Radia Client Automation Enterprise Messaging Server

For the Windows® operating system

Software Version: 9.00

Reference Guide

Document Release Date: April 2013

Software Release Date: June 2013

Legal Notices

Warranty

Restricted Rights Legend

Copyright Notice

Trademark Notices

Acknowledgements

This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com).

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/).

This product includes software written by Tim Hudson (tjh@cryptsoft.com).

This product includes software written by Daniel Stenberg (daniel@haxx.se).

This product includes OVAL language maintained by The MITRE Corporation (oval@mitre.org).

Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, go to:

http://support.persistentsys.com/

This site requires that you register for a Persistent Passport and sign in. Register online at the above address.

For more details, contact your Persistent sales representative.

Support

Persistent 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 valued support customer, you can benefit by being able to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Submit enhancement requests online
- Download software patches
- Look up Persistent support contacts
- Enter into discussions with other software customers
- Research and register for software training

To access the Self-solve knowledge base, visit the Persistent Support home page.

Note: Most of the support areas require that you register as a Persistent Support user and sign in. Many also require an active support contract. More information about support access levels can be found on the Persistent Support site.

To register for a Persistent Support ID, go to: Persistent Support Registration.

Contents

Chapter 1

Introduction

About the Messaging Server

The Radia Client Automation Messaging Server (Messaging Server) is a robust messaging service that provides a means to forward data from one piece of the RCA infrastructure to another. It can be used as a point to aggregate different types of data as well as to segregate data accumulated from various RCA servers by type. Its job is to monitor predefined data queues and dynamically route data objects to one or more external destinations. The Messaging Server provides retry, rerouting, and failover capabilities to ensure all data is transferred efficiently and reliably.

On the RCA Configuration Server (Configuration Server), the Messaging Server operates hand-in-hand with the executable, `QMSG`, to handle the transfer of data reported from RCA agents to the appropriate ODBC reporting database or external RCA Server.

Features and Benefits

The Messaging Server provides an efficient and flexible messaging system that can be used by HPCA infrastructure modules. For example, it can:

- Route a single message to multiple destinations.
- Automatically retry a message delivery.
- Re-route messages to a new host after an unsuccessful delivery attempt (failover capability).
- Route data from one Messaging Server to another one (store and forward capability).
- Maintain multiple data queues on Store and Forward Messaging Servers.

The Messaging Server provides the following benefits:

- Multiple, specialized queues that enable separate workers to operate on each queue, and therefore allows for parallel processing of object messages.
- Additional workers can be configured to drain a queue more quickly.
- Independent data delivery agents allow for modular upgrades.
- Eliminates bottlenecks on the Configuration Server.
- Reliability of processing Inventory and Patch data is maintained, due to:
  - Built-in retry capability.
  - Ability to reroute messages remaining in a queue to a failover location.
  - Retry, holding, and re-routing features eliminate potential loss of data caused by network failures.
- Improved memory control related to creating the Inventory Manager SQL tables and loading the SQL commands upon startup. As a result, these SQL tasks are always performed upon

Messaging Server and Data Delivery Agent startup, and support for the previous STARTUPLOAD parameter has been removed.

- Improved queue control and throttling capability for posting RCA Portal data.
- New support for customized message routing to multiple DSNs using ODBC.

Messaging Server Processing on the Configuration Server

The Messaging Server acts as a delivery service between the Configuration Server and external ODBC databases or HPCA services. It is a separate component from the Configuration Server.

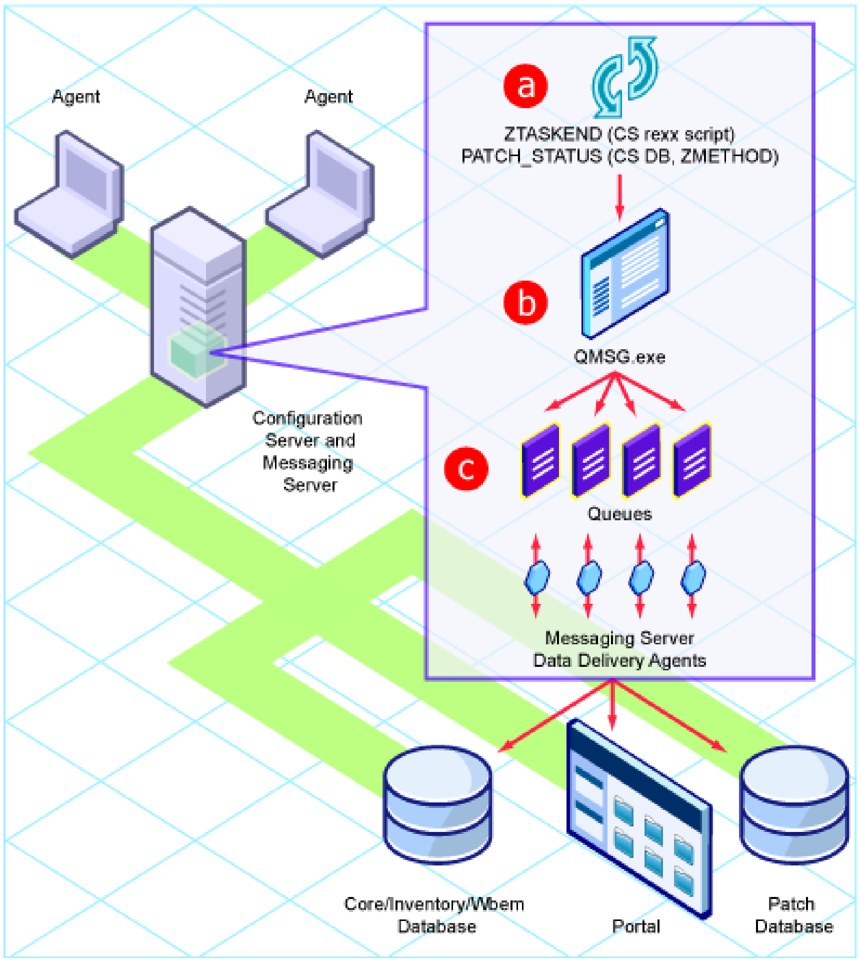
When a customer has multiple Configuration Servers, each one will have a co-located Messaging Server and the ability to route object data to the appropriate target location.

The figure "Messaging Server ProcessingLegend" on next page provides an overview of Messaging Server Processing.

The RCA agent connects to the Configuration Server to resolve its desired state. At the end of each agent connection, the agent passes objects back to the Configuration Server. Different agent objects are passed according to each specific phase of the agent connect process.

The Messaging Server refers to CORE objects as the standard RCA agent objects that are exchanged between the agent and the Configuration Server. Examples of CORE objects are ZMASTER, PREFACE, and SESSION. Other types of objects that can be exchanged are multi-heap WBEM reporting objects, FILEPOST objects created by an inventory agent audit process (called INVENTORY objects) and the DEERROR, BUSTATUS, DESTATUS, and PASTATUS agent objects collected for RCA Patch Manager processing.

Messaging Server Processing


Agent
Agent
a
ZTASKEND (CS rexx script)
PATCH_STATUS (CS DB, ZMETHOD)
b
QMSG.exe
Configuration Server and Messaging Server
c
Queues
Messaging Server Data Delivery Agents
Core/Inventory/Wbem Database
Portal
Patch Database

Legend

a - ZTASKEND is called

b - QMSG.EXE assembles object data

c - Messaging Server Data Delivery Agents poll the queues and transfer data

ZTASKEND is Called
On the Configuration Server, the ZTASKEND rexx method is called at the end of the agent connect process. ZTASKEND creates the commands to invoke the QMSG executable. The commands to QMSG contain parameters that define the appropriate objects to send as well as the designated queues in which to place the objects. ZTASKEND is responsible for naming all objects forwarded to QMSG—with the exception of objects needed for Patch reporting. For Patch objects, five methods in the PRIMARY.SYSTEM.ZMETHOD class of the Configuration Server Database call QMSG for the PATCH objects.

QMSG.EXE Assemble Object Data
QMSG assembles object data into XML files and creates header files with the appropriate metadata “address” needed to deliver the message by the Messaging Server. When the two message files (XML and meta data files) are created, QMSG places these files in one or more predefined data queues on the Configuration Server.

Note: `QMSG` places objects in separate queues, based on the parameters specified in ZTASKEND and in the five PATCH_* methods (DEERROR, BUSTATUS, DESTATUS, and PASTATUS). The Messaging Server is configured to use individual Data Delivery Agents (DDAs) to process messages from these queues. The table "Data Queues and Data Delivery Agents" lists the Data Delivery Agents that operate on each data queue location.

