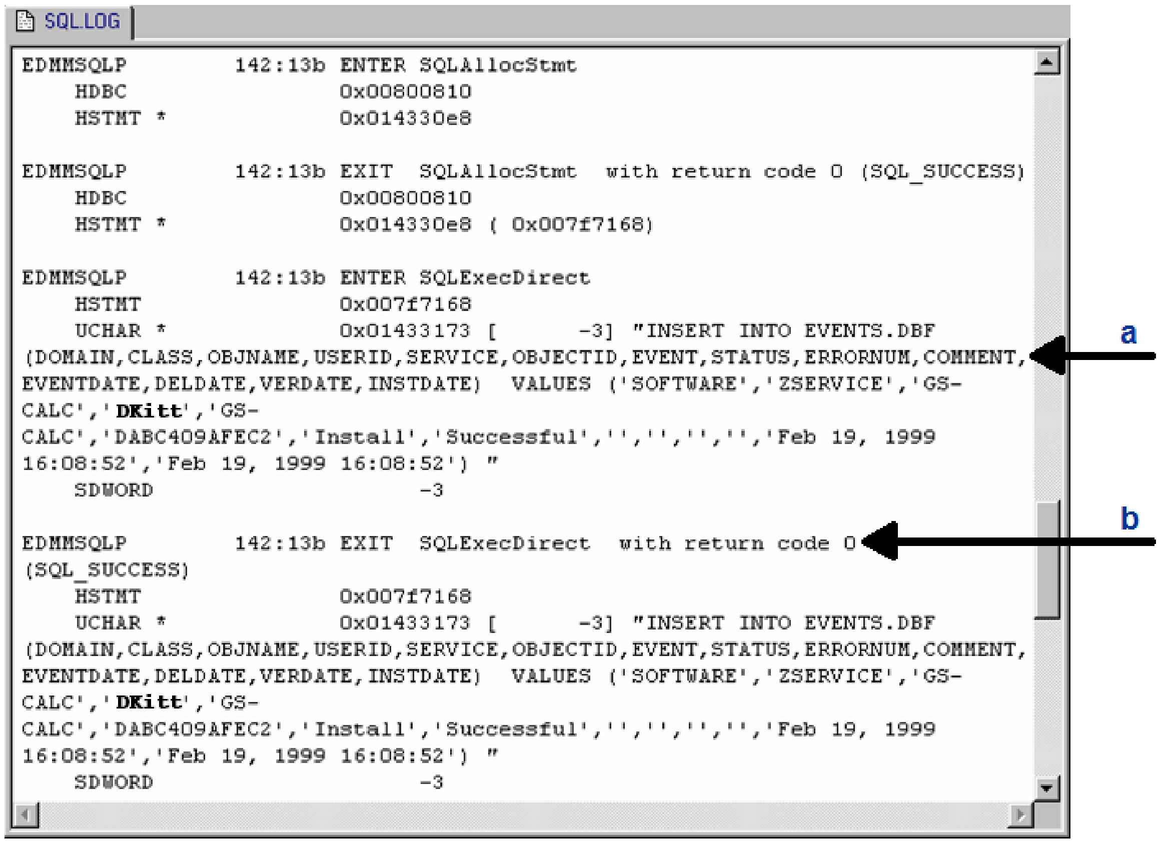
ODBC tracing allows you to create logs of the calls to ODBC drivers for use by support personnel or to aid you in debugging your applications. Note: log files can become very large.

Figure 31: The ODBC Data Source Administrator window – Tracing tab.

- In the **When to trace** area select **All the time**.
- Set the **Log file Path** to identify the ODBC log file. This example will store a log file named `SQL.LOG` in the root directory of the current drive.
- Then, in the **When to trace** area, click **Start Tracing Now**. The button face changes to say **Stop Tracing Now**. All ODBC operations from this point forward will be logged in the `SQL.LOG` file.

To turn off ODBC tracing, return to the **Tracing** tab, and click **Stop Tracing Now**.

Figure 32 on page 272 presents a sample of the `SQL.LOG` file for a successfully processed SQL statement generated by EDMMSQLP.



```
EDMMSQLP        142:13b ENTER SQLAllocStmt
    HDBC                0x00800810
    HSTMT *             0x014330e8

EDMMSQLP        142:13b EXIT  SQLAllocStmt  with return code 0 (SQL_SUCCESS)
    HDBC                0x00800810
    HSTMT *             0x014330e8 ( 0x007f7168)

EDMMSQLP        142:13b ENTER SQLExecDirect
    HSTMT               0x007f7168
    UCHAR *             0x01433173 [      -3] "INSERT INTO EVENTS.DBF
(DOMAIN,CLASS,OBJNAME,USERID,SERVICE,OBJECTID,EVENT,STATUS,ERRORNUM,COMMENT,
EVENTDATE,DELDATE,VERDATE,INSTDATE)  VALUES ('SOFTWARE','ZSERVICE','GS-
CALC','DKitt','GS-
CALC','DABC409AFEC2','Install','Successful','','','','','Feb 19, 1999
16:08:52','Feb 19, 1999 16:08:52') "
    SDWORD                    -3

EDMMSQLP        142:13b EXIT  SQLExecDirect  with return code 0
(SQL_SUCCESS)
    HSTMT               0x007f7168
    UCHAR *             0x01433173 [      -3] "INSERT INTO EVENTS.DBF
(DOMAIN,CLASS,OBJNAME,USERID,SERVICE,OBJECTID,EVENT,STATUS,ERRORNUM,COMMENT,
EVENTDATE,DELDATE,VERDATE,INSTDATE)  VALUES ('SOFTWARE','ZSERVICE','GS-
CALC','DKitt','GS-
CALC','DABC409AFEC2','Install','Successful','','','','','Feb 19, 1999
16:08:52','Feb 19, 1999 16:08:52') "
    SDWORD                    -3
```

Figure 32: SQL.LOG file with processed SQL statement generated by EDMMSQLP.

a The SQL statement generated by EDMMSQLP.

b The SQL statement was processed successfully

Iterative Simplification

Finally, a productive way to isolate a failure is to iteratively simplify the connection until the problem disappears. The last simplifying change you make before a successful connection locates the error.

6 Radia Database Utilities

At the end of this chapter, you will:

- Be able to manage the Radia database using the Radia database utilities.

HP strongly recommends shutting down the Configuration Server to ensure that the database contents do not change during the execution of the database import and export utilities.

HP further recommends backing up the Radia database prior to running any of the import utilities against an existing database.

Radia Database Utility Programs

The Configuration Server database utilities are initially used by the installation program to install the Radia database. You can then use these utilities to take a test environment database to the production environment, overlay an existing database, and move a database configuration from one operating system to another. The database utilities are located in the Configuration Server's `bin` subdirectory (Windows); `exe` subdirectory (UNIX); and in the MVS LOAD dataset.

If you do not specify optional command-line parameters for the utilities in this chapter, the default value for the optional parameter will be used.

Table 109: The Radia Database Utility Programs List

Utility	Description
EDMMEXPC	Exports class data from the Radia database and generates a comma-separated variable file that can be imported into another Radia database (using EDMMIMPC). Contents include class template and base instance values.
EDMMEXPI	Exports instance data (objects) from the Radia database as a comma-separated variable file for input into a third party tool for reporting purposes or for input into another Radia database (using EDMMIMPR) .
EDMMEXPR	Exports resources from the Radia database, and generates a binary file that can be imported into another Radia database (using EDMMIMPR).
EDMMIMPC	Imports class data and its base instance from a comma-separated variable file produced when using EDMMEXPC. Imported files or records are updated with a new object ID and time and date stamp if the TIME=NEW parameter is used.
EDMMIMPI	Reads a comma-separated variable list and creates instances from the list created by EDMMEXPI.
EDMMIMPR	Imports resources into the Radia database. Instances with the same names might be replaced with resource data contained in the input file.

It is imperative that you periodically back up your production database when working with mission-critical production applications, and prior to making changes to the Radia database.

Furthermore, to avoid changes to the Radia database while you are working with its contents, do not invoke these utilities while the Configuration Server is running.

Import and Export Utilities

The following pages document the HP utilities that enable you import/export data to and from data sets and files.

- If using a log scraper, it is important to analyze the revised logs.
- If you are depending on specific non-zero return codes from an import or export operation, you should appropriately revise your dependencies.

HP strongly recommends that you use the appropriate ZEDMAMS verbs (see Chapter 7, EDM Access Method Services) rather than the six import and export utilities (EDMMEXPC, EDMMEXPI, EDMMEXPR, EDMMIMPC, EDMMIMPI, and EDMMIMPR) documented here. These utilities are being phased out and might not be supported in future versions of the Configuration Server.

EDMMEXPC

EDMMEXPC is a utility that exports classes from the Radia database. The resulting file contains the exported class templates and base instances, and can be imported into another Radia database using the EDMMIMPC import facility.

HP recommends that the ZEDMAMS verb, EXPORT_CLASS, be used instead of this utility.

You will need to specify certain parameters when executing EDMMEXPC. The literal parameters, such as FILE=, as well as their values, such as PRIMARY, must be specified in uppercase.

- For Intel Configuration Servers, be sure that the `edmprof` file (`edmprof`.dat) is in the same directory from which the program is executed. The database path is specified in the DBPATH parameter.
- For UNIX Configuration Servers, be sure that the `edmprof` file (`~/.edmprof`) has the DBPATH parameter set to the appropriate database path.
- For MVS Configuration Servers, the DBPATH parameter in the `edmprof` file (PARMLIB) will determine the database path.

Syntax

```
EDMMEXPC FILE=,(DOMAIN=),(CLASS=),(PREVIEW=YES/NO),
OUTPUT=output_file),(COMMENT=comment),(INPUT=input_file),(HEAD
ER=YES/NO),(LOGFILE=)
```

Specifying the EDMMEXPC Utility

The following table lists and describes the command parameters for the EDMMEXPC utility.

Table 110: EDMMEXPC Command Parameters

Parameter	What To Specify
file	Specify the name of the source Radia database by name. PRIMARY is the only valid option. Does not allow wildcards.
domain	Specify the name of the source domain, such as SYSTEMX under the PRIMARY file. Allows only the asterisk wildcard (*).

Parameter	What To Specify
class	Specify the name of the source class, such as USER, DEPT, WORKGRP, or other under the SYSTEMX domain. Allows only the asterisk wildcard (*).
preview	Specify YES if you want a preview listing of the exported classes in the input file. Specify NO if you do not want to perform the export. The default is YES.
output	Specify the name of the destination output file (with extension) where you want the exported data to reside. The default is the current window (stdout). Ignored if PREVIEW=YES.
comment	Specify this parameter if you want to add a comment to the output file header.
input	Specify, in the following format, a file that contains the classes `FILE=`*`file_name`*`,DOMAIN=`*`domain_name`*`,CLASS=`*`class_name`*`, INSTANCE=.`
header	Specify YES if you want to produce an output header file. Specify NO if you do not want an output header file. The default is NO.

MVS User's Note

The following is a sample JCL for EDMMEXPC:

```
//*
//IMPCPRIM EXEC PGM=EDMMIMPC,
//
PARM='PREVIEW:NO,FILE://DSN:YOUR.PDS(CPRIM)',
//              COND=(0,NE)
//STEPLIB   DD DSN=YOUR.EDM.LOAD,DISP=SHR
//          DD DISP=SHR,DSN=SASC.C650.LINKLIB
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//PREVIEW   DD SYSOUT=*
//LOG       DD SYSOUT=*
//SYSTERM   DD SYSOUT=*
//STGRPT    DD SYSOUT=*
//PARMLIB   DD DSN=YOUR.EDM.PARMLIB(STARTUP),DISP=SHR
//PRIMARY   DD DISP=SHR,DSN=YOUR.EDM.PRIMARY,
//    AMP=('STRNO=10,BUFNI=30,BUFND=100')
//*
```

Examples

The following table shows some of the ways to specify the EDMMEXPC command options.

Table 111: EDMMEXPC Command Options

Command	Explanation
`EDMMEXPC PREVIEW=NO, FILE=PRIMARY,OUTPUT=MYFILE.bin`	Exports all instances under the PRIMARY file.
`EDMMEXPC PREVIEW=NO, FILE=PRIMARY,DOMAIN=SYSTEMX, CLASS=ZRSOURCE,OUTPUT=MYFILE.bin`	Exports the ZRSOURCE class template and base instance attribute values under the PRIMARY file, SYSTEMX domain.
`EDMMEXPC PREVIEW=YES, FILE=PRIMARY,OUTPUT=MYFILE.bin`	Creates a preview of the expected export. The preview results will be shown to the edmmexpc.log file.
`EDMMEXPC PREVIEW=NO,FILE=PRIMARY, COMMENT="My comment here"`	Inserts the comment specified at the head of the output file.

EDMMEXPI

EDMMEXPI is a utility that enables the exporting of objects from the database. The EDMMEXPI database tool is an extension of the functionality of the original EDMMEXPO tool.

HP recommends that the ZEDMAMS verb, EXPORT_INSTANCE, be used instead of this utility.

This program generates a modified comma-separated variable (CSV) file of an object. You can restrict the attribute data retrieved by EDMMEXPI or it can retrieve all attribute data contained within the instances that you select at the command line.

The resulting modified CSV file can be imported into another Radia database using the import facility, EDMMIMPI, or used to generate reports by using third-party vendor software (for example, Microsoft Access, Microsoft Excel).

Do not modify the exported CSV file prior to importing it.

To format the modified CSV output file for import into third-party vendor software, you can run a HP-provided REXX program, called EXPORT.REX, against the output file of EDMMEXPI. This REXX will reformat the output of EDMMEXPI into a true CSV file.

In generating a CSV output file, EDMMEXPI will create unique names for connects and methods in a class. Connects can be identified by the CONNnnnn format, and methods by METHnnnn.

This is only when the REPORT=YES switch is selected. Select REPORT=NO if you intend to import to another Radia database.

EDMMEXPI exports the instances only. It does not export the classes or any underlying resources for the objects. To export the resources, refer to the EDMMEXPR program.

You will need to specify certain parameters when executing EDMMEXPI. The literal parameters, such as FILE=, as well as their values, such as PRIMARY, must be specified in uppercase.

- For Intel Configuration Servers, be sure that the `edmprof` file (`edmprof.dat`) is in the same directory from which the program is executed. The database path is specified in the DBPATH parameter.

- For UNIX Configuration Servers, be sure that the `edmprof` file (`~/.edmprof`) has the DBPATH parameter set to the appropriate database path.
- For MVS Configuration Servers, the DBPATH parameter in the `edmprof` file (PARMLIB) will determine the database path.

Syntax

```
EDMMEXPI FILE=,(DOMAIN=),(CLASS=),(INSTANCE=),(FROMINST=)
,(TOINST=),(PREVIEW=YES/NO),(FROMDATE=),(TODATE=),(FROMTIME=)
,(TOTIME=),(OUTPUT=),(KEEP=),(DROP=),(ORDER=),(COMMENT=)
,(REPORT=YES/NO),(CSVL=YES/NO),(INPUT=),(HEADER=YES/NO)
,(PHEX=YES/NO),(DELETE=YES/NO),(LOGFILE=)
```

Specifying the EDMMEXPI Utility

Table 112 describes the command parameters for the EDMMEXPI utility.

Table 112: EDMMEXPI Command Parameters

Parameter	What To Specify
base	Specify this parameter if you want to include all the attributes that are predefined in the base instance.
class	Specify the name of the source class, such as USER, DEPT, WORKGRP, under the respective domain. Allows only the asterisk wildcard (*).
comment	Specify this parameter if you want to add a comment to the optional output file header.
CSVL	Specify YES to produce a CSV listing for all attribute values. Currently, the output file specified with CSVL=YES is created in a subdirectory of the current working directory. Specify NO if you do not want a CSV output file. The default is NO.
domain	Specify the name of the source domain, such as SYSTEMX under the PRIMARY file. Allows only the asterisk wildcard (*).
drop	Specify the instance attributes that you do not want to retain in the resulting file.

Parameter	What To Specify
file	Specify the name of the source Radia database by name. The only valid options are PRIMARY, PROFILE, and HISTORY. Does not allow wild cards.
fromdate	Specify the start of a range of instances (by date) to export. The correct date format is YYYYMMDD.
frominst	Specify the start of a range of instances (by instance name) to export. Allows wild cards.
fromtime	Specify the start of a range of instances (by time) to export. The correct time format is HH:MM:SS.
header	Specify YES if you want to produce an output header file. Specify NO if you do not want an output header file. The default is NO.
input	Specify a file that contains the instances. Use the following format `FILE=`*`file_name`*`,DOMAIN=`*`domain_name`*`,CLASS=`*`class_name`*`,INSTANCE =`*`instance_name`*.
instance	Specify the name of the source instance, such as ZCONFIG of the PROFILE file. Allows prefixes and the asterisk wildcard (*).
keep	Specify the instance attributes that you want to retain in the resulting file.
order	Specify YES if you want the resulting file to be ordered by attribute names.
output	Specify the name of the destination output file (with extension) where you want the exported data to reside. The default is the current window (stdout).
phex	Specify YES if you want to output the data portion of variables in printable hexadecimal. Specify NO if you do not want an output of the data portion of variables in printable hexadecimal. The default is NO.
preview	Specify YES if you want a preview listing of the exported object instances in the input file. Specify NO if you do not want to perform the export. The default is YES.

Parameter	What To Specify
todate	Specify the end of a range of instances (by date) to export. The correct date format is YYYYMMDD.
toinst	Specify the end of a range of instances (by instance name) to export. Allows wild cards.
totime	Specify the end of a range of instances (by time) to export. The correct time format is HH:MM:SS.

MVS User's Note

The following is a sample JCL for EDMMEXPI:

```
//*
//IMPIPRIM EXEC PGM=EDMMEXPI,
//PARM='PREVIEW:NO,FILE:PRIMARY,OUTPUT://DSN:YOUR.PDS(IPRIM)'
//             COND=(0,NE)
//STEPLIB  DD DSN=YOUR.EDM.LOAD,DISP=SHR
//         DD DISP=SHR,DSN=SASC.C650.LINKLIB
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//PREVIEW  DD SYSOUT=*
//LOG      DD SYSOUT=*
//SYSTERM  DD SYSOUT=*
//STGRPT   DD SYSOUT=*
//PARMLIB  DD DSN=YOUR.EDM.PARMLIB(STARTUP),DISP=SHR
//PRIMARY  DD DISP=SHR,DSN=YOUR.EDM.PRIMARY,
//   AMP=('STRNO=10,BUFNI=30,BUFND=100')
//*
```

Examples

Table 113 shows some of the ways you can specify the EDMMEXPI command options.

Table 113: EDMMEXPI Command Options

Command	Explanation
`EDMMEXPI PREVIEW=NO,0 FILE=PRIMARY ,0 OUTPUT=report.fil`	Exports all objects under the PRIMARY file placing the result of the export in the file `report.fil`.

Command	Explanation
`EDMMEXPI PREVIEW=NO,FILE=PRIMARY ,DOMAIN=SYSTEMX, CLASS=USER, FROMINST="DIFF*",TOINST="DIFF*" ,OUTPUT=report.fil`	Exports all the objects that begin with a prefix of DIFF under the USER class, SYSTEMX domain, PRIMARY file.
`EDMMEXPI PREVIEW=NO,BASE=YES, FILE=PRIMARY,DOMAIN=SYSTEMX, CLASS=USER,OUTPUT=report.fil`	Exports all the objects under the USER class, SYSTEMX, PRIMARY file, inheriting the predefined values from the base instance.
`EDMMEXPI PREVIEW=YES, FILE=PRIMARY,DOMAIN=SYSTEMX, CLASS=USER,INSTANCE="*"`	Exports all the objects under the USER class, SYSTEMX, PRIMARY file, displaying the expected results to the edmmexpi.log file.

EDMMEXPR

EDMMEXPR is a utility that exports resources from the Configuration Server resource database. The resulting file can be imported into another Radia database using the EDMMIMPR import facility.

HP recommends that the ZEDMAMS verb, EXPORT_RESOURCE, be used instead of this utility.

You will need to specify certain parameters when executing EDMMEXPR. The literal parameters, such as FILE=, as well as their values, such as PRIMARY, must be specified in uppercase.

- For Intel Configuration Servers, be sure that the `edmprof` file (`edmprof.dat`) is in the same directory from which the program is executed. The database path is specified in the DBPATH parameter.
- For UNIX Configuration Servers, be sure that the `edmprof` file (`~/.edmprof`) has the DBPATH parameter set to the appropriate database path.
- For MVS Configuration Servers, the DBPATH parameter in the `edmprof` file (PARMLIB) will determine the database path.

EDMMEXPR is used to join resource data from the HP configuration file (for example, PRIMARY) into a single platform-independent binary file that can be used between Configuration Server platforms.

Syntax

```
EDMMEXPR FILE=,(DOMAIN=),(CLASS=),(INSTANCE=),(FROMINST=),
(TOINST=),(PREVIEW=YES/NO),(FROMDATE=),(TODATE=),(FROMTIME=),
(TOTIME=),(OUTPUT=),(COMMENT=YES/NO),(INPUT=),(HEADER=YES/NO),
(DELETE=YES/NO),(LOGFILE=)
```

Specifying the EDMMEXPR Utility

Table 114 describes the command parameters for the EDMMEXPR utility.

Table 114: EDMMEXPR Command Parameters

Parameter	What To Specify
preview	Specify YES if you want a preview listing of the exported resources in the input file. Specify NO if you do not want to perform the export. The default is YES.
file	Specify the name of the source Radia database. PRIMARY and RESOURCE are the only valid options. Does not allow wildcards.
domain	Specify the name of the source domain, such as SYSTEMX under the PRIMARY file. Allows only the asterisk wildcard (*).
class	Specify the name of the source class, such as the ZRSOURCE. Allows only the asterisk wildcard (*).
frominst	Specify the start of a range of resources (by instance name) to export. Allows wildcards. (Parameters are not valid when FILE=RESOURCE.)
toinst	Specify the end of a range of resources (by instance name) to export. Allows wildcards. (Parameters are not valid when FILE=RESOURCE.)
fromdate	Specify the start of a range of resources (by date) to export. The correct date format is YYYYMMDD.
fromtime	Specify the start of a range of resources (by time) to export. The correct time format is HH:MM:SS.
todate	Specify the end of a range of resources (by date) to export. The correct date format is YYYYMMDD.
totime	Specify the end of a range of resources (by time) to export. The correct time format is HH:MM:SS.
output	Specify the name of the destination output file (with extension) where you want the exported data to reside. The default is the current window (stdout).
comment	Specify this if you want to add a comment to the output file header.
input	Specify a file which contains the instances, using the following format `FILE=`*`file_name`*`,DOMAIN=`*`domain_name`*`,CLASS=`*`class_name`*`,FROMINST =`*`instance_name`*`,TOINST=`*`instance_name`*.
header	Specify YES if you want to produce an output header file. Specify NO if you do not want an output header file. The default is NO.

MVS User’s Note

The following is a sample JCL for EDMMEXPR:

This JCL is an example of when it is necessary to call in a PDS member.

```
//RUNEXPR   JOB ('EDM'),'EDM:MANAGER UTILITY',
//  NOTIFY=&SYSUID,TIME=(1439),MSGCLASS=A
//*
//EDMMEXPR EXEC PGM=EDMMEXPR,REGION=0M,
// PARM=('INPUT=//DDN:SYSIN')
//*
//STEPLIB  DD DISP=SHR,DSN=EDM.V4X.LOAD
//         DD DISP=SHR,DSN=EDM.V4X.SASCLINK
//*
//PREVIEW  DD SYSOUT=*
//LOG      DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSTERM  DD SYSOUT=*
//STGRPT   DD SYSOUT=*
//*
//PARMLIB  DD DISP=SHR,DSN=EDM.V4X.PARMLIB(STARTUP)
//*
//PRIMARY  DD DISP=SHR,DSN=EDM.V4X.PRIMARY
//*
//RESOURCE DD DISP=SHR,DSN=EDM.V4X.RESOURCE
//*
//HISTORY  DD DISP=SHR,DSN=EDM.V4X.HISTORY
//*
//AUDIT    DD DISP=SHR,DSN=EDM.V4X.AUDIT
//*
//EXPORT   DD DISP=SHR,DSN=EDM.V4X.EDMEXPR.EXPORTED
//*
//SYSIN    DD DISP=SHR,DSN=XXX.XX.PDS(INFILE)
//*
//
```

```
All this information should be coded on one line in the
PDS member INFILE:FILE=PRIMARY,PREVIEW=NO,DOMAIN=SYSTEMX,
CLASS=ZRSOURCE,FROMINST=ACISS_A*,TOINST=ACISS_9*,OUTPUT=/
/DDN:EXPORT.
```

Examples

Table 115 shows some of the ways to specify the EDMMEXPR command options.

Table 115: EDMMEXPR Command Options

Command	Explanation
EDMMEXPR PREVIEW=NO ,FILE=PRIMARY, OUTPUT=MYFILE.bin	Exports all RESOURCE data for the instances under the PRIMARY file.
EDMMEXPR PREVIEW=NO ,FILE=PRIMARY,FROMINST= "TSO*",TOINST="TSO*" ,OUTPUT=MYFILE.bin	Exports all the RESOURCE data that begin with a prefix of TSO under the PRIMARY file, and produces an export file called `MYFILE.bin`.
EDMMEXPR PREVIEW=NO ,FILE=PRIMARY,DOMAIN= SYSTEMX,CLASS=ZRSOURCE, OUTPUT=MYFILE.bin	Exports all the RESOURCE data under the PRIMARY file, SYSTEMX domain, ZRSOURCE class.
EDMMEXPR PREVIEW=YES, FILE=PRIMARY,OUTPUT= MYFILE.bin	Creates a preview of the expected export. The preview results will be shown to the file `edmmexpr.log`.
EDMMEXPR PREVIEW=YES, FILE=PRIMARY,OUTPUT= MYFILE.bin	Creates a preview of the expected export into the file `edmmexpr.log`.
EDMMEXPR PREVIEW=NO, FILE=RESOURCE,OUTPUT= MYFILE.bin	Exports all RESOURCE data for the instances under the RESOURCE file.
EDMMEXPR PREVIEW=NO, FILE=RESOURCE,DOMAIN= SYSTEMX,CLASS=ZRSOURCE, OUTPUT=MYFILE.bin	Exports all the RESOURCE data under the RESOURCE file, SYSTEMX domain, ZRSOURCE class.
EDMMEXPR PREVIEW=YES, FILE=RESOURCE,OUTPUT= MYFILE.bin	Creates a preview of the expected export. The preview results will be shown to the `edmmexpr.log` file.

EDMMIMPC

EDMMIMPC is a utility that is used to import classes into a Radia database. Any classes existing on the receiving Radia database that have the same name as a class being imported, will be replaced by the input class.

HP recommends that the ZEDMAMS verb, IMPORT_CLASS, be used instead of this utility.

You will need to specify certain parameters when executing EDMMIMPC. The literal parameters, such as FILE=, as well as their values, such as MYFILE, must be specified in uppercase.

When using EDMMIMPC, if REPLACE=YES is specified (or if it defaults to YES), a class being imported that has the same name as an existing class, will overwrite all instance and class information within the context of that class in the destination Radia database.

- For Intel Configuration Servers, be sure that the `edmprof` file (`edmprof.dat`) is in the same directory from which the program is executed. The database path is specified in the DBPATH parameter.
- For UNIX Configuration Servers, be sure that the `edmprof` file (`~/.edmprof`) has the DBPATH parameter set to the appropriate database path.
- For MVS Configuration Servers, the DBPATH parameter in the `edmprof` file (PARMLIB) will determine the database path.

Syntax

```
EDMMIMPC FILE=,(PREVIEW=YES/NO),(TIME=OLD/NEW/MOD),
(REPLACE=YES/NO),(XDF=YES/NO),(Y2K=YES/NO),(FROMDOMA=),
(TODOMA=),(LOGFILE=)
```

Specifying the EDMMIMPC Utility

Table 116 describes the command parameters for the EDMMIMPC utility.

Table 116: EDMMIMPC Command Parameters

Parameter	What To Specify
file	The keyword file should be followed by the filename that contains the output from the execution of the EDMMEXPC command.
fromdoma	Domain from which to import the data.
preview	Specify YES if you want to preview the expected results in the input file. No import will be performed. Specify NO if you want to apply the database changes contained in the input file. The default is YES.
replace	Specify YES if you want to replace the class template, if it already exists. The old class template and all instances will be deleted. Specify NO if you do not want to modify the class template. The default is NO.
time	Specify OLD if you want to retain the class object time stamps and object ID. Specify NEW if you want to update the class time stamp and the object ID based on the import. The default is OLD. The format is HH:MM:SS.
todoma	Domain into which to import the data. Note: This must be specified if FROMDOMA= is specified.
xdf	Effective only when creating a Primary file. The default is YES.
y2k	Specify YES to convert YY/MM/DD or MM/DD/YY to Y2K format. Please note that the Y2K date format is YYYYMMDD. Specify NO to retain the YY/MM/DD or MM/DD/YY format. The default is YES.

TIME=NEW should be used with caution. Introducing new class object IDs for previously deployed class and instance data will cause a redeployment of previously deployed instance data on the next client connect (for example, ZSERVICE, ZRSOURCE).

MVS User's Note

The following is a sample JCL for EDMMIMPC:

```
//*
//IMPCPRIM EXEC PGM=EDMMIMPC,
//           PARM='PREVIEW:NO,FILE://DSN:YOUR.PDS(CPRIM)',
//           COND=(0,NE)
//STEPLIB  DD DSN=YOUR.EDM.LOAD,DISP=SHR
//         DD DISP=SHR,DSN=SASC.C650.LINKLIB
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//PREVIEW  DD SYSOUT=*
//LOG      DD SYSOUT=*
//SYSTERM  DD SYSOUT=*
//STGRPT   DD SYSOUT=*
//PARMLIB  DD DSN=YOUR.EDM.PARMLIB(STARTUP),DISP=SHR
//PRIMARY  DD DISP=SHR,DSN=YOUR.EDM.PRIMARY,
//   AMP=('STRNO=10,BUFNI=30,BUFND=100')
```

Examples

Table 117 shows some of the ways to specify the EDMMIMPC command options.

Table 117: EDMMIMPC Command Options

Command	Explanation
`EDMMIMPC FILE=myfile, REPLACE=NO,PREVIEW=NO`	Imports the class templates and base instance data contained in the file `MYFILE` into the Radia database. It will not overwrite a class if the class in the input file already exists in the Radia database.
`EDMMIMPC PREVIEW=YES, FILE=MYFILE,REPLACE=NO`	Creates a preview listing of input file contents, and expected results can be viewed in the file `edmmimpc.log`.
`EDMMIMPC PREVIEW=NO, FILE=MYFILE,TIME=OLD, REPLACE=NO`	Imports the class templates and base instance data from the file, `MYFILE`. All object IDs and time/date information will be retained for the newly imported class and instance data. The import will not overwrite the templates and instances in the existing class.
`EDMMIMPC PREVIEW=NO, FILE=MYFILE,TIME=OLD, REPLACE=NO`	Imports the class template and base instance data from the file, `MYFILE`. All object IDs, and time/date information will be retained for the classes that existed prior to the import.

Command	Explanation
`EDMMIMPC FILE=MYFILE, TIME=NEW,PREVIEW=NO, REPLACE=NO`	Imports the class template data from the file, `MYFILE`. All new object IDs, and time/date information will be created for the newly imported class and instance data.

EDMMIMPI

EDMMIMPI is a utility that reads a comma-separated variable list and database objects from the input file.

HP recommends that the ZEDMAMS verb, IMPORT_INSTANCE, be used instead of this utility.

You will need to specify certain parameters when executing EDMMIMPI. The literal parameters, such as FILE=, as well as their values, such as MYFILE, must be specified in uppercase.

Prior to importing instance data into the Radia database, the class template that corresponds to the instance data being imported must already exist within the Radia database.

- For Intel Configuration Servers, be sure that the `edmprof` file (`edmprof.dat`) is in the same directory from which the program is executed. The database path is specified in the DBPATH parameter.
- For UNIX Configuration Servers, be sure that the `edmprof` file (`~/.edmprof`) has the DBPATH parameter set to the appropriate database path.
- For MVS Configuration Servers, the DBPATH parameter in the `edmprof` file (PARMLIB) will determine the database path.

Syntax

```
EDMMIMPI FILE=,(PREVIEW=YES/NO),(TIME=OLD/NEW/MOD),
(FROMMDOMA=)(TODOMA=),(CHGCONS=YES/NO),(REPLACE=YES/NO),
(Y2K=YES/NO),(VERIFY=),(LOGFILE=)
```

Specifying the EDMMIMPI Utility

Table 118 describes the command parameters for the EDMMIMPI utility.

Variable names that are present in the input data file but not defined in the class template will be dropped when the object is created.

Table 118: EDMMIMPI Command Parameters

Parameter	What To Specify
file	Specify the file containing the input information; that is, the objects you want to create or update.
fromdoma	Domain from which to import the data.
preview	Specify YES if you want a preview listing of the contents of the input file. Specify NO if you do not want to perform the import. The default is YES.
time	Specify OLD if you want to retain the class object time/date stamp and object ID. Specify NEW if you want to update the instance time/date stamp and the object ID based on the time/date of import. Specify MOD if you want to update the instance time/date stamp only leaving the original object ID. This value is recommended in change-control centric environments. The default is OLD. The format is HH:MM:SS.
todoma	Domain in which to import the data. Note: This must be specified if FROMDOMA= is specified.
chgcons	Change the connect values to the new domain (TODOMA=). Note: If FROMDOMA= and TODOMA= are specified, CHGCONS must be specified.
y2k	Specify YES if you want to convert YY/MM/DD or MM/DD/YY to Y2K format. Y2K date format is YYYYMMDD. Specify NO to retain the YY/MM/DD or MM/DD/YY format. The default is YES.

MVS User's Note

The following is a sample JCL for EDMMIMPI:

```
//*
//IMPIPRIM EXEC PGM=EDMMIMPI,
//          PARM='PREVIEW:NO,FILE://DSN:YOUR.PDS(IPRIM)',
//          COND=(0,NE)
//STEPLIB  DD DSN=YOUR.EDM.LOAD,DISP=SHR
//         DD DISP=SHR,DSN=SASC.C650.LINKLIB
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//PREVIEW  DD SYSOUT=*
//LOG      DD SYSOUT=*
//SYSTERM  DD SYSOUT=*
//STGRPT   DD SYSOUT=*
//PARMLIB  DD DSN=YOUR.EDM.PARMLIB(STARTUP),DISP=SHR
//PRIMARY  DD DISP=SHR,DSN=YOUR.EDM.PRIMARY,
//   AMP=('STRNO=10,BUFNI=30,BUFND=100')
```

Examples

Table 119 shows some of the ways to specify the EDMMIMPI command options.

Table 119: EDMMIMPI Command Options

Command	Explanation
EDMMIMPI FILE=myfile	Imports a file into a Configuration Server.
EDMMIMPI PREVIEW=YES, FILE=myfile	Creates a preview listing of input file contents, and expected results can be viewed in the file `edmmimpi.log`.
EDMMIMPI PREVIEW=YES, FILE=MYFILE	Creates a preview listing of the input file contents into `edmmimpi.log` for printing.
EDMMIMPI FILE=MYFILE, TIME=OLD,REPLACE=YES, PREVIEW=NO	Imports the instance data in the file `MYFILE`, and all object IDs. Time and date information will be retained for the imported instance data, and pre-existing instance data will be replaced.
EDMMIMPI FILE=MYFILE, TIME=NEW,PREVIEW=NO, Y2K=NO	Imports the instance data in the file `MYFILE` and all new object IDs. Time and date information will be created for the newly imported instance data. The old, non-Y2K format is retained or already converted.

EDMMIMPR

EDMMIMPR is a utility that is used to import resources into a Radia database. If a resource on the receiving Radia database has the same name as a file being imported, it will be replaced by that incoming file.

HP recommends that the ZEDMAMS verb, IMPORT_RESOURCE, be used instead of this utility.

You will need to specify certain parameters when executing EDMMIMPR. The literal parameters, such as FILE=, as well as their values, such as MYFILE, must be specified in uppercase.

Prior to importing resource data into a Configuration Server's RESOURCE file, the instance data that corresponds to the physical resource data must already exist in the context of the Radia database.

- For Intel Configuration Servers, be sure that the `edmprof` file (`edmprof.dat`) is in the same directory from which the program is executed. The database path is specified in the DBPATH parameter.
- For UNIX Configuration Servers, be sure that the `edmprof` file (`~/.edmprof`) has the DBPATH parameter set to the appropriate database path.
- For MVS Configuration Servers, the DBPATH parameter in the `edmprof` file (PARMLIB) will determine the database path.

Syntax

```
EDMMIMPR FILE=,(FROMDOMA=),(TODOMA=),(PREVIEW=YES/NO),
(REPLACE=YES/NO),(VERIFY=),(LOGFILE=)
```

Specifying the EDMMIMPR Utility

Table 120 describes the command parameters for the EDMMIMPR utility.

Table 120: EDMMIMPR Command Parameters

Parameter	What To Specify
file	Specify the file containing the input information, that is, the RESOURCE data that you want to create or update.

Parameter	What To Specify
fromdoma	Domain from which to import the data.
preview	Specify YES if you want a preview listing of the resources in the input file. Specify NO if you want to perform the import. The default is YES.
replace	Specify YES to replace the RESOURCE data, if it already exists. Specify NO if you do not want to modify the RESOURCE data, if it already exists. The default is NO.
todoma	Domain in which to import the data. Note: This must be specified if FROMDOMA= is specified.

Caution should be used with the REPLACE=YES option. If this option is chosen all existing instances in the receiving file will be deleted.

MVS User's Note

The following is a sample JCL for EDMMIMPR:

```
//*
//IMPR    EXEC PGM=EDMMIMPR,
//            PARM='PREVIEW:NO,FILE://DSN:YOUR.PDS(RRES)',
//            COND=(0,NE)
//STEPLIB  DD DSN=YOUR.EDM.LOAD,DISP=SHR
//         DD DISP=SHR,DSN=SASC.C650.LINKLIB
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//PREVIEW  DD SYSOUT=*
//LOG      DD SYSOUT=*
//SYSTERM  DD SYSOUT=*
//STGRPT   DD SYSOUT=*
//PARMLIB  DD DSN=YOUR.EDM.PARMLIB(STARTUP),DISP=SHR
//PRIMARY  DD DISP=SHR,DSN=YOUR.EDM.PRIMARY,
//   AMP=('STRNO=10,BUFNI=30,BUFND=100')
//RESOURCE DD DISP=SHR,DSN=YOUR.EDM.RESOURCE,
//   AMP=('STRNO=10,BUFNI=30,BUFND=30')
//
```

Examples

Table 121 shows some of the ways to specify the EDMMIMPR command options.

Table 121: EDMMIMPR Command Options

Command	Explanation
`EDMMIMPR PREVIEW=NO, FILE=myfile.bin`	Imports a file into the Configuration Server. For any of the parameters not specified, the default values will be used.
`EDMMIMPR PREVIEW=YES, FILE=myfile.bin`	Previews the expected results sent to the `edmmimpr.log` file, but does not perform the import.
`EDMMIMPR PREVIEW=YES, FILE=myfile.bin, REPLACE=YES`	Previews the expected results sent to the `edmmimpr.log` file, but does not perform the import, though the preview will show what resources will be replaced.

7 EDM Access Method Services (EDMAMS)

At the end of this chapter, you will:

- Be able to use the EDMAMS verbs to manage the Radia database.

HP strongly recommends shutting down the Configuration Server in order to ensure that the Radia database contents are not changing during the execution of the EDMAMS utilities.

However, ZEDMAMS can be run as a method (except on MVS, see MVS Considerations on page 304), in which case the Configuration Server must be running.

Furthermore, HP strongly recommends backing up the Radia database prior to running any of the EDMAMS utilities that will update the Radia database.

All UNIX platforms with file systems that support file sizes greater than 2 GB.

The EDMAMS code sequentially accesses the XPR file. If the C-runtime, or kernel I/O routines, or file system do not support file sizes in excess of 2 GB, the sequential I/O will fail at the 2 GB limit.

If the file system supports file sizes greater than 2 GB, an AMS fix will be required to enable the large file size C-runtime interface.

Overview

EDMAMS is a core set of utilities that can be used to create, delete, copy, change, and list objects in the Radia database. The functionality of these utilities is invoked using one of the EDMAMS verbs, which are called by the module, ZEDMAMS. The ZEDMAMS module is located in the directory in which the Configuration Server was installed.

- On a Win32 system, the default is:

  `system_drive:\Novadigm\ConfigurationServer`

- On a UNIX system, the default is:

  `/opt/Novadigm/ConfigurationServer`

Terminology

Table 122 lists terms that will be encountered in this chapter.

Table 122: EDMAMS Verb Terminology

Term	Definition
Verb	The action to be performed on the database object. For example: `DELETE_INSTANCE`, `EXPORT_CLASS`, and `SYNC_CLASS`.
Keyword	The (required and optional) pre-defined database location or object designators, such as: `FILE`, `DOMAIN`, `FROMDOMA`, and `TOINST`. Also, a pre-defined instruction to be considered during the action. For example: `KEEPDATE`, `PREVIEW`, `BASEONLY`, and `HEADER`.

Term	Definition
Value	The user-specified database location or object upon which the verb will act, such as: SYSTEMX, USER, RAD*, and EXPC.DAT. Also, a user-specified consideration that accompanies the action. For example: • Retaining an object's creation date and time (**KEEPDATE=YES**), • Applying a comment to a copied object (**COMMENT=***copied_object*), and • Replacing all existing data in the target object (**REPLACE=YES**)
Unmated Instance	An instance that does not have a resource.
Orphaned Resource	A resource that does not have a parent instance.
Component Orphans	A component instance that does not have a parent (for example, a package instance).

Invoking the EDMAMS Verbs

For a list of the EDMAMS verbs that are available with the version of the Radia database running in an environment, on the command line, type:

```
ZEDMAMS VERB=
```

The ZEDMAMS module is subject to continuous development. Due to this on-going development, some of the verbs detailed in this chapter might not be applicable to the Radia database version that is running in your environment.

Subsequent printings of the Configuration Server Guide will document enhancements to the EDMAMS functionality.

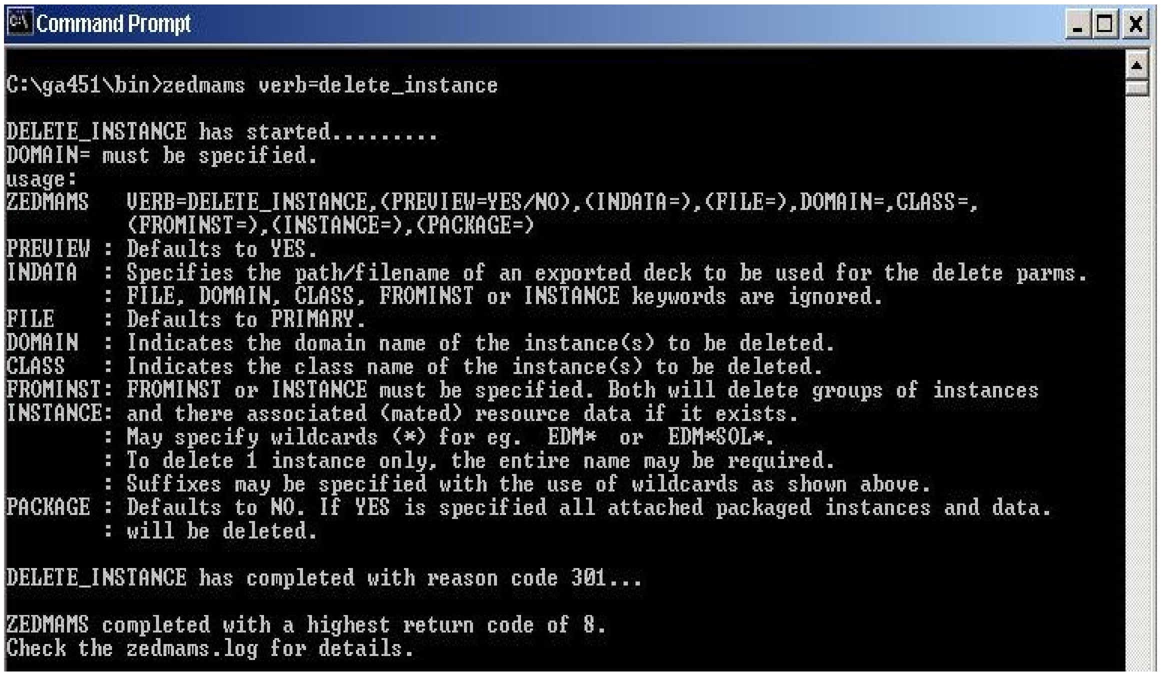
To display a verb's (required and optional) keywords, type:

```
ZEDMAMS VERB=VERB_NAME
```

For example, typing,

```
ZEDMAMS VERB=DELETE_INSTANCE
```

on the command line, yields the following:



```
C:\ga451\bin>zedmams verb=delete_instance

DELETE_INSTANCE has started.........
DOMAIN= must be specified.
usage:
ZEDMAMS   VERB=DELETE_INSTANCE,(PREVIEW=YES/NO),(INDATA=),(FILE=),DOMAIN=,CLASS=,
          (FROMINST=),(INSTANCE=),(PACKAGE=)
PREVIEW : Defaults to YES.
INDATA  : Specifies the path/filename of an exported deck to be used for the delete parms.
        : FILE, DOMAIN, CLASS, FROMINST or INSTANCE keywords are ignored.
FILE    : Defaults to PRIMARY.
DOMAIN  : Indicates the domain name of the instance(s) to be deleted.
CLASS   : Indicates the class name of the instance(s) to be deleted.
FROMINST: FROMINST or INSTANCE must be specified. Both will delete groups of instances
INSTANCE: and there associated (mated) resource data if it exists.
        : May specify wildcards (*) for eg.  EDM*  or  EDM*SOL*.
        : To delete 1 instance only, the entire name may be required.
        : Suffixes may be specified with the use of wildcards as shown above.
PACKAGE : Defaults to NO. If YES is specified all attached packaged instances and data.
        : will be deleted.

DELETE_INSTANCE has completed with reason code 301...

ZEDMAMS completed with a highest return code of 8.
Check the zedmams.log for details.
```

Figure 33: The keywords and conditions for the DELETE_INSTANCE verb.

Figure 33 shows the results of invoking the verb, DELETE_INSTANCE. The keywords are listed, as well as an explanation of each, with examples and applicable defaults. For more information on using the verbs, see the sections, Using the EDMAMS Verbs and Usage Considerations.

Using the EDMAMS Verbs

All of the EDMAMS verbs are specified in the following format:

ZEDMAMS VERB=*VERB_NAME*,*KEYWORD*=*VALUE*,*KEYWORD*=*VALUE*

For example,

ZEDMAMS VERB=DELETE_INSTANCE,CLASS=USER,INSTANCE=RADIA

There are no rules governing the order in which the keyword-value combinations are specified.

Usage Considerations

UNIX: Important Notes

On the command line, "ZEDMAMS" must be specified in uppercase. The verbs, keywords, and values, however, are not case sensitive.

When specifying values for Radia database locations and objects, such as, FILE=, CLASS=, FROMDOMA=, and TOINST=, the value must exactly match the location/object, as it is labeled in the Radia database. If it doesn't, the verb will fail. For example, to act on the instance, **JohnDoe**, in the USER class of the SYSTEMX domain, specifying:

```
DOMAIN=SYSTEMX,CLASS=USER,INSTANCE=johndoe
```

will result in a fail, because the value of INSTANCE=, as specified, does not exist in the Radia database. Rather, the following must be specified.

```
DOMAIN=SYSTEMX,CLASS=USER,INSTANCE=JohnDoe
```

- Optional keywords are enclosed in parentheses **( )**.
- Verbs, keywords, and data can be typed in UPPERCASE, lowercase, or a coMbinATioN.

  However, the value of string keywords, such as FROMDATA=, TODATA=, and STRING= are case-sensitive, where indicated.
- A comma (without a space) must follow each keyword-value combination, with the exception of the last keyword-value combination, which must be followed by a space.
- An asterisk (*) is not required for those values that recognize partial specification.
- For all of the EDMAMS verbs, the default value of FILE is **PRIMARY**.
- The EDMAMS verbs act on only one database object per execution, unless otherwise noted.

  For example, to change the names of the instances, **East_Sales** and **North_Sales** to **US_Sales**, the verb RENAME_INSTANCE would have to be run once for each of these instances.
- Output is written to `zedmams.log`, unless a different log is specified for LOGFILE=.

  Error conditions are written to **STDERR**.

MVS Considerations

- The ZEDMAMS utility cannot be run as a method.
- The ZEDMAMS utility cannot run while the Configuration Server is running.
- The ZEDMAMS utility will not work if used thru object transfer or a REXX script.
- All input and output files (including MPI and MPR files) must be pre-allocated.
- All parameters that are passed in, either by direct or indirect reference, are limited to 80 characters.

  Using more than 80 characters in the parameter requires the ZFILE option and its associated files.

- In order to function properly, input and output files must be in specified file formats. For example, resource files (XPR and MPR) must be specified in the fixed-block standard, **1** by *n* sequential-type files.

Input Files

Another way to run the EDMAMS utilities is to contain the commands in an input file—a file that has been edited by a text editor. This file can then be run to successively execute several EDMAMS verb functions.

- In an input file, the keywords can be specified on one or more lines for a single function.
- Like the command-line method, a comma must follow each keyword-value combination, with the last keyword-value combination followed by a space.

EXPORT Verbs

Input files are very useful when using the exporting EDMAMS verbs (EXPORT_CLASS, EXPORT_INSTANCE, and EXPORT_RESOURCE) because they allow multiple domain-class combinations to be exported during a single export function.

Multiple Verbs

More than one verb can be specified in an input file. For example, the COPY_DOMAIN verb can be followed by the verbs, DELETE_CLASS and LIST_CONS_VARS. Any combination of verbs can be included in one run, as long as the data sets being accessed do not conflict. If an asterisk (*) is placed in the first column, it is considered a comment and no action is taken.

To execute commands that are contained in an input file, enter the following on the command line:

```
ZEDMAMS ZFILE "DRIVE:\FILE_PATH\FILE_NAME"
```

ZEDMAMS is the executable. ZFILE is the second argument and must be in uppercase.

Example of multiple VERBS executed from one file:

```
VERB=COPY_DOMAIN,FROMDOMA=SYSTEMX,TODOMAIN=SYSTEMA,BASEONLY=
YES,REPLACE=NO
*
VERB=DELETE_CLASS,DOMAIN=SYSTEMA,CLASS=TESTCLAS
*
VERB=LIST_CONS_VARS,DOMAIN=SYSTEMX,CLASS=ZRSOURCE
,INSTANCE=CICS
```

Wildcards

EDMAMS supports two types of wildcards: *implicit* and *explicit*.

- **Implicit wildcards**
  are available for the COPY_INSTANCE, DELETE_INSTANCE, and LIST_INSTANCE verbs. Specify any portion of the value to select all occurrences that contain that portion of the value. Implicit wildcards do not require an asterisk, and are expressed as follows:

  KEYWORD=<wildcard_string>.

  For example, specify FROMINST=RAD to include all the fields that contain RAD as any part of the string.

- **Explicit wildcards**
  are available for the CHANGE_INS_FIELD, COPY_NEW_SUFFIX, COPY_RESOURCE, LIST_CONS_VARS, LIST_INST_DATA, LIST_ZRSC_FIELDS, and SEARCH_INSTANCES verbs. Explicit wildcards require an asterisk, and are expressed as follows:

  KEYWORD=<wildcard_string>.*

For example, specify FROMINST=RAD* to select all the instances that contain RAD as the first part of the string.

LOGFILE

This keyword can be used with all of the EDMAMS verbs. Specify the fully qualified path to, and name of, the log file to which the information is to be reported. The default log file name is **ZEDMAMS.LOG**.

If this keyword is not specified, or specified without a value, the following default locations are assumed:

- Windows: the `bin` folder in which the `zedmams.exe` utility is running.
- UNIX: the `home` directory of the user ID that installed the Configuration Server.

To send the information to a location\file other than the default, specify:

LOGFILE=*FILE_PATH\FILE_NAME*

HP recommends that if LOGFILE is specified as a path other than the default, that its existence be verified. If the directory does not exist, the log creation process will fail, and the command will not execute.

Specifying the ZEDMAMS Utility

Table 123 below lists and describes the verbs for the ZEDMAMS utility.

Table 123: ZEDMAMS Verbs

Verb	**Description**
ADD_FIELD	Adds a variable at the end of a template, including an automatic README file.
BUILD_PATCH	Builds a PATCH file in a Radia database, or a file from two Radia databases.
BUILD_STAGING_POINT	Creates a staging point, as needed, on the hard drive, and allows the resources (represented as an export deck) to be converted into a tree of files located in a stager-style directory.

Verb	Description
CHANGE_FIELDNAME	Changes variable names in class templates.
CHANGE_FLD_VALUE	Changes a template's variable length, type, Configuration Server, and client flags.
CHANGE_INST_DATA	Globally changes data in instance records, by class.
CHANGE_INS_FIELD	Changes one field in each instance of a class and verifies connects.
CHECK_RESOURCES	Verifies the size of resources against ZRSCSIZE and ZCMPSIZE.
CLONE_INSTANCE	Clones an instance with a four-digit suffix (max. 2000).
COPY_CLASS	Copies a class, its instances, and, if it exists, its resource data.
COPY_DATA	Copies only resource data from one `domain.class.instance` to another.
COPY_DOMAIN	Copies domains.
COPY_FIELD	Copies attribute data to a new attribute or to an existing attribute.
COPY_INSTANCE	Copies an instance or a range of instances and associated resource records.
COPY_NEW_SUFFIX	Copies and renames the suffix for ZRSOURCE class and mated instances.
COPY_RESOURCE	See COPY_INSTANCE.
CREATE_INSTANCES	Writes and populates instances from data in an edited text file.
DELETE_CLASS	Deletes a class and its instances.
DELETE_COMP_ORPHS	Deletes component orphans from the PACKAGE class.
DELETE_DOMAIN	Deletes one, or a range of, domains.
DELETE_FIELD	Deletes an attribute from a template.
DELETE_INSTANCE	Deletes/displays a range of instances within a class and, if it exists, its resource data.
DELETE_ORPHANS	Deletes unmated resource records (orphans) from the RESOURCE file.