Messaging Server Data Delivery Agents poll the queues and transfer data
The Messaging Server DDA's poll the message queues and automatically pick up and transfer data files to the appropriate external locations using the routing type and locations defined in the specific data delivery agent's configuration file.

Data Queues and Data Delivery Agents

Data Queue	Data Delivery Agent
`..\Data\MessagingServer\core`	`core.dda`
`..\Data\MessagingServer\patch`	`patch.dda`
`..\Data\MessagingServer\usage`	`usage.dda`

The Messaging Server runs on all Windows platforms supported by the Configuration Server.

About the Data Delivery Agents

The Data Delivery Agents are function-specific modules created to transfer certain types of message data. There are Data Deliver Agents available for CORE, PATCH, and USAGE message data.

- The CORE message data refer to agent objects typically exchanged during a standard agent connect process. The CORE message data also includes the FILEPOST objects created during the agent audit process and the WBEM reporting object data. Examples of CORE type message objects include ZMASTER, SESSION, and ZCONFIG. CORE objects support reports for Vulnerability Management, Windows CE Thin Clients and Application Management Profiles, among others.
- The PATCH message data is comprised of `DESTATUS`, `BUSTATUS`, `DEERROR` and `PASTATUS` agent object data.

The Data Delivery Agents (DDAs) have the ability to post messages using ODBC into a SQL database that can be used for reporting. For processing efficiency, each DDA works independently on its own queue.

The Messaging Server loads these independent data delivery agents, whose configurations define how and where the messages for CORE, INVENTORY, WBEM, and PATCH data objects will be delivered.

Note: The CORE data delivery agent is configured to post CORE object data to an Inventory Manager database, as well as CORE object data to a Portal directory.

The data delivery agent modules are located in the `\MessagingServer\modules` folder. The modules are loaded using "dda.module load" statements in the Messaging Server configuration file (`rms.cfg`).

Each data delivery agent is configured from its own configuration file (`*.dda.cfg`). These configuration files are located in the `\MessagingServer\etc` folder.

About Routing Inventory Data

This Messaging Server supports direct ODBC posting of Inventory Manager data to a back-end SQL Inventory Database. The CORE.DDA has the ability to route CORE, INVENTORY and WBEM data messages to a back-end SQL database using ODBC. It is recommended to use this routing option for best performance.

About the SQL Database Tables and Scripts for Inventory

The CORE Data Delivery Agent posts the message data into the SQL table, for which the default definitions and associated SQL queries are found in the following Messaging Server directories:

`/etc/`*`<module name>`*`/hp`

Customized scripts can be placed in the directories:

`/etc/`*`<module name>`*`/`

The script necessary to map the CORE object data to the related SQL table column is `taskend.tcl`. The script necessary to map the INVENTORY object data (FILEPOST object) is called `filepost.tcl`. Both of these scripts are found in the `/etc/`*`<module name>`*`/lib` directory of the Messaging Server.

All inventory and the application deployment related tables have a foreign key constraint with the table `DeviceConfig`. This foreign key constraint ensures that when a device is deleted, all the records corresponding to that device are deleted from all the other tables.

Creating SQL Tables for the Inventory Database

The SQL tasks to create the tables for the Inventory Database and load the SQL commands into memory are performed when the Messaging Server service and the Data Delivery Agents are initially started.

Note: All SQL tasks are performed at service startup and any STARTUPLOAD value found in a configuration file is ignored.

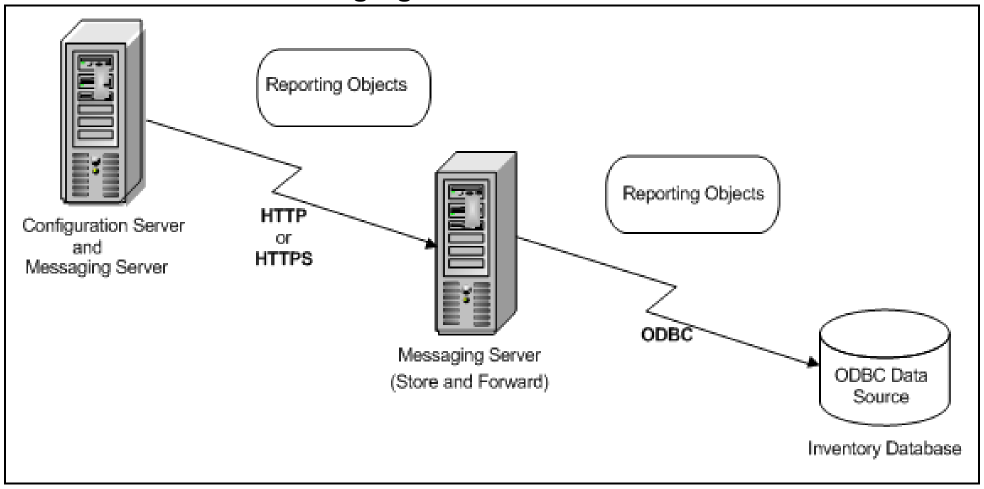
About Using Store and Forward

The Messaging Server includes store and forward capabilities that allow you to forward messages to another Messaging Server using HTTP or HTTPS. In addition, if the Messaging Server is not located close the SQL database, it is a good practice to forward the messages to one that is located as close to the SQL database as possible.

The Messaging Server supports forwarding and receiving messages using multiple queues.

The following figure Store and Forward Messaging Server illustrates a typical configuration that forwards messages to another Messaging Server, prior to posting data to the Inventory database using ODBC.

Store and Forward Messaging Server



1. The Messaging Server on the Configuration Server is configured to have the CORE data delivery agent forward the data to another Messaging Server using HTTP or HTTPS. This configuration makes use of the *coreforward*, *inventoryforward*, and *wbemforward* sections that are provided in the data delivery agent configuration files.
2. The Store and Forward Messaging Server is located close to the Inventory Manager SQL database. It receives the CORE, INVENTORY, and WBEM objects.
3. The attending CORE data delivery agent on the receiving Messaging Server posts the data objects to the Inventory Manager database using ODBC.

About this Guide

In addition to this chapter, this book contains the following information:

- **Configuring and Tuning the Messaging Server**
  This chapter describes how the Configuration Server ZTASKEND REXX and the QMSG executable work hand-in-hand with the Messaging Server. It also discusses how to configure the Messaging Server configuration file, which loads the Data Delivery Agents. In addition, this chapter also describes how to configure the DDA modules to route CORE, INVENTORY, WBEM, and PATCH message data to the Inventory, Portal, and Patch Manager databases or directories. Additional tuning options are included.
- **Additional Messaging Server Configuring Options**
  This chapter describes alternate configurations, including how to install and configure for Store and Forward, and other customized configurations.

Abbreviations and Variables

Abbreviations Used in this Guide

Abbreviation	Definition
RCA	Radia Client Automation
Core and Satellite	RCA Enterprise environment consisting of one Core server and one or more Satellite servers.
CSDB	Configuration Server Database

Variables Used in this Guide

Variable	Description	Default Values
`InstallDir`	Location where the RCA server is installed	For a 32-bit OS: `C:\Program Files\Hewlett-Packard\HPCA` For a 64-bit OS: `C:\Program Files(x86)\Hewlett-Packard\HPCA`
`SystemDrive`	Drive label for the drive where the RCA server is installed	C:

Summary

- The Messaging Server routes the object data collected from Client Automation Agents and placed into queues into the appropriate Client Automation server or SQL database. Messages can also be forwarded to another Messaging Server.
- Messages processed include agent objects collected for core, inventory, wbem and patch data.
- Configuration settings in the `rms.cfg` file allow you to load the Data Delivery Agents needed to process the queues on that server. There are separate Data Delivery Agents for core, inventory (filepost), wbem and patch data objects.
- Configuration settings in the `*.dda.cfg` files for the specific data delivery agents specify how and where to route the data processed by that data delivery agent.
- Additional tuning options address load balancing when processing high-volumes of data as well as large-sized objects.

Chapter 2

Configuring and Tuning the Messaging Server

Note: To configure the Usage Data Delivery Agent, see the *Radia Client Automation Enterprise Application Usage Manager Reference Guide*.

Understanding the Configuration Server Modules that Support the Messaging Server

This topic explains how the methods on the Configuration Server work hand-in-hand with the Messaging Server to collect, queue, and then deliver data to the appropriate external location.