Verb	Description
EDIT_CLASS_PREFIX	Updates the fields in the first 60 bytes of the prefix.
EXPORT_CLASS	Exports classes to an output file or data set for import to another file or data set.
EXPORT_INSTANCE	Exports instances to an output file or data set for import to another file or data set for reporting.
EXPORT_RESOURCE	Exports resource data to an output file or data set for import to another file or data set.
IMPORT_CLASS	Imports classes from an exported output file or data set to a PRIMARY file specified in the `edmprof` file.
IMPORT_INSTANCE	Imports instances from an exported output file or data set to a PRIMARY file specified in the `edmprof` file.
IMPORT_RESOURCE	Imports resource data from an exported output file or data set to a RESOURCE file specified in the `edmprof` file.
LIST_CLASSES	Displays a list of classes with the class names, object IDs, ZOBJDATE, ZOBJTIME, persistence flag, sequence-sensitive flag, Distributed Configuration Server flag, database type, and count totals. Note: If DOMAIN=* is specified, all classes in the domain will be listed.
LIST_CONS_VARS	Displays connect and variable field (types C and V) values from instance records and stores output in `zedmams.log`.
LIST_DOMAINS	Display a list of domains in the log of a specified file.
LIST_FLAGS	Lists Configuration Server and client flags in selected class record.
LIST_INST_DATA	Lists instance data including variable names to `zedmams.log`.
LIST_INSTANCE	Lists instance names by domain, class, and instance (prefix).
LIST_PACKAGE	List PACKAGE class instances and all mated components.
LIST_PREFIX	Lists the Distributed Configuration Servers prefix by file, domain, and class.

Verb	Description
LIST_RESOURCES	Lists resource prefix information (promote date, instance, and so forth).
LIST_ZRSC_FIELDS	Lists field values of fields beginning with ZRSC.
MATCH_RESOURCES	Matches resource records against ZRSOURCE class records.
PACKAGE_UNMATES	List all PACKAGE class instances without mated components.
REFRESH_DMA	Recounts the instances and classes in a file and updates the Distributed Configuration Server prefix.
RENAME_INSTANCE	Renames or displays instances by full name or prefix.
SEARCH_INSTANCES	Searches selected instances by domain and class for specified string
SORT_OBJECT_ID	Sorts object IDs in ascending or descending order by file or domain.
SYNC_CLASS	Synchronizes an existing class with a newly formatted class, and re-organizes all (existing class) instances according to the mapping in the newly formatted class. All existing class attributes that have a match in the new template will adopt the characteristics of the new template attribute, whereas any existing class attributes that do not have a match in the new template will be deleted.
UPDATE_INSTANCES	Updates the existing instance fields from data in an edited text file.
UPDATE_MGRIDS	Updates the Configuration Server ID and name by file, domain, and class.
VERIFY_CLASS	Verifies class templates for gaps, overlaps, and other anomalies.
VERIFY_DATABASE	Verifies the integrity of a Radia database.
ZRSOURCE_UNMATES	Matches ZRSOURCE instances with the resource.

The sections that follow describe each ZEDMAMS verb.

In the **Syntax** for each verb, the default values are presented in **bold**.

ADD_FIELD

This verb adds a variable (attribute) to the end of a class template and unconditionally includes a README attribute.

- The README attribute is 35 characters long.
- The LENGTH of the specified attribute cannot be greater than 255 characters, or less than one character.
- The Configuration Server and client flags (MFLAGS and CFLAGS) are optional and will default according to the attribute TYPE. The default flags are presented when requesting a usage display.
- The FLDNAME can be any one- to eight-character name and can be changed with the CHANGE_FIELDNAME function.

Syntax:	`(FILE=,)DOMAIN=,CLASS=,FLDNAME=,LENGTH=,TYPE=(,MFLAGS=)(,CFLAGS=)(,KEEPDATE=YES/`**`NO`**`)(,README=)(,DEFAULT=)`
Example:	Add an **M** type attribute, NEWFIELD, to the specified class, giving it a length of 165 characters, assigning default Configuration Server and client flags, and updating the object date and time: **`DOMAIN=SOFTWARE,CLASS=USER,FLDNAME=NEWFIELD,LENGTH=165,TYPE=M`**
Tip:	Run LIST_FLAGS against the class before and after running this to view the old and new template.

BUILD_PATCH

The functionality of this verb has been replaced with the **Service Optimization** feature of the System Explorer. For more information, refer to the *System Explorer Guide*.

BUILD_STAGING_POINT

This verb allows resources (represented as an export deck) to be converted into a tree of files located in a stager-style directory. It creates, on the hard drive, a staging point at a location designated by OUTFILE, then the data can be transferred to a CD using standard CD-writing software. This verb checks for the staging point and, if it does not exist, creates it.

- Specify PREVIEW=YES to see the expected results of running the verb with specific parameters.
- INFILE is the fully qualified path and filename of an exported resource (XPR) file.
- OUTFILE is the destination directory location of the staging files.

  OUTFILE defaults to the `location_of_edmprof\staging_point`. Therefore, if `edmprof` is located in `C:\Radia\bin`, the default staging point is `C:\Radia\bin\staging_point`.

  If PREVIEW=YES, OUTFILE is ignored.
- XPI is an exported instance deck that is used to timestamp the staged resources.

  This setting's value must be specified as the fully qualified path and filename of an exported instance deck.

  If XPI is not specified, or if the specified deck does not contain instance data for the staged resource, the timestamp will be determined using the date and time from the input deck resource header.
- REPLACE=YES replaces the existing same-named file at the specified staging point.
- MULTICAST=YES specifies that the OUTFILE will contain the name of a directory where resource data will be stored in File.Domain.Class.Instance format, and each file will contain an embedded 60-character prefix, identical to the prefix saved in the RESOURCE file.

Syntax:	`(PREVIEW=`**`YES`**`/NO,)INFILE=(,OUTFILE=)(,XPI=)(,REPLACE =`**`YES`**`/NO)(,MULTICAST=YES/`**`NO`**`)`

Example:	Accept the default staging point on the hard drive as the destination for files from `C:\NovaFiles\NewCD`: **`INFILE=C:\NovaFiles\NewCD`**
Tip:	To see what would be the result of running the verb (without actually doing so), specify PREVIEW=YES.

Resource Naming

BUILD_STAGING_POINT will remove the 60-byte prefix from each resource that is delivered to the staging point, and rename it according to the following format:

1. The first byte of the object ID is added to the **000** string, and used as a *branch root*.
2. Bytes 2 – 9 will be used as a *name*, and followed by a period.
3. The last three bytes will be used as an *extension*.

   For example, a resource with an object ID of

   ```
   DABC12345678
   ```

   will be renamed:

   ```
   000D\ABC12345.678.
   ```

In accordance with step 1, the first byte (**D**) was added to the string **000**. The back slash (\) was automatically inserted to make **000D** a branch root. A period was placed in front of the final three bytes (**678**), making them the extension, as described in step 3. The bytes in between the branch root and the extension (**ABC12345**) become the name.

CHANGE_FIELDNAME

This verb changes the specified class template field name (FROMNAME) to a new field name (TONAME).

- If there are multiple-occurrences of a field name (such as, README), all of them will be changed.

  However, README names should not be changed; only user-created variable field names should be changed.

Syntax:	`(FILE=,)DOMAIN=,CLASS=,FROMNAME=,TONAME=(,KEEPDATE= YES/`**`NO`**`)`
Example:	Change the template field name in the specified class from GROUPID to GROUP: **`DOMAIN=SOFTWARE,CLASS=USER,FROMNAME=GROUPID ,TONAME=GROUP`**
Tip:	Run LIST_INST_DATA to display template field names with INSTANCE=_BASE.

CHANGE_FLD_VALUE

This verb changes a variable's length, type, and Configuration Server and client flags by FLDNAME.

- The LENGTH must be in the range of 1 to 255 characters.
- The Configuration Server (Manager) and client flags (MFLAGS and CFLAGS) are optional, and will retain their current settings if omitted.
- The keyword DESC changes the DESCRIPTION field in the template for a maximum of 20 characters. The DESCRIPTION field can be changed either alone or in addition to other fields.

  If blanks are included in the text, it must be enclosed in quotation marks (" ").
- The keyword TYPE is required except when DESC is the only field being changed. The values are:
  - For connect types – the characters **C**, **A**, **I**, **R**, and **O**.
  - For method types – the characters **M**, **T**, **H**, and **D**.
  - For variable types – the characters **V**, **U**, and **W**.
- INDEX changes the nth occurrence of variable FLDNAME. When omitted, the first variable with a matching fieldname will be changed.
- KEEPDATE=YES will prevent the OBJDATE and OBJTIME from being updated.
- README is the 1- to 35-character description of the changed field that was placed in the _BASE_INSTANCE_. If blanks are included in the text, it must be enclosed in quotation marks (" ").
- DEFAULT the length of the default data must be less than or equal to the length specified by LENGTH.

  This value populates FLDNAME in the _BASE_INSTANCE_. If blanks are included in the text, it must be enclosed in quotation marks (" ").

Syntax:	`(FILE=,)DOMAIN=,CLASS=(,DESC=,)FLDNAME=(,LENGTH=,) TYPE=(,MFLAGS=)(,CFLAGS=)(,INDEX=)(,KEEPDATE=YES/`**`NO`**`) (,README=)(,DEFAULT=)`

Example:	In the template specified, change the TYPE value of the third attribute (named EDMSETUP) from V to C. (The length, and Configuration Server and client flags retain their current value.): **`DOMAIN=SOFTWARE,CLASS=ZSERVICE,FLDNAME=EDMSETUP,TYPE=C,INDEX=3`**
Tip:	Run LIST_FLAGS against the class before and after running this to view the old and new template.

CHANGE_INST_DATA

This verb changes instance data.

- A match occurs only when the value of FROMDATA begins the instance field and is not embedded, or is not part of other data in the field.
- If PREVIEW=YES, only the instances to be changed will be displayed.
- If FIELD is specified, this verb will change all instances whose FROMDATA criteria is equal to the data portion of the heap specified by FIELD.
- FROMDATA is the data, in the heap, that is to be replaced.

  All instances that meet the criteria of FIELD and FROMDATA will be changed with TODATA.

  If FROMDATA=BLANKS, the change criteria will be all blank characters.
- Omitting TODATA will set the field to blanks.
- FROMDATA and TODATA can be entered in any case. Keep in mind that the comparison made against FROMDATA is based on the case entered.
- If either FROMDATA or TODATA contain embedded spaces, the string must be enclosed in quotation marks.
- If KEEPDATE=NO, a new ZOBJDATE and ZOBJTIME are generated.

Syntax:	`(FILE=,)DOMAIN=,CLASS=(,FIELD=,)FROMDATA=(,TODATA=)` `(,PREVIEW=`**`YES`**`/NO)(,KEEPDATE=YES/`**`NO`**`)`
Example:	Change all instance data in the specified class from *JohnPublic* to *John Q. Doe*: **`DOMAIN=SOFTWARE,CLASS=USER,FROMDATA=JohnPublic,`** **`TODATA="John Q. Doe"`**
Tips:	Run LIST_INST_DATA to get a display of instance data and the class field names to which the data belongs. Run first with PREVIEW=YES to display the current variable data values.

CHANGE_INS_FIELD

This verb changes the instance data of the specified field in specific instances within the same domain.

- INDEX is the relative number (1– 99) of multiple same-named fields (such as, EDMSETUP or README).
- The keyword PREFIX:
  - Can be specified wildcards (*). For example, DIFF* and DIFF*SOL*.
  - If specified with just an asterisk (*), all instances in the specified class are changed.
  - To change only one instance, the entire name must be specified.
- The keyword FIELD is the name of the attribute to be changed.
- VERIFY=YES verifies that a connection value in TODATA exists. If the verify fails, the program aborts.
- KEEPDATE=YES will prevent the OBJDATE and OBJTIME from being updated.
- TODATA can be entered in any case; the data is placed in the field as entered. If TODATA contains embedded spaces, the string must be enclosed in quotation marks.

  Specify only CLASS.INSTANCE, not the domain.
- PREVIEW=YES displays instance names and the current data before the change and PREVIEW=NO displays instance names and the current data after the change.

Syntax:	`(FILE=,)DOMAIN=,CLASS=(,PREVIEW=`**`YES`**`/NO,)PREFIX=,FIELD=,TODATA=(,INDEX=)(,VERIFY=YES/`**`NO`**`)(,KEEPDATE=YES/`**`NO`**`)`
Example:	Change the third EDMSETUP field (INDEX=3) in the specified instances to ZLOCMGR.RAD_RESOURCE_FILE. Verify the existence of the connection, and update the OBJDATE and OBJTIME: **`DOMAIN=SOFTWARE,CLASS=USER,PREFIX=TSO,FIELD=EDMSETUP,TODATA=ZLOCMGR.RAD_RESOURCE_FILE,VERIFY=YES,INDEX=3`**

Tip:	This verb allows wildcards for the PREFIX parameter. Therefore, you can specify PREFIX=RAD* to view all the prefixes that contain RAD as the first part of the string.

CHECK_RESOURCES

This verb verifies the actual size of the resource data against ZRSCSIZE or ZCMPSIZE.

The entire PRIMARY file is checked for the presence of the variable field names ZRSCSIZE and ZCMPSIZE. If these fields exist, and if ZCMPSIZE contains a value, it is compared against the actual size. If ZCMPSIZE is zero or blank, the value in ZRSCSIZE is used. If there is a mismatch, the name of the resource and the appropriate sizes are listed to the log and a return code of 8 is passed.

- If LISTALL=YES, all objects checked are listed, and objects with resources are listed with their respective sizes.
- If UPDATE=YES, and if the appropriate of either ZRSCSIZE or ZCMPSIZE does not contain the correct value, the field will be updated with the correct value. In addition, the 8-byte, printable hex field in the instance prefix will be updated if in error.
- **CRC** is a toggle to activate/turn-off the Cyclical Redundancy Check.
- If CRC=YES, a CRC is calculated for each resource and if UPDATE=YES, incorrect CRCs are updated to their correct values in the OBJRCRC field.
- DOMAIN is a 1- to 8-character (domain) name that is to have its resources checked.
- CLASS is a 1- to 8-character (class) name that is to have its resources checked.

  If specified, DOMAIN must be specified.
- INSTANCE is a 1- to 32-character (instance) name that is to have its resources checked.

  If specified, DOMAIN and CLASS must be specified.

Syntax:	`LISTALL=YES/`**`NO`**`(,UPDATE=YES/`**`NO`**`)(,CRC=YES/`**`NO`**`,)DOMAIN= ,CLASS=,INSTANCE=`
Example:	List all the resources that have a mismatch: **`CHECK_RESOURCES,LISTALL=YES`**
Tip:	N/A

CLONE_INSTANCE

This verb clones the instance a specified number (nnnn) of times and each new instance name is suffixed with one to four digits: 0000 through nnnn-1.

The cloned instance name, plus its suffix, cannot exceed 32 characters: 1 to 28 digits for the instance name + 1 to 4 digits for the suffix. Therefore, instance name length = 32, the length of the suffix.

- A new OBJID, OBJDATE, and OBJTIME are generated for the cloned objects.
- COUNT is the number of clones to spawn, and can range from 1 to 2000.

Syntax:	`DOMAIN=,CLASS=,INSTANCE=,COUNT=nnnn`
Example:	Clone 100 instances in the specified class, naming the cloned instances DIFF80 through DIFF899 type: **`DOMAIN=SOFTWARE,CLASS=USER,INSTANCE=DIFF8,COUNT =100`**
Tip:	Run LIST_INSTANCE to display the instance names before and after the cloning.

COPY_CLASS

This verb copies a class template, its component instances, and resource data from one domain to another.

This verb is applicable to a Radia environment only; it will not function in an EDM environment.

Although this verb is supported, HP recommends using the verbs EXPORT_CLASS and IMPORT_CLASS to copy a class from the Radia database.

As of version 4.4 of the Radia database, this verb will copy the component instances and resource data also.

- TODB

  Non-MVS users: specifies the path to a destination file other than the one in the `edmprof` file. If omitted, it defaults to the DBPATH specified in the `edmprof` file.

  MVS users: TODB should specify a DDNAME (other than PRIMARY) that points to an appropriate VSAM configuration file, and must include a Resource destination DDNAME named RESOOUT. If omitted, it defaults to the DD Name (PRIMARY) in your JCL.
- If TOCLASS is omitted, the destination class will be assumed to be the same as the FROMCLAS. In this case, TODOMAIN and FROMDOMA must be different.
- BASEONLY=YES copies the class template and base instance only.
- A new object ID, ODJDATE, and OBJTIME are generated for the copied objects.
- If TODOMAIN does not exist, the ZBASE class and base instance will be copied from the source domain (FROMDOMA) in order to create a valid domain.
- If REPLACE=YES, all existing data is replaced.

Syntax:	`(TODB=)(,FILE=,)FROMDOMA=,FROMCLAS=,TODOMAIN=(,TOCLASS=)(,REPLACE=YES/`**`NO`**`)(,BASEONLY=YES/`**`NO`**`)`

Example:	Copy the specified class template and only the base instance from SOFTWARE to SYSTEM: **`FROMDOMA=SOFTWARE,FROMCLAS=USER,TODOMAIN=SYSTEM,BASEONLY=YES`**
Tip:	Run once with REPLACE=NO to determine the preexistence of the class in the destination domain.

COPY_DATA

This verb is applicable to an EDM environment only; it will not function in a Radia environment.

This verb copies only resource data, from one domain.class.instance to another. The destination instance must exist, and will inherit the CRC and printable hex size of the incoming data.

- TODB

  Non-MVS users: specifies the path to a destination file other than the one in the `edmprof` file. If omitted, it defaults to the DBPATH specified in the `edmprof` file.

  MVS users: TODB should specify a DDNAME (other than PRIMARY) that points to an appropriate VSAM configuration file, and must include a Resource destination DDNAME named RESOOUT. If omitted, it defaults to the DD Name (PRIMARY) in your JCL.

- FROMINST (mandatory) can copy a single resource, or group of resources. For groups, wildcards (*) are accepted. For example, DIFF* or DIFF*SOL*.

  To copy a single resource, the entire name is required.

  FROMINST cannot accept a wildcard if TOINST is specified.

- TOINST specifies an existing receiving instance full name in the TOCLASS.

  TOINST is not permitted when a FROMINST wildcard is specified.

- If REPLACE=YES all existing data is replaced.

Syntax:	`(PREVIEW=`**`YES`**`/NO)(,TODB=,)FROMDOMA=,FROMCLAS=,FROMINST=,TODOMAIN=,TOCLASS=(,TOINST=)(,REPLACE=YES/`**`NO`**`)`
Example:	Copy resource data from one domain.class.instance to another domain.class.instance: **`FROMDOMA=SOFTWARE,TODOMAIN=NEWDOM,FROMCLAS=ZSERVICE,TOCLASS=NEWCLASS,FROMINST=QA101_ENORMAL_EXT,TOINST=NEW_QA101_ENORMAL_EXT,PREVIEW=NO`**
Tip:	N/A

COPY_DOMAIN

This verb copies only the class template and _BASE_INSTANCE_ of a domain within a database, and optionally, to a different destination database. To copy the entire contents of a domain, see the section, Copying a Domain and its Contents.

This verb is applicable to a Radia environment only; it will not function in an EDM environment.

- A new object ID, OBJDATE, and OBJTIME are generated for the copied objects.
- If REPLACE=YES, all existing data is replaced.
- TODB

  Non-MVS users: TODB specifies the path to a destination file other than the one in the `edmprof` file. If omitted, it defaults to the DBPATH specified in the `edmprof` file.

  MVS users: TODB should specify a DDNAME (other than PRIMARY) that points to an appropriate VSAM configuration file, and must include a Resource destination DDNAME named RESOOUT. If omitted, it defaults to the DD Name (PRIMARY) in your JCL.

Syntax:	`(FILE=,)FROMDOMA=,TODOMAIN=(,REPLACE=YES/`**`NO`**`)(,TODB=)`
Example:	Copy the class template and _BASE_INSTANCE_ of the SOFTWARE domain to the SYSTEM domain: **`FROMDOMA=SOFTWARE,TODOMAIN=SYSTEM`**
Tip:	Run once with REPLACE=NO to determine the preexistence of the destination domain.

Copying a Domain and its Contents

To copy a domain, run this verb with FROMDOMA and TODOMAIN specified, then use the export and import verbs as detailed below.

To copy a domain with contents

1 Run this verb with FROMDOMA and TODOMAIN specified.

This will copy just the _BASE_INSTANCE_ and class template.

2 Verify that the new domain has been created by checking the ZEDMAMS log's return code or physically querying the Radia database using an explorer tool.

3 Use the EDMAMS verbs, EXPORT_INSTANCE (detailed on page 345) and EXPORT_RESOURCE (detailed on page 347) with **DOMAIN=<**`old_domain`**>** (the value of FROMDOMA).

4 Import the domain using the EDMAMS verb, IMPORT_INSTANCE (detailed starting on page 349). Be sure to specify:

— the exported instance (**FILE=**),

— the resource files (**XPR=**),

— **MAP_DOMAIN=**`old_domain/new_domain`.

COPY_FIELD

This verb copies an attribute (FROMFLD) and its instance data to a new attribute (TOFLD). The length, type, and flags of the new attribute will be inherited from the existing attribute.

- INDEX is the nth occurrence of a FROMFLD multiple-named attribute.
- PREVIEW will display the value of the FROMFLD attribute before changing an existing TOFLD attribute.
- REPLACE=YES will overlay existing data in the TOFLD attribute with the values in the FROMFLD attribute.
- If KEEPDATE=NO, a new zobjdate and zobjtime are generated.

Syntax:	`(FILE=,)DOMAIN=,CLASS=,FROMFLD=,TOFLD=(,INDEX=)(,PREVIEW=`**`YES`**`/NO)(,REPLACE=YES/`**`NO`**`)(,KEEPDATE=YES/`**`NO`**`)`
Example:	Copy an attribute named OLDFLD (and the associated data in the specified class) to a new attribute named NEWFLD: **`DOMAIN=SOFTWARE,CLASS=USER,FROMFLD=OLDFLD,TOFLD=NEWFLD`**
Tip:	Run LIST_INST_DATA against the class to view the contents of each attribute.

COPY_INSTANCE (COPY_RESOURCE)

These verbs copy a range of specified instances and resource data from one domain-class pair to another, in the PRIMARY and RESOURCE files, and optionally, to a different destination database. The function assumes that the destination class name is the same as the source, and that the templates are identical.

This verb is applicable to a Radia environment only; it will not function in an EDM environment.

Although this verb is supported, HP recommends using the verbs EXPORT_INSTANCE and IMPORT_INSTANCE to copy an instance from the Radia database.

This verb can be specified as either COPY_INSTANCE or COPY_RESOURCE, as these verbs have been combined and perform identical functions.

As of version 4.4 of the Radia database, this verb will copy the component instances and resource data also.

- TODB

  Non-MVS users: specifies the path to a destination file other than the one in the `edmprof` file. If omitted, it defaults to the DBPATH specified in the `edmprof` file.

  MVS users: TODB should specify a DDNAME (other than PRIMARY) that points to an appropriate VSAM configuration file, and must include a Resource destination DDNAME named RESOOUT. If omitted, it defaults to the DD Name (PRIMARY) in your JCL.

- The value of FROMINST can be partially specified. For example, you can specify FROMINST=DIFF to specify all the instances that contain DIFF as any part of the string.
- A new object ID is generated for the copied objects. A new OBJDATE and OBJTIME will be generated unless KEEPDATE=YES.
- PREVIEW=YES will display a list of instance names that will be copied from the source class. PREVIEW=NO will display instances that have been copied.
- If the ZRSOURCE class does not exist in the destination domain, the ZRSOURCE class and its BASE_INSTANCE will be copied from the source domain.

- If REPLACE=NO and an existing instance is found, the function will abort, indicating the existing instance name to STDERR and the log, and listing in the log any instances that have been copied.
- The existing instance name will be written to STDERR, and instances that have been copied will be listed in the log.
- If PREVIEW=NO and the destination domain does not contain FROMCLAS, the source class (FROMCLAS) and its BASE_INSTANCE will be copied to the destination domain (TODOMAIN).
- NEWINST renames an instance (if TODB is specified).

  If wildcards are used for FROMINST, NEWINST is not allowed.

Syntax:	`(PREVIEW=`**`YES`**`/NO)(,TODB=)(,FILE=,)FROMDOMA=,FROMCLAS=,TODOMAIN=,FROMINST=(,NEWINST=)(,REPLACE=YES/`**`NO`**`)(,KEEPDATE=YES/`**`NO`**`)`
Example:	Copy instances and resource data from the SOFTWARE domain to the SYSTEM domain, with the from-instance wildcard specified as CICS*ICO*: **`FROMDOMA=SOFTWARE,FROMCLAS=ZSERVICE,TODOMAIN=SYSTEM,FROMINST=CICS*ICO*`**
Tips:	Run once with PREVIEW=YES to get a list of instances to be copied. Also, run LIST_RESOURCES against the source domain to display a list of existing resources and instance names that can be copied, and then against the destination domain to see what was copied. Destination domain instance names that match the new instance name will be deleted.

COPY_NEW_SUFFIX

This verb copies the specified ZRSOURCE instances and resource data from one domain to another in the PRIMARY and RESOURCE files; and adds a new suffix to the destination instance. This verb allows wildcards for FROMINST.

- The value of FROMINST can be partially specified. For example, you can specify FROMINST=DIFF to select all the instances that contain DIFF as the first part of the string.
- If the length of NEWSUFF is longer than that of OLDSUFF, the new instance name must not exceed 32 characters. If it does, a message will be placed in the log and the instance will not be copied.
- PREVIEW=YES will display the old instance name and the new instance name and the total number of instances to be copied.
- If the class does not exist in the TODOMAIN, the class and BASE_INSTANCE will be copied from the source domain (FROMDOMA).
- Instance names must match the prefix (FROMINST) and the suffix (OLDSUFF) in order to be copied and renamed.

Syntax:	`(PREVIEW=`**`YES`**`/NO,)FROMDOMA=,FROMCLASS=,FROMINST=,TODOMAIN=,OLDSUFF=,NEWSUFF=`
Example:	From the SOFTWARE domain, copy and re-suffix instances and resource data prefixed with TSO and suffixed with ICO, to the SYSTEM domain with the new suffix, TSO_(*)_NEW_ICO; and displaying only those that would be copied: **`FROMDOMA=SOFTWARE,FROMCLAS=ZSERVICE,TODOMAIN=SYSTEM,FROMINST=TSO,OLDSUFF=ICO,NEWSUFF=NEW_ICO`**
Tips:	Run once with PREVIEW=YES to verify that the new instance names will not exceed 32 characters, and that the instances required will be copied. Also, run LIST_RESOURCES against the source domain to display a list of existing resources and instance names that can be copied and then against the destination domain to see what was copied. Destination domain instances that match the new instance name will be deleted.

COPY_ZRSOURCE

This verb copies the entire ZRSOURCE class (including instances and resource data) from one domain to another in the PRIMARY and RESOURCE files, and adds a new suffix to the destination instance.

This verb was deleted from the Radia database as of version 4.3. Its functionality is handled by the verb, COPY_CLASS (on page 323).

- If the domain specified in TODOMAIN does not exist, the ZBASE class and _BASE_INSTANCE_ will be copied from the source domain in order to create a valid domain.
- FROMDOMA and TODOMAIN must be different domain names.
- If BASEONLY=YES, only the class template and _BASE_INSTANCE_ will be copied.

Syntax:	`FROMDOMA=,TODOMAIN=(,BASEONLY=YES/`**`NO`**`)`
Example:	Copy the ZRSOURCE class from the SYSTEMX domain to the SYSTEMA domain: **`FROMDOMA=SYSTEMX,TODOMAIN=SYSTEMA`**
Tip:	To view what will be copied, run LIST_RESOURCES before the copying. To view what was copied, run LIST_RESOURCES after the copying.

CREATE_INSTANCES

This verb creates instances in the specified domain from data in an edited text file (INFILE). The INFILE must conform to the following specifications.

- The first line must contain a new instance name starting in column 1 for a maximum of 32 characters.
- The next line contains the variable field name, beginning in column 2, for a maximum of eight characters; then a blank in column 10; followed by a two-byte index value (01-99) in columns 11 and 12; followed by the data to be populated beginning in column 13.
- Existing instance names will be bypassed and the function will stop unless a **/*** (in columns 1 and 2) line follows the last data line of an instance. In the example below, the function will continue to the next input instance.

Column Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Line 1	U	S	E	R	_	N	A	M	E															
Line 2		N	A	M	E						J	.	A	.	D	E	V	E	L	O	P	E	R	
Line 3		E	D	M	S	E	T	U	P		D	O	M	A	I	N	.	C	L	A	S	S		
Line 4		Z	P	R	I	O	R	I	T		9	9	9											
Line 5	U	S	E	R	_	N	A	M	E		L	.	Z	I	M	M	E	R						
Line 6		N	A	M	E																			
Line 7		E	D	M	S	E	T	U	P		U	S	E	R	.	U	S	E	R	3				
Line 8		Z	T	R	A	C	E				N	N	N											

Syntax:	`INFILE=,DOMAIN=,CLASS=`
Example:	Create the instances in the specified text file in CLASS=USER: **`INFILE=MYINPUT.TXT,DOMAIN=SYSTEMX,CLASS=USER`**
Tip:	N/A

DELETE_CLASS

This EDMAMS verb deletes a class template and all of its instances.

As of version 4.4 of the Radia database, this verb will delete the component instances and resource data also.

- If PACKAGE=YES, all component and component data is deleted.

Syntax:	`(FILE=)(,PREVIEW=`**`YES`**`/NO)(,PACKAGE=YES/`**`NO`**`,)DOMAIN= ,CLASS=`
Example:	Delete the class, USERTEST, from the SOFTWARE domain: **`DOMAIN=SOFTWARE,CLASS=USERTEST`**
Tip:	N/A

DELETE_COMP_ORPHS

This verb deletes component orphans from the PACKAGE class. Orphans are defined as RESOURCE file data that have no mated instances in the associated PRIMARY file PACKAGE class.

- CLASS defaults to all classes of the specified domain.

Syntax:	`(FILE=)(,PREVIEW=`**`YES`**`/NO,)DOMAIN=(,CLASS=)`
Example:	Delete orphans from the TSTRESLT Class of the SOFTWARE domain: **`DOMAIN=SOFTWARE,CLASS=TSTRESLT,PREVIEW=NO`**
Tip:	N/A

DELETE_DOMAIN

This verb deletes a domain or an alphabetical range of domains (class templates and instances) from the file specified.

As of version 4.4 of the Radia database, this verb will delete the component instances and resource data also.

- If TODOMAIN is omitted, only FROMDOMA will be deleted.

  If TODOMAIN is specified, the range is inclusive.

Syntax:	`(FILE=)(,PREVIEW=`**`YES`**`/NO,)FROMDOMA=(,TODOMAIN=)`

Example:	From the PRIMARY file, delete only SOFTWARE:
	`FILE=PRIMARY,FROMDOMA=SOFTWARE`
	From the PROFILE file, delete USERA through USERF:
	`FILE=PROFILE,FROMDOMA=USERA,TODOMAIN=USERF`

Tip:	Run with PREVIEW=YES first.

Use caution when specifying a range of domains. Be certain of the range specified, because there is no confirmation request.

DELETE_FIELD

This verb deletes an attribute from a template along with its README field, and reorganizes the class accordingly.

- FIELD must refer to an attribute, not a README (description) field.
- INDEX refers to multiple occurrences of the same attribute name. For example, if there were three occurrences of EDMSETUP, the third would be INDEX=3.

  Values for INDEX are the numerals 1 through 99.

Syntax:	`DOMAIN=,CLASS=,FIELD=(,INDEX=)`
Example:	Delete the attribute USERATTR from the USER class: **`DOMAIN=POLICY,CLASS=USER,FIELD=USERATTR`**
Tip:	Use with discretion because deletion of an attribute causes a restructuring and rewrite of all the instances in the class.

DELETE_INSTANCE

This verb deletes an instance or a range of alphabetical instances within a specific class.

The keywords, TOINST and SUFFIX, were deleted from this verb's functionality with version 4.3 of the Radia database.

As of version 4.4 of the Radia database, this verb will delete the component instances and resource data also.

- FROMINST and INSTANCE perform the same function; either one must be used. Both will delete groups of instances and any existing, associated resource data. To delete only one instance, the entire name must be specified.

  FROMINST and INSTANCE can specify wildcards (*). For example, DIFF* and DIFF*SOL*.

- PREVIEW=YES displays a list of instances that would be deleted with PREVIEW=NO.

- INDATA specifies the fully qualified path to, and name of, a file that contains the delete parameters. This file can be either an exported instance deck (.XPI), or a manually created file that contains the FILE, DOMAIN, CLASS, and INSTANCE values.

  If using INDATA, the values that are specified for FILE, DOMAIN, CLASS, FROMINST, and INSTANCE on the command line are ignored because these values are extracted from the INDATA file.

  When not using INDATA, either FROMINST or INSTANCE *must* be specified. Both can delete groups of instances.

  FROMINST (instead of INSTANCE) can be specified inside the INDATA file in order to define a group of instances to be deleted.

Syntax:	`(PREVIEW=`**`YES`**`/NO)(,INDATA=)(,FILE=,)DOMAIN=,CLASS=(,FROMINST=)(,INSTANCE=)`
Example:	Delete all USER class instances, beginning with the instance prefix name of **`OLD_USER:DOMAIN=POLICY,CLASS=USER,FROMINST=OLD_USER`**

Tip:	Run once with PREVIEW=YES to view which instances will be deleted. Also, run LIST_INSTANCES against the class to display a list of instance names that might be deleted.

Use caution when specifying a range of domains. Be certain of the range specified, because there is no confirmation request.

DELETE_ORPHANS

This verb will delete all orphans in all domains. Orphans are defined as RESOURCE file data that have no mated instance in the associated PRIMARY file.

- TRACE=YES provides additional diagnostic (tracing) confirmation in the log.

Syntax:	`(PREVIEW=`**`YES`**`/NO)(,TRACE=YES/`**`NO`**`)`
Example:	Delete all orphans: **`PREVIEW=NO`**
Tip:	N/A

DELETE_RESOURCE

This verb deletes instances and resources from the PRIMARY and RESOURCE files.

The keyword, DELETE_RESOURCE, was deleted from the Configuration Server as of version 4.2. Its functionality is handled by the verb, DELETE_INSTANCE (on page 338).

- A partial name can be specified for FROMINST and TOINST.
- If TOINST is omitted, only the instances specified in FROMINST will be deleted.

  If TOINST is specified, the range is inclusive.
- If the instance exists in the PRIMARY file, it will be deleted—regardless of whether it has any resource data connected (mated) to it.

Syntax:	`(PREVIEW=`**`YES`**`/NO,)DOMAIN=,CLASS=,FROMINST=(,TOINST=)`
Example:	In the ZRSOURCE class of the SYSTEMX domain, delete all instances (and mated resource data) with the prefix RAD*: **`PREVIEW=NO,DOMAIN=SYSTEMX,CLASS=ZRSOURCE, FROMINST=RAD*`**
Tip:	N/A

EDIT_CLASS_PREFIX

This verb enables you to edit the first 60 bytes of the template's prefix.

- FIELD is the name of the class prefix field.
- VALUE will not be read if FIELD is not specified.

  Refer to the **Values** column in Table 124 below for the valid values for each FIELD type.
- If KEEPDATE=NO, a new OBJDATE and OBJTIME are generated.

Table 124: Changeable Fields

Field	Values
CLASTYPE	**P** (POLICY_CLASS_TYPE), **C** (CONFIGURATION_CLASS_TYPE),**T** (COMPONENT_CLASS_TYPE), **B** (CLASS_TYPE_BLANK)
CLASSPRI	**5** (PATH), **10** (METACLAS), **50** (FILE), **50** (ZSERVICE), **60** (DESKTOP), **70** (MACALIAS),**70** (REGISTRY), **50** (DEFAULT_PRIORITY), **B** (Blank)
DBTYPE	**E** (MVS VSAM), **U** (UNICODE), **A** (ASCII)
OBJDM	**D** (DISTRIBUTED_MANAGER)
OBJTYPE	**S** (SINGLE_DIMENSIONAL_OBJECT), **M** (MULTIPLE_DIMENSIONAL_OBJECT)
SEQ_SENS	**S** (SEQUENCE_SENSITIVE), **I** (SEQUENCE_INSENSITIVE) Note: Specifies whether to process variables in the order in which they occur in the class template (**S**), or in order of attribute type (**I**)—that is, VARs then CONNs, and so on.
OBJNAME	Any text up to 20 characters in length. If there are embedded spaces in the text, enclose the text with quotation marks (" "), as in the example.

Syntax:	`DOMAIN=,CLASS=,FIELD=,VALUE=(,PREVIEW=YES/NO)(,KEEP DATE=YES/`**`NO`**`)`
Example:	**`DOMAIN=PLOICY,CLASS=USER,FIELD=OBJNAME,VALUE="User names",PREVIEW=NO`**
Tip:	N/A

EXPORT_CLASS

This verb enables the exporting of class template data from a file or data set for importing to another file or data set. This verb has the same functionality as the utility, EDMMEXPC.

- If PREVIEW=YES, OUTPUT is ignored.
- OUTPUT is the name of the destination output file (with extension) where the exported data is to reside.
- INPUT references a pre-defined input file, which enables multiple FILE.DOMAIN.CLASS combinations to be specified.

  The input file must be specified in the following format:

  FILE=*file_name*,DOMAIN=*domain_name*,CLASS=*class_name*
- HEADER=YES produces an output header file.
- COMMENT=YES adds a comment to the optional output file header.

Syntax:	`FILE=(,DOMAIN=)(,CLASS=)(,PREVIEW=`**`YES`**`/NO,)OUTPUT=(,COMMENT=)(,INPUT=)(,HEADER=YES/`**`NO`**`)`
Example:	Export all the classes (templates) in the PRIMARY file to a file named `EXPC.DAT`: **`FILE=PRIMARY,PREVIEW=NO,OUTPUT=C:\EXPC.DAT`**
Tip:	N/A

EXPORT_INSTANCE

This verb enables the exporting of instances to an output file or data set for importing to another file or data set or for reporting purposes. This verb has the same functionality as the utility, EDMMEXPI.

- If PREVIEW=YES, OUTPUT is ignored.
- OUTPUT is the name of the destination output file (with extension) where the exported data is to reside.
- KEEP specifies a text file that contains a list of the instance attributes to be retained in the resulting output file. These names are case-sensitive.
- DROP specifies the instance attributes that are not to be retained in the output file.
- If ORDER=YES, the resulting file will be ordered by attribute name.
- COMMENT=YES adds a comment to the optional output file header.
- Specify REPORT=YES to export instances to third-party vendor software.
- CSVL=YES produces a Comma-Separated Variable listing for all attribute values. Currently, the output file you specify with CSVL=YES is created in a subdirectory of the current working directory.
- If BASE=YES, the values of the _BASE_INSTANCE_ will be inherited.
- INPUT references a pre-defined input file, which enables multiple FILE.DOMAIN.CLASS.INSTANCE combinations to be specified. The input file must be specified in the following format:

  FILE=*file_name*,DOMAIN=*domain_name*,CLASS=*class_name*, INSTANCE=*instance_name*
- HEADER=YES produces an output header file.
- If SKIP_ERRORS=YES and database or consistency errors are encountered in the PROFILE file, a bad-object event (return code=4) will be recorded in the log, but processing will continue.

  If SKIP_ERRORS=NO (the default) and errors are encountered, the exporting will stop.
- PHEX=YES outputs the data portion of variables in printable hex format.

Syntax:	`FILE=(,DOMAIN=)(,CLASS=)(,INSTANCE=)(,FROMINST=)(,TOINST=)(,FROMDATE=)(,PREVIEW=`**`YES`**`/NO)(,TODATE=)(,FROMTIME=)(,TOTIME=,)OUTPUT=(,KEEP=YES/`**`NO`**`)(,DROP=)(,ORDER=)(,COMMENT=)(,CSVL=YES/`**`NO`**`)(,PHEX=YES/`**`NO`**`)(,BASE=YES/`**`NO`**`)(,INPUT=)(,HEADER=YES/`**`NO`**`)(,REPORT=YES/`**`NO`**`)(,SKIP_ERRORS=YES/`**`NO`**`)`
Example:	Export all the instances in the PRIMARY file to a file named EXPI.DAT: **`FILE=PRIMARY,PREVIEW=NO,OUTPUT=C:\EXPI.DAT`**
Tip:	N/A

EXPORT_RESOURCE

This verb enables the exporting of resource data to an output file or data set for importing to another file or data set. This verb has the same functionality as the utility, EDMMEXPR.

- If PREVIEW=YES, OUTPUT is ignored.
- OUTPUT is the name of the destination output file (with extension) where the exported data is to reside.
- Specify COMMENT=YES to add a comment to the optional output file header.
- INPUT references a pre-defined input file, which enables multiple FILE.DOMAIN.CLASS.INSTANCE combinations to be specified.

  The input file must be specified in the following format:

  FILE=*file_name*,DOMAIN=*domain_name*,CLASS=*class_name*, INSTANCE=*instance_name*
- HEADER=YES produces an output header file.

Syntax:	`FILE=(,DOMAIN=)(,CLASS=)(,INSTANCE=)(,FROMINST=)(,TOINST=)(,PREVIEW=`**`YES`**`/NO)(,FROMDATE=)(,TODATE=)(,FROMTIME=)(,TOTIME=,)OUTPUT=(,COMMENT=YES/`**`NO`**`)(,INPUT=)(,HEADER=YES/`**`NO`**`)`
Example:	Export all resources in the RESOURCE file to a file named EXPR.DAT: **`FILE=PRIMARY,PREVIEW=NO,OUTPUT=C:\EXPR.DAT`**
Tip:	N/A

IMPORT_CLASS

This verb allows you to import template data from an exported data set or output file to a PRIMARY file specified in the `edmprof` file. This verb has the same functionality as the utility, EDMMIMPC

- FILE is the name of the file or data set that contains the import class data..
- TIME=NEW – generates a new OBJDATE, OBJTIME, and OBJID.

  TIME=OLD – retains the original OBJDATE, OBJTIME, and OBJID.

  TIME=MOD – generates a new OBJDATE and OBJTIME, but retains the original OBJID.
- TODOMA is the domain with which matching source domains are replaced.
- If the FROMDOMA domain exists in the destination database, specify TIME=NEW to avoid duplicate object IDs.

  If FROMDOMA is specified, TODOMA must also be specified.
- If REPLACE=NO, the class template will not be replaced.

Syntax:	`FILE=(,PREVIEW=`**`YES`**`/NO)(,TIME=`**`OLD`**`/NEW/MOD)(,REPLACE=YES/`**`NO`**`)(,FROMDOMA=)(,TODOMA=)`
Example:	Import all the classes in the file specified by FILE to a PRIMARY file specified in the `edmprof` file: **`FILE=input_file,PREVIEW=NO`**
Tip:	N/A

IMPORT_INSTANCE

The IMPORT_INSTANCE verb enables an administrator to import instance and resource data from an exported data set or output file (an *import deck*) to a location in the Radia database. The import deck will be imported to the Radia database PRIMARY File that is specified for the DBPATH setting in the MGR_DIRECTORIES section of the `edmprof` file, such as:

```
[MGR_DIRECTORIES]
DBPATH = C:\Novadigm\ConfigurationServer\DB
```

This verb is a replacement for the database utility, **EDMMIMPI**.

The entire process is compromised of the following phases.

Verify the Import Deck
This verification checks the internal integrity (such as, size, referential integrity, and validity of data) of the incoming deck.

Preview the Import Deck
This phase is a comprehensive *analysis* that scans the database and the entire import deck, and reports the results, detailing differences that are relevant to this import session. This analysis checks for duplicate object IDs (OIDs), determines if fixes are possible, and determines if a new deck is required. The entire import deck is analyzed before this phase completes.

For the import deck, an *all-or-nothing* rule applies. Therefore, if an error condition is realized for one instance of the import deck, the entire deck is invalid.

When OIDs in the deck are the same as OIDs in the database, this verb's behavior is dictated by the keywords REPLACE and CONTINUE. These keywords are discussed in detail on page 352.

Import the Instances and Resources
The instances and resources from the deck will be imported only if the integrity check (performed during the **Preview the Import Deck** phase) is free of errors.

The IMPORT_INSTANCE information is presented as follows:

- Verb History (starting on page 350) describes when the keywords were introduced or discontinued.
- Syntax (starting on page 350) details all the keywords that are associated with this verb. This includes their expected behavior, and their interactions with, and dependencies on, one another.

- Retired Syntax (starting on page 354) covers the keywords that have been retired from use in this version, and includes information about which new keywords have replaced them.
- Usage Considerations (starting on page 355) addresses some of the more noticeable and critical effects that might result from using this verb, as well as some of the **new features.**
- Examples (starting on page 357) presents a few examples of how to express the keywords.

Verb History

This verb was introduced with version 4.3 of the Configuration Server to manage its database.

- The keywords, VERIFY and LOGFILE, were added to this verb's functionality in version 4.4 of the Configuration Server.
- In version 4.5.2 of the Configuration Server:
  - The keywords, FROMDOMA, TODOMA, TIME, CHGCONS, FORCE, Y2K, and IMPORT_RESOURCE were removed from this verb's functionality—but continue to be supported. See the section, Retired Syntax, on page 354.
  - The keywords, XPR, DUPLICATES, FORCE, CONTINUE, NEW, AUTOFIX, MAP_DOMAIN, and COMMIT_CHANGES were added in order to provide enhanced behavior management in the event of duplicate object IDs, multiple domains, and import decks from older systems and databases.

Syntax

This section details the syntax (**keywords** and **values**) that is associated with this verb, including the most efficient and effective ways to use it.

FILE=(,PREVIEW=**YES**/NO)(,DUPLICATES=**STOP**/MANAGE)(,XPR=)(,NEW=)(,REPLACE=YES/**NO**)(,VERIFY=YES/**NO**)(,AUTOFIX=YES/**NO**)(,MAP_DOMAIN=)(,CONTINUE=YES/**NO**)(,COMMIT_CHANGES=**YES/**NO)(,LOGFILE=)

FILE is the only keyword that must be specified on the command line in order for this verb to execute.

The other keywords are optional (as denoted by their inclusion in parentheses); if they are not specified, their defaults will be assumed and they will effect the processing of this verb.

- PREVIEW creates a preview listing of the input file contents, the expected results, and its ability to be imported. The results are written to the log file. The default is **YES**.

  To better understand the functionality of this keyword, think of it as asking, "Preview *only*?"

  If PREVIEW=NO, the processing will run and the import deck data *can* be written to the database—depending on the other keywords and the results of the comparison.

  If PREVIEW=YES (the default), the only result is a log file being generated—no action is taken on the database.

  If PREVIEW=NO, the processing will run and, provided there are no errors, the import deck can be written to the database.

- FILE is the fully qualified path and filename of the file that contains the collection of instances for the import deck. This file is commonly suffixed with the extension **XPI**, as shown in the examples starting on page 357.

  If a fully qualified path is not specified (as shown below), the location of the import file is assumed to be that from which ZEDMAMS is running.

  ```
  ZEDMAMS VERB=IMPORT_INSTANCE,FILE=DB_001.XPI
  ,PREVIEW=YES
  ```

- DUPLICATES enables an administrator to indicate the action to be taken when duplicate OIDs of instances are encountered (in the import deck and the database). The default is **STOP**.

  DUPLICATES=STOP (the default) will result in this operation stopping. The return code **8** will be reported.

  DUPLICATES=MANAGE will result in a new deck being created in order to avoid the re-use of a previously allocated OID. If the instance is data-bearing, it can be corrected only if the resource is available, in which case, XPR must be specified.

  A new deck could be created for any of the following reasons: duplicate OIDs; domain change; OBJRCRC is NULL or empty.

- XPR is the fully qualified path and filename of the resource deck.
- REPLACE dictates the behavior of the process (as defined in the following conditions) when identical instances are discovered in the database and the deck. The default is **NO**.

  If REPLACE=YES, the data in the import deck can be written to the database.

  If REPLACE=NO (the default), the database and import deck will be queried for differences, and the following logic will apply.

  — If no differences are found, processing will continue with the next instance in the import deck.

  — If differences are found, the value of CONTINUE is checked.

    – If CONTINUE=NO, each instance with a difference will be ignored and processing will continue with the next instance in the import deck, but the process will not proceed to the next phase. A return code of **8** will be returned.

    – If CONTINUE=YES, each instance with a difference will be ignored and processing will continue with the next instance in the import deck.
- CONTINUE dictates the behavior of the process when matching records are discovered. The default is **NO**.

  If CONTINUE=YES:

  — For any class attribute found in the target class template, the import will continue as long as it doesn't result in the truncating of any significant (non-blank) data. In this case, the process will continue and an error will be reported.

  — For any class attribute found in the target class template, but the import cannot import the data without truncating significant (non-blank) data, the process will continue, an error will be reported, and the import will fail.

  — For any class attribute not found in the target class template, the field will be dropped—even if it contains significant (non-blank) instance data. (A warning or error message will be issued, indicating this occurrence.)

  Fields and data that are dropped will be documented in the log file.

  If CONTINUE=NO (the default) *and…*

— the class attribute is not defined in the target class template, the import will fail.

- VERIFY compares the date (ZOBJDATE) and time (ZOBJTIME) of incoming files with those of the database files, if specified as YES. The default is **NO**.

  If VERIFY=YES and…

  — the dates and times do not match (**rc=8**), a warning message is issued, and processing will continue with the next instance in the import deck.

  — the dates and times match and XPR was specified (with a valid value), the VERIFY_IMPORT verb will run in order to check the integrity of the decks.

  — the dates and times match, but XPR has not been specified (or its value is invalid), verification is not possible.

  The results are reported to ZEDMAMS.LOG (the default), unless a different log has been specified for LOGFILE.

  If the dates and times match, the ZEDMAMS.LOG will report a successful verify; if not, the ZEDMAMS.LOG will report a failed verify. These verifications are done on a per-instance basis and reported at the end of the process.

  If VERIFY=YES, PREVIEW=YES and REPLACE=YES are assumed—but nothing is imported to the database.

  If VERIFY=NO, no verification is done.

- NEW is the fully qualified path and filename prefix of the new decks that will be created.

  Optionally, just the filename prefix can be specified, in which case the new decks will be created in the current directory (by default, the `bin` directory).

  This keyword is applicable only if DUPLICATES=MANAGE.

  — If either the filename prefix or the fully qualified path have embedded blanks, the entire string must be enclosed in quotation marks (**" "**).

  — The decks will have the suffixes `.MPI` and `.MPR`.

— The defaults are the fully qualified paths of **XPI** (as specified by the keyword, FILE) and **XPR** (as specified by the keyword, XPR), respectively.

- AUTOFIX dictates whether to delete orphaned resources. If importing a data-bearing instance and the resource file exists in the database, the existing resource will be deleted and the incoming resource will be written to the database. The default is **NO**.

  This keyword should be used with extreme caution and only by an experienced administrator in a controlled manner. Incorrect use could result in the accidental removal of database elements that are critical to performance and operation.

  If AUTOFIX=YES, orphaned resources will be deleted.

- MAP_DOMAIN enables an administrator to import all instances from one domain into a different domain, thereby facilitating application management.

  Use MAP_DOMAIN=*`source_domain`*/*`target_domain`* to import all instances that originated in one domain, *`source_domain`*, into a domain with a different name, *`target_domain`*. Doing so triggers the creation of a new deck. All object IDs from a domain that matches *`source_domain`* are placed in the new deck. In the new deck, their domain value is replaced by that of *`target_domain`*.

  For example, import all instances of the SOFTWARE domain to the domain, SOFTBACK, by specifying,

  MAP_DOMAIN=SOFTWARE/SOFTBACK.

  The new domain must exist in the database. If it doesn't, the process will stop.

- COMMIT_CHANGES dictates whether to commit to the database, the data in the import deck. The default is **YES**.

  COMMIT_CHANGES=**YES** (the default) will write the changes to the database.

  If COMMIT_CHANGES=NO, the changes will not be written to the database, but new **.MPI** and **.MPR** decks will be produced (if required).

  If there are no changes, this keyword is ignored.

- LOGFILE (see the section, LOGFILE, on page 306).

As the EDMAMS verbs have evolved, changes have been made in order to enhance their processing. Because of this, and in order to maintain logic for the user, there have been changes to the syntax of some of the verbs.

This section details the keywords that have been retired from use for the IMPORT_INSTANCE verb.

Although retired, these keywords are still supported.

These are superseded by new keywords where indicated.

- TIME=OLD (the default) retains the original OBJDATE, OBJTIME, and OBJID. (This is superseded by DUPLICATES=STOP.)

  TIME=NEW generates a new OBJDATE, OBJTIME, and OBJID. (This is superseded by DUPLICATES=MANAGE.)