Getting Agent Information to the Messaging Server

Agent objects exchanged with or created on the Configuration Server during an agent connect session are formatted into messages for the Messaging Server via the Configuration Server binary executable QMSG. The QMSG executable is part of the HPCA release. This executable is invoked during agent taskend processing by the rexx method ZTASKEND or a connection to the Patch Manager methods. The calls to QMSG can include parameters specifying what queue to place the messages in, the priority in which the message is to be processed, the objects that are to be included in the messages, and the “destination address” or routing identifier for the file.

The QMSG executable formats the object data into an XML file. Each XML file can be made up of multiple objects, such as when processing the CORE objects (for example, ZMASTER, SESSION and ZCONFIG) or it can be a single multi-heap object such as a wbem or filepost object. Each call to QMSG will produce two files, the XML file created from the object data and a file which contains the metadata. Metadata are attributes describing the XML file, how big it is, when it was created and the routing identifier. The actual file names are created with a timestamp format. This enables the Messaging Server to process the oldest messages first. The Messaging Server always processes in a “First In First Out” mode when the messages have the same processing priority.

When queue designations are specified when invoking QMSG, messages are placed in a directory with the specified queue name. When no queue identifier is used, messages are placed in a directory named default. Each data delivery agent uses its own unique queue for its particular messages. This segregation of messages according to type allows for the simultaneous processing of all queues and leads to more efficient operation.

In Summary:

- For agent information needed by the Patch Manager, `QMSG` is executed as a result of the `SYSTEM.ZMETHOD.PATCH*` instances: `PATCH_DEERROR`, `PATCH_BUSTATUS`, `PATCH_DESTATUS`, and `PATCH_PASTATUS`.

For more information on creating the method connection needed for Patch message processing, see the *Radia Client Automation Enterprise Patch Management Reference Guide*.

- For information needed by the Inventory Manager, Portal, and Application Management Profiles, the Configuration Server REXX method, ZTASKEND, is used to trigger the call to QMSG. The format of the QMSG call is included in the discussion of ZTASKEND which follows.

About the Patch Method for Collection

Five methods call the QMSG executable with the parameters in the ZMTHPRMS attribute of the method. The default value of this attribute looks like this:

```
ZMTHPRMS      -to PATCH5 -queue patch <<object-name>>
```

The `-to PATCH5` parameter specifies that the messages will be placed in a queue called `../data/patch` relative to the location of where QMSG is executed. This is the default location specified in the install of the `patch.dda`.

The <<*object-name*>> parameters (`DEERROR`, `BUSTATUS`, `DESTATUS`, and `PASTATUS`) specify the objects that will be included in the message files created by QMSG.

About the ZTASKEND REXX method

The ZTASKEND REXX method on the Configuration Server is called at the end of each agent connect, while objects associated with the present session are still available in storage on the Configuration Server. ZTASKEND invokes `QMGS` when agent data needs to be collected for another service, such as Inventory Manager, Application Management Profiles, or the Portal. QMSG collects the data and places the messages in specified queues for pickup and processing by the Messaging Server and its data delivery agents.

Note: Different object types (core, inventory and wbem objects) are placed in separate queues whenever the Data Delivery Agents have been installed for those data objects.

An important job of ZTASKEND is to ensure unique agent data is collected at the appropriate agent connect phase. For efficiency, ZTASKEND also groups identical messages, having the exact same object content, to minimize the number of calls made to QMSG.

This topic explains:

- How ZTASKEND determines when to call QMSG for the various agent connect phases and which objects are collected.
- The basic syntax of calls to QMSG.
- How the QMSG -to parameter establishes one or more destinations for message processing.
- How the QMSG -priority parameter establishes Messaging Server processing order.

ZTASKEND calls to QMSG

This topic reflects the ZTASKEND method.

Processing Phase-Dependent Objects

Each time an RCA agent connects to the Configuration Server, the agent declares its connection intent or phase. An important role of the ZTASKEND REXX code is to minimize the collection of duplicate information. With that goal in mind, the ZTASKEND method invokes QMSG depending on the agent connection phase and the objects present. ZTASKEND restricts message posting to five specific connection phases. The phases of interest are:

- `BOOTSTRAP (Client Operations Profiles or COP)`
- `AGENT SELF MAINTENANCE`
- `CATALOG RESOLUTION`
- `SERVICE RESOLUTION`
- `AGENT REPORTING`

Processing CORE Objects by Phase

There are "critical objects" collected for each of the core targets for Inventory Manager (RIM) and Portal (RMP). The potential critical objects are:

For RIM: `APPEVENT MSIEVENT SYNOPSIS RNPEVENT`

For RMP: `SYNOPSIS IDENTITY`

In addition to the above critical objects, the table "Critical Objects collected by phase" identifies critical objects collected for each phase.

Critical Objects collected by phase

Phase	Critical Objects
`BOOTSTRAP (COP)`	`SESSION PREFACE ZSTATUS SMINFO`
`AGENT SELFMAINTENANCE`	`SESSION PREFACE ZSTATUS SMINFO`
`CATALOG RESOLUTION`	`SESSION PREFACE ZSTATUS ZCONFIG ZMASTER SMINFO`
`SERVICE RESOLUTION`	`SESSION PREFACE ZSTATUS SMINFO`
`AGENT_REPORTING`	`SESSION PREFACE ZSTATUS SAPSTATS ZRSTATE SMINFO`

With this in mind, ZTASKEND uses the following logic:

1. Whenever a critical object is presented, the critical object and the additional objects are processed by QMSG and deposited into queues for processing by Messaging Server (RMS).
2. If a critical object is not present, then the code invokes QMSG when an agent connects during the phases CATALOG_RESO and AGENT_REPORTING.

3. If a critical object is not found, then code does not invoke QMSG for the following agent phases: BOOTSTRAP (COP), SERVICE RESOLUTION and AGENT_SELF MAINTAINANCE.
4. Finally, ZTASKEND does not invoke QMSG to obtain error message objects (ZERRORM & ZERROR).

The table "ZTASKEND calls to QMSG for CORE Data for RIM and RMP" summarizes the Agent Connect phases and the objects collected during the ZTASKEND calls to QMSG, for CORE data going to Inventory Manager or the Portal.

ZTASKEND calls to QMSG for CORE Data for RIM and RMP

Agent Connect Phase	QMSG call if not critical object?	QMSG call if critical object
Bootstrap - COP Resolution: for Client Operations Profile.	No	Collects these objects for CORE.RIM and CORE.RMP destinations: APPEVENT | MSIEVENT | SYNOPSIS | RNPEVENT SESSION PREFACE ZSTATUS SMINFO
Agent Maintenance:	No	Collects these objects for CORE.RIM and CORE.RMP destinations: APPEVENT | MSIEVENT | SYNOPSIS | RNPEVENT SESSION PREFACE ZSTATUS SMINFO
Catalog Resolution: Agent connects to the Configuration Server to obtain service resolution list.	**Always**. Collects these objects for CORE.RIM and CORE.RMP destinations: SESSION PREFACE ZCONFIG ZMASTER ZSTATUS SMINFO	See previous column, but also collects APPEVENT| MSIEVENT | SYNOPSIS | RNPEVENT.
Single Service Resolution: For each service to be resolved, agent makes another connection to the Configuration Server.	No	Collects the following objects for CORE.RIM and CORE.RMP destinations: APPEVENT | MSIEVENT | SYNOPSIS | RNPEVENT SESSION

Agent Connect Phase	QMSG call if not critical object?	QMSG call if critical object
		PREFACE ZSTATUS SMINFO
Agent Reporting: At the end of service resolution. Agent data is reported back to the Configuration Server.	**Always**. Collects these objects for CORE.RIM destination: SESSION PREFACE ZSTATUS SAPSTATS ZRSTATE SMINFO	See previous column, but also collects APPEVENT | MSIEVENT | SYNOPSIS | RNPEVENT

Adding Items to a Critical Object List

The ZTASKEND rexx code can be configured to add object names to the critical and additional object lists. The rexx variables `CriticalRIMObjects` and `CriticalRMPObjects` contain the object names for each of these targets. The rexx function call to `BuildObjectList` is used to build the object list for each phase.

```
Call BuildObjectList  ....
   :
   :
   :
   :
  CriticalRIMObjects    = "APPEVENT MSIEVENT SYNOPSIS RNPEVENT"
  CriticalRMPObjects    = "SYNOPSIS"
```

The ZTASKEND rexx code contains additional information on how to alter these items.

Processing Always Objects

There is a section of the ZTASKEND rexx code to define objects that will always be processed, independent of the phase being processed. The larger Inventory Manager objects, FILEPOST, WBEMAUDT and CLISTATS are processed this way. If these objects exist, then they are sent to the specified target. In addition to the larger Inventory Manager objects, the job objects: JOBSTAT, JOBPARM, and JOBTASK are also processed this way.