  TIME=MOD generates a new OBJDATE and OBJTIME, but retains the original OBJID.
- FROMDOMA=<source_domain> and TODOMA=*target_domain* have been replaced by MAP_DOMAIN.
- FORCE=YES/NO. (This is superseded by CONTINUE.)
- CHGCONS specifies whether any embedded references to the name specified by FROMDOMA should be changed to the name specified by TODOMA.
- IMPORT_RESOURCE dictates whether this operation should import resources.

Usage Considerations

This section addresses some of the more noticeable and critical effects that might result from using this verb, as well as some of the new features.

- Once the data from an import deck has been written to the Radia database, the changes are considered permanent. Therefore, it is imperative that a Radia database administrator using this verb be certain of the changes that are being considered.

HP Recommendations

- Shut down the Configuration Server to ensure that the Radia database contents are not changing during the processing.
- Back up the Radia database prior to running this verb.
- Specify PREVIEW=YES and check the resulting log before committing any changes to the database.

- Executing this verb might result in the creation of new decks (**MPI/MPR**). The circumstances under which this might happen are:
  - There exist duplicate object IDs (ZOBJID) in the database instances and the import deck instances, and DUPLICATES=MANAGE.
  - There is a domain name change for the imported data (using the MAP_DOMAIN keyword).
  - The ZOBJRCRC (the object resource CRC) is NULL or empty and the value can be calculated and assigned in the process.

  All of these occur in conjunction with the existence of the XPR deck, it being specified on the command line, and the values of the CONTINUE and REPLACE allowing the processing to continue.

  The XPR deck is necessary so that if changes are required, it is available to be updated at that time.

- This version of IMPORT_INSTANCE has more consistency checks that must be passed before the import deck is considered valid for import.
- Combining instances and resources on the same command line allows the import deck to be more thoroughly examined.
- Use the CONTINUE option to prevent data loss during import. (See the description for the keyword CONTINUE on page 352.)
- In this version, an administrator:
  - Can manage the processing behavior if an import deck OID collides with a database OID.
  - Can specify the filename prefix if a new deck is generated.
  - Has a single keyword to facilitate changing domains.
- A new time-based format of object ID generation has been introduced, thereby eliminating the chance of randomly generating a duplicate OID.

For more information on this feature, consult the MGR_STARTUP section on page 102.

Examples

The following examples offer a look at the ways this verb can be used.

Even though some keywords are dependent on another, the order in which they are specified on the command line is not significant.

If a keyword is not specified on the command line, but has a default, the default will be assumed.

Some keywords, although optional, become mandatory based on the specifications of others and the results of processing. For an example, see DUPLICATES=MANAGE.

Example 1

Import the instance data that is contained in `DB_001.XPI` and `DB_001.XPR` to the PRIMARY file that is specified in the `edmprof` file. Do not write the changes to the database. Do not manage duplicate object IDs. Write the results to `C:\Temp\EDMAMS\DB001\Test01.log`.

```
ZEDMAMS VERB=IMPORT_INSTANCE,FILE=DB_001.XPI
,XPR=DB_001.XPR,PREVIEW=YES,LOGFILE=C:\Temp\EDMAMS
\DB001\Test01.log
```

In this run, the implied values (defaults) that affected the processing are: DUPLICATES=STOP and CONTINUE=NO.

Example 2

Import the instance data that is contained in `DB_001.XPI` and `DB_001.XPR` to the PRIMARY file that is specified in the `edmprof` file. Do not write the changes to the database. Manage any duplicate object IDs that are encountered. Query the database and deck for matching records and, if any are found, continue processing. Delete any orphaned resources that are encountered. Write the results to `C:\Temp\EDMAMS\DB001\Test02.log`.

```
ZEDMAMS VERB=IMPORT_INSTANCE,FILE=DB_001.XPI
,XPR=DB_001.XPR,PREVIEW=YES,DUPLICATES=MANAGE
,CONTINUE=YES,AUTOFIX=YES,LOGFILE=C:\Temp\EDMAMS
\DB001\Test02.log
```

In this run, the implied values (defaults) that affected the processing are: REPLACE=NO.

Example 3

Assume that the Example 2 command line has completed as specified. Execute the same run, but this time, write the changes to the database and the results to `C:\Temp\EDMAMS\DB001\Test02.log`.

```
ZEDMAMS VERB=IMPORT_INSTANCE,FILE=DB_001.XPI
,XPR=DB_001.XPR,PREVIEW=NO,DUPLICATES=MANAGE
,CONTINUE=YES,AUTOFIX=YES,LOGFILE=C:\Temp\EDMAMS
\DB001\Test02.log
```

In this run, the implied values (defaults) that affected the processing are: COMMIT_CHANGES=YES.

Note: The only difference between examples 2 and 3 is the value of PREVIEW=.

Example 4

Import the instance data (contained in `DB_001.XPI` and `DB_001.XPR`) from the SOFTWARE domain to the SOFTBACK domain in the PRIMARY file in the database. Do not write the changes to the database. Do not manage duplicate object IDs. Query the database and deck for matching records and, and if any are found, continue processing. Delete any orphaned resources that are encountered. Write the results to `C:\Temp\EDMAMS\DB001\Test03.log`.

```
ZEDMAMS VERB=IMPORT_INSTANCE,FILE=DB_001.XPI
,XPR=DB_001.XPR,MAP_DOMAIN=SOFTWARE/SOFTBACK,PREVIEW=YES
,REPLACE=NO,CONTINUE=YES,AUTOFIX=YES,LOGFILE=C:\Temp
\EDMAMS\DB001\Test03.log
```

In this run, the implied values (defaults) that affected the processing are: DUPLICATES=STOP.

Example 5

Import the instance data that is contained in `DB_001.XPI` and `DB_001.XPR` to the PRIMARY file that is specified in the `edmprof` file. Write the changes to the database. Do not query the database and deck for matching records. Write the results to `C:\Temp\EDMAMS\DB001\Test04.log`.

```
ZEDMAMS VERB=IMPORT_INSTANCE,FILE=DB_001.XPI
,XPR=DB_001.XPR,PREVIEW=NO,REPLACE=YES,LOGFILE=C:\Temp
\EDMAMS\DB001\Test04.log
```

In this run, the implied values (defaults) that affected the processing are: COMMIT_CHANGES=YES, DUPLICATES=STOP, and CONTINUE=NO.

Import and Export Files

Table 125 presents a list of the six default import/export files that are generated by the Radia database. Their level in the Radia database is part of the logic in their naming.

Table 125: Import and Export File Names

Database Level	Import File Name	Export File Name
Class	`.MPC` – modified class file	`.XPC` – exported class file
Instance	`.MPI` – modified instance file	`.XPI` – exported instance file
Resource	`.MPR` – modified resource file	`.XPR` – exported resource file

The export files (`XPI` and `XPR`) are the original decks that might be generated during the export process, and are the files that are imported either back into the existing database or into another database. So, each `XPI` and `XPR` file can be an export *and* import file.

The `MPI` and `MPR` files are decks that are generated during a database import that resulted in correcting duplicate OID issues, changing domains, or correcting empty or NULL OBJRCRCs.

IMPORT_RESOURCE

This verb imports resource data from an exported data set or file to a RESOURCE file specified in `edmprof`. This verb has the same functionality as the utility, EDMMIMPR.

The keyword, VERIFY, was added to this verb's functionality in version 4.4 of the Configuration Server.

- If REPLACE=NO, the class template will not be replaced.
- If VERIFY=YES, an implied PREVIEW=YES is set.

  If a resource exists, its size is compared to the size of the incoming resource to verify that they match.

Syntax:	`FILE=(,FROMDOMA=)(,TODOMA=)(,PREVIEW=`**`YES`**`/NO)(,REPLACE=YES/`**`NO`**`)(,VERIFY=YES/`**`NO`**`)`
Example:	Import all resources in the file specified by FILE to a RESOURCE file specified in `edmprof`: **`FILE=RESOURCE,PREVIEW=NO`**
Tip:	N/A

LIST_CLASSES

This verb displays a list of class names, object IDs, and other 60-byte prefix information, such as ZOBJDATE, ZOBJTIME, persistence flag, sequence sensitive flag, Distributed Configuration Server flag and db type and count totals.

Syntax:	FILE=,DOMAIN=
Example:	List all classes in the PRIMARY file: **`DOMAIN=*`**
Tip:	N/A

LIST_CONNECTS

This verb displays a list of connect-to values (type C) for the specified instances.

- INSTANCE can be partially specified.

  To display only one instance, the entire name must be specified.

Syntax:	`DOMAIN=,CLASS=(,INSTANCE=)`
Example:	List all of the connect-to values in the ZRSOURCE class of the SYSTEMX domain: **`DOMAIN=SOFTWARE,CLASS=ZSERVICE`**
Tip:	To list data for all type C values in all instances of the specified class, omit INSTANCE.

LIST_CONS_VARS

This verb automatically displays a list of connect-to data (type C) and, optionally, variable data (type V) for the specified instances.

- Wildcards (*) can be specified in INSTANCE.

  For example, specify DIFF* to select all the instances that contain DIFF as the first part of the string.

  To display only one instance, the entire name must be specified.

- If VTYPE=YES, variable data will be included in the display.

Syntax:	`(FILE=,)DOMAIN=,CLASS=(,INSTANCE=)(,VTYPE=YES/`**`NO`**`)`
Example:	From the USER class in the SOFTWARE domain, list the connect-to and variable values only for those instances prefixed with DIFF: **`DOMAIN=SOFTWARE,CLASS=ZSERVICE,INSTANCE=DIFF ,VTYPE=YES`**
Tip:	To list all C- and V-type data in all the instances of the specified class, omit INSTANCE.

LIST_DOMAINS

This verb displays an alphabetical list of domains for a specified file (the default is the PRIMARY file).

- FROMDOMA is the domain from which to start the list of domains.

  If omitted, all the domains through the TODOMAIN will be listed.

- TODOMAIN is the domain at which to end the list of domains.

  If omitted, all the domains following FROMDOMA will be listed.

Syntax:	`FILE=(,FROMDOMA=)(,TODOMAIN=)`

Example:	In the PRIMARY file, list all of the domains that follow the domain ACCT:
	`FILE=PRIMARY,FROMDOMA=ACCT`
	In the PROFILE file, list the domains in the range ACCT through SALES (inclusive):
	`FILE=PROFILE,FROMDOMA=ACCT,TODOMAIN=SALES`

Tips:	To list all the domains of the selected file, simply omit FROMDOMA and TODOMAIN. To list one domain, specify it for FROMDOMA and TODOMAIN.

LIST_FLAGS

This verb displays the attribute name, length, type, and Configuration Server and client flags for a specific class template.

- Specify README=YES to include the README attributes.

Syntax:	`DOMAIN=,CLASS=(,README=YES/`**`NO`**`)`
Example:	List the attribute information for the attributes of the ZSERVICE class of the SYSTEMX domain and omit the README attributes: **`DOMAIN=SOFTWARE,CLASS=ZSERVICE`**
Tip:	N/A

LIST_INST_DATA

This verb displays, in a concise format, the attribute data of the specified instances.

- Wildcards (*) can be specified for INSTANCE.

  For example, specify DIFF* to select all the instances that contain DIFF as the first part of the string.

  To display only one instance, the entire name must be specified.

- Use FIELDS to specify up to six attribute names.

  Specify the fields with a space separating each name, and the entire string enclosed in quotation marks, as in:

  ```
  "field1 field2 field3 field4 field5 field6"
  ```

  To list all attribute data of all instances of the specified class, omit FIELDS.

  If FIELD is omitted, all attribute data of all instances of the specified class will be displayed. This might produce a very large log, which might hinder locating data.

  If a fieldname does not exist in the template, it will be ignored.

Syntax:	`DOMAIN=,CLASS=(,INSTANCE=)(,FIELDS=)`
Example:	From the USER class in the SOFTWARE domain, list the attribute data of all instances with the prefix, RAD: **`DOMAIN=SOFTWARE,CLASS=ZSERVICE,INSTANCE=RAD*`**
Tip:	To list all instances of the specified class, omit INSTANCE.

LIST_INSTANCE

This verb displays a list of instance names and object IDs for the class specified. It also displays the ZOBJTIME and resource size.

- The value of FROMINST can be partially specified.

  For example, specify FROMINST=DIFF to select all the instances that contain DIFF as any part of the string.

  Use a wildcard (*) to specify this value as a prefix, as in RAD*.

- The value of SUFFIX can be partially specified.

  For example, specify SUFFIX=INT to select all the instances that have INT as a suffix.

Syntax:	`DOMAIN=,CLASS=(,FROMINST=)(,SUFFIX=)`
Example:	From the USER class of the POLICY domain, list the instance names and object IDs for all instances with the prefix RAD, and all instances with the suffix, PORT: **`DOMAIN=POLICY,CLASS=USER,FROMINST=RAD*,SUFFIX=PORT`**
Tip:	To list all instances, omit FROMINST.

LIST_PACKAGE

This verb lists the instances and all mated components of the PACKAGE class.

- The value of INSTANCE can be partially specified.

  For example, specify INSTANCE=DIFF to select all the instances that contain DIFF as any part of the string.

  Use a wildcard (*) to specify this value as a prefix, as in, RAD*; and as a suffix, as in, *INT.

Syntax:	`(FILE=,)DOMAIN=,INSTANCE=`
Example:	From the SOFTWARE domain, list all instances with the prefix RAD: **`DOMAIN=SOFTWARE,INSTANCE=RAD*`**
Tip:	N/A

CLASS is not an option because this verb applies to the PACKAGE class only.

LIST_PREFIX

This verb displays data from the Distributed Configuration Server prefix.

- If CLASS is omitted, all class prefixes will be displayed.

Syntax:	`(FILE=,)DOMAIN=(,CLASS=)`
Example:	List the Distributed Configuration Server prefixes for all classes in SOFTWARE domain of the PRIMARY file: **`DOMAIN=SOFTWARE`**
Tip:	N/A

LIST_RESOURCES

This verb displays a list of resource names (with promote dates, times, data sizes, and object IDs) from the RESOURCE and PRIMARY files.

As of version 4.3 of the Radia database, this verb was re-named. Its original name was LIST_RESOURCE.

- The specified domain should be that in which the ZRSOURCE class resides.
- If there is no mated instance in the PRIMARY file, an appropriate message will be generated.
- CLASS will default to ZRSOURCE.
- If ORPHANS=YES, only the orphans will be listed.
- If CHKSIZE=YES, the size listed (from ZOBJRSIZ) is compared to the actual size (from the NvdDBFind).

  Only those resources that have a size anomaly will be listed.
- SIZE is the size of the resource.

  SIZE must be specified as a range (not a single byte size), and the range must be defined with a dash (**-**), as in, 100-500.

Syntax:	`DOMAIN=(,CLASS=)(,ORPHANS=YES/`**`NO`**`)(,CHKSIZE=YES/`**`NO`**`)(,SIZE=`*`nnnn`*`)`
Example:	From the SOFTWARE domain of the PRIMARY file, list all resources that are between 64 and 1024 bytes in length, and compare this with the actual size: **`DOMAIN=SOFTWARE,CHKSIZE=YES,SIZE=64-1024`**
Tip:	N/A

LIST_ZRSC_FIELDS

This verb displays the data in all fields that begin with **ZRSC**, such as ZRSCSIZE and ZRSCVRFY. This verb allows wildcards for INSTANCE.

- INSTANCE can be specified with a partial name.

  For example, specify INSTANCE=DIFF to select all the instances that contain DIFF as any part of the string.

  Use a wildcard (*) to specify this value as a prefix, as in, RAD*; and as a suffix, as in, *INT.

  To display only one instance, the entire name must be specified.

Syntax:	`DOMAIN=,CLASS=(,INSTANCE=)`
Example:	From the ZSERVICE class of the SOFTWARE domain, list the instances that begin with CICS: **`DOMAIN=SOFTWARE,CLASS=ZSERVICE,INSTANCE=CICS*`**
Tip:	To list all instances of the specified class, do not specify INSTANCE.

MATCH_RESOURCES

This verb will compare resource data from the RESOURCE file against instance names from the PRIMARY file to determine and display whether those resources are mated (orphaned). This comparison is made by reading the resource data, extracting the instance name from the resource prefix, and attempting to find the mated instance.

The keyword, PREVIEW, was added to this verb as of version 4.3 of the Radia database.

- If PREVIEW=NO, and there is resource data that has been determined to be orphaned, a search of the PRIMARY file instances will occur, in order to locate a matching instance object ID.

  If a match is made, the instance name is placed in the resource data prefix and the resource is updated, with the time stamp for the resource data being updated with the ZRSCDATE and ZRSCTIME.
- Totals at completion include resources found, as well as total resources (mated and orphaned).
- When using PREVIEW=NO, a great deal of I/O might occur.
- CLASS defaults to ZRSOURCE.

Syntax:	`DOMAIN=(,CLASS=)(,PREVIEW=`**`YES`**`/NO)`
Example:	Match and display resource data names for all the classes of the SOFTWARE domain in the RESOURCE file against instance names under the SOFTWARE domain in the PRIMARY file: **`DOMAIN=SOFTWARE,PREVIEW=YES`**
Tip:	N/A

If many orphans are detected with PREVIEW=YES, a more efficient way to update the resource data file is to use the verb ZRSOURCE_UNMATES, detailed later in this chapter.

PACKAGE_UNMATES

This verb lists all PACKAGE class instances that do not have mated components in the domain that is specified.

- DOMAIN must be one that has a PACKAGE class.
- CLASS defaults to PACKAGE.

Syntax:	`DOMAIN=(,CLASS=)`
Example:	From the SOFTWARE domain, list all the PACKAGE class instances that have no mated components: **`DOMAIN=SOFTWARE`**
Tip:	N/A

REFRESH_DMA

This verb will recount all the instances, classes, and domains in the PRIMARY file and, optionally, refresh the **count** (TotalInstanceCount, TotalClassCount, and TotalDomainCount) and **date** (LastUpdateDate, LastInstanceUpdateDate, and LastClassUpdateDate) fields in the appropriate Distributed Configuration Server-prefix areas. Additionally, the updated output can be displayed in the log.

This verb was introduced with version 4.2 of the Configuration Server. It replaced the verb, REFRESH_COUNTS.

The keywords, DOMAIN and CLASS, were added with version 4.3; and the keyword, COUNTS_ONLY, was added with version 4.5.2.

- If PREVIEW=YES, a count of instances by class, domain, and file will be listed to the log, but no data will be written.

  The log will display the actual count (as calculated by running this verb) and the current count (current values in the total count fields [mentioned in the introductory paragraph] in the database) in the Distributed Configuration Server prefix. If the actual count differs from the current count, the latter will be flagged with an asterisk (*).

- If PREVIEW=NO, the TotalInstanceCount, TotalClassCount, and TotalDomainCount (for each applicable Distributed Configuration Server prefix) will be computed and updated, in addition to updating the current counts.

- If COUNTS_ONLY=**NO** (the default) and PREVIEW=NO, the current count and date fields in the Distributed Configuration Server area for each class will be updated.

  If COUNTS_ONLY=NO and PREVIEW=YES, the current count and date fields in the Distributed Configuration Server area for each class will be displayed.

- If COUNTS_ONLY=YES and PREVIEW=NO, the current count fields only will be updated. This is effective for very large database files because, by not refreshing the date fields and not having to read every instance in the database, the function executes in less time.

  If COUNTS_ONLY=YES and PREVIEW=YES, the current count and actual count fields will be displayed, but not updated; the date fields are not touched.

- Each class for each domain will be previewed separately, and at the end of each domain, a domain summary will be presented.

- After the last domain, a file summary will be presented by listing the ZBASE.ZBASE template information.

Syntax:	`PREVIEW=`**`YES`**`/NO(,DOMAIN=)(,CLASS=)(,COUNTS_ONLY=YES/`**`NO`**`)`
Example:	Preview the counts and update dates in the Distributed Configuration Server prefix for the entire PRIMARY file: **`PREVIEW=YES`**
Tip:	N/A

RENAME_INSTANCE

This verb will rename instances and the internal name of any mated resource data.

- KEEP indicates whether the old instance will be deleted.
- OLDPREFIX specifies existing instances that are to be renamed.

  If a single instance is to be renamed, specify the entire name.

  If multiple instances are to be renamed, a wildcard (*) is required.

  For example, to change the names of the instances, **east_sales** and **north_sales** to **US_Sales**, specify OLDPREFIX=*_sales,NEWPREFIX=US_Sales.
- NEWPREFIX is that which will replace OLDPREFIX.

Syntax:	`DOMAIN=,CLASS=,OLDPREFIX=,NEWPREFIX=(,KEEP=YES/`**`NO`**`)(, PREVIEW=`**`YES`**`/NO)`
Example:	In the ZSERVICE class of the SOFTWARE domain, rename all instances prefixed with EAST to NORTH_EAST, and delete the old prefix: **`DOMAIN=SOFTWARE,CLASS=ZSERVICE,OLDPREFIX=EAST,NEWPREFIX=NORTH_EAST,PREVIEW=NO`**
Tip:	N/A

SEARCH_INSTANCES

This verb will search the specified instances for the data contained in STRING.

- The data specified is not case-sensitive; if it contains embedded spaces, it must be enclosed in quotation marks (`""`).
- The output log will contain the name of the instance, attribute, and the specified STRING value.
- If the value of STRING is not found in the specified instances, this will be reported in the log.
- FROMINST can be specified with a partial name.

  For example, specify FROMINST=DIFF to select all the instances that contain DIFF as any part of the string.

  Use a wildcard (*) to specify this value as a prefix, as in, RAD*; and as a suffix, as in, *INT.

  To display only one instance, the entire name must be specified.

Syntax:	`DOMAIN=,CLASS=(,FROMINST=,)STRING=`
Example:	Search for the string “J. Q. Public” in all instances that end in EAST in the USER class of the SOFTWARE domain: **`DOMAIN=SOFTWARE,CLASS=USER,STRING="J. Q. Public",FROMINST=*EAST`**
Tip:	To list all instances of the specified class, omit FROMINST.

SORT_OBJECT_ID

This verb will sort object IDs within a domain.

- DOMAIN can be a single domain or all domains of the specified file.

  To sort the object IDs of multiple (but not all) domains in a file, execute this verb once for each.

  To sort the object IDs of multiple domains in multiple files, execute this verb once for each.

- ORDER=NOSORT means that the OBJIDs will be listed in CLASS.INSTANCE groups.

  ORDER=ASCEND/DESCEND specifies the order of sorting, based on OBJIDs.

- Duplicate OBJIDs will be flagged in ascending or descending order.
- ALLIDS is valid only if ORDER=ASCEND or DESCEND.

  ALLIDS=YES will duplicate the entire file.

  ALLIDS=NO will duplicate OBJIDs only.

Syntax:	`(FILE=,)DOMAIN=(,ORDER=`**`ASCEND`**`/DESCEND/NOSORT)`
	`(,ALLIDS=`**`YES`**`/NO)`

Example:	Sort, in descending order, all the object IDs of all domains, and duplicate the entire file: **`DOMAIN=ALL,ORDER=DESCEND,ALLIDS=YES`**
Tip:	All domains can be selected by specifying **`DOMAIN=ALL`**.

SYNC_CLASS

This verb will synchronize an existing (target) class with a newly formatted (source) class, and re-organize all existing class instances according to the mapping in the source class template.

- All target-class attributes that have a match in the new template will adopt the characteristics of that matching attribute.

  Any target-class attributes that do not have a match in the new template will be deleted.
- TODOMAIN specifies the class that contains the target class template that is to be synchronized.
- SYNCDOMA specifies the class that contains the source class template. The default is **ZEDMSYNC**.
- CLASS is the target class (within the domain that is specified by TODOMAIN *and* SYNCDOMA) that will be synchronized.

  The value of CLASS must exist in the TODOMAIN and SYNCDOMA domains; if it doesn't, the function will fail.
- CACHE=YES will update loaded cache when running as a Configuration Server method.
- If BASE=YES, this verb will copy the _BASE_INSTANCE_ from the source (SYNCDOMA) to the target (TODOMAIN).

Syntax:	`TODOMAIN=(,PREVIEW=`**`YES`**`/NO)(,SYNCDOMA=`**`ZEDMSYNC`**`,)CLASS=(,CACHE=YES/`**`NO`**`)(,BASE=YES/`**`NO`**`)`
Example:	In the POLICY domain, synchronize the existing USER class with a newly formatted USER class, imported from ZEDMSYNC. Include the _BASE_INSTANCE_ and update the loaded cache: **`TODOMAIN=POLICY,CLASS=USER,PREVIEW=NO,CACHE=YES,BASE=YES`**
Tip:	N/A

UPDATE_INSTANCES

This verb will update instances in the specified domain from data in an edited text file, INFILE. INFILE must conform to the following specifications.

- The first line must contain a new instance name (max. 32 characters) starting in column 1.

  The next line contains the variable field name (max eight characters) starting in column 2; then a blank in column 10; a 1-byte index value (1-9) in column 11; a blank in column 12; and the data to be populated beginning in column 13.
- REPLACE=YES specifies that an attribute that contains existing data (non-blank) be overlaid.
- Use a slash-asterisk combination (**/***) in columns 1 and 2 to denote the end of instance data (see lines 5 and 10 below).

Column Number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
Line 1	U	S	E	R	_	N	A	M	E															
Line 2		N	A	M	E								J	.	A	.	D	E	V	E	L	O	P	
Line 3		E	D	M	S	E	T	U	P		5		5	T	H	A	T	T	R	I	B	U	T	E
Line 4		Z	P	R	I	O	R	I	T				9	9	9									
Line 5	/	*																						
Line 6	U	S	E	R	_	N	A	M	2															
Line 7		N	A	M	E								L	.	Z	I	M	M	E	R				
Line 8		E	D	M	S	E	T	U	P		2		2	N	D	A	T	T	R	I	B	U	T	E
Line 9		E	D	M	S	E	T	U	P		3		3	R	D	A	T	T	R	I	B	U	T	E
Line 10	/	*																						
Line 11	U	S	E	R	_	N	A	M	E	_	3													
Line 12		N	A	M	E								J	I	M		D	O	E					
Line 13		E	D	M	S	E	T	U	P		4		4	T	H	A	T	T	R	I	B	U	T	E

Syntax:	`INFILE=,DOMAIN=,CLASS=(,REPLACE=YES/`**`NO`**`)`
Example:	Update the instances in the MYINPUT.TXT file in the SYSTEMX domain's USER class: **`INFILE=MYINPUT.TXT,DOMAIN=SYSTEMX,CLASS=USER`**
Tip:	Run LIST_INST_DATA or LIST_FLAGS to determine the index of a like-named variable to be updated.

UPDATE_MGRIDS

This verb updates the specified Manager ID, Manager name, managing Manager ID, and managing Manager name in the HP OpenView Distributed Configuration Server Using Radia prefix.

The Configuration Server was previously called the *Manager*.

Therefore, the keywords associated with this verb retain the 'Manager' designation, as in, Manager name (MNAME).

- All keywords are optional. However, at least one keyword other than DOMAIN and CLASS must be specified in order to avoid a usage error being displayed to STDERR.

Syntax:	`(FILE=)(,DOMAIN=)(,CLASS=)(,MID=)(,MMID=)(,MNAME=)(,MMNAME=)`
Example:	Update the managing Configuration Server IDs and managing Configuration Server names in the USER class of the POLICY domain in the PRIMARY file: **`FILE=PRIMARY,DOMAIN=POLICY,CLASS=USER,MMID=010,MMNAME=New_Mngng_RCS`**
Tip:	To update an entire file, omit DOMAIN and CLASS.

VERIFY_CLASS

This verb will display class templates, as specified, to determine if any gaps, overlays, or other anomalies exist. The output log will consist of a template entry number, the attribute name, its length, and its displacement in the heap. If an error is found, it will be indicated in the appropriate place.

- DOMAIN must be specified.

  All domains can be selected by specifying the value as ALL.
- CLASS must be specified.

  All classes can be selected by specifying the value as ALL.

Syntax:	`(FILE=,)DOMAIN=,CLASS=`

Example:	In the POLICY domain of the PRIMARY file, verify all the class templates:
	`FILE=PRIMARY,DOMAIN=POLICY,CLASS=ALL`
	Verify the ZSERVICE class templates in the PRIMARY file:
	`DOMAIN=ALL,CLASS=ZSERVICE`

Tip:	N/A

VERIFY_DATABASE

This verb will validate the integrity of a Radia database. It can be used at any time to check database integrity, and can run stand-alone or as a Configuration Server method. This database validation must be done in read-only mode.

This verb was introduced with version 4.5.1 of the Configuration Server.

- DOMAIN can be a single domain or all domains of the Radia database.

  To verify the integrity of multiple (but not all) domains in the Radia database, execute this verb once for each domain.

  This is an exhaustive check of the database and, as such, might take a long time to run, possibly several hours.

  It makes two passes over the database – first, checking the PRIMARY file and any associated resources; and second, checking the RESOURCE file for orphans.

  As a result of the two passes, it has enough information to do a check for duplicate object IDs.

Syntax:	`DOMAIN=(ALL/`*`any_domain`*`)`

Example:	Verify the integrity of the Radia database's SOFTWARE domains:
	`DOMAIN=SOFTWARE`
	Verify the integrity of all domains in the database:
	`DOMAIN=ALL`

Tip:	Specify DOMAIN=ALL to select all domains in the Radia database.

ZRSOURCE_UNMATES

This verb will match instances in the PRIMARY file with the resource data in the RESOURCE file, based on the specified domain and class, and the object ID from the PRIMARY file instances.

- If resource data is not found, the instance is considered *unmated*.
- If resource data is found, the instance name in the resource data prefix is compared with the instance name from the PRIMARY file.

  If it does not match, it is reported in the log as an inconsistency.
- If PREVIEW=NO, the resource prefix is updated to reflect the true instance name from the PRIMARY file.
- At completion, totals are listed in the log to indicate the number of unmated instances, inconsistencies, and so forth.
- DOMAIN is that which contains resource data, usually SYSTEMX.
- If CLASS is omitted, the default is ZRSOURCE.
- If DEBUG=YES, all instances are listed as they are verified.

Syntax:	`DOMAIN=(,CLASS=)(,PREVIEW=`**`YES`**`/NO)(,DEBUG=YES/`**`NO`**`)`
Example:	In the USER class of the POLICY domain in the PRIMARY file, list all unmated instances and inconsistencies: **`PREVIEW=YES,DOMAIN=POLICY,CLASS=USER,DEBUG=YES`**
Tip:	N/A

JCL Examples for Running EDMAMS (MVS Only)

The following are JCL examples for running the EDMAMS verbs.

Example Number 1

For running EDMAMS on an MVS operating system using the **KEYWORD DD** statement method.

```
//JOBNAME1 JOB CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1),NOTIFY=&SYSUID
        //EDMAMS   EXEC PGM=ZEDMAMS,COND=(0,NE),PARM=(ZFILE)
//STEPLIB  DD DSN=YOUR.EDM.LOAD.LIBRARY,DISP=SHR
//         DD DISP=SHR,DSN=SASC.C650.LINKLIB
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//PREVIEW  DD SYSOUT=*
//SYSTERM  DD SYSOUT=*
//STGRPT   DD SYSOUT=*
//LOG      DD SYSOUT=*
//INPDS    DD DSN=YOUR.INPUT.PDS.(PDSINP),DISP=SHR
//KEYWORD  DD DSN=YOUR.KEYWORD.PDS(KW1),DISP=SHR
//PARMLIB  DD DSN=YOUR.EDM.PARMLIB(STARTUP),DISP=SHR
//PRIMARY  DD DISP=SHR,DSN=YOUR.EDM.PRIMARY.FILE,
//   AMP=('STRNO=10,BUFNI=30,BUFND=100')
//RESOURCE DD DISP=SHR,DSN=YOUR.EDM.RESOURCE.FILE,
//   AMP=('STRNO=10,BUFNI=30,BUFND=30')
//*
```

Note: The KEYWORD DD statement contains the name of a data set member with EDMAMS keywords, as the following example indicates:

```
*
VERB=CHANGE_INST_DATA,DOMAIN=SYSTEMA,CLASS=USER,
FROMDATA="J. FILBERT",TODATA="JOHN B. MCDONALD"
*
VERB=CHANGE_INS_FIELD,DOMAIN=SYSTEMA,CLASS=USER,PREFIX=DIF,SUFFIX=F8, FIELD=NAME,TODATA="H.
        Kapelcro",PREVIEW=NO
*
VERB=LIST_INST_DATA,DOMAIN=SYSTEMA,CLASS=USER,INSTANCE=DIFF
*
        VERB=SEARCH_INSTANCES,DOMAIN=SYSTEMA,CLASS=USER,STRING="J. SMITH"
*
VERB=CHECK_RESOURCES,LISTALL=NO
*
VERB=COPY_CLASS,FROMDOMA=SYSTEMX,TODOMAIN=SYSTEMC,FROMCLAS=USER,REPLACE=YES
*
VERB=COPY_INSTANCE,FROMDOMA=SYSTEMX,TODOMAIN=SYSTEMA,FROMCLAS=USER,FROMINST=DIFF2
*
VERB=DELETE_DOMAIN,FROMDOMA=SYSTEMC,PREVIEW=NO
*
VERB=CREATE_INSTANCES,INFILE=INPDS,DOMAIN=SYSTEMA,CLASS=USER
```

```
*
VERB=DELETE_CLASS,DOMAIN=SYSTEMC,CLASS=ZSERVICE
*
VERB=DELETE_INSTANCE,DOMAIN=SYSTEMA,CLASS=USER,FROMINST=DIFF
*
VERB=LIST_CONS_VARS,DOMAIN=SYSTEMX,CLASS=ZRSOURCE,INSTANCE=CICS
*
VERB=LIST_DOMAINS
*
VERB=LIST_FLAGS,DOMAIN=SYSTEMX,CLASS=ZRSOURCE
*
VERB=LIST_INSTANCE,DOMAIN=SYSTEMA,CLASS=USER
*
VERB=LIST_RESOURCES,DOMAIN=SYSTEMX
*
VERB=LIST_ZRSC_FIELDS,DOMAIN=SYSTEMX,CLASS=ZRSOURCE,INSTANCE=EDM
```

Example Number 2

For running EDMAMS on an MVS operating system using the **PARM=** method.

This method is the least convenient, because the syntax for entering the keywords is very exacting.

```
//JOBNAME1 JOB CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//EDMAMS   EXEC PGM=ZEDMAMS,COND=(0,NE),
//        PARM=('VERB=COPY_CLASS','FROMDOMA=SYSTEMX','FROMCLAS=USER',
        'TODOMAIN=SYSTEMB','REPLACE=YES')
//STEPLIB  DD DSN=YOUR.EDM.LOAD.LIBRARY,DISP=SHR
//         DD DISP=SHR,DSN=SASC.C650.LINKLIB
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//PREVIEW  DD SYSOUT=*
//SYSTERM  DD SYSOUT=*
//STGRPT   DD SYSOUT=*
//LOG      DD SYSOUT=*
//PARMLIB  DD DSN=YOUR.EDM.PARMLIB(STARTUP),DISP=SHR
//PRIMARY  DD DISP=SHR,DSN=YOUR.EDM.PRIMARY.FILE
//RESOURCE DD DISP=SHR,DSN=YOUR.EDM.RESOURCE.FILE
//*
```

Example Number 3

For running EDMAMS on an MVS operating system using the **redirection** method. This method is convenient when running single EDMAMS utilities, but the PARMIN line is limited to 80 characters with no intervening spaces.

```
//JOBNAME1 JOB CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//EDMAMS   EXEC PGM=ZEDMAMS,COND=(0,NE),
//  PARM='=<//DDN:PARMIN'        REDIRECT INLINE - 80 CHARS MAX
//PARMIN   DD  *
  VERB=LIST_INSTANCE,DOMAIN=SYSTEMX,CLASS=USER
```

```
//*
//STEPLIB  DD DSN=YOUR.EDM.LOAD.LIBRARY,DISP=SHR
//         DD DISP=SHR,DSN=SASC.C650.LINKLIB
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//PREVIEW  DD SYSOUT=*
//SYSTERM  DD SYSOUT=*
//STGRPT   DD SYSOUT=*
//LOG      DD SYSOUT=*
//INPDS    DD DSN=YOUR.INPUT.DATA.SET(PDSINP),DISP=SHR
//PARMLIB  DD DSN=YOUR.EDM.PARMLIB(STARTUP),DISP=SHR
//PRIMARY  DD DISP=SHR,DSN=YOUR.EDM.PRIMARY.FILE,
//   AMP=('STRNO=10,BUFNI=30,BUFND=100')
//RESOURCE DD DISP=SHR,DSN=YOUR.EDM.RESOURCE.FILE,
//   AMP=('STRNO=10,BUFNI=30,BUFND=30')
//*
```

8 Configuration Server Performance

At the end of this chapter, you will:

- Have a better understanding of how CPU and network considerations can impact various performance aspects of the HP OpenView Configuration Server Using Radia (Configuration Server).

An Overview of Performance Issues

The purpose of this chapter is to discuss system performance issues as they relate to the Configuration Server. The next chapter, Chapter 9, Troubleshooting the Configuration Server, explores some problem determination issues.

Performance issues are associated with enhancing the efficiency of a working system, while troubleshooting deals with features, functions, and components that are not operating as expected. Taking into account performance and usage considerations prior to configuring the Configuration Server might prevent many of the conditions that require troubleshooting.

The Configuration Server is a multi-platform, multi-processing server framework for:

- Policy Management
- Component Management
- Network Management
- Version Management
- Asset Management
- State Management

There are many aspects of Configuration Server performance. In addition, there are specific phases of Configuration Server operations. Each phase has different performance characteristics and requirements. Because of these many variables, there is no easy 'cook book' approach to Configuration Server performance.

General Performance-and-Usage Considerations

Three important performance-and-usage considerations must be taken into account before beginning any discussion of Configuration Server issues.

- *What is the overall system infrastructure?*
  This includes numbers, types, and speeds of processors; total size and type of memory; and network capability and configuration.
- *What are the performance benchmarks?*
  These include average performance levels, as well as the high and low levels.

- *What are the workload parameters?*
  These include average demand, peak load requirements, and idle times.

Before undertaking any further performance initiatives, become familiar with the above considerations as they apply to your Configuration Server.

How this Chapter is Organized

This chapter is divided into three areas that dramatically influence performance:

- CPU Requirements, starting below.
- Memory, starting on page 393.
- Networking, starting on page 395.

CPU Requirements

There are minimum CPU requirements specified at installation for each Configuration Server platform. It must be noted, however, that these are *minimum values*. The real requirements for CPU utilization can only be determined by workload—essentially, the number of resolutions that a Configuration Server can process.

The CPU and Object Resolution

As each client connects to the Configuration Server, an identifier object (ZMASTER) is sent from the client to the Configuration Server triggering the dynamic construction of an object model for that client. This process is known as *object resolution*. The object resolution process exhausts most of the processor time required by the Configuration Server.

When trying to determine how many object resolutions can take place simultaneously, use the simple formula outlined below.

Total Number of Available CPU Seconds

x <u>Number of CPU Seconds required (per user)</u>

= **Total Number of Possible User Resolutions**

Total Number of Available CPU Seconds

This value is obtained by multiplying the number of CPUs by the number of seconds in the connection window of opportunity. The window of opportunity is the timeframe in which the clients need to connect.

For example: a two-processor machine with a six-hour window of opportunity (e.g., 12:00 AM - 6:00 AM) would result in a total number of available CPU seconds of 43,200. (2 processors X 6 hours (21,600 seconds) = 43,200 seconds).

Total Number of CPU Seconds Required Per User

This value is a little more complicated to obtain. Some benchmarks for object resolution speed have been determined by HP running a Configuration Server on an HP/K200 system with 85 MHz processors. We have determined that approximately 860 objects can be resolved in one second. With this benchmark, we can make an approximation of how many CPU seconds are required to resolve a user's object model.

By taking the average number of ZRSOURCE objects per user (ZRSOURCE being the most common object in a user's model) and multiplying it by three (an estimate of how many objects are resolved in order to end up with a fully resolved ZRSOURCE), we get the average number of objects to be resolved per user. We then divide that by the number of objects the Configuration Server can resolve in a CPU second, and get the number of CPU seconds required to resolve the average client's object model.

For example, let us assume that the average number of ZRSOURCE objects per user in your environment is 1000. We multiply that by 3, and get 3000 objects per user. Now divide 3000 by 860 (the average number of objects resolved per second by the Configuration Server), and you get approximately 3.5 seconds of CPU time required to resolve the average user's model.

1000 x 3 = 3000
3000/860 = ~3.5 (3.488…)

Total Number of Possible User Resolutions

This value is obtained by dividing the (*Total Number of Available CPU Seconds*) by the (*Total Number of CPU Seconds Required per User*). Using the results of our previous examples, we would divide the 43,200 (available CPU Seconds based on a two-processor machine with a six-hour window) by 3.5 (number of CPU seconds required to resolve the average user's model) resulting in a (*Total Number of Possible User Resolutions*) of approximately 12,000.

43,200/3.5 = ~12,000 (12,342.8571…)

This calculation does not mean that 12,000 users could be successfully configured in the six-hour period. Other variables must be considered, such as disk I/O for data being transferred/received to/from clients, the platform's network card capability (for concurrent conversations between the Configuration Server and clients), and the system's available memory (process/memory swapping requires system overhead).

Also, note that other tasks running on the machine will be sharing system resources with the Configuration Server.

Memory

There are two features that deal with memory usage, **content caching** and **index caching**. They are established in the MGR_CACHE section of the Configuration Server `edmprof` file.

- **Content Caching**
  refers to loading a portion of the Radia database (class templates, base instances, and other instances) into memory to speed up the resolution process. This enhances performance by eliminating disk I/O. In configuring index caching, the size used for each content cache entry needs to reflect the size of the instance in the database before any resolution has been performed. This provides a resolution boost across the Configuration Server.
- **Index Caching**
  is used to keep in memory all names of the instances of the class that have been cached. Caching the names of all instances of cached classes eliminates the need to read the directory in order to determine which instances begin with the specified prefix. Index caching makes a significant performance improvement when the generic resolution feature is utilized. Generic resolutions are those that use a partially specified CLASS.INSTANCE naming format that terminates in an asterisk (*), indicating that all instances with the same prefix are to be resolved.

MGR_CACHE

The MGR_CACHE section of the Configuration Server `edmprof` file defines the values that determine how much virtual storage is reserved for content cache. The two values are CACHE_SEGMENTS, which determines the number of separate memory areas, and CACHE_SIZE, which is allocated at startup, and used exclusively for content cache. The product of CACHE_SEGMENTS x CACHE_SIZE is the amount of memory that will not be available for resolution purposes during connection. In the NT environment, the maximum virtual storage available for any single process (such as the NT Manager) is 2 GB. If the MS NT Enterprise Server is used, this is increased to 3 GB of virtual memory per process, but might be constrained by the size of real memory and the size of the page space available. The NT Manager will attempt to use all of the virtual storage available to it and might cause all (up to 2 GB) of the defined page space to be in use. The value of AVERAGE_OBJECT_SIZE in this section should be set to the size of the largest instance of the classes being cached. This defaults to 2048 bytes if not specified.

A parameter called ICACHE_SIZE is available for index caching. It is activated by simply specifying a value for the keyword. The easiest means to correctly size this is to take the total of all instances to be cached, multiply by 100, and place the result as the ICACHE_SIZE value size. ICACHE_SIZE is of benefit when generic resolution is active, that is, any connection to ZRSOURCE.PREFIX_* which requires the Configuration Server to process all of the potential instances prefixed by the string. While ICACHE is most important for ZRSOURCE, the caching mechanism (described below in MGR_CLASS) uses the same criteria for selecting which DOMAIN.CLASS instances to cache.

MGR_CLASS

The MGR_CLASS section controls two separate processes: initial classes to be cached and the amount of storage to be used for in-storage objects during resolution of each client (as differentiated from database classes and instances where the class name might be the same as the in-storage object name as is the case of ZSERVICE and ZRSOURCE). In-storage objects of interest are generally persistent and multi-heap. Controlling the storage and processing of these objects provides for performance improvements.

For index caching and content caching purposes, the contents of MGR_CLASS are processed in the order in which they are presented, so the first DOMAIN.CLASS is processed completely (index cache is loaded and content cache is loaded) before the second, and so on.

For each DOMAIN.CLASS (for example, SYSTEMX.ZRSOURCE) identified in this section, four parameters are specified. The first two control the caching behavior, and the second two are used exclusively for persistent object virtual storage allocation during resolution.

For a detailed explanation of the MGR_CLASS settings, including performance and usage considerations, see MGR_CLASS on page 46. The first of the four parameters (Value1) allows one to specify whether the class template and base instance are to be cached. A value of Y is recommended since it is generally necessary to load the class template and base instance and this eliminates a tremendous amount of disk I/O for classes that are commonly involved in resolution.

Networking

Bandwidth Throttling

Bandwidth throttling refers to the reservation of a certain percentage of the available TCP/IP bandwidth for use by other processes on the desktop. It was designed to help you maximize your network resources while running the Configuration Server.

Bandwidth throttling is configured in the Configuration Server using the SEND_THROTTLE setting of the MGR_TIMEOUT section of the Configuration Server `edmprof` file. This setting specifies the number of milliseconds that the Configuration Server will wait before sending packets. The default is 0, meaning no delay. The range of values is 0 to 4 GB.

There are three variables in the client's ZMASTER object that also have an impact on bandwidth throttling, ZBWMGR, ZBWMAX, and ZBWPCT.

- ZBWMGR=YES means the Configuration Server will be controlling the bandwidth.
- ZBWMAX is the maximum speed (in bytes per second) of the sends (0 - 4 GB).
- ZBWPCT is the percentage of the maximum to use (0 - 100).

The ZBWMGR, ZBWMAX, and ZBWPCT values will override the SEND_THROTTLE setting.

9 Troubleshooting the Configuration Server

At the end of this chapter, you will:

- Have a better idea of some of the common causes of Configuration Server processing problems, and be able to quickly recognize and remedy them.

Troubleshooting Issues

The purpose of this chapter is to explore problem-determination issues as they relate to the Configuration Server. Chapter 8, Configuration Server Performance discusses system performance issues.

Performance issues are associated with enhancing the efficiency of a working system while troubleshooting deals with features, functions, and components that are not operating satisfactorily. Before troubleshooting, see General Performance-and-Usage Considerations on page 390.

General Troubleshooting Considerations

There are several things that you should consider before attempting to troubleshoot a specific problem:

- *What, specifically, is the problem?*
  Sometimes, different problems have similar symptoms. For example, if a client resolution does not complete due to timing out, the timeout could be based on either a Configuration Server value or a client setting.
- *At what point did the problem occur?*
  If you can determine at what point a process failed, you might be able to eliminate prior steps.
- *How is the problem reflected in the Configuration Server log?*
  You can use the Configuration Server log and the search tools provided by HP Technical Support to isolate exactly where the problem is reported in the Configuration Server log.
- *Are there external causes for the problem?*
  You might be able to determine if a cause unrelated to the Configuration Server is responsible for the problem.

How this chapter is organized

This chapter is organized into three general scenarios:

- The Configuration Server Does Not Start
- The Configuration Server Does Not Process Tasks as Expected
- The Configuration Server Does Not Respond

Each scenario contains individual conditions, possible causes, and recommended actions.

The Configuration Server Does Not Start

Table 126: The Configuration Server Does Not Start

Condition	Possible Cause	Recommended Action
The Configuration Server does not start.	The Radia database did not verify correctly.	Reset VERIFY_DEPTH setting in the Configuration Server `edmprof` file.
	There is insufficient disk space.	Free up sufficient disk space.
	The Configuration Server `edmprof` file is not processed.	Ensure that the Configuration Server `edmprof` file is in the same directory as ZTOPTASK.
The Configuration Server does not start. (*NT-specific*)	The Configuration Server was installed under a user account that is not part of the Windows Admin Group.	Reinstall the Configuration Server under a user account that is part of the Windows Admin Group.
The Configuration Server does not appear in the Windows Services List. (*NT-specific*)	The Configuration Server was not installed as a Windows service.	Reinstall the Configuration Server as a Windows service.
The Configuration Server does not start automatically when rebooted. (*NT-specific*)	The Configuration Server Service in the Windows Services List is set to manual.	Set the Configuration Server Service to Automatic. Then reboot the Configuration Server.

The Configuration Server Does Not Process Tasks as Expected

There are two aspects to this scenario: either the Configuration Server does not perform the process correctly, or the data received is not correct.

Table 127: The Configuration Server Does Not Process Tasks as Expected

Condition	**Possible Cause**	**Recommended Action**
No Console or Admin functions.	Incorrect MGR_ACCESS values.	Change MGR_ACCESS values.
Configuration Server tasks not starting.	Tasks not listed in MGR_ATTACH_LIST section.	Add slots in MGR_ATTACH_LIST section.
Configuration Server does not accept client tasks.	No Configuration Server communications tasks specified in MGR_ATTACH_LIST section.	Specify appropriate Configuration Server communications tasks in MGR_ATTACH_LIST section.
Configuration Server does not accept additional client tasks.	Setting in TASKLIM is too low.	Increase TASKLIM setting.
Too much processor time required to load commonly used classes.	Classes not listed in MGR_CLASS section.	Add classes to MGR_CLASS section.
Methods not executing properly.	TIMEOUT setting in MGR_METHODS section is too low.	Increase TIMEOUT setting in MGR_METHODS section.
Too many messages in Configuration Server log.	Tracing is set to YES.	Set tracing to `NO` for unnecessary trace settings.
Configuration Server Log is slow to respond.	FLUSH_SIZE is set too low.	Increase FLUSH_SIZE in MGR_LOG section.
Lost portions of Configuration Server log.	Log has been reused.	Change THRESHOLD setting in MGR_LOG section to a positive value.

The Configuration Server Does Not Respond

Table 128: The Configuration Server Does Not Respond

Condition	Possible Cause	Recommended Action
The Configuration Server does not respond to communications requests.	Configuration Server communication tasks are not enabled.	Add appropriate Configuration Server communications tasks in MGR_ATTACH_LIST section.
The Configuration Server does not respond to client task requests.	Other clients have a RETRY value that is too low.	Raise RETRY value to at least 1.

10 Multi-Mode Configuration Servers

At the end of this chapter, you will:

- Be able to establish a single Configuration Server to service EDM and Radia clients.

  The proper EDM and Radia licensing has to be acquired in order to operate the Multi-Mode Configuration Server.

Introduction

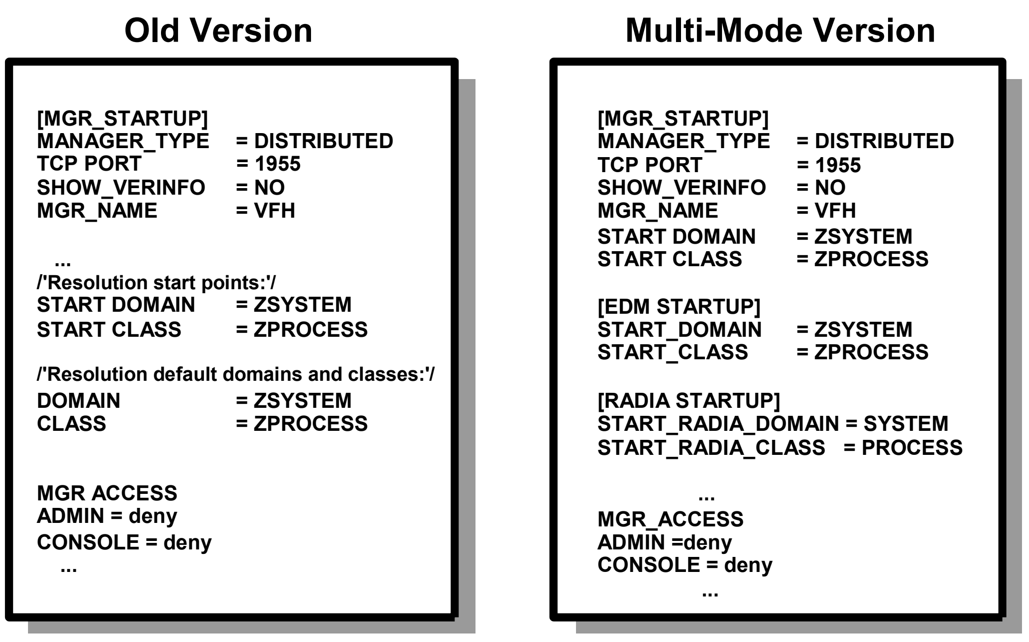
The Configuration Server is fully capable of processing EDM, Radia, and other transactions (clients) during a single Configuration Server session. Some features in the Configuration Server apply to Radia only, and conversely, some apply to EDM only. If EDM is seen as a multi-service batch type of environment, then Radia represents single-service processing. Since Radia is oriented to interactive processing, the database for it contains some additional features to allow for easier and friendlier usage. The EDM configurations are oriented to complicated, multi-dimensional processes that fully define the states of desktops and/or servers.

Single Configuration Server Approach

The Configuration Server can work in a multi-function mode, serving two types of clients. EDM and Radia clients, for example, have different settings that direct the paths of resolution that are stored in the Radia database. Generally, even the definitions can be shared as long as the starting point of object resolution for each type of client is defined separately.

In prior versions of the Configuration Server, the resolution starting point was defined in the MGR_STARTUP section of the `edmprof` file. The Multi-Mode Configuration Server uses the EDM_STARTUP and RADIA_STARTUP sections to define the DOMAIN.CLASS starting point for all EDM and Radia clients (respectively). This way, the resolution starting point will be different depending on the type of connection.

Figure 34 on page 405 presents a comparison of old and new resolution starting points.