Adding Custom Objects to the BuildAlways Object List

This part of the rexx code can also be configured to add "custom" objects that are always delivered to the specified target. The rexx function `BuildAlways` is used to configure the target, queue and objects to process. The rexx code contains additional information on how to alter these items.

```
Call BuildAlways  "inventory",  InventoryQueue, "FILEPOST"
```

QMSG Method Syntax

The QMSG command/method is used to post Configuration Server objects for Inventory Manager, Portal and Application Management Profiles. QMSG reads the specified object and converts it to XML and then writes it to the specified queue.

The syntax of the QMSG method is given below:

```
qmsg -to <destination(s)> -queue <queue> -priority <priority> object1
object2 ... objectn
```

-to <destination(s)>
must be explicitly coded with one or more destinations, or targets. Messages going to multiple destinations have comma-separated entries. For example:

```
-to CORE.RIM,CORE.RMP
```

The Messaging Server destination values used by the delivered QMSG include:

```
-to CORE.ODBC
-to CORE.RMP
-to INVENTORY.ODBC
-to WBEM.ODBC
```

Each message destination requires an equivalent ROUTE defined for it in the msg::register router section of the appropriate Data Delivery Agent's `*.dda.cfg` file. The table "QMSG Destinations and DDA Configuration Locations" gives the destination values of QMSG and the configuration file and section used to define its ROUTE.

QMSG Destinations and DDA Configuration Locations

QMSG –to destination	Configuration File in *MessagingServer*`\etc` folder	Required ROUTE section
-to CORE.ODBC	`core.dda.cfg`	msg::register corerouter
-to CORE.RMP	`core.dda.cfg`	msg::register corerouter
-to INVENTORY.ODBC	`core.dda.cfg`	msg::register inventoryrouter
-to WBEM.ODBC	`core.dda.cfg`	msg::register wbemrouter

-queue <queue>
The queue is a directory relative to where `QMSG.EXE` exists. `QMSG` is located in the `\bin` directory of the Configuration Server. For example, if `QMSG` resides at

```
C:\Program Files\Hewlett-
Packard\HPCA\ConfigurationServer\bin\qmsg.exe
```

Then the queues would reside at:

```
C:\Program Files\Hewlett-Packard\HPCA\ConfigurationServer\data\blue
```

```
C:\Program Files\Hewlett-
Packard\HPCA\ConfigurationServer\data\default
```

```
C:\Program Files\Hewlett-Packard\HPCA\ConfigurationServer\data\green

C:\Program Files\Hewlett-Packard\HPCA\ConfigurationServer\data\red
```

The parent directory of all queues is "data". For illustrative purposes, this example shows the existence of the (fictitious) blue, green and red queues. Note that the queue named "default" is the default queue.

Note: Queue locations are defined near the top of ZTASKEND. For Patch routing, the queue location is named in the parameters passed to QMSG from the five `SYSTEM.ZMETHOD.PATCH_*` instances. The instances are named DEERROR, BUSTATUS, DESTATUS, and PASTATUS. For example:

```
Method = qmsg
Parameter = -to PATCH5 -queue patch <<object>>
```

To modify the parameters to pass, edit the value of the `ZMTHPRMS` attribute in the `PATCH_*` instance.

-priority
The priority is available to establish Messaging Server processing priority. If omitted, a default priority of 10 is given to the message. Valid values are 00 (highest priority) to 99 (lowest priority). For more information on Messaging Server processing priority, see the topic "How Priority Establishes Messaging Server Processing Order" below.

object1 [objectn]
The rest of the command line includes the names of the objects to queue, object1 object2... object*n*. The objects are processed in the order specified; thus, depending on the destination, there might be a dependent order.

How Priority Establishes Messaging Server Processing Order

When ZTASKEND calls QMSG, the optional **–priority** parameter in the call assigns a processing priority to the message. Priority values can range from 00 to 99, with 00 reserved for critical processing and 99 being the lowest priority.

For messages waiting to be processed in the same queue, the Messaging Server processes all messages assigned to a higher priority (such as 10) *before* processing any messages assigned to a lower priority (such as 20). Within a given priority, messages are processed using first in, first out (FIFO) order.

Note: The message priority remains the same for the life of the message. For example, if a message is forwarded from one Messaging Server to another, the message priority remains the same.

- Priority 10 is the default if a priority is not specified.
- Previously, ZTASKEND called QMSG with parameters to assign a lower priority of 20 to the larger objects collected for Inventory Manager Reports: these include file audits, wbem reporting data, and agent statistics (CLISTATS).
- This is no longer necessary because of the segregation of queues by object type.

If the Messaging Server is not able to process the messages as fast as they are delivered from QMSG, the lower priority messages will accumulate at the bottom of a queue location, even though newer messages with higher priorities are still being processed.

Configuring the Messaging Server

Use these topics to reconfigure or tune the Messaging Server after installation, or reconfigure or tune the data delivery agents for core or patch data.

Editing the Configuration Files for the Messaging Server and Data Delivery Agents

The Messaging Server and Data Delivery Agents standard installation allows configuration of several of the configuration parameters contained in the respective configuration files. You will need to edit the configuration file with a text editor to achieve a more customized environment.

All of the configuration files for the Messaging Server and Data Delivery Agents are found in the `\etc` directory of where the Messaging Server was installed.

All the configuration files for the Messaging Server and the associated Data Delivery Agents have similar configuration sections. Understanding these sections and the syntax used to configure them will aid in customizing your environment.

The structure for the RMS configuration file sections is given below:

```
msg::register <unique identifier> {
 TYPE        <RMS registered TYPE>
 <Configurable Variable for the registered TYPE>      <Value for that
Variable>
 <Configurable Variable for the registered TYPE>      <Value for that
Variable>
 }
```

Each configuration section starts with the command `msg::register`. This signals the start of a configuration parameter for the Messaging Server and its modules.

A unique identifier follows the msg::register command. Within an instance of the Messaging Server, which includes the configuration files for the Messaging Server and all of the DDA modules, this unique identifier label can only be used once. The unique identifier is followed by a curly brace "{". The configuration section for this TYPE must be ended by a closing curly brace for the entire configuration to work.

All configuration file sections have a TYPE identifier, which indicate the kind of work to be done by this section. These are the current acceptable TYPE designations:

- QUEUE
- ROUTER
- HTTP
- HTTPS
- HTTPD

- FILTER
- SQLBALANCER (only configurable in `core.dda.cfg`)

The table "Glossary of Section TYPEs and Configurable Parameters" describes the different section TYPES and their configurable parameters. The sections are listed in the general order in which they appear in the configuration files.

Glossary of Section TYPEs and Configurable Parameters

Section TYPE	Configurable Parameters
QUEUE Defines the directory where messages are placed for processing.	**DIR** The directory name of the queue. **USE** Specify the name of the unique identifier that will signify how to dispatch the message. Usually this is a ROUTER type. **POLL** By default, the Messaging Server is configured to poll the queue location every 10 seconds and post up to 100 objects at a time. To change the poll interval, modify the POLL parameter. **COUNT** Maximum number of objects to post at a time (during the polling interval defined by POLL). Default is 100 objects. **ATTEMPTS** Maximum number of attempts to retry a failed message delivery before discarding the message. Default is 200 attempts. (By default, the Messaging Server and Data Delivery Agents are configured to retry any failed posts every hour, and make up to 200 attempts.) **DELAY** Number of seconds to wait between attempts to retry a failed message delivery. Default is 3600 seconds, or one hour. **Note:** To calculate the maximum time that a message could stay in the queue before being discarded, take the DELAY time and multiply it by the ATTEMPTS value. Using the default settings, this is a DELAY of 3600 seconds x 200 ATTEMPTS, or approximately eight days.
ROUTER Defines where the messages are going to sent.	**ROUTE** Delimit each Route by curly braces **TO** This is the address of the message. It is contained in the