Old Version

```
[MGR_STARTUP]
MANAGER_TYPE     = DISTRIBUTED
TCP PORT         = 1955
SHOW_VERINFO     = NO
MGR_NAME         = VFH

  ...
/'Resolution start points:'/
START DOMAIN     = ZSYSTEM
START CLASS      = ZPROCESS

/'Resolution default domains and classes:'/
DOMAIN           = ZSYSTEM
CLASS            = ZPROCESS


MGR ACCESS
ADMIN = deny
CONSOLE = deny
  ...
```

Multi-Mode Version

```
[MGR_STARTUP]
MANAGER_TYPE     = DISTRIBUTED
TCP PORT         = 1955
SHOW_VERINFO     = NO
MGR_NAME         = VFH
START DOMAIN     = ZSYSTEM
START CLASS      = ZPROCESS

[EDM STARTUP]
START_DOMAIN     = ZSYSTEM
START_CLASS      = ZPROCESS

[RADIA STARTUP]
START_RADIA_DOMAIN = SYSTEM
START_RADIA_CLASS   = PROCESS

             ...
MGR_ACCESS
ADMIN =deny
CONSOLE = deny
             ...
```

Figure 34: Resolution starting points.

In the Old Version example (above, left) the resolution starting point (START DOMAIN and START CLASS) for all clients is:

```
ZSYSTEM.ZPROCESS.OBJECT_NAME
```

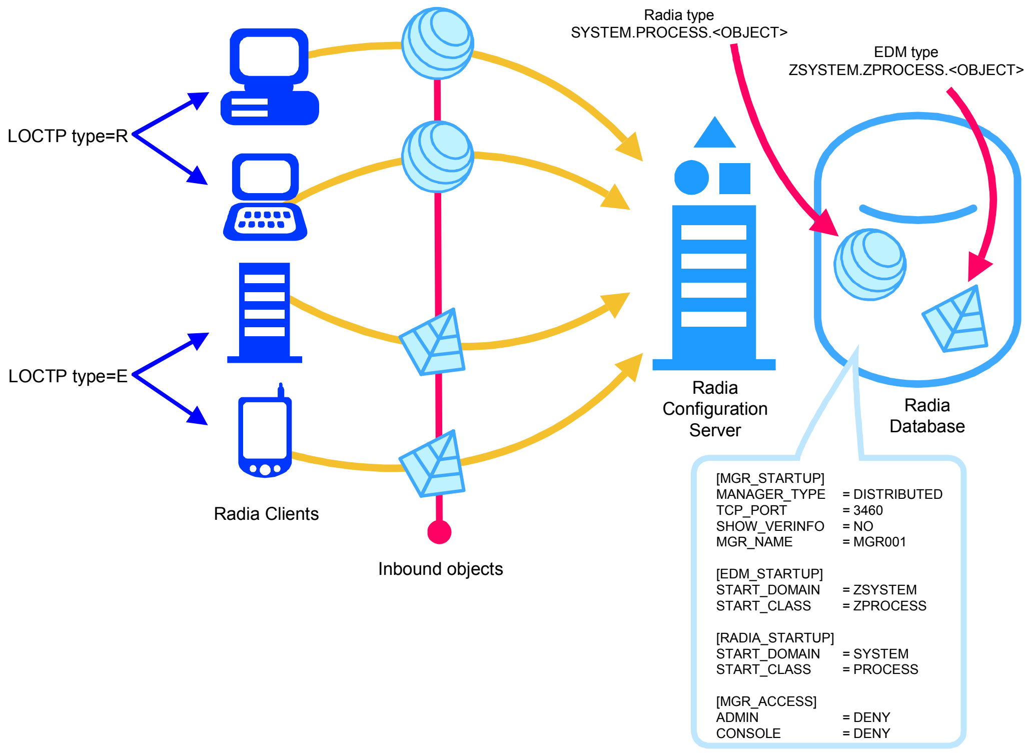
In the Multi-Mode Version example (above, right), the resolution starting point for EDM clients is:

```
ZSYSTEM.ZPROCESS.OBJECT_NAME
```

and resolution starting point for Radia clients is:

```
SYSTEM.PROCESS.OBJECT_NAME
```

In this case, the Multi-Mode Configuration Server's processing might look like:



```
[MGR_STARTUP]
MANAGER_TYPE    = DISTRIBUTED
TCP_PORT        = 3460
SHOW_VERINFO    = NO
MGR_NAME        = MGR001

[EDM_STARTUP]
START_DOMAIN    = ZSYSTEM
START_CLASS     = ZPROCESS

[RADIA_STARTUP]
START_DOMAIN    = SYSTEM
START_CLASS     = PROCESS

[MGR_ACCESS]
ADMIN           = DENY
CONSOLE         = DENY
```

Figure 35: Multi-mode Configuration Server's processing.

- *How does the Configuration Server distinguish between EDM and Radia clients?*
  The client type is defined in the initial "handshake" transactions (called EDMLOCTP) that every client sends to the Configuration Server immediately after the connection is established. The EDMLOCTP contains the client product field that sets the client type (E or R). As mentioned, the resolution starting point for each type of client is defined in the Configuration Server `edmprof` file. Then, for any in-bound object, the resolution starts in the specified domain and class depending on the type of client.

11 SSL Managers

At the end of this chapter, you will:

- Have a better understanding of the configuration and use of the Configuration Server's SSL Manager task.

  ➤ The proper licensing is required in order to operate the SSL Manager.

For more information on SSL Managers, proxies, and firewalls, refer to the HP OpenView Adapter for SSL Using Radia, which covers:

- Installing and implementing SSL in an HP OpenView Using Radia environment.
- SSL components and terminology.
- Configuring a Radia client.
- OSI and TCP/IP reference models.
- The purpose and benefits of proxies and firewalls.
- OpenSSL

Introduction

The Configuration Server is a powerful resource for the storage and dissemination of information. Its versatility is enhanced by the introduction of security in the information exchange, and enabling the Configuration Server to act as a web server.

This feature is enabled via the SSL Manager, a new task that must be entered in the MGR_ATTACH_LIST section of the Configuration Server's `edmprof` file. For instructions on how to do this, see the section, Configuration Server Changes, starting on page 409; or the MGR_ATTACH_LIST section, starting on page 37.

Virtual IP Addresses in UNIX

With virtual IP addresses, a machine with a single Network Interface Card (**NIC**) can have multiple IP addresses. This is especially useful when multiple server programs have to listen on the same port. To resolve the port conflicts, machines are set up with *virtual IP addresses*, whereby multiple IP addresses are assigned to a machine.

The ztcpmgr can support virtual IP addresses. It accepts the IP address and port number on the command line, as shown in this example MGR_ATTACH_LIST section entry:

```
CMD_LINE=(ztcpmgr addr=1.1.1.10,port=4438) RESTART=YES
```

If the **address** is not specified, the machine address is used.

To configure virtual IP addresses, use the **ifconfig** command. This command has to be run under **root** privileges, as shown in the following example.

```
/usr/sbin/ifconfig hme0:1 inet 208.244.225.163 netmask
0xffffff00 broadcast +

/usr/sbin/ifconfig hme0:2 inet 208.244.225.175 netmask
0xffffff00 broadcast +

/usr/sbin/ifconfig hme0:1 up

/usr/sbin/ifconfig hme0:2 up
```

hme0 is the device name. This can be **le0** on other systems. To get the proper device name, type:
ifconfig -a

Starting the Configuration Server with Root Privileges on UNIX Systems

When a Radia client has to connect to the Configuration Server using an intermediary Web server, it uses TCP/IP tunneling. The TCP/IP tunneling works by blindly funneling requests between the client and the Configuration Server. It is important that the Configuration Server's port number be properly set.

For using ports below 1024, which are reserved ports, you would have to start the Configuration Server as root, either using the rc scripts or logging in as root. In addition, you would have to set the LD_LIBRARY_PATH to the Configuration Server executable directory before you run ZTOPTASK, as in the following example.

```
LD_LIBRARY_PATH=/mgrbuild/V4.11/exe:/usr/lib:lib
export LD_LIBRARY_PATH
./ztoptask
```

For Microsoft proxy servers, the port number has to be 443, which is the secure HTTP port. This requires that the Configuration Server run on port 443, so that the proxy can contact it; otherwise, the proxy won't let the clients establish the tunnel.

SSL Manager

Enabling SSL in Configuration Server and Client

Secure Sockets Layer (**SSL**) capability increases security in the Configuration Server's information exchange. It is a communication DLL (shared library), similar to HP TCP/IP DLL. The SSL protocol is actually an extension of HP existing TCP/IP DLL, and is called `nvdtcps.dll`. SSL is used by the client and the Configuration Server, and is implemented using the public domain, **OpenSSL**.

To use SSL, the client and the Configuration Server each need a Certificate Authority root certificate (CA root certificate). These certificates enable the Configuration Server–client handshake, so they can communicate. The Configuration Server needs a Server certificate also.

Configuration Server Changes

To enable SSL on the Configuration Server, add a task to the MGR_ATTACH_LIST section, as below.

```
[MGR_ATTACH_LIST]
CMD_LINE=(zsslmgr) RESTART=YES
```

With the SSL Manager, there are two important components: `nvdtcps.dll` (the SSL DLL) and zsslmgr (the SSL Manager Task). The MGR_SSL section of the `edmprof` file allows you to configure SSL.

Client Changes

To enable SSL on the client, the parameters listed in Table 129 must be in its ZMASTER object.

Table 129: ZMASTER Object Parameters

Parameter	Function
CAFILE	Use to specify the Certificate Authority certificate file.
ZDEVICEN	Use to specify the device number for SSL (094).

Radia-Specific Changes

The Certificate Authority root certificates are stored in the CACERTIFICATES directory. The client should store all the CA certificates in this directory. If there are multiple CAs, they should be stored with unique names. The default certificate file is `CACERT.PEM`.

HP OpenView Proxy Server Using Radia

The HP OpenView Proxy Server Using Radia (Proxy Server) functions as an extension of the Configuration Server. When it is used, it becomes the primary repository for Radia client data. Once a Radia client determines which resources it needs in order to achieve its desired state, it can request the resources from the Proxy Server. This feature allows the Configuration Server to allocate more resources to other tasks.

Client requests are made using either HTTP or TCP/IP. The Proxy Server can service multiple, concurrent client requests using both protocols simultaneously. For extensive information on the Proxy Server, refer to the

Installation and Configuration Guide for the HP OpenView Proxy Server Using Radia (*Proxy Server Guide*).

A Configuration Server Methods

This appendix is a reference for Configuration Server Methods. For information on configuring and using Configuration Server Methods, see Chapter 3, Managing Configuration Server Processing. Table 130 provides an alphabetical list of Configuration Server methods and a description of the method's use.

Table 130: Configuration Server Methods

Method	Description
EDMMAILQ	Deposits e-mail in the mail queue (outbox) so it can be sent to a remote system user.
EDMMALLO	Allocates an external data set (MVS only).
EDMMCACH	Refreshes or disables cache.
EDMMCMPR	Compresses an in-storage object.
EDMMCOPY	Copies an in-storage object.
EDMMDALO	'De-allocates' an external data set (MVS only).
EDMMDB	Locks and unlocks the database against all components except Radia Distributed Configuration Server (Distributed Configuration Server).
EDMMDCLA	Deletes a class from the database.
EDMMDELI	Deletes an instance from an in-storage object.
EDMMDELV	Deletes a variable from all instances of an in-storage object.
EDMMDINS	Deletes an instance or instances from within a database class.
EDMMDOBJ	Deletes an in-storage object.
EDMMDPRO	Deletes an object in the PROFILE file.
EDMMEXIS	Verifies the existence of a given class or instance in the database.
EDMMGNUG	Retrieves a list of local and global groups to which a specified user belongs.
EDMMGPRO	Creates an in-storage object from the PROFILE file.

Method	Description
EDMMNFYT	Executes a TCP/IP notification on a client.
EDMMOLOG	Displays the contents of an in-storage object.
EDMMPHIS	Puts an in-storage object into the HISTORY file.
EDMMPPRO	Puts an in-storage object into the PROFILE file.
EDMMPROM	Adds or updates an instance to the database.
EDMMPUSH	Puts an inbound object into a notify queue.
EDMMPUTD	Receives multiple data types that are sent by the Radia Inventory Manager and stores the data in files on the Configuration Server (EDM only).
EDMMRESO	Resolves specified objects.
EDMMRPRO	Adds, updates, or deletes instances in the PRIMARY file based on the variables of an in-storage object.
EDMMSORT	Sorts instances, by stems, of in-storage objects.
EDMMSQLG	Imports data from an external SQL database.
EDMMSQLP	Exports data to an external SQL database.
EDMMTUCH	Updates the date/time stamp of an instance.
EDMMULOG	Used to write to the user log file.
EDMMVDEL	Deletes all in-storage objects.
EDMMVGBL	Migrates values from one in-storage object to another and deletes the source object.
EDMMXREF	Cross-references class and instance usage during the object-resolution process.
EDMSIGN	Authenticates users against the database.
EDMSIGNR	Authenticates users against external security systems.
ZUPDPROF	Updates profile information, only; it does not perform any type of deletion.

The following pages describe each Configuration Server method, providing examples of use, a description, its parameters, and the associated possible return codes.

All arguments are expected to be in the format, KEYWORD=VALUE, and delimited by commas.

Quotation marks (**“ ”**) are required when a value contains commas or embedded blanks.

EDMMAILQ

This method deposits e-mail in the mail queue (outbox). For this method to execute correctly, the MGR_SMTP_MAIL section must be added to the Configuration Server `edmprof` file.

EDMMAILQ Parameters

attach	Specifies attachment files. Multiple attachments can be listed by using a semi-colon (;) delimiter between each attachment. For example, **`c:\config.sys;c:\autoexec.bat`**. Attachments are sent using MIME, and can be in binary. This parameter is optional.
from	The sender’s address. This parameter is required.
mesgfile	Specifies the file that contains the message. Used in place of the parameter, message, if the message is greater than 255 characters. This parameter is optional.
message	Specifies a brief message (limited to 255 characters). This parameter is required.
subject	Specifies the subject of the e-mail. This parameter is optional.
to	Specifies the e-mail recipients. Multiple users can be listed by using a semi-colon (;) delimiter between each recipient. This parameter is required.

Parameters are used like keywords and are not case-sensitive.

Example

In the example below, an e-mail with a brief message is sent from **user1@company1.com** to **user2@company2.com**.

```
EDMMAILQ from=user1@company1.com,to=user2@company2.com,
Message="This is a brief message"
```

Example

In the example below, a text file (`c:\report.txt`) with a subject (**`My report`**) is sent between the same users.

```
EDMMAILQ from=user1@company1.com,to=user2@company2.com,
Mesgfile=c:\report.txt,Subject="My report"
```

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

EDMMALLO (MVS Only)

This method enables you to allocate a file dynamically for use in a REXX executable. The file will be freed automatically after the EXECIO command is issued with the FINIS option, or when the EDMMALLO method is issued. The maximum number of files a single REXX executable can allocate is 50. Use this method to set aside storage for the resulting output of a REXX method or program.

EDMMALLO Parameter

alloparm	One string with the three former parameters (DSN || "," || DISP || "," || MEM;) passed with commas separating the values.

Example

```
/************************* REXX *********************/
SAY 'SAMPLE FILE ALLOCATION'
/********** ALLOCATE THE DATA SET ******************/
DSN = 'DSN=USER1.EDM.TESTLIB';
DISP= 'DISP=SHR';
MEM= "MEMBER=TESTMEM"
ALLOPARM = DSN || "," || DISP || "," || MEM;
/***************************************************/
/* ALLOCATE THE FILE                               */
/***************************************************/
INDD = EDMMALLO(ALLOPARM)
/***************************************************/
/* READ ALL THE RECORDS FROM THE FILE INTO A       */
/* STRUCTURE WITH RECS. AS THE STEM VARIABLE FOR   */
/* EACH RECORD. THE FINIS OPTION WILL CAUSE THE    */
/* DATA SET TO BE CLOSED AND DEALLOCATED. IF THE   */
/* FINIS OPTION IS NOT SPECIFIED REXX KEEPS THE    */
/* DATA SET OPEN AND THE "EXECIO 0" COMMAND MUST   */
/* BE USED TO CLOSE THE FILE.                      */
```

```
/****************************************************/
"EXECIO * DISKR" INDD "(STEM RECS. FINIS"
SAY 'ALLOCATED   DDN = ' INDD;

/* "EXECIO * DISKR" INDD "(STEM RECS. " ←-THIS    */
/* WOULD NOT CLOSE/FREE                            */
/****************************************************/
/* DE-ALLOCATE THE FILE. THIS IS NOT NECESSARY AND */
/* WILL FAIL IF THE DATA SET ALREADY FREED. IT     */
/* SHOULD BE USED IN CASE AN ERROR IN THE REXX     */
/* EXEC DOES NOT EXECUTE THE EXECIO COMMAND AT ALL,*/
/* NEVER OPENING/CLOSING AND FREEING THE FILE. IF  */
/* THE DATA SET IS NOT FREED OTHER USERS MAY NOT BE*/
/* ABLE TO ACCESS THE DATA SET UNTIL THE MANAGER   */
/* IS TERMINATED.                                  */
/****************************************************/
DDNF = EDMMDALO(DDN=INDD);
/****************************************************/
/* ALLOCATE THE FILE                               */
/****************************************************/
  SAY 'DEALLOCATED DDN = ' DDF;
END
```

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

EDMMCACH

This method refreshes or disables caching.

EDMMCACH Parameters

Parameter	Description
option	Caching option values are ENABLE or DISABLE.

Example

```
ADDRESS EDMLINK EDMMCACH 'option=ENABLE' ;
```

MVS User’s Note

ADDRESS must be replaced with **CALL**, as in:

```
CALL EDMLINK EDMMCACH 'option=ENABLE'
```

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

EDMMCMPR

This method compresses an in-storage object.

EDMMCMPR Parameter

object	The name of the in-storage object to be compressed.

Example

```
ADDRESS EDMLINK EDMMCMPR 'ZTEST' ;
```

MVS User's Note

ADDRESS must be replaced with **CALL**, as in:

```
CALL EDMLINK EDMMCMPR 'ZTEST'
```

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

EDMMCOPY

This method copies an in-storage object. The resulting object has the same variables and number of heaps as the original object.

EDMMCOPY Parameters

fromobject	The name of the existing in-storage object to be copied.
toobject	The name of the in-storage object to be created.

Example

```
ADDRESS EDMLINK EDMMCOPY 'OBJECT1,OBJECT2' ;
```

MVS User's Note

ADDRESS must be replaced with **CALL**, as in:

```
CALL EDMLINK EDMMCOPY 'OBJECT1,OBJECT2'
```

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

EDMMDALO (MVS Only)

This method enables you to dynamically "de-allocate" files. It should be used only in case an error in the REXX executable does not execute the EXECIO command. Use this method to free the data set and terminate the Configuration Server, so other users will not be denied access to the data set.

If the data set is already freed, this method is not necessary, and, if specified, will result in a *fail*.

EDMMDALO Parameter

ddname	Dataset name (allocated by EDMMALLO) to be "de-allocated."

Example

```
EDMMDALO (DDN=NDD)
```

Return Codes

0	The method was successful.

EDMMDB

This method locks and unlocks the database to all tasks, except the Distributed Configuration Server.

EDMMDB Parameter

option	LOCK locks the database to any incoming tasks, except Distributed Configuration Server. UNLOCK makes the database accessible to all incoming tasks.

Example

```
EDMLINK EDMMDB "OPTION=LOCK"
```

MVS User's Note

ADDRESS must be replaced with **CALL**, as in:

```
CALL EDMLINK EDMMDB "OPTION=LOCK"
```

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

EDMMDCLA

This method deletes a class and its associated instances from the database.

ZDCLASS will not delete ZRSOURCE instances.

EDMMDCLA Parameters

domain	The eight-character (maximum) name of the domain that houses the class to be deleted.
class	The eight-character (maximum) name of the class to be deleted.
file	The file name that contains the class to be deleted. This parameter is optional.

Example

```
/***************************** REXX ****************/
 DOMAIN  = 'SYSTEMX ';
 CLASS   = 'TESTCLAS';
PARM     = SUBSTR(DOMAIN,1,8)  ||  SUBSTR(CLASS,1,8);
 SAY 'PARM STRING IS ' PARM;
 ADDRESS EDMLINK EDMMDCLA PARM;
```

MVS User's Note

ADDRESS must be replaced with **CALL**, as in:

```
CALL EDMLINK EDMMDCLA PARM
```

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

EDMMDELI

This method deletes a heap of an in-storage object.

EDMMDELI Parameters

object	The name of the in-storage object from which to delete a heap.
instance#	The heap number to delete.

Example

```
/************************ REXX ********************/
DPARM = 'TESTOBJ,'||1|| ' ';
ADDRESS EDMLINK EDMMDELI DPARM;
```

MVS User's Note
ADDRESS must be replaced with **CALL**, as in:
`CALL EDMLINK EDMMDELI DPARM`

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

EDMMDELV

This method deletes a variable from all heaps of an in-storage object. It verifies the existence of the specified object and finds the specified variable in that in-storage object. The variable value is then removed from each heap of the in-storage object.

EDMMDELV Parameters

object	The object that contains the variable to be deleted.
variable	The name of the variable to be deleted.

Example

```
/************************* REXX ********************/

ADDRESS EDMLINK EDMMDELV 'TESTOBJ,VAR00001' ;

SAY 'QAREXX ****** VAR00001 DELETED FROM OBJECT
     TESTOBJ' ;
```

MVS User's Note

ADDRESS must be replaced with **CALL**, as in:

```
CALL EDMLINK EDMMDELV 'TESTOBJ,VAR00001'
```

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

EDMMDINS

This method displays or deletes (from the Radia database) an instance, or range of instances, within a class. It permits the use of wildcards (*).

Displayed instances will be written to the Configuration Server log even if all other TRACE settings are OFF.

EDMMDINS Parameters

Parameter	Description
file	The file that contains the instances to be displayed or deleted.
domain	The domain that contains the instances to be displayed or deleted.
class	The class that contains the instances to be displayed or deleted.
option	DISPLAY if instances are to be displayed. DELETE if instances are to be deleted.
frominst	The name or starting name of the instance to be deleted or displayed.
toinst	The instance to be deleted or displayed. Blanks in this field indicate that it is a single instance to display or delete, not a range.

Example

```
/************************ REXX ********************/
FILE     = 'PRIMARY'
 DOMAIN  = 'SYSTEMX'
 CLASS      = 'ZRSOURCE';
 FROMIN  = 'TSO_                 ;
 TOINS      = '                ;
 OPTION  = 'DISPLAY';
 PARM       = FILE||DOMAIN||CLASS||OPTION||FROMIN||TOINS;
 SAY 'PARM STRING IS 'PARM;
```

```
ADDRESS EDMLINK EDMMDINS PARM;
```

MVS User's Note

ADDRESS must be replaced with **CALL**, as in:

```
CALL EDMLINK EDMMDINS PARM
```

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

EDMMDOBJ

This method deletes an in-storage object.

EDMMDOB Parameters

object	The name of the in-storage object to be deleted.

Example

```
ADDRESS EDMLINK EDMMDOBJ 'ZTEST';
```

MVS User's Note

ADDRESS must be replaced with **CALL**, as in:

```
CALL EDMLINK EDMMDOBJ 'ZTEST'
```

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

EDMMDPRO

This method deletes a class in the PROFILE file of the Radia database.

EDMMDPRO Parameters

domain	The domain that contains the object to be deleted.
class	The class that contains the object to be deleted.

Example

```
EDMLINK EDMMDPRO 'TESTP1,TESTPROF'
```

MVS User's Note

ADDRESS must be replaced with **CALL**, as in:

```
CALL EDMLINK EDMMPPRO 'TESTP1,TESTPROF'
```

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

EDMMEXIS

This method verifies the existence of classes and instances in the Radia database.

EDMMEXIS Parameters

file	The file that contains the class or instance to be verified.
domain	The domain that contains the class or instance to be verified.
type	The type of file.
class	The class that contains the instance or class record to be verified.
instance	The instance to be verified.

Example

```
FILE = 'PRIMARY'
DOMAIN = 'SYSTEMX ' ;
CLASS = 'USER' ;
INST = 'USER1' ;
PARM = FILE  || '.' || DOMAIN || '.' || CLASS || '.' || INST
;

ADDRESS EDMLINK EDMMEXIS PARM ;

IF RC = 0 THEN

    SAY 'QAREXX ****** OBJECT ' INST ' EXISTS;
```

MVS User's Note

ADDRESS must be replaced with **CALL**, as in:

```
CALL EDMLINK EDMMEXIS PARM
```

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

EDMMGNUG

This method retrieves a list of local and global groups to which a specified user belongs.

This method is functional in a Windows network environment only.

Usage

This method is used to issue a function call to a specific server to collect information about the Windows group membership of a user. It does not provide authentication of a particular user ID. These calls are issued under security provisions of the user used to start the Configuration Server when it runs as a normal task; or under a system account when the Configuration Server runs as a service. See the Security Requirements below for security limitations as defined by Microsoft.

Security Requirements

Windows NT

No special group membership is required to successfully execute the EDMMGNUG method.

Windows 2000

If you invoke this function on a Windows 2000 domain controller that is running Active Directory, access is determined based on the access-control list (**ACL**) for the securable object. The default ACL permits all authenticated users and members of the "Pre-Windows 2000 compatible access" group to view the information. By default, the “Pre-Windows 2000 compatible access” group includes everyone as a member. This enables anonymous access to the information if the system allows anonymous access.

If you invoke this function on a Windows 2000 member server or workstation, all authenticated users can view the information. Anonymous access is also permitted if the Restrict Anonymous policy setting allows anonymous access.

For more information on restricting anonymous access, go to the following web site:
http://msdn.microsoft.com/library/psdk/network/ntlmapi_13zn.htm.

Method Input Parameters

The only parameter passed to the method is the name of the object containing the request. The following table details the input parameters and defaults for the method.

ZUSERID	Required variable used as user name.
NTSRVNAM	Name of the remote server on which the function is to execute. If this parameter is NULL, the local computer is used. The default is the local server.
ZOBJREQ	Name of the response object that will contain information about local and global (network) groups to which the user specified in ZUSERID belongs. The default is **NTGROUPS**.

Method Return Values

The EDMMGNUG method returns group membership information about a user in the object specified in ZOBJREQ (usually NTGROUPS). The following are the variables delivered by the method.

NTGRP**L**CT	Number of local groups on the server to which the user belongs.
NTGRP**G**CT	Number of global (network) groups on the server to which the user belongs.
NTGRP**S**CT	Total number of groups on the server to which the user belongs. (NTGRPLCT + NTGRPGCT)
NTGRP**L**xx	There are as many of these variables as there are local groups that a user belongs to on the specified server. xx = {1, NTGRPLCT}
NTGRP**G**xx	There are as many of these variables as there are global groups that a user belongs to on the specified server. xx = {1, NTGRPGCT }
MSGGRP**L**E	An error message, returned by the Network Management Functions, for request for the local group list to which the user belongs.

MSGGRPGE	An error message, returned by the Network Management Functions, for request for the global group list to which the user belongs.
NTUSER	Name of the user for which the function was executed. (This is the same as ZUSERID)
NTSRVNAM	Name of the remote server on which the function was executed.
ZMRC	Return code (set in in-bound and response objects).