Section TYPE	Configurable Parameters
	meta data file of each message when the message is created. **USE** Specify the unique name of a Messaging Server TYPE that will be used to dispatch the message, such as HTTP or HTTPS. (In a DDA file, the USE entries may also be COREODBC, WBEMODBC and PATCHDDAODBC).
HTTP This is a way to post the message to another server location using http protocol. The target server can be another Messaging Server where the message can be re-queued or it can be another part of the infrastructure, such as the Portal.	**ADDRESS** Delimit each address using curly braces. Multiple URL destinations can be specified within each HTTP TYPE as long as each has its own ADDRESS label and a different priority. **PRI** Denotes the priority in which to sent messages to the companion URL. The default priority is 10. The priority setting only matters if multiple ADDRESS entries are configured. If multiple ADDRESS entries are configured the lowest priority is tried first and if that fails the next priority URL is tried. This allows failover capability if there are network problems. **URL** Specifies the URL to use to send the message.
HTTPS This is a way to post the message using a secure socket. The HTTPS parameters are configured the same as the HTTP parameters--with the exception of the URL specification. The URL uses the `https://` convention.	**ADDRESS** Same as TYPE of HTTP. **PRI** Same as TYPE of HTTP. **URL** Specifies the URL using the `https://` convention. For detailed information, see the *Radia Client Automation Enterprise SSL Implementation Guide*.
HTTPD Defines the parameters the Messaging Server will use to receive incoming messages.	**PORT** Defines the Port used to “listen” for messages. Only one port specification can be used for a given Messaging Server. **URLHANDLER** Delimit with curly braces. If the incoming messages are to be deposited into different queues, depending upon the URL, the URLHANDLER must be used to delimit the USE and URL parameters. **USE** Specify the name of the QUEUE type that will receive

Section TYPE	Configurable Parameters
	the incoming messages. **URL** Specify the URL prefix that that will be accepted by the Messaging Server. When messages are received with the designated URL they are deposited in the associated queue. The QUEUE type must be defined when messages are received. All URL's specified must start with **/proc/**.
FILTER A means to route PATCH data into multiple SQL tables.	**USE** Specify the name of the unique identifier that will signify how to dispatch the message. **TO** This is the address of the message. It is contained in the meta data file of each message when the message is created. **FILTER** Used for PATCH object processing into multiple tables
COREODBC Used to post CORE messages into a SQL database.	**DSN** Data Source Name **USER** User ID for the DSN **PASS** Password for the DSN **DSN_ATTEMPTS** Number of attempts to connect to the DSN. Default is 1. **DSN_DELAY** Delay in seconds between attempts to connect to the DSN. Default is 5 seconds. **DSN_PING** Delay in seconds between pinging the database connection to verify the DSN is available. Default is 300 seconds. **STARTUPLOAD** No longer supported; any existing value is ignored. **Note:** The SQL tasks to create the tables and load the scripts for the Inventory Database are always performed upon Messaging Server and Data Delivery Agent startup.
PATCHDDAODBC	**DSN**

Section TYPE	Configurable Parameters
Used to post PATCH data messages into a SQL or Oracle database. Only configurable in `patch.dda.cfg`	Data Source Name **USER** User ID for the DSN **PASS** Password for the DSN **DBTYPE** Database type of MSSQL (for Microsoft SQL Database) or ORACLE (for an Oracle Database).
SQL The SQL database connection that handles all operational and inventory related data flows.	**DSN** Data Source Name **SERVER** The name of the SQL server **USER** User ID for the DSN **PASS** Password for the DSN **AUTOCOMMIT** The transaction management mode for the SQL server that decides whether to commit or roll back the data. **DSN_DELAY**
	Delay in seconds between attempts to connect to the DSN. Default is 5 seconds. **DSN_PING** Delay in seconds between pinging the database connection to verify the DSN is available. Default is 300 seconds. **TRUNCATE** Truncates the data if it exceeds the column size. If disabled, the Messaging Server throws an error message. **DISABLE_TABLE_UPDATES** The Messaging Server disables the provision to update the specified tables in the database. You can specify the tables for which you want to disable the updates. For example, to disable the tables `HDeviceConfig` and

Section TYPE	Configurable Parameters
	`HAppEvent` from being updated, add the following entry in the `*.cfg` file: `DISABLE_TABLE_UPDATES {HDeviceConfig HAppEvent}` **ENABLE-CORE** Flag to enable the routing of CORE data objects placed in the queue locations for the core objects. **ENABLE-INVENTORY** Flag to enable routing of INVENTORY data objects placed in the queue locations for filepost (file audit inventory). **ENABLE-WBEM** Flag to enable routing of WBEM data objects placed in `\data\wbem` queue location. **ENABLE-VM** **AUTOCREATE** A switch to enable the creation of a new SQL file and table in the Inventory ODBC database when a new object class is received. Default is 0. 0 – Does not create a SQL file or table entry for a new object class. 1 – Creates a new SQL file and table entry for a new object class.
	AUTOLOAD Enables autoload of schemas **SQLPATH** The path where the schemas are stored. **MAPPATH** The path where the mappings (`*.tcl` files) are stored. **WBEM-PKEYS** Enables creation of WBEM tables on demand with primary keys instead of unique indexes. **ARCHIVE** Enables archiving of messages. If enabled, the messages are moved to the Archive location.

Section TYPE	Configurable Parameters
	REJECTS Moves the messages to the REJECTS handler. The messages that fail to get processed, after maximum number of attempts have been tried to deliver this message, are moved to the REJECTS handler. Maximum number of attempts to retry a failed message **VALIDATE_NLS_PARAMS** National Language Support (NLS) connection settings that can be set for Oracle.
SQLBALANCER Routes all messages for a given device to a single sub-queue based on the host name, with a single worker process processing the data in the queue.	**QUEUE_DIR** The directory where the queue is stored. For example, `<InstallDir>//Data/MessagingServer/core`. **QUEUE_POLL** By default, the Messaging Server is configured to poll the queue location every 1 second and post up to 100 objects at a time. To change the poll interval, modify the POLL parameter. **QUEUE_COUNT** Maximum number of objects to post at a time (during the polling interval defined by POLL). Default is 5000 objects. **QUEUE_DELAY** Number of seconds to wait between attempts to retry a failed message delivery. Default is 3600 seconds.
	QUEUE_ATTEMPTS Maximum number of attempts to retry a failed message delivery before discarding the message. Default is 200 attempts. (By default, the Messaging Server and Data Delivery Agents are configured to retry any failed posts every hour, and make up to 200 attempts.) **QUEUE_REJECTS** Moves the messages to the REJECTS handler. The messages that fail to get processed are moved to the REJECTS handler. **SQL_DSN** Data Source Name **SQL_SERVER** **SQL_USER**

Section TYPE	Configurable Parameters
	User ID for the DSN **SQL_PASS** Password for the DSN **SQL_AUTOCOMMIT** The transaction management mode for the SQL server that decides whether to commit or roll back the data. **SQL_DSN_DELAY** Delay in seconds between attempts to connect to the DSN. Default is 5 seconds. **SQL_DSN_PING** Delay in seconds between pinging the database connection to verify the DSN is available. Default is 300 seconds. **SQL_ENABLE-CORE** Flag to enable the routing of CORE data objects placed in the queue locations for the core objects. **SQL_ENABLE-WBEM** Flag to enable routing of WBEM data objects placed in \data\wbem queue location. **SQL_ENABLE-INVENTORY** Flag to enable routing of INVENTORY data objects placed in the queue locations for filepost (file audit inventory).
	SQL_ENABLE-VM **SQL_AUTOCREATE** A switch to enable the creation of a new SQL file and table in the Inventory ODBC database when a new object class is received. Default is 0. 0 – Does not create a SQL file or table entry for a new object class. 1 – Creates a new SQL file and table entry for a new object class. **SQL_AUTOLOAD** Enables autoload of schemas

You can adjust default values and routing options by editing the `*.cfg` files, located in the `\etc` directory of where the Messaging Server was installed.

The base Messaging Server configuration file (`rms.cfg`) loads the individual Data Directory Agent (DDA) modules for posting the following object types: core, inventory, wbem, and patch. Each DDA has it own configuration file (`*.dda.cfg`) that defines where and how the objects are routed.

See the following topics for more information on how to configure or modify each configuration file.

- "About the Sections in the RMS.CFG File" on next page
- " About the Sections in the CORE.DDA.CFG File" on page 32
- " About the Sections in the PATCH.DDA.CFG File" on page 34
- " Additional Tuning Topics" on page 35

Note: To configure the `USAGE.DDA.CFG` file, see Configuring Aggregation in *Radia Client Automation Enterprise Application Usage Manager Reference Guide*.

To edit a Messaging Server or Data Delivery Agent `*.cfg` file:

1. Stop the Messaging Server before editing the `rms.cfg` file.
2. Edit the appropriate `*.cfg` file using any text editor. By default, the `*.cfg` files are located at: `<InstallDir>\MessagingServer\etc`.
3. Modify the sections using the information given in the following topics.

   Note: All path entries in the configuration files must be specified using forward slashes.

4. Save your changes and restart the Messaging Server.

To encrypt a password entry for a database DSN in a configuration file:

1. The PASS value in all the `.cfg` files where specification of DSN parameters is necessary has to be encrypted. When the value is entered during the install process, the installation program takes care of encryption using AES, by default. If you need to modify the password, you can use the `nvdkit` utility to create an encrypted password using the AES or DES encryption method, and specify this encrypted value within the appropriate section of the configuration file. Enclose the encrypted value in quotation marks.
2. Open a command prompt and go to the directory where the Messaging Server is installed.
3. Enter the following command: `nvdkit`
4. At the % prompt, type the following command for AES encryption (the installation default):
   `nvdkit encrypt <password_value> aes`

   Alternatively, type the following command for DES encryption:
   `nvdkit encrypt <password_value>`

   The utility will return an encrypted password value.
5. Cut and paste this encrypted password value into the `configuration` file as the PASS value. Enclose the value in quotation marks.

About the Sections in the RMS.CFG File

The Messaging Server Configuration file, `rms.cfg`, has the following main sections after the header. The configuration file loads separate modules for data delivery agents (DDAs), whose job is to post the objects to the configured locations.

There are separate data delivery agents for core data (Inventory Manager and Portal objects) and patch data (for Patch Manager)

Optional entries in the `rms.cfg` file can include SSL support, a "`msg:register httpd`" section if this Messaging Server is receiving forwarded messages from another Messaging Server, and a "`msg:register default`" section if this Messaging Server is posting messages from a single queue location of `\data\default`.

Additional Sections in the RMS.CFG File

- **Required packages**
  Do not remove the following lines at the top of the `rms.cfg` file
  ```
  package require nvd.msg

  package require nvd.httpd
  ```
- **SSL Configuration Parameters**
  If the Messaging Server is SSL enabled, this Overrides Config {   } section in the `rms.cfg` is used to define the necessary certificates and parameters for SSL support. It also includes the command `module load tls`, which loads the code necessary to support SSL. For more information, see the *Radia Client Automation Enterprise SSL Implementation Guide*.
- **log::init**
  Logging configuration.These settings apply to all Messaging Server logging. For details on changing the logging configurations, see "Configuring the Log Level, Log Size and Number" on page 37. The log files are located in the Logs directory of where the Messaging Server was installed.
  - **loglevel 3**
    Default logging level for entries written to the log files. Default is 3. Normally, this is not changed.
  - **loglines 200000**
    The default number of lines contained in a log before the log is rolled over.
  - **loglimit 7**
    The default number of rolled logs to keep.
- **Load Data Delivery Agents for posting objects**
  Include the following lines at the end of `rms.cfg` to load the data delivery agents needed to post each type of object.
  - **dda.module load core**
    Posts CORE message data to a SQL or Oracle database and, optionally, CORE message data to the Portal directory. See " About the Sections in the CORE.DDA.CFG File" on next page for more information on how to configure the posting of core objects.
  - **dda.module load patch**
    Post PATCH message data to a SQL or Oracle database. See " About the Sections in the

PATCH.DDA.CFG File" on page 34 for information on how to configure the posting of patch objects.

- **(Optional) msg::register default**, **msg::register router**, and **msg::register** <*rim*|*rmp*|*other*>
These sections, if they exist, define how the Messaging Server handles the messages placed by QMSG in an existing `/data/default` location (or `/data/default` queue).
- **(Optional) Configure the maximum log size and number of logs**
Note that these options apply for each Worker assigned to process the `\data\default` queue. For more information, see "Configuring the Log Level, Log Size and Number" on page 37.
- **(Optional) Configure the frequency to check the Satellite server health**
You can configure the frequency when the Satellite server health is checked and the data is sent to the Core server. The default frequency is set to 21600 seconds (six hours). You can set the HEARTBEAT attribute in the Overrides Config section in rms.cfg on the Satellite server. You can set the minimum value as 1 second and the maximum value as 86400 seconds.

  Snippet to set the frequency to 12 hours:

```
 # Set HEARTBEAT frequency to 12 hours
Overrides Config {
 HEARTBEAT 43200
 }
```

About the Sections in the CORE.DDA.CFG File

The `core.dda.cfg` file defines where and how to route objects placed in queue locations for core objects. As mentioned previously, the core objects are objects created on the agent, available during the agent connect process and used in reports. Examples of core objects are ZMASTER, ZCONFIG, and SESSION.

- To activate the `core.dd`a module, the command `dda.module load core` must be included at the end of the `rms.cfg` file.
- For a Messaging Server co-located with a Configuration Server, the queue locations are folders where the QMSG executable places messages:
Queue folder for core messages: `<`*`InstallDir`*`>\data\core`
- For a Messaging Server receiving messages forwarded from another Messaging Server `core.dda` module, URLs define the locations on which to listen for messages:
URL for core messages: `http://localhost:3461/proc/core`

Several configurations are possible.

- Core object data can be routed using ODBC directly to the back-end SQL database. This option is best used when the database is close to the current location.
- Core data can be forwarded to another Messaging Server. This option is used to place the objects as close as possible to the SQL database, before ODBC posting, in order to avoid slow network response.
- Core messages for the Portal can be routed using HTTP to a Portal Zone, or discarded using the built-in `/dev/null` location.

The `core.dda.cfg` file has the following main sections:

- **msg::register core**
  This is the SQLBALANCER type for the `core.dda` configuration. Routes all messages for a given device to a single sub-queue based on the host name, with a single worker process processing the data in the queue.
  The parameters are summarized below:
  - DIR defines the full path of the `/data/core` location. This is set at installation time.
  - USE defines where the routing information for each TO label is located.
  - POLL and COUNT establish the polling interval and post quantity for the Messaging Server, which determines how often and how many messages are posted at a time. To adjust this, see the topic "Configuring the Poll Interval and Post Quantity" on page 35.
  - Retry Attempts (DELAY and ATTEMPTS), after maximum retry attempts, a message is automatically discarded
- **msg::register core.odbc**
  This is the SQL type. Routes messages to sub-queues based on the hostname. Defines a DSN, User, and Password to post core data directly to an ODBC database. For details on the entries, see the topic " ODBC Settings for CORE, INVENTORY and WBEM Objects" below.
- **(Optional) Configure the maximum log size and number of logs**
  Note that these options apply for each Worker assigned to process the specific queue. For more details, see "Configuring the Log Level, Log Size and Number" on page 37.

ODBC Settings for CORE, INVENTORY and WBEM Objects

The following settings are configured in the **msg::register core.odbc** section of `core.dda.cfg`:

DSN	Specify the Data Source Name (DSN) for the Inventory ODBC database. Enclose the entry in quotes.
USER	Specify the user name for the Inventory ODBC database identified in the DSN parameter.
PASS	Specify the password for the user of the Inventory ODBC database. When modifying a password entry, obtain an encrypted entry using the procedure "To encrypt a password entry for a database DSN in a configuration file: " on page 30. Both AES (the installation default) and DES password encryption methods are supported.
DSN_ ATTEMPTS	Number of attempts to connect to the Inventory Manager database DSN. Default is 1.
DSN_DELAY	Delay in seconds between attempts to connect to the Inventory Manager database DSN. Default is 5 seconds.
DSN_PING	Delay in seconds between pings to the database connection to verify the DSN is available. Default is 300 seconds.
AUTOCREATE	A switch to enable the creation of a new SQL file and table in the Inventory ODBC database when a new object class is received. Default is 0.

	0 – Does not create a SQL file or table entry for a new object class. 1 – Creates a new SQL file and table entry for a new object class.
STARTUPLOAD	No longer supported; any coded value is ignored. **Note:** The SQL tasks to create the tables and load the scripts for the Inventory Database are always performed upon Messaging Server and Data Delivery Agent startup.
SQLPATH	The path where the schemas are stored.
MAPPATH	The path where the mappings (`*.tcl` files) are stored.
WBEM-PKEYS	If true, enables creation of WBEM tables on demand with primary keys instead of unique indexes.
ARCHIVE	Enables archiving of messages. If enabled, the messages are moved to the Archive location.
REJECTS	Moves the messages to the REJECTS handler. The messages that fail to get processed, after maximum number of attempts have been tried to deliver this message, are moved to the REJECTS handler.
VALIDATE_ NLS_PARAMS	The NLS connection setting that can be set for Oracle.

About the Sections in the PATCH.DDA.CFG File

The `patch.dda.cfg` file defines where and how to route the objects placed in the `\data\patch` queue location. Most of the configuration is done automatically during the installation of the Messaging Server.