EDMMGPRO

This method accesses the PROFILE file of the Radia database, gets the *dbobject* object and puts it in storage creating an in-storage object, *inobject*. The domain in the PROFILE database is the USERID, ZUSERID, which is found in the current object or in the ZMASTER object. If ZUSERID is not found, the method will return an error message, “Profile error: user ID not found” in the log.

EDMMGPRO Parameters

dbobject	The PROFILE file object name.
inobject	The name of the in-storage object to be created.
domain name	The name of the domain in the PROFILE file (usually the ZUSERID of the ZMASTER object).
instance	The name of the instance. This parameter is *optional*.

Example

```
/*************** REXX ******************************/
    PARM='ZSTATUS,ZSTATUS,'ZMASTER.ZUSERID;
    ADDRESS EDMLINK EDMMGPRO PARM;
/* GET OLD PROFILE.?.ZSTATUS                      */
```

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

EDMMNFYT

This method enables you to initiate a PUSH notification (the execution of a program or programs on a client desktop from another location). To execute notify successfully, EDMEXECD must be running on the clients on which you are executing a PUSH.

EDMMNFY Parameters

"process to run"	The application that you are forcing the client desktop to execute.
domain	The name of the domain where the notification results are stored. If not specified, this will be automatically generated as a function of date/time.
instance	The instance name containing the results of the single notification. If not specified, this will be automatically generated as an eight-digit number. The default is 00000001. Note: Only numeric names 00000000 to 99999999 are valid. Specifying anything else will result in 00000000 replacing the erroneous name.
password	The ZNFYPWD for the target terminal's ZMASTER object.
port	The port on which the client notify daemon is listening. This should be the same as the client's ZMASTER.ZNTFPORT port number.
target IP address	The IP address of the client desktop on which you are executing a PUSH.
user ID	The client user ID.

If the **domain** and **instance** parameters are omitted, the resulting instance will be written as:

```
NOTIFY.mmddyyhhmmss.NOTIFY.00000001
```

This does not guarantee the uniqueness of the domain name. In addition, the instance name does not represent anything significant, other than sequence.

Example

```
/*trace i*/
RC           = EDMGET("EDMMNFYT",0)
NHEAPS          = EDMMNFYT
DO CURRHEAP     = 1 TO NHEAPS BY 1
   RC        = EDMGET("EDMMNFYT",CURRHEAP)
   NIPADDR      = EDMMNFYT.IPADDR
   NPORT     = EDMMNFYT.PORT
   NUSER     = EDMMNFYT.USER
   NPASSW       = EDMMNFYT.PASSW
   NCMDLINE = strip(EDMMNFYT.CMDLINE)
   NHANDLE      = EDMMNFYT.HANDLE
   /*** CALL ZNFYT TO ISSUE THE NOTIFY ***/
   ADDRESS EDMLINK "EDMMNFYT" NIPADDR || ',' || NPORT || ','
|| NUSER || ',' || NPASSW || ',"' || NCMDLINE || '"'|| ','
NHANDLE || ',' || CURRHEAP;
END
```

MVS User's Note

ADDRESS must be replaced with **CALL**, as in:

```
CALL EDMLINK "EDMMNFYT" NIPADDR || ',' || NPORT ||
',' || NUSER || ',' || NPASSW || ',"' || NCMDLINE ||
'"'|| ',' NHANDLE || ',' || CURRHEAP;EDMLINK
"EDMMNFYT" NIPADDR
```

Configure NFYTTST on the console.

UNIX User's Note

To ensure that the UNIX process was started, type the following command at the UNIX prompt:

```
ps -u [username]
```

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

EDMMOLOG

This method writes the contents of an in-storage object to the Configuration Server log.

EDMMOLOG will write to the Configuration Server log even if all other TRACE settings are OFF.

EDMMOLOG Parameter

object	The name of the in-storage object to be displayed.

Example

```
ADDRESS EDMLINK EDMMOLOG 'ZMASTER';
```

MVS User's Note

ADDRESS must be replaced with **CALL**, as in:

```
CALL EDMLINK EDMMOLOG 'ZMASTER'
```

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

EDMMPHIS

This method puts an in-storage object into the HISTORY file. It takes the *inobject* that is in storage and puts it in the HISTORY file as *dbobject*. The domain used in the HISTORY file is the DATE/TIME stamp found in current object, or in ZMASTER object.

EDMMPHIS Parameters

dbobject	The object name that will be put in the HISTORY file. This parameter is optional.
inobject	The object name of the in-storage object.

Example

```
ADDRESS EDMLINK EDMMPHIS 'ZCOMPARE,ZCOMPARE';
```

```
ADDRESS EDMLINK EDMMPHIS 'ZSTATUS';
```

MVS User's Note

ADDRESS must be replaced with **CALL**, as in:

```
CALL EDMLINK EDMMPHIS 'ZCOMPARE,ZCOMPARE'
```

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

EDMMPPRO

This method puts an in-storage object (inobject) into the PROFILE file of the Radia database, as dbobject. The domain in the PROFILE file is the ZUSERID found in the inobject, or in the ZMASTER object. If ZUSERID is not found, the dbobject is put in the domain, _UNKNOWN.

The EDMMPPRO method allows you to specify multiple objects simultaneously.

EDMMPPRO Parameters

dbobject	The object name that will be put into the PROFILE database. The default is the object name.
domainid	The value of an optional third operand can be used to specify a domain other than ZUSERID or to eliminate the search for a ZUSERID value.
inobject	The name of the in-storage object.

Example

```
/************** REXX ***********************************/
ADDRESS EDMLINK EDMMPPRO 'OBJECTS=ZMASTER,ZCONFIG,ZUSERID'
```

MVS User's Note

ADDRESS must be replaced with **CALL**, as in:

```
CALL EDMLINK EDMMPPRO
'OBJECTS=ZMASTER,ZCONFIG,ZUSERID'
```

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

EDMMPROM

This method allows the addition and updating of instances in the PRIMARY database, based on the contents of the parameter object that was passed. Each heap in the object specifies an instance to be added or updated.

EDMMPROM requires that spaces be entered after commas in order to blank out existing services in fields on the Radia database.

EDMMPROM Parameter

object	The name of the in-storage object.

Example

```
EDMLINK EDMMPROM 'ZANYOBJECT'
```

MVS User's Note

ADDRESS must be replaced with **CALL**, as in:

```
CALL EDMLINK EDMMPROM 'ZANYOBJECT'
```

EDMMPUSH

This method receives input requests, gets the required parameters, and then puts the requests to the right queues for processing by a specific Notify Manager. An in-bound object, or even a dynamic object, created because of the object resolution can be used to deliver requests to EDMMPUSH. The return code associated with the in-bound object might initiate further action. (See Chapter 4, Notifying Clients for more information.)

EDMMPUSH Parameters

nfydelay	The interval for delay before trying to re-notify a client. The default is the value specified in the NFYT_TIMEOUT setting of the MGR_NOTIFY section of the `edmprof` file.
nfyhndl	The domain name of the NOTIFY file where the results of notifications will be stored. The heap number of the request object will become the instance name.
nfymrtry	The maximum number of retries. The default is the value specified in the NFY_RETRY setting of the MGR_NOTIFY section of the `edmprof` file.
ntfyrtim	HP timestamp defining the time after which the notification should occur.
nfyproc	Controls the processing of the current heap request. If Y, the heap will be processed. If N, the request for the current heap is ignored. The default is Y.
nfytype	Defines the type of the notify requested. Valid values are: TCP and EMAIL. Note: Only the first three bytes of the type are used for the identification. Therefore, EMAIL and EMA will be treated as the same. There is no default for this variable. If it is not defined, the current heap of the object will be ignored.
nfyuinfo	Allows you to enter user information.

Example

```
EDMLINK EDMMPUSH ZNOTIFY
```

MVS User's Note

ADDRESS must be replaced with **CALL**, as in:

```
CALL EDMLINK EDMMPUSH ZNOTIFY
```

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

EDMMPUTD (EDM only)

This method is called when EDMSENDF is used to send the ZTRANSF object to the Configuration Server. It closes a security loophole. EDMSENDF will not work with the version 4.4 Configuration Server, therefore, it is necessary to modify your Radia database by adding a new instance in ZPROCESS and linking it to a ZMETHOD object that invokes the EDMMPUTD method. This will have to be done for each object that is sent to the Configuration Server using EDMSENDF.

EDMMPUTD allows you to receive multiple data types sent by the Radia Inventory Manager, and enables you to specify where on the Configuration Server to store this data.

- With EDMMPUTD, you can inject a REXX method before EDMMPUTD gets dispatched to alter the location of the data based on appropriate criteria or to do security validation.
- The EDMMPUTD method handles the object and data sent from the client's EDMSENDF method. Thus, the specifications for the inbound object are the same.
- PROCESS class instances must be configured for each inbound object needing EDMMPUTD in order to receive the appended inbound data and store it in a file.

The following table lists the attributes that EDMMPUTD expects:

Table 131: Attributes Associated with EDMMPUTD

ZRSCMFIL	ZRSCMLOC	ZRSCMMEM
ZRSCRASH	ZRSCSTYP	ZOBJCLAS
ZOBJDOMN	ZOBJFILE	ZOBJID
ZOBJNAME	ZPERUID	ZPERGID
ZEDMTYPE		

The ZRSCDATE and ZRSCTIME fields are not referenced, nor are they used to update the date/time of the file received. ZRSCSIZE and ZCMPSIZE are ignored also. However, these attributes might be used in the future.

EDMMRESO

This method resolves the Radia database instance specified by the parameter string, and the resulting objects are left in storage. Any prerequisite objects needed for a successful resolution must already be built and in storage. For example, to resolve:

USER.&ZUSERID

an object containing the ZUSERID variable might have to be constructed. Otherwise, the resolution might not be completely successful.

EDMMRESO Parameters

file	The file that contains the instance to resolve.
domain	The domain that contains the instance to resolve.
class	The class that contains the instance to resolve.
instance	The instance to resolve.
message	This specifies the message for conditional resolution paths.

Example

```
ADDRESS EDMLINK EDMMRESO PRIMARY,SYSTEMX,USER,USER1,EDMSETUP
```

MVS User's Note

ADDRESS must be replaced with **CALL**, as in:

```
CALL EDMLINK EDMMRESO
PRIMARY,SYSTEMX,USER,USER1,EDMSETUP
```

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

EDMMRPRO

This method allows the adding, updating, and deleting of heaps in the PRIMARY database based on the contents of the parameter object that is passed. Each heap in the object specifies an instance to be added, updated, or deleted.

EDMMRPRO Parameter

object	The name of the in-storage object. The object can have multiple heaps where each heap in the object represents a Radia database instance to be altered.

Example

```
EDMLINK EDMMRPRO 'ANYOBJECT'
```

MVS User's Note

ADDRESS must be replaced with **CALL**, as in:

```
CALL EDMLINK EDMMRPRO 'ANYOBJECT'
```

Return Codes

0	The method was successful.
>0	An error was detected, the method failed.

The instance indicates which Radia database instance will be altered by specifying five control variables:

ZOBJCLAS	Target class, for example, ZRSOURCE or ZSERVICE.
ZOBJDOMN	Target domain, for example, SYSTEMX.
ZOBJFILE	Target file, for example, PRIMARY.
ZOBJNAME	Target instance. This can be any valid instance name.
ZOBJFILE	Target file, for example, PRIMARY.

- On a **delete** request, only the control variables are used to identify the instance to be deleted, the remaining variables are ignored.
- On **add** and **update** requests, the variables in each instance contain the values used to update the instance in the Radia database.

- The fields that can be updated are variables, class connections, expressions, and methods.
- There are specification differences for the three field types.

  However, regardless of field type, the target instances' field lengths determine the amount of data moved, and length adjustment is performed, including blank padding and truncation.

Variables

Any variable name found in the parameter object and found in the target instance will be used to update the target instance. Any variable not found in the target instance will be ignored.

Method Fields

Method fields found in the parameter object and found in the target instance will be used to update the target instance. Any method not found in the target instance will be ignored.

The string before the equals sign (**=**) must be eight characters long.

The methods are indicated by variables in the parameter object that are named MTHD*nnnn*, where *nnnn* is 0001 to 9999. The value of the variable in the object should contain

```
methodfieldname=xxxxxxxxxx
```

where *methodfieldname* is used to identify the target method field. For example:

```
_ALWAYS_=ZSYSTEM.ZMETHOD.SIGNON_METHOD
```

OR

```
EDMSETUP=ZSYSTEM.ZMETHOD.CHECK_APPL_STATUS.***
```

Connection Fields

Class fields found in the parameter object and found in the target instance will be used to update the target instance. Any variable not found in the target instance will be ignored. The connections are indicated by variables in the parameter object that are named CONN*nnnn*, where *nnnn* is 0001 to 9999. Types of connection fields include CONN*nnnn* (connection), INCL*nnnn* (Includes), _ALW*nnnn* (Always), and REQU*nnnn* (Requires). The value of the variable in the object should contain

```
connectionfieldname=xxxxxxxxxx
```

where `connectionfieldname` is used to identify the target class field. For example,

`_ALWAYS_=ZSYSTEM.ZSERVICE.MY_SERVICE`.

These variable names are the same format as object resolution with a message type = _NONE_. This is designed to allow for the output of these resolutions (sometimes referred to as reporting resolutions) to be used unchanged as input to EDMMRPRO.

You can update specific target instances while not overwriting some existing values (for example, EDMSETUP=) via EDMMRPRO in one of two ways:

- Each class named by the CONN*nnnn*, the value of the variable in the object *connectionfieldname*=*xxxxxxxx*, needs to be specified even if it is not the target of change and will be updated with the specified content. Each CONNnnnn needs to be specified for each variable in the sequence to provide a placeholder for updating the variables. For example,

  CONN0001   EDMSETUP=COUNTRY.USA_EAST_COAST

  CONN0002   EDMSETUP=ZSERVICE.XYZ

  CONN0003   EDMSETUP=ZSERVICE.ABC

- Another way of updating specific target instances while not overwriting some existing values is to specify a CSV (comma-separated variable) string for the instance. Empty values specified before a comma indicates that the connection should skip over the existing value and not update it. For example, the format for skipping over the first two variables and updating the third would appear as:

  CONN0001   “EDMSETUP=, ,ZSERVICE.ABC”

Notes on EDMMRPRO Usage

- The file, domain, and class must already exist; EDMMRPRO will not add any of these levels dynamically. In addition, any fields being processed must already be defined in the target class; EDMMPROM will not modify classes.
- Different class instances can be altered during one execution of EDMMRPRO. However, it is not advisable to do so as certain field names might overlap (particularly methods and connections).
- Different databases cannot be altered in one execution of EDMMRPRO.

EDMMSORT

This method sorts the heaps of an in-storage object by the values of specified variables and according to the desired collating sequence.

EDMMSORT Parameters

sort sequence	Ascending (SORT) or descending (SORTD).
object	The name of the object to be sorted.
variable	Up to three variable names can be specified.

Example

```
PARM1 = 'SORT,'
PARM2 = 'ZSERVICE'
PARM3 = ',ZOBJNAME,ZOBJDATE,ZOBJTIME ';
PARM  = PARM1 || PARM2 || PARM3;
ADDRESS EDMLINK EDMMSORT PARM;
```

MVS User's Note

ADDRESS must be replaced with **CALL**, as in:

```
CALL EDMLINK EDMMSORT PARM
```

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

EDMMSQLG

For a detailed description of this method, including usage, see Chapter 5, HP SQL Methods.

EDMMSQLP

For a detailed description of this method, including usage, see Chapter 5, HP SQL Methods.

EDMMTUCH

This method updates the date/time stamp of an instance in the Radia database.

EDMMTUCH Parameters

class	The name of the class that contains the instance to be updated.
domain	The name of the domain that contains the instance to be updated.
instance	The name of the instance to be updated.

Example

```
DOMAIN   = "SYSTEMX";
CLASS = "ZRSOURCE";
INST  = "TEST_OBJECT";
PARM  = SUBSTR(DOMAIN,1,8)||SUBSTR
(CLASS,1,8)||SUBSTR(INST,1,32);
```

```
ADDRESS EDMLINK EDMMTUCH PARM ;
```

MVS User's Note

ADDRESS must be replaced with **CALL**, as in:

```
CALL EDMLINK EDMMTUCH PARM
```

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

EDMMULOG

This method writes a message returned from the execution of a REXX method to a user log file. For this method to work, the MGR_USERLOG section must be added to the `edmprof` file, and ACTIVATE= must be YES.

EDMMULOG Parameter

msg	The message that will be written to the user log file. This message is returned by a method after its execution.

Example

```
ADDRESS EDMLINK "EDMMULOG" MSG
```

MVS User's Note

ADDRESS must be replaced with **CALL**, as in:

```
CALL EDMLINK "EDMMULOG" MSG
```

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

Configuration Server EDMPROF File Sample

```
[MGR_USERLOG]
ACTIVATE     = YES
COLUMN_WIDTH = 128
DIRECTORY    =
FLUSH_SIZE      = 128
PIPE_SIZE    = 100000
THRESHOLD    = 500000
```

EDMMVDEL

This method deletes all in-storage objects. There are no parameters associated with EDMMVDEL.

Example

```
ADDRESS EDMLINK "EDMMVDEL"
```

MVS User’s Note

ADDRESS must be replaced with **CALL**, as in:

```
CALL EDMLINK "EDMMVDEL"
```

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

EDMMVGBL

This method migrates values from one in-storage object to another, and then deletes the source object.

EDMMVGBL Parameters

destination	Object to which the values are being migrated.
source	Object from which the values are being migrated.

Only variables with appropriate flag settings will be migrated.

Example

```
EDMLINK EDMMVGBL 'TESTSORT,TESTVGBL'
```

MVS User's Note

ADDRESS must be replaced with **CALL**, as in:

```
CALL EDMLINK EDMMVGBL 'TESTSORT,TESTVGBL'"
```

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

EDMMXREF

EDMMXREF cross-references class and instance usage during object resolution, and collects information on the cross-referenced objects. It will generate objects that enable administrators to cross-reference users with any, and all, Departments, Workgroups, and Services with which they are affiliated (connected).

To implement the EDMMXREF method

1. Add new method instances to the METHOD class. Some are methods to create the objects from the PRIMARY database and others are to write these objects to the PROFILE database.
2. Update the Configuration Server process (ZMASTER) used during the client-connect process by adding new methods to SYSTEM.PROCESS.ZMASTER.
3. Update your class templates, if necessary, to add new method attributes.
4. Update the _BASE_INSTANCE_ to specify the method instances to be executed.

EDMMXREF Parameter

object	The name of the object containing the cross-referenced information.

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

EDMSIGN

This method enables the Configuration Server to authenticate Radia client and Radia administrator sessions against the Radia database. The password that is stored in the ZPWD variable in the specified object on the client/administrator is compared to that which is stored in the user's profile in the Radia database.

- If the passwords match, the session continues.
- If the passwords do not match, the message "PASSWORD INVALID" is returned.

Changing Passwords

Passwords can be changed by either of the following methods.

- In the HP OpenView Client Explorer Using Radia (Client Explorer): specifying a new password in ZNEWPWD and the old password in ZPWD.
- On the System Explorer and HP OpenView Publisher Using Radia (Publisher) login panels: selecting the **Change Password** option, and specifying the password information.

EDMSIGN Parameter

object	Specifies the name of the object from which the ZPWD variable is extracted. If no object name is specified, the ZMASTER object is used by default.

Example

```
REXX

EDMSIGN

    (Uses the ZMASTER Object)

EDMSIGN & (ZCURRENT>ZCUROBJ)
```

Return Codes

0	The method was successful.
4	New user.
8	An error was detected, the method failed.

EDMSIGNR

This method enables the Configuration Server to authenticate sessions with Radia client and Radia administrator against an external security system such as RACF, Top Secret, and Windows security. The password that is stored in the ZPWD variable in the specified object on the client/admin. is compared to that which is stored in the native security system for that user ID.

- If the passwords match, the session continues.
- If the passwords do not match, the message “PASSWORD INVALID” is returned.

MVS User’s Note

If this method is used, the Configuration Server **load library** must be APF authorized. If it is not, the MVS system will generate an “S683 abend” at the first client-connect attempt.

Changing Passwords

Passwords can be changed by either of the following methods.

- In the Client Explorer: specifying a new password in ZNEWPWD and the old password in ZPWD.
- On the System Explorer and Publisher login panels: selecting the **Change Password** option, and specifying the password information.

EDMSIGNR Parameter

object	Specifies the name of the object from which the ZPWD variable is extracted. If no object name is specified, the ZMASTER object is used by default.

Example

```
/******************REXX***************************/
EDMSIGNR
   (Uses the ZMASTER Object)
EDMSIGNR & (ZCURRENT>ZCUROBJ)
```

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

ZUPDPROF

This method only updates profile information. It does not perform any type of deletion.

- If toobject is not specified, the fromobject (source) name will be used.
- If user ID is not specified, the current user ID will be used.

ZUPDPROF Parameters

fromobject	The source object name.
toobject	The destination object name.
user ID	The user ID.

Example

```
ADDRESS ZUPDPROF OBJECT1,OBJECT
```

MVS User's Note

ADDRESS must be replaced with **CALL**, as in:

```
CALL ZUPDPROF OBJECT1,OBJECT
```

Return Codes

0	The method was successful.
8	An error was detected, the method failed.

B MVS Operations

MVS Considerations

This version of the Configuration Server for MVS does not contain all of the same functionality as its counterparts for other platforms. These differences are:

- The Configuration Server for MVS supports internal policy-management only (it doesn't integrate with Policy Server),
- The ZEDMAMS utility (Chapter 7) cannot be run as a Configuration Server method (only as a stand-alone operation), and
- The Configuration Server for MVS cannot be integrated with an ODBC database.

The Configuration Server, version 4.5.4 for MVS can, however, support Application Manager, Software Manager, and EDM clients.

MVS Commands to Modify a Configuration Server

This section describes the various commands that can modify the Configuration Server (versions 4.4.2 and 4.5.x) on MVS operating systems. These commands affect the Configuration Server processes listed in Table 132.

Table 132: Configuration Server Processes

Stopping	Switching
Flushing	Requesting Module Version Information
Queuing REXX Requests	Refreshing PARMLIB Options
Changing Storage-Pool Options	

Using the Commands

Table 133 details the commands that can be used to modify an MVS Configuration Server.

Considerations

- The format is the same for all the commands listed in Table 133.
- Some commands and sub-commands have a required minimum number of characters.
- Some commands and sub-commands can be abbreviated to their shortest, unique form.

Table 133: Configuration Server-MVS Commands

Command	**Description and Usage**
SHUTDOWN	Use this command to shut down the Configuration Server.
	Examples: **P** <*jobname*> **F** <*jobname*>**,SHUTDOWN** **F** <*jobname*>**,SHUT**
SWITCH	Use this command to switch between Configuration Server logs (for example, from EDMLOGA to EDMLOGB and vice versa). Note: This command causes the Configuration Server to switch from the DDNAME log it is currently logging to. This is useful for capturing logs during off-peak times, then saving them for subsequent referencing.
	Examples: **F** <*jobname*>**,SWITCH** **F** <*jobname*>**,SWI**
FLUSH	Use this command to flush the Configuration Server log. **Note**: This command causes all of the log messages that are still buffered in the Configuration Server to be written to the Configuration Server log. This is useful when working in real-time with the Configuration Server.

Command	Description and Usage

	Examples:
	`F <jobname>,FLUSH`
	`F <jobname>,FLU`

VERSION	Use this command to request that the Configuration Server module version information be printed in the log. Note: If SHOW_VERINFO=NO in the MGR_STARTUP section of PARMLIB, use this command to extract the Configuration Server version information before sending a log to HP Technical Support.
	Example: `F <jobname>,VER`
QUEUE	Use this command to queue a global REXX request. Note: This command passes the parameter string as a whole, and lets the REXX command do the parsing.

	Examples:
	`F <jobname>,QUEUE,REXX,<command>,<parms>`
	`F <jobname>,Q,R,<command>,<parms>`
	For clarification:
	`F <jobname>,Q,R,Audit,test21,ran ok`
	Note: In the above example, **Audit** is the name of the REXX command and **test21** and **ran ok** are the two parameters that are being passed to it. Regardless of how these parameters (**test21** and **ran ok**) are specified (as one, two, or three parameters) on the command line, they will be parsed according to how the REXX command parses parameters.

REFRESH	Use this command to refresh three of the sections that are set in the Configuration Server PARMLIB. Note: This command is applicable to the following sections only: **MGR_TRACE**, **MGR_MESSAGE_CONTROL**, and **MGR_POOLS**.

Command	Description and Usage

	Examples:
	`F <jobname>,REFRESH,<memname>`
	`F <jobname>,R,<memname>`
	where *memname* is a member in the PARMLIB with which the Configuration Server was started.
	Notes: The PARMLIB with which the Configuration Server was started is the dataset on the PARMLIB DD card in the Configuration Server JCL. The Configuration Server trace options, message filtering, and routing and pools options will be updated from any corresponding sections found in the member name specified on the modify command.
	To revert to the original startup options, refresh from the member with which the Configuration Server started.

POOLS	Use this command to change pool-storage options while the Configuration Server is running.
	Example: `F <jobname>,POOLS,<subcommand>,<parms>` The values for *subcommand* are: **STATS**, **DISABLE**, **ENABLE**, **CLEAR**, and **ADJUST**, with the following parameters:

STATS	No parameters.
DISABLE	Specify the pool value (for example, 2048) that is to be no longer used.
ENABLE	Specify the pool value that is to be re-enabled (for example, 2048, from above).
CLEAR	Specify a pool value whose counts are to be cleared.

Command	Description and Usage	
	ADJUST	Specify the number of elements in a pool and the waste-tolerated percentage. For example, assume a pool of 2048 with 60 elements and a waste tolerance of 95%. To increase the number of elements (to 80) and decrease the waste-tolerance percentage (to 85), specify: `F <jobname>,POOLS,adjust ,2048,80,85` Doing so would add 20 elements and decrease the waste-tolerance percentage by 10. Note: The new required totals must be specified—not the amount by which the parameter is to be increased or decreased.

The Radia Database's LICENSE File

This file is important in the control Radia product licensing by ensuring tighter control, while permitting a shared license pool among users in a distributed network (see License Reclamation, starting on page 67). Product licensing is controlled by information in the HP-issued license string.

The LICENSE File and the Configuration Server

On an MVS operating system, the LICENSE file must be added to the Radia database by an administrator. The Configuration Server will act on this file, but this action that is taken (for example to create new or update existing LICENSE file entries) depends on the type and level of the Radia client that is connecting. This activity has no impact on the Configuration Server's performance.

The Configuration Server will verify licensing- and report-warnings and errors in the Configuration Server log, and act upon them as necessary.

LICENSE File Considerations

- This file is a repository of information about the type and level of Radia clients that connect to the Configuration Server, and as such, its contents should not be deleted.
- This file should be the approximately the same size as the HISTORY and NOTIFY files.
- This file should be maintained with standard backup procedures.

Index