The `patch.dda.cfg` file has the following main sections after the header and required lines:

```
# DO NOT REMOVE FOLLOWING LINE
package require nvd.msg.patchodbc
package require nvd.msg
```

- **msg::register patchq**
  This is the QUEUE type for the `patch.dda.cfg`. Defines how the Messaging Server handles the messages placed by QMSG in the `/data/patch` location (or `/data/patch` queue). The parameters are summarized below:
  - TYPE of QUEUE defines a Messaging Server location and polling values for picking up messages.
  - DIR defines the full path of the `/data/patch` message location. This is set at installation time.
  - USE defines where the routing information for the TO PATCH objects are located.

- POLL and COUNT establish the polling interval and post quantity for the Messaging Server, which determines how often and how many messages are posted at a time. To adjust this, see the topic "Configuring the Poll Interval and Post Quantity" below.
- Retry Attempts (DELAY and ATTEMPTS), after maximum retry attempts, a message is automatically discarded

- **msg::register patchrouter**
  This is the ROUTER type for the `patch.dda.cfg`. Configures routing assignments for each –To patch type of message. At least one route is specified for each –To type:
  -To PATCH

  -To PATCH5
- **msg::register patchddasummarize**
  Translates any ZOBJSTAT patch reporting objects from pre-5.0 agents into the PATCH reporting objects used in Version 5.0 or higher patch reporting. Not configurable.
- **msg::register patchddaodbc**
  Defines a DSN, User, Password and DBTYPE to post patch data directly to an ODBC database. For details on the entries, see "ODBC Settings for PATCH Objects" below.

  Note: This section may be pre-configured during the install.
- **(Optional) Configure the maximum log size and number of logs.**
  Note that these options apply for each Worker assigned to process the specific queue. For more information, see "Configuring the Log Level, Log Size and Number" on page 37.

ODBC Settings for PATCH Objects

The following settings are configured in the **msg::register patchddaodbc** section of `patch.dda.cfg:`

DSN	Specify the Data Source Name (DSN) for the Patch Manager ODBC database. Enclose the entry in quotes.
USER	Specify the user name for the Patch Manager ODBC database identified in the DSN parameter. Enclose the entry in quotes. Default value is {sa}.
PASS	Specify the password for the user of the Patch Manager ODBC database. Enclose the entry in quotes.
DBTYPE	Specify `SMSSQL` for a Microsoft SQL Server, or `ORACLE` for an Oracle SQL Database. Enclose the entry in quotes. Default is MSSQL.

Additional Tuning Topics

Configuring the Poll Interval and Post Quantity

By default, the Messaging Server is configured to poll the queue location every 10 seconds and post up to 100 objects at a time. To change the poll interval, modify the POLL parameter in the appropriate configuration file and msg::register section with a TYPE of QUEUE.

To change the maximum number of objects to be posted at a time, modify the COUNT parameter in the same section.

If the objects being posted are very large, it is recommended to increase the POLL interval to give sufficient time to complete the posting.

Configuring for Retry Attempts

The Messaging Server and the Data Delivery Agents are configured to retry any message that fails to post. By default, the Messaging Server will retry posting the message every hour, and make up to 200 attempts. These values are defined by the DELAY and ATTEMPTS entries in the sections of the configuration files that have a TYPE of QUEUE. See the table "Configuration File and Section Used to Modify Queue Processing" for a list of msg::register sections used to modify QUEUE processing.

Note: After the last attempt, the message is automatically discarded from the queue without being posted.

To calculate the maximum time that a message could stay in the `/data/default` queue, take the DELAY time and multiply it by the ATTEMPTS value. Using the default settings, this is a DELAY of 3600 seconds x 200 ATTEMPTS, or approximately eight days.

You can adjust the DELAY and ATTEMPTS values in configuration files to establish a different maximum time that a message could stay in the `/data/default` queue. Specify the DELAY in seconds.

Configuring for Failover

You can configure the Messaging Server with one or more servers defined for failover support when defining HTTP types. When defining failover servers, each one is assigned a PRI value. If the Messaging Server fails to connect with the first server (that is, the server with the lowest PRI value) it will try the next server on the list (or, the next higher PRI value).

Note: This PRI value is separate from the –priority value assigned by QMSG for processing priority. The PRI value and the QMSG -priority value are not related.

To set failover in a `*.dda.cfg` file:

Failover support is added by inserting additional ADDRESS entries to the appropriate section of an HTTP type in a DDA cfg file.

The URL entries will be tried in order of PRI (priority) starting with the *lowest* PRI value.

The code sample below shows sample modifications to the msg:register rim section of `core.dda.cfg` for failover. The code in **bold** was added to define a failover server for Inventory Manager processing.

```
msg::register rim {
    TYPE        HTTP
    ADDRESS     {
        PRI     10
        URL     http://rim1:3466/proc/rim/default
```

```
      }
    ADDRESS    {
        PRI    20
        URL    http://rim2:3466/proc/rim/default
     }
  }
```

Configuring the Log Level, Log Size and Number

The log files for the Messaging Server (`rms.log`) reside in the Logs folder of the `MessagingServer` directory, `<InstallDir>\ConfigurationServer\MessagingServer\logs`.

Changing the Logging Level

The log::init section at the beginning of the configuration file establishes the logging level for all Messaging Server logging. The default logging level is 3. Valid levels are 0 (no logging) to 10 (maximum logging level). Normally, the log level is not changed unless requested by a customer support person for troubleshooting purposes. The following lines show the log level increased to 4:

```
log::init {
    -loglevel   4
}
```

Changing the Size and Number of Log Files

The Messaging Server writes entries to a set of log files for each WORKER. There is generally one WORKER attending each queue location. Queue locations include:

```
\data\core
\data\inventory
\data\patch
\data\wbem
```

or

```
\data\default
```

By default, the Messaging Server creates and retains seven log files per worker, each file having a maximum of 5000 lines. The log files are located in the Messaging Server `\logs` directory.

The following line in the `rms.cfg` file establishes the default logging:

```
-loglines 200000
```

To control the size and number of logs created for each worker, add or modify the following entries below the log::init section of the `rms.cfg` file:

```
-loglines maximum_lines
-loglimit maximum number of logs
```

where *maximum_lines* is the maximum number of lines for a given log file. After the maximum is reached, another log file is created, until the *maximum number of logs* specified in the log.configure

–size entry is reached. After the maximum number of logs is reached, the oldest log files are deleted.

The next code sample illustrates an RMS.CFG file containing entries to limit each log file to 1000 lines, and allow up to 10 log files to be retained.

```
log::init {
    -loglevel  3
    -loglines 1000
    -loglimit 10
}
```

Configuring the Messaging Server to Discard or Drain Messages

The location of `/dev/null` is built into the Messaging Server for discarding messages. When the USE parameter is set to `/dev/null` in any of the ROUTER type sections of the Messaging Server or Data Delivery Agent configuration files, the messages being processed will be successfully discarded without an error.

Example: Discarding Messages for the Portal

For example, to discard all RMP messages placed in the `\data\core` queue (these messages have a TO label of CORE.RMP), specify the following ROUTE in the `msg::register corerouter` section of `core.dda.cfg`:

```
msg::register corerouter {
    TYPE    ROUTER
. . .
    ROUTE    {
            TO      CORE.RMP
            USE     /dev/null
}
```

Example: Draining a Message Queue

As another example, to quickly drain an entire queue, temporarily replace `USE router` in the msg::register section for the queue with `USE   /dev/null`. See the table "Configuration File and Section Used to Modify Queue Processing" for a list of the configuration files and sections that control each queue type. After draining the queue, reset it back to `USE router`.

Configuring the Messaging Server to Route Portal Messages

About the Portal Data Queue (rmpq) in CORE.DDA.CFG

The default `core.dda.cfg` configuration re-queues CORE messages that are to be posted to the Portal directory into its own queue, named rmpq. This allows for separate throttling of the CORE messages being posted via HTTP to the Portal directory, as opposed to the CORE messages being posted using ODBC to another database.

The following code shows the sections from `core.dda.cfg` used for this purpose.

```
# Requeue and process just RMP data to throttle the data flow
msg::register rmpq {

TYPE         QUEUE

DIR        ../ConfigurationServer/data/rmp
USE          rmpqrouter

POLL         10
COUNT        30
DELAY        3600
ATTEMPTS     200
}

msg::register rmpqrouter {
TYPE         ROUTER

ROUTE        {
TO       CORE.RMP
USE      rmpqhttp
}
}

msg::register rmpqhttp {
TYPE         HTTP

ADDRESS      {
PRI      10
URL      http://localhost:3471/proc/xml/obj
}
}
```

Restoring Routing for Portal Messages

If you initially installed the Messaging Server to discard Portal messages, use the steps below to begin routing Portal data to a Portal Server and Port.

To modify `core.dda.cfg` for posting Portal data:

1. Use a text editor to edit the `core.dda.cfg` file, located in the `etc` folder of the Messaging Server directory.
2. Look for the section starting with `msg:register corerouter`, and then find the entry for the ROUTE defining `CORE.RMP` messages. The Portal data that is being discarded will show the following entry with a USE value set to `/dev/null`:
   ```
   ROUTE        {
         TO       CORE.RMP
         USE      /dev/null
   ```

3. Change the USE value from `/dev/null` to `rmpq`, as shown below:
   ```
      TO      CORE.RMP
          USE    rmpq
   ```

4. Next, locate the `msg::register rmpqhttp` section near the end of the `core.dda.cfg` file, and find the default URL entry, shown below:
   ```
   msg::register rmpqhttp {
       TYPE         HTTP

       ADDRESS      {
           PRI      10
           URL      http://localhost:3466/proc/xml/obj
   ```

5. Edit the URL value of `localhost:3466` to indicate the host and port of your Portal server. For example, a Portal with a hostname of PORTALSVR on port 3471 is defined with the following URL entry:
   ```
   URL      http://PORTALSVR:3471/proc/xml/obj
   ```
   Host names can be specified using an IP address or a DNS name.

6. Save your changes and restart the Messaging Server.

Enhancing Messaging Server Performance

If the Messaging Server posts large amount of objects to the ODBC and the Portal, the Portal throughput for Messaging Server object degrades. To improve Messaging Server performance for posting objects to the Portal, apply the following configurations in the `rmp.cfg` file. The modified configuration decouples the Messaging Server object posting to the Portal from Messaging Server's main ODBC path. The Messaging Server stores the Portal outbound objects to the disk. The Portal then invokes a separate thread dedicated for processing these messages at its own pace.

The Messaging Server posts the messages into the Portal queue; the Portal then initiates a thread that reads from this folder and updates the destination (Database, OpenLDAP, or Configuration Server database).

These configuration should be applied only to the core Messaging Server.

To improve the Messaging Server performance for posting objects to the Portal, complete the following steps:

1. Stop the HPCA Portal service.

2. Add the following parameters in the `<`*`InstallDir`*`>/ManagementPortal/etc/rmp.cfg` file in the `rmp::init { }` block:
   ```
   ENABLE_RMSQUEUE 1
   RMSQUEUE_PATH
   RMSQUEUE_REJECTS_PATH
   ```
   where,
   `RMSQUEUE_PATH` is the path to the Messaging Server folder where the Portal objects are stored.
   `RMSQUEUE_REJECTS_PATH` is the path to the rejects folder
   For example:
   ```
   ENABLE_RMSQUEUE 1
   ```

```
RMSQUEUE_PATH {<InstallDir>/Data/MessagingServer/rmp}
RMSQUEUE_REJECTS_PATH {<InstallDir>/Data/MessagingServer/rejects}
```

3. Add the following line of code in the `<InstallDir>/HPCA/ManagementPortal/etc/HPCA-RMP.rc` file in the `Overrides Config { }` block:
   `RMS_STORE_SUBSCRIBERS 0`
4. Restart the HPCA Portal Service.
5. Ensure that ATTEMPTS is set to the desired value in the `<InstallDir>/HPCA/ManagementPortal/etc/rms.cfg` in the `msg::register rmp { }` block.

Disabling Processing of Messages in a Queue

The objects in a disabled queue are not polled or posted. You may want to disable processing during peak agent connect periods if resources are in contention, or if you know a target server is down.

You can re-enable the processing at night or during slower periods to allow the Messaging Server to transfer the messages.

To disable queue processing using WORKERS -1 (minus 1) value:

To disable a queue from being polled and its contents posted, set a WORKERS value of -1 (minus one) at the end of the appropriate msg::register section for that queue.

Configuration File and Section Used to Modify Queue Processing

Queue	Configuration File in `MessagingServer\etc` folder	msg::register section
`\data\core`	`core.dda.cfg`	msg::register coreq
`\data\inventory`	`core.dda.cfg`	msg::register inventoryq
`\data\patch`	`patch.dda.cfg`	msg::register patchq
`\data\wbem`	`core.dda.cfg`	msg::register wbemq
`\data\default` (in earlier Versions)	`rms.cfg`	msg::register default

To add a WORKERS value to create multiple processes (WORKERS):

1. Use a text editor to open the appropriate *.cfg file for the Messaging Server or Data Delivery Agent processing the queue to be disabled. The table "Configuration File and Section Used to Modify Queue Processing" identifies the configuration file to use for each queue type.
2. Locate the 'msg::register' section named in the table "Configuration File and Section Used to Modify Queue Processing"; it will be defined with { TYPE QUEUE }.
3. Add or modify a line for WORKERS with a value of -1 (minus 1) to the end of the section. A sample entry for disabling a wbem queue is shown below with a WORKERS value of -1.

```
msg::register wbemq   {
TYPE    QUEUE

DIR  C:/Progra~/Hewlet~/CM/ConfigurationServer/data/wbemq
USE    router

POLL     10
COUNT    100
DELAY    3600
ATTEMPTS  200
WORKERS   -1
}
```

4. Save your changes and exit the editor.

To enable a disabled queue:

To enable a previously disabled queue, change the WORKERS value in the msg::register section of appropriate configuration file from –1 to 1. The number of WORKERS indicates the number of independent and lightweight processes to be started for this queue. It is recommended to use 1 as the default configuration value for a Messaging Server that is processing multiple queues with Data Delivery Agents.

Modifying the Priority in which Messages are Processed

Note: With the adoption of specific queues for each object type, the priority feature is no longer applicable. However, the code for processing messages in increasing priority order has not been disabled.

To modify the priority in which messages are processed, see the earlier topic "How Priority Establishes Messaging Server Processing Order" on page 21.

Configuring Messaging Server to Report Differenced Objects

RCA Messaging Server provides an efficient and flexible messaging system to route the data objects. It monitors pre-defined data queues and dynamically routes data objects to one or more external destinations. RCA agent does not check whether the data objects have changed since last scan to send the difference in the data objects to the Messaging Server. Instead, RCA agent sends complete data to the Messaging Server. Sometimes the Messaging Server queue grows too large, which impacts performance and causes delay in reporting. With the Reporting Differencer feature enabled, RCA agent sends only the differenced data objects to the Messaging Server.

RCA agent invokes the `RepObjDiff.exe` based on the settings configured in the REPTDIFF class in the CLIENT domain. `RepObjDiff.exe` creates a SQLite database on the managed device that contains all the data objects. The data objects in the `OUTBOX` folder are replaced with the differenced data objects containing fewer messages instead of complete data. The Messaging

Server is configured to handle the 'Differenced' Reporting Objects. The `PATCH.DDA` file processes the differenced object data as well as full object data to generate the reports.

Complete the following steps to configure the Messaging Server to report difference data objects:

1. Click **Start** > **Programs** > **Radia Client Automation Administrator** > **Radia Client Automation Administrator CSDB Editor**. The logon dialog box opens.
2. Type your User ID and Password. By default, the user name is `ADMIN` and the password is `secret`.
3. Click **OK**. The RCA Admin CSDB Editor window opens.
4. Navigate to the PRIMARY.CLIENT.REPTDIFF.
5. Set the following parameters:
   a. Set `FLUSHDB` to `Y` to recreate the reporting reference information.
   b. Set `FULLREFR` to `Y` to refresh the complete data and sends it to Messaging Server.
   c. Set `REPTDIFF` to `Y` to enable the Reporting Differencer.
   d. Set `REPTOBJS` to one or more data objects for differencing, such as `DESTATUS.edm`, `BUSTATUS.edm`, and `PASTATUS.edm`. RCA agent invokes the `RepObjDiff.exe` only if one of these objects are available in the `OUTBOX` folder and the Reporting Differencer is enabled.

For more information on REPTDIFF class, see *Radia Client Automation Enterprise Configuration Server Database Reference Guide*.

We appreciate your feedback!

If an email client is configured on this system, by default an email window opens when you click here.

If no email client is available, copy the information below to a new message in a web mail client, and then send this message to radiadocfeedback@persistent.co.in.

Product name and version: Radia Client Automation Enterprise Messaging Server, 9.00

Document title: Reference Guide

Feedback: