Universal Messaging Administration Guide

Version 10.3

October 2018

This document applies to Software AG Universal Messaging 10.3 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Copyright © 2013-2021 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at https://softwareag.com/licenses/.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at https://softwareag.com/licenses/ and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at https://softwareag.com/licenses/ and/or in the root installation directory of the licensed product(s).

Document ID: NUM-AG-103-20210930

Table of Contents

About this Documentation **5**
Online Information and Support 6
Data Protection 6

Overview **7**

2 Universal Messaging Enterprise Manager **9**
Introduction 10
Starting the Enterprise Manager 10
Tab-by-Tab Overview 11
Administration Using Enterprise Manager 14
Using the Enterprise Viewer 351

3 Using Command Central to Manage Universal Messaging **353**
Managing Universal Messaging Using Command Central 354
Securing Communication Between Command Central and Universal Messaging 354
Securing Access to Command Central 356
Instance Management 358
Authentication Configuration 360
Universal Messaging Configuration Types 360
Universal Messaging Administration Types 387
Snooping on Channels 391
Snooping on Queues 393
Publishing Events 396
Universal Messaging Cloud Transformation 398
Universal Messaging Logs 399
Universal Messaging Inventory 399
Universal Messaging Lifecycle Actions 399
Universal Messaging KPIs 400
Universal Messaging Run-time Monitoring Statuses 401
Universal Messaging and the Command Line Interface 401

4 Comparison of Enterprise Manager and Command Central Features **453**

5 Setting up Active/Passive Clustering with Shared Storage **461**
About Active/Passive Clustering 462
Overview of Active/Passive Clustering on Windows 466
Overview of Active/Passive Clustering on UNIX 467
Configuring a Universal Messaging Active/Passive Cluster on UNIX 469

6 Command Line Administration Tools **473**
Introduction to the Administration Tools 474

Starting the Tools using the Tools Runner Application....474
Performing Standard Administration Tasks on Realms and Clusters....476
Running a Configuration Health Check....482
The "Realm Information Collector" Diagnostic Tool....492
The ExportEventsFromOfflineMemFile Tool....499
The RepublishEventsFromOfflineFile Tool....502
Syntax reference for command line tools....503

7 Universal Messaging Administration API....579
Introduction....580
Administration API Package Documentation....583
Namespace Objects....583
Realm Server Management....589
Security....597
Management Information....601

8 Thread Pool Monitoring....609

9 Data Protection and Privacy....613

About this Documentation

- Online Information and Support 6
- Data Protection 6

Online Information and Support

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at http://documentation.softwareag.com. The site requires credentials for Software AG's Product Support site Empower. If you do not have Empower credentials, you must use the TECHcommunity website.

Software AG Empower Product Support Website

If you do not yet have an account for Empower, send an email to empower@softwareag.com with your name, company, and company email address and request an account.

Once you have an account, you can open Support Incidents online via the eService section of Empower at https://empower.softwareag.com/.

You can find product information on the Software AG Empower Product Support website at https://empower.softwareag.com.

To submit feature/enhancement requests, get information about product availability, and download products, go to Products.

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the Knowledge Center.

If you have any questions, you can find a local or toll-free number for your country in our Global Support Contact Directory at https://empower.softwareag.com/public_directory.asp and give us a call.

Software AG TECHcommunity

You can find documentation and other technical information on the Software AG TECHcommunity website at http://techcommunity.softwareag.com. You can:

- Access product documentation, if you have TECHcommunity credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

Overview

This administration guide covers the following areas:

- "Universal Messaging Enterprise Manager" on page 9: This section describes the Enterprise Manager, which is Universal Messaging's native graphical user interface for management of your Universal Messaging environment. There is also a read-only version of the Enterprise Manager, called the Enterprise Viewer, which allows unprivileged users to view the Universal Messaging environment (see the section "Using the Enterprise Viewer" on page 351 for details).
- "Using Command Central to Manage Universal Messaging" on page 353: This section describes the parts of Command Central that are specific to Universal Messaging. Command Central is a generic tool used by many Software AG products. It provides a web browser and command-line interface to configure and manage Universal Messaging.
- "Setting up Active/Passive Clustering with Shared Storage" on page 461: This section describes how to set up an active/passive cluster, using third party solutions that supply additional hardware and software for cluster management.
- "Command Line Administration Tools" on page 473: This section describes a set of command line tools that allow you to perform many of the common actions available through Universal Messaging.
- "Universal Messaging Administration API" on page 579: This section describes the powerful administration API that allows you to build applications to manage your Universal Messaging environment programmatically.

2 Universal Messaging Enterprise Manager

- Introduction 10
- Starting the Enterprise Manager 10
- Tab-by-Tab Overview 11
- Administration Using Enterprise Manager 14
- Using the Enterprise Viewer 351

Introduction

The Enterprise Manager is a powerful, graphical management tool that enables the capture of extremely granular metrics, management and audit information from multiple Universal Messaging realms. The Enterprise Manager also allows you to control, configure and administer all aspects of any Universal Messaging realm or clusters of realms.

The Enterprise Manager has been completely written using the Universal Messaging administration API and so any of its functionality can be easily integrated into bespoke or 3rd party systems management services.

The Enterprise Manager and administration API use in-band management. This ensures that the flexibility of Universal Messaging connections is also made available from a management / monitoring perspective. Universal Messaging realms can be managed remotely over TCP/IP sockets, SSL enables sockets, HTTP and HTTPS as well as through normal and user-authenticated HTTP/S proxies.

This guide contains information on all aspects of using the Enterprise Manager.

The read-only Enterprise Viewer

The Enterprise Viewer is a read-only version of the Enterprise Manager. It allows unprivileged users to view the same information as with the Enterprise Manager, but does not allow you to change the Universal Messaging environment in any way. For further information, see the section "Using the Enterprise Viewer" on page 351.

Starting the Enterprise Manager

In order to start administering and monitoring your Universal Messaging realm servers you need to launch the Enterprise Manager. The Enterprise Manager is capable of connecting to multiple Universal Messaging realms at the same time, whether these are part of a cluster / federated namespace or simple standalone realms. A configuration file called `realms.cfg` is created in your home directory which stores the Enterprise Manager's connection info, however the very first time you launch it a bootstrap RNAME environment variable can be used to override the default connection information. Subsequent launches will not depend on the environment variable as long as you save your connection information (see "Realm Profiles" on page 43).

Launching on Windows platforms can be done by selecting the Enterprise Manager shortcut in the Start Menu.

You can also open a client command prompt and type a command of the following form:

```
<InstallDir>\UniversalMessaging\java\<InstanceName>\bin\nenterprisemgr.exe
```

where <*InstallDir*> is the installation root location and <*InstanceName*> is the name of the Universal Messaging server.

Launching on UNIX platforms can be done by executing the `nenterprisemgr` executable, which you can find under the installation directory at the following location:

```
java/umserver/bin/nenterprisemgr
```

Logging In

When you start the Enterprise Manager, there is a login dialog in which you can enter a user ID and password. The user ID and password are only required for logging in if you have activated basic authentication. If you have not activated basic authentication, the password is ignored, but the user ID is still subject to the usual ACL checks in the Enterprise Manager.

See the section *Basic Authentication* in the Developer Guide for information about setting up basic authentication.

Tab-by-Tab Overview

This section provides a high level overview of Enterprise Manager functionality on a tab by tab basis, for each of the following node types (as displayed in Enterprise Manager's navigation pane).

- "Enterprise Node" on page 11
- "Realm Nodes" on page 11
- "Container (Folder) Nodes" on page 13
- "Channel Nodes" on page 13
- "Queue Nodes" on page 13

Enterprise Node

Highlighting the **Enterprise** node in the tree provides an **Enterprise Summary** view of all realms to which Enterprise Manager is connected, and includes information such the total number of realms, clusters, channels, queues, events published and received, and more.

Realm Nodes

Highlighting a Realm node in the navigation tree in the left hand panel will bring up a context-sensitive set of tabs in the right hand panel:

- **Status Tab**

  Provides a snapshot and historical view of statistics such as the number of events published or consumed, numbers of connections, and memory usage.

- **Monitoring Tab**

  A container for multiple panels that enable you to view live information on the selected realm:

  - **Logs**

    Provides a rolling view of Universal Messaging Logs and Plugin Logs including Access and Error logs.

  - **Connections**

Provides a list of all current connections to the realm, along with details such as protocol, user, and host. Allows connections to be "bounced" (forcing them to reconnect).

- **Threads**

  Provides details such as the number of idle and active threads per thread pool, task queue size per thread pool and a total number of executed tasks for the respective thread pool. It also provides details of scheduled operations each task has within the system.

- **Top**

  A "UNIX top"-like view of realm memory usage, JVM garbage collection statistics, channel and connection usage.

- **Audit**

  Displays the contents of the remote audit file and receives real time updates as and when audit events are generated.

- **Metrics**

  Provides metrics on current memory usage, such as on-heap event memory usage.

- **ACL Tab**

  Displays the realm ACL and the list of subjects and their associated permissions for the realm. Permits editing of ACLs.

- **Comms Tab**

  Provides access to management tools for TCP interfaces, IP Multicast and Shared Memory communication methods:

  - **Interfaces**

    Management of TCP Interfaces (creation, deletion, starting/stopping) as well as configuration of advanced interface properties.

  - **Multicast**

    Management of IP Multicast Configurations (creation/deletion) and advanced configuration tuning.

  - **Shared Memory**

- **Realms Tab**

  Provides a summary of memory, event and interface information for each realm to which Enterprise Manager is connected.

- **Config Tab**

  Manage the settings for many groups of advanced realm configuration parameters.

- **Scheduler Tab**

Permits the user to view, add, delete and edit scheduler scripts.

- **JNDI Tab**

  Enables the creation of references to JMS TopicConnectionFactory and QueueConnectionFactory, as well as references to Topics and Queues.

Container (Folder) Nodes

- **Totals Tab**

  Provides status information for resources and services contained within the selected container branch of the namespace tree.

- **Monitor Tab**

  A "Unix top"-like view of the usage of Channels or Queues found within the container node.

Channel Nodes

Highlighting a Channel node in the navigation tree in the left hand panel will bring up a context-sensitive set of tabs in the right hand panel:

- **Status Tab**

  Provides a snapshot and historical view of statistics such as the number of events published or consumed, rates, and event storage usage.

- **Joins Tab**

  Permits the user to view, add, delete and edit joins between Channels.

- **ACL Tab**

  Permits the user to add, remove or modify entries within the Channel ACL.

- **Durables**

  Enables the viewing and deletion of durables , which provide state information for durable consumers for the channel.

- **Snoop Tab**

  Permits snooping of events on the Channel

- **Connections**

  Enables the creation of references to JMS TopicConnectionFactory and QueueConnectionFactory, as well as references to Topics and Queues.

Queue Nodes

Highlighting a Queue node in the navigation tree in the left hand panel will bring up a context-sensitive set of tabs in the right hand panel:

- **Status Tab**

  Provides a snapshot and historical view of statistics such as the number of events published or consumed, rates, and event storage usage.

- **Joins Tab**

  Permits the user to view, add, delete and edit joins from any Channels to this Queue.

- **ACL Tab**

  Permits the user to add, remove or modify entries within the Queue ACL.

- **Snoop Tab**

  Permits snooping (a non-destructive read) of events on the Queue.

Administration Using Enterprise Manager

Enterprise View

The Enterprise view is the first screen you see whenever the Enterprise Manager is launched. The screen is designed to provide an overview of the characteristics as well as current status of the set of Universal Messaging realms that Enterprise Manager is currently connected with, your Universal Messaging enterprise. This summary view includes any Universal Messaging realms you have added to your connection information whether they are standalone development realms or production clustered realms. Adding a Universal Messaging realm to the Enterprise Manager's connection info will result in the realm's data being included in this view (see "Connecting to Multiple Realms" on page 84 and "Disconnecting from Realms" on page 86).

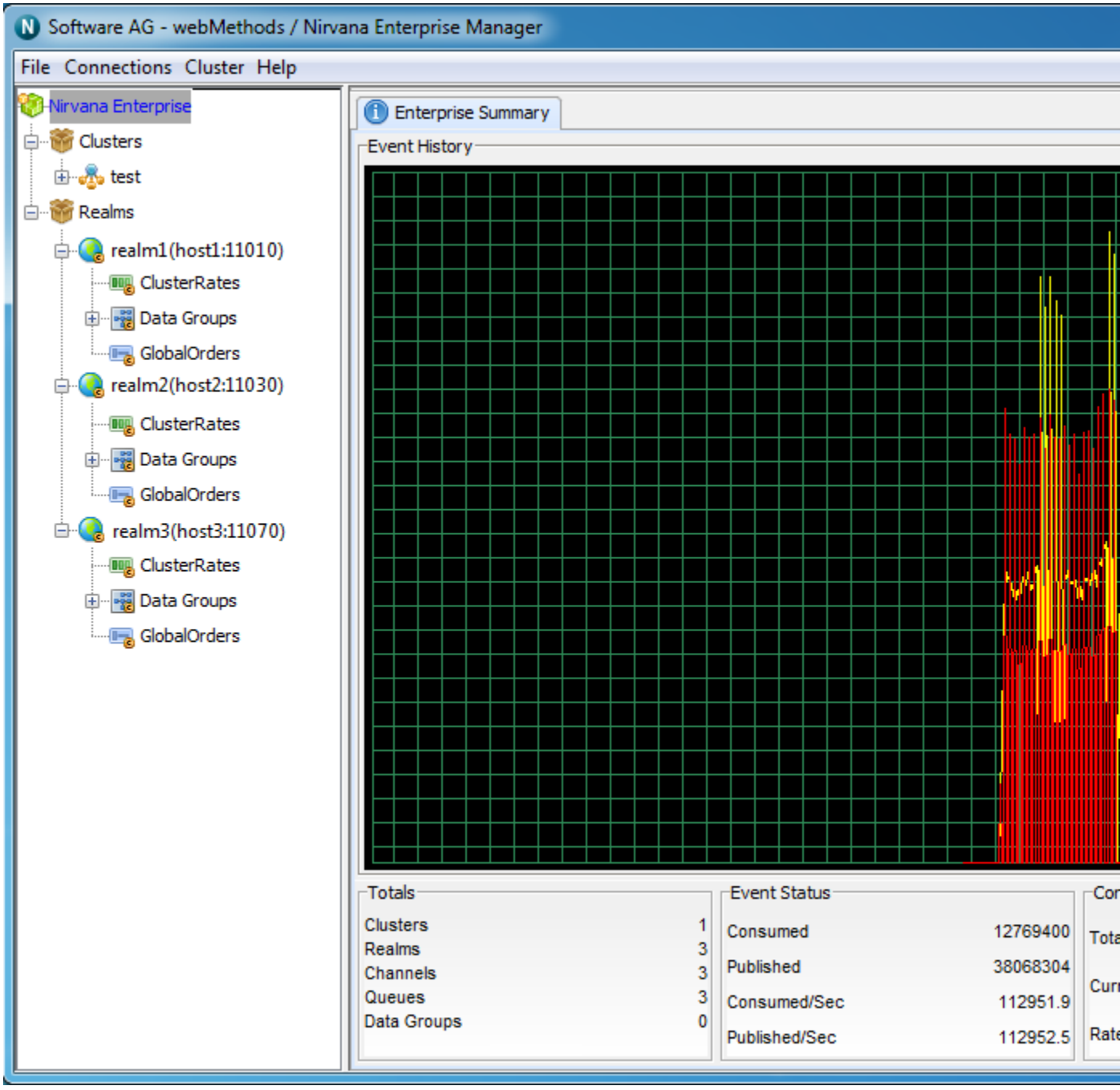
As you navigate through more specific parts of the Universal Messaging enterprise, you can always return to this screen by selecting the root node of the navigation tree called **Universal Messaging Enterprise**.

The view shows a large real time graph illustrating the total number of events published (yellow) and consumed (red) across all Universal Messaging realms. The bottom of the screen displays 3 panels named **Totals**, **Event Status** and **Connection Status** respectively.

The **Totals** panel displays the total number of clusters, realms and resources across all Universal Messaging realms.

The **Event Status** panel displays the total number of events consumed and published, as well as the current consume and publish rates (events per second).

The **Connection Status** panel displays the total number, the current number as well as the number of connections (sessions) being made per second across all realms at this point in time, whether application or administrative.



Realm Administration

Creating and Starting a Realm Server

Universal Messaging provides the following tools for performing general administrative tasks on realms, such as creating a realm, checking the status of a realm, and deleting a realm.

- **The Universal Messaging Instance Manager:**

For related information, see the section *Universal Messaging Instance Manager* in the *Installation Guide*.

- **Command Central:** If your installation of Universal Messaging includes the optional Command Central component, you can use the command line tool of Command Central to perform administrative tasks on realms.

  For related information, see the section "Universal Messaging and the Command Line Interface" on page 401 in the *Command Central* part of the documentation.

Creating a Realm Server

You can use either the Universal Messaging Instance Manager or Command Central to create the realm server. See the examples in the corresponding documentation pages at the locations mentioned above.

Starting a Realm Server

After you have created the realm server, start the realm server as follows:

On Windows systems:

1. From the Windows Start menu, navigate to the node **Start Servers** that is located under the **Universal Messaging** node.
2. Navigate in the hierarchy to find the node labelled **Start *<RealmName>***, and click it. Here, <*RealmName*> is the name you assigned to the realm server when you created it.

On UNIX systems:

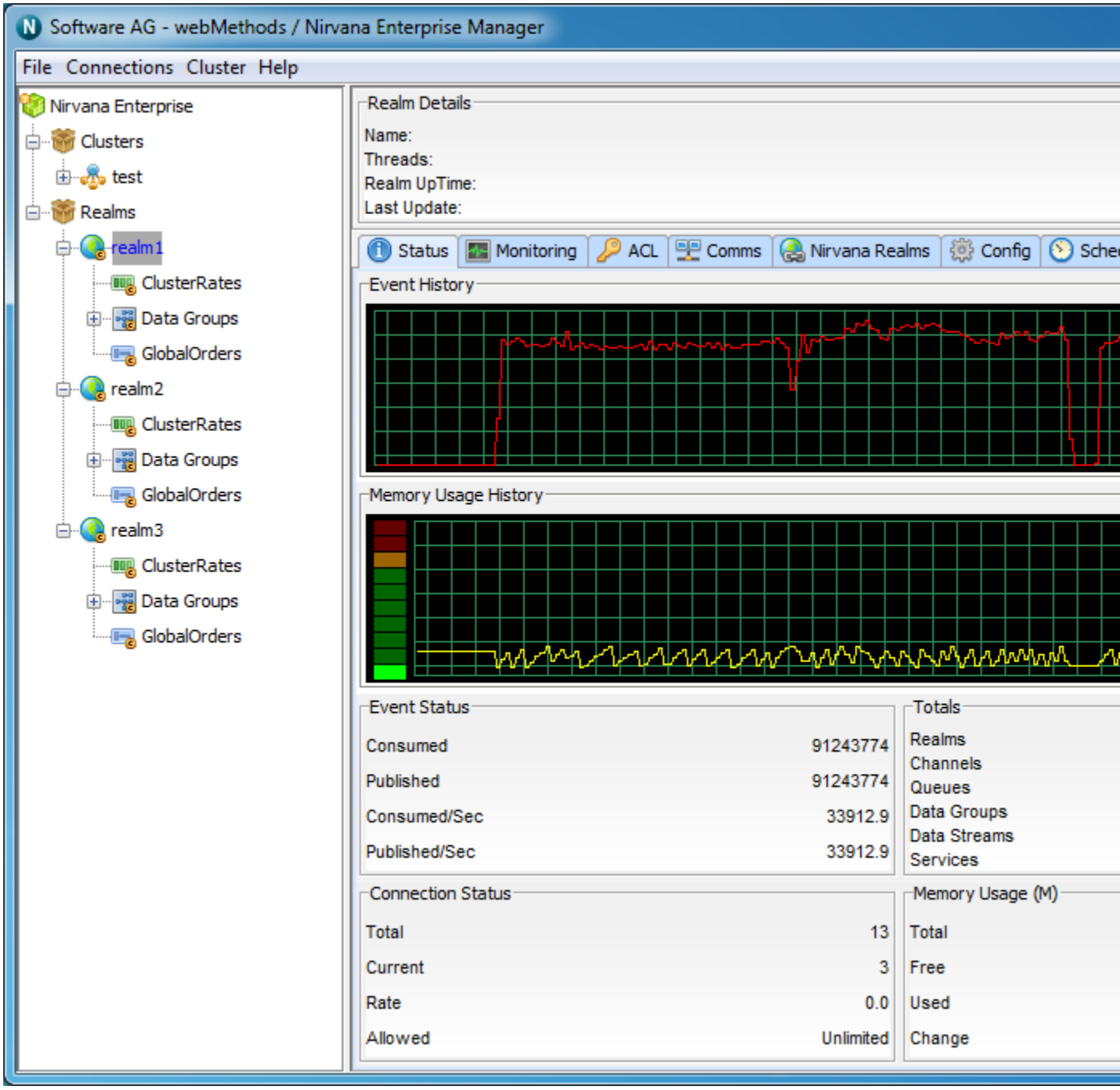
1. Start the script `nserver.sh` that is located in `UniversalMessaging/server/<RealmName>/bin/` under the product installation directory.

Related information on starting and stopping a realm server

For additional information on starting and stopping a realm server, see the sections *Starting the Realm Server* and *Stopping the Realm Server* in the *Installation Guide*.

Viewing a Realm

The **Realm** view provides an overview of the characteristics the current status of the set of Universal Messaging realms that Enterprise Manager is monitoring. When you select a realm node from the namespace, the status panel is displayed by default for the realm.



The top of the screen displays a panel containing several values. These values are :

- **Name** - The name of the selected realm
- **Threads** - Number of threads within the Realm Server's JVM
- **Realm Up Time** - How long the realm has been running for
- **Last Update** - The time that the last status update was sent by the realm

The Status panel contains real time graphs illustrating the total number of events published (yellow) and consumed (red) across the Universal Messaging Realm, as well as the memory usage history for the selected realm.

The bottom of the screen displays 4 panels named **Event Status**, **Totals**, **Connection Status** and **Memory Usage**. These panels and the information displayed are described below.

Event Status

The Event Status section describes the following values :

- **Consumed** - The total number of events consumed from all channels, queues and services within the realm
- **Published** - The total number of events published to all channels, queues and services within the realm
- **Consumed/Sec** - The number of events consumed from all channels, queues and services, per second within the realm
- **Published/Sec** - The number of events published to all channels, queues and services, per second within the realm

Totals

The Totals section describes the following values :

- **Realms**- The number of realms mounted within this realm's namespace
- **Channels**- The number of channels that exist within this realm
- **Queues**- The number of queues that exist within this realm
- **Data Groups**- The number of data groups that exist within this realm
- **Data Streams**- Total number of data streams that exist within this realm

Connection Status

The Connection Status section contains the following values :

- **Total** - The total number of connections made to this realm
- **Current** - The current number of connections to this realm
- **Rate** - The number of connections being made per second to this realm
- **Allowed** - The permitted number of concurrent connections

Memory Usage(MB)

The Memory Usage section contains the following values :

- **Total** - The total amount of MB allocated to the Realm JVM, specified by the -Xmx value for the JVM
- **Free** - The amount of JVM memory available for the Realm
- **Used** - The amount of JVM memory used by the Realm
- **Used/sec** - The amount of memory used per second by the Realm between newest update and previous update

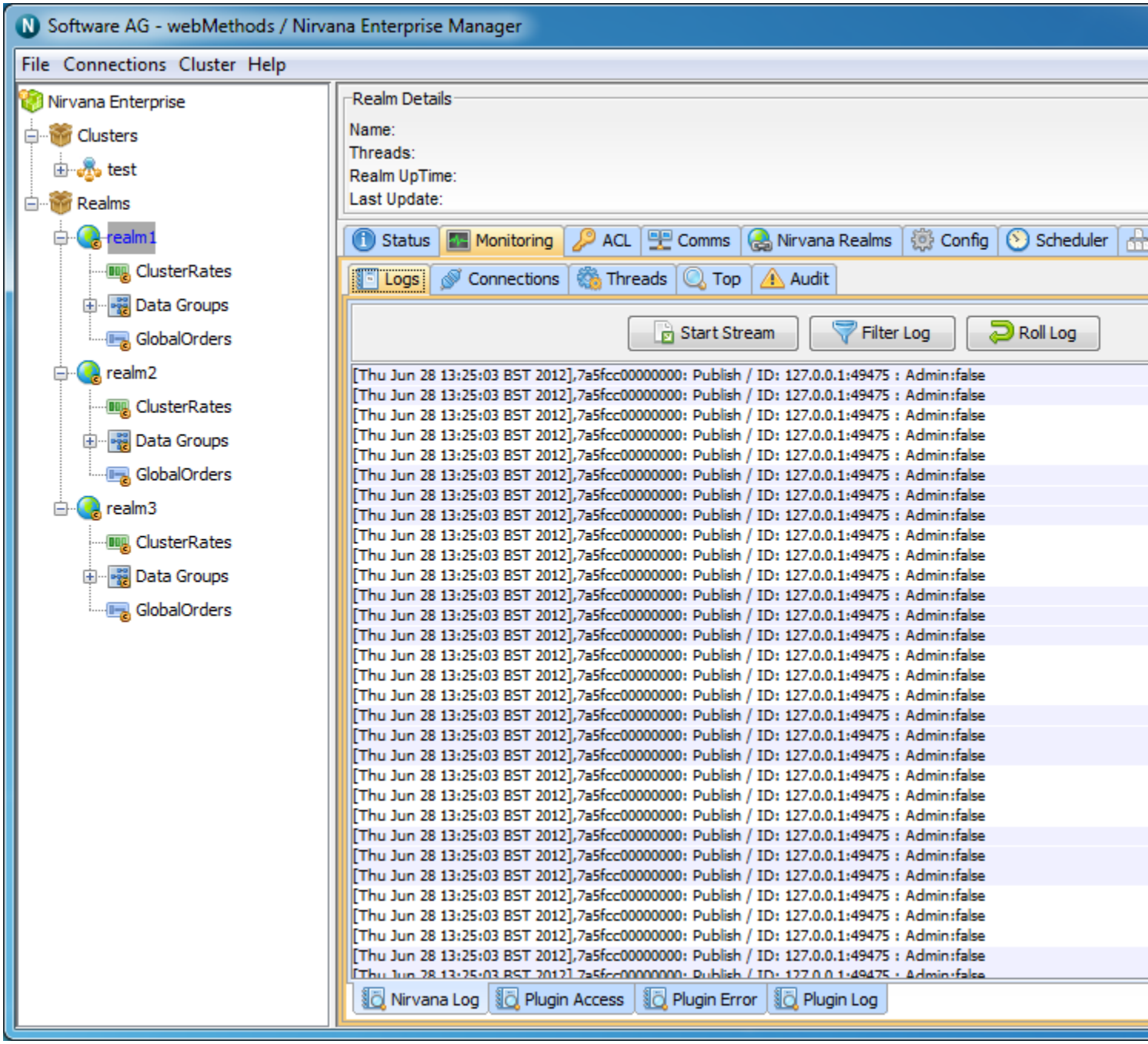
Monitoring a Realm

Overview

When you select a Universal Messaging realm node from the namespace, one of the available panels to select is labeled 'Monitoring'. This panel is a container for multiple panels that enable you to view live information on the selected realm.

There are 5 tabs available under the Monitoring section, as shown in the image below.

- "Logs" on page 20
- "Realm Connections" on page 25
- "Threads Panel" on page 30
- "Top" on page 32
- "Audit" on page 37



Universal Messaging Enterprise Manager : Logs Panel

Each Universal Messaging realm server has a log file called `nirvana.log` within the directory `<InstallDir>\UniversalMessaging\server\<InstanceName>\data`, where *<InstallDir>* is the disk root location of your Universal Messaging installation and *<InstanceName>* is the name of the realm server.

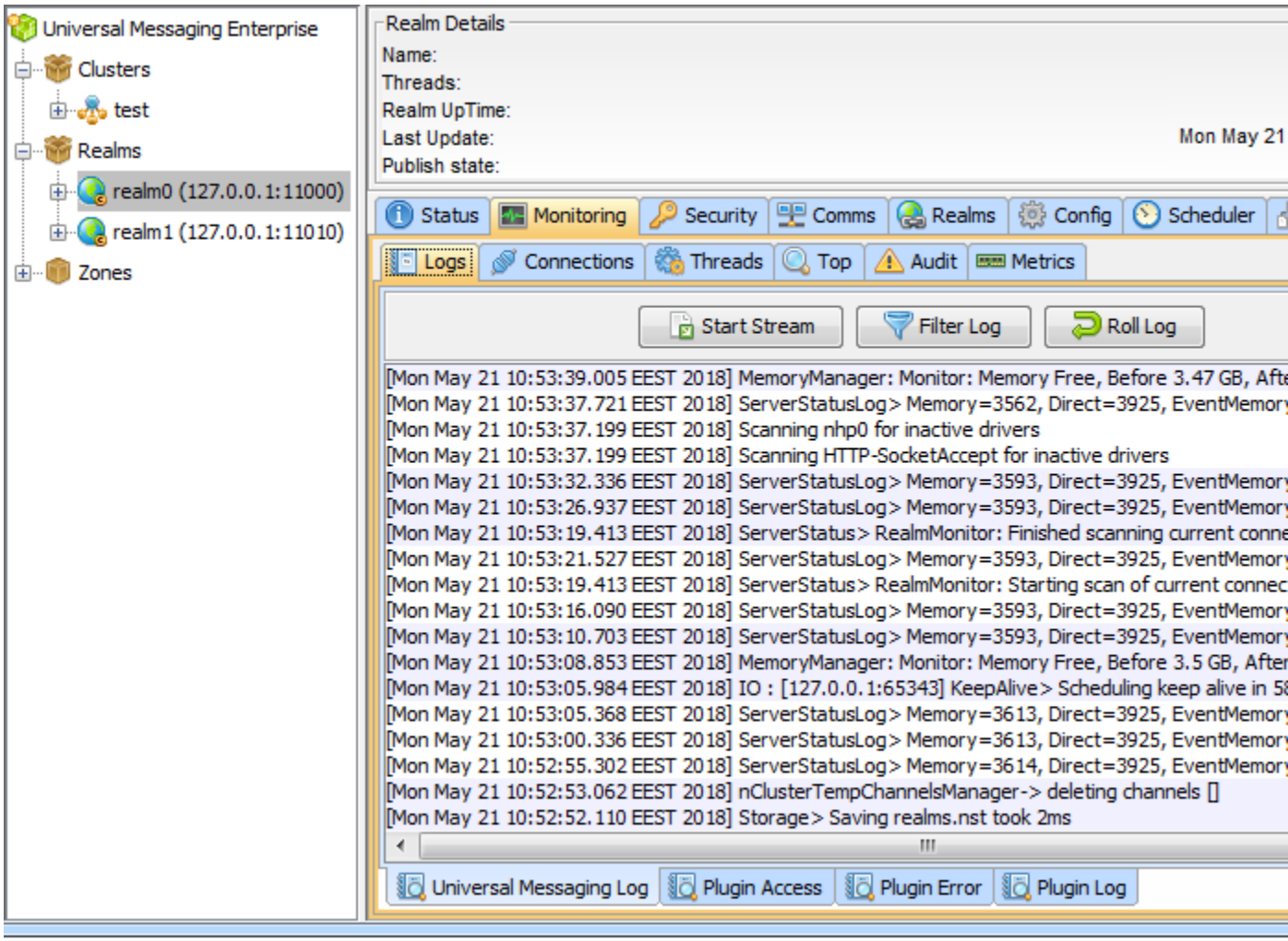
The Enterprise Manager provides a panel that displays real time log messages as they are written to the log file. This enables you to remotely view the activity on a realm as it is happening. The Universal Messaging Administration API also provides the ability to consume the log file entries

from an `nRealmNode`. See the code example "Monitor the Remote Realm Log and Audit File" for an illustration of usage.

The Universal Messaging log file contains useful information about various activities, such as connection attempts, channels being located and subscribed to, as well as status and warning information.

The Logs Panel

The Enterprise Manager provides a panel for each realm where the realm's log file can be viewed. To view the log file, click on the realm node from the namespace and select the panel labeled **Monitoring** and then select the **Logs** tab. This will show the live log messages for the selected realm. The log panel will automatically replay the last 20 log entries from the realm server and then each entry thereafter. The image below shows an example of the log panel for a selected realm:



The log panel also provides the ability to stream the log messages to a local file. Clicking on the button labeled **Start Stream** from the log panel will prompt you to enter the name of the file you wish to stream the log messages to. The stream can be stopped by clicking the same button again.

Understanding the log file

Entries in the log file have the following general format:

```
Timestamp LogLevel ThreadName Message
```

Where:

- `Timestamp` gives the date and time that the entry was created, for example:

  ```
  [Fri May 18 09:03:46.610 EEST 2018]
  ```

  The time of day is given in the format `hh:mm:ss.ttt`, representing hours, minutes, seconds, thousandths of a second.

- `LogLevel` determines the depth of information being logged. It is displayed only if the `EmbedTag` logging configuration property is set to true (default is false). See the description later in this section for details of logging levels.

- `ThreadName` is the name of the internal processing thread that generated the log message. This is displayed only if the `DisplayCurrentThread` logging configuration property is set to true (default is true).

- `Message` contains the actual information that is being logged.

See the section "Realm Configuration" on page 49 for information about configuration properties.

When a server is started, the initial entries in the log file contain useful information about the server's configuration. The following text is an excerpt from a realm server log during startup (the entries for `LogLevel` and `ThreadName` have been suppressed here for clarity) :

```
[Fri May 18 09:03:46.610 EEST 2018] ================================================
[Fri May 18 09:03:46.610 EEST 2018]   Copyright (c) Software AG Limited. All rights
reserved
[Fri May 18 09:03:46.610 EEST 2018] Start date            = Fri May 18 09:03:46 EEST
 2018
[Fri May 18 09:03:46.610 EEST 2018] Process ID             = 9040
[Fri May 18 09:03:46.610 EEST 2018]
[Fri May 18 09:03:46.610 EEST 2018] Realm Server Details :
[Fri May 18 09:03:46.610 EEST 2018]   Product              = Universal Messaging
[Fri May 18 09:03:46.610 EEST 2018]   Realm Server name    = umserver
[Fri May 18 09:03:46.610 EEST 2018]   Release Identifier   = 10.3.0.0.106659
[Fri May 18 09:03:46.610 EEST 2018]   Build Date           = May 17 2018
[Fri May 18 09:03:46.610 EEST 2018]   Data Directory       =
                        C:\SoftwareAG\UniversalMessaging\server\umserver\data
[Fri May 18 09:03:46.610 EEST 2018]   Extension Directory  =
                       C:\SoftwareAG\UniversalMessaging\server\umserver\plugins\ext
[Fri May 18 09:03:46.610 EEST 2018]   Low Latency Executor = false
[Fri May 18 09:03:46.610 EEST 2018]   Has License Manager  = true
[Fri May 18 09:03:46.610 EEST 2018]   Interfaces Running   :
[Fri May 18 09:03:46.610 EEST 2018]     0) nhp0: nhp://0.0.0.0:9000 Running
[Fri May 18 09:03:46.610 EEST 2018]
[Fri May 18 09:03:46.610 EEST 2018]   Realm(s) Reloaded    = 1
[Fri May 18 09:03:46.610 EEST 2018]   Channels Reloaded    = 0
[Fri May 18 09:03:46.610 EEST 2018]   Queues Reloaded      = 0
[Fri May 18 09:03:46.610 EEST 2018]   Data Groups Reloaded = 0
[Fri May 18 09:03:46.610 EEST 2018]   Interfaces Reloaded  = 1
```

```
[Fri May 18 09:03:46.610 EEST 2018]
[Fri May 18 09:03:46.610 EEST 2018] Operating System Environment :
[Fri May 18 09:03:46.610 EEST 2018]   OS Name              = Windows 7
[Fri May 18 09:03:46.610 EEST 2018]   OS Version           = 6.1
[Fri May 18 09:03:46.610 EEST 2018]   OS Architecture      = amd64
[Fri May 18 09:03:46.610 EEST 2018]   Available Processors = 4
[Fri May 18 09:03:46.610 EEST 2018]
[Fri May 18 09:03:46.610 EEST 2018] Java Environment :
[Fri May 18 09:03:46.610 EEST 2018]   Java Vendor         = Oracle Corporation
[Fri May 18 09:03:46.610 EEST 2018]   Java Vendor URL     = http://java.oracle.com/
[Fri May 18 09:03:46.610 EEST 2018]   Java Version        = 1.8.0_151
[Fri May 18 09:03:46.610 EEST 2018]   Java Vendor Name    =
                       Java HotSpot(TM) 64-Bit Server VM 1.8.0_151-b12
[Fri May 18 09:03:46.610 EEST 2018]   Memory Allocation   = 981 MB
[Fri May 18 09:03:46.610 EEST 2018]   Memory Warning      = 834 MB
[Fri May 18 09:03:46.610 EEST 2018]   Memory Emergency    = 922 MB
[Fri May 18 09:03:46.610 EEST 2018]   Nanosecond delay    = Not Supported
[Fri May 18 09:03:46.610 EEST 2018]   Time Zone           = Eastern European Time
[Fri May 18 09:03:46.610 EEST 2018]   Security Provider 0 = SUN version 1.8
[Fri May 18 09:03:46.610 EEST 2018]   Security Provider 1 = SunRsaSign version 1.8
[Fri May 18 09:03:46.610 EEST 2018]   Security Provider 2 = SunEC version 1.8
[Fri May 18 09:03:46.610 EEST 2018]   Security Provider 3 = SunJSSE version 1.8
[Fri May 18 09:03:46.610 EEST 2018]   Security Provider 4 = SunJCE version 1.8
[Fri May 18 09:03:46.610 EEST 2018]   Security Provider 5 = SunJGSS version 1.8
[Fri May 18 09:03:46.610 EEST 2018]   Security Provider 6 = SunSASL version 1.8
[Fri May 18 09:03:46.610 EEST 2018]   Security Provider 7 = XMLDSig version 1.8
[Fri May 18 09:03:46.610 EEST 2018]   Security Provider 8 = SunPCSC version 1.8
[Fri May 18 09:03:46.610 EEST 2018]   Security Provider 9 = SunMSCAPI version 1.8
[Fri May 18 09:03:46.610 EEST 2018] ================================================
[Fri May 18 09:03:46.610 EEST 2018] Startup: Realm Server Startup sequence completed
```

The above sequence of log entries can be found at the beginning of the Universal Messaging log file, and shows information such as when the realm was started, the build number and build date of the Universal Messaging realm server, as well as environmental information like, OS, Java version, timezone.

Log Levels

The Universal Messaging log level is a level from 0 to 6 that determines what information is written to the log. Log level 0 is the most verbose level of logging and on a heavily utilized server will produce a lot of log output. Log level 6 is the least verbose level, and will produce low levels of log output. The log level of each log message corresponds to a value from 0 to 6. The following list explains the log file messages levels and how they correspond to the values:

- 0 - TRACE (Log level 0 will output any log entries with a level in the range 0-6; this is the most verbose level)
- 1 - DEBUG (Log level 1 will output any log entries with a level in the range 1-6)
- 2 - INFO (Log level 2 will output any log entries with a level in the range 2-6)
- 3 - WARN (Log level 3 will output any log entries with a level in the range 3-6)
- 4 - ERROR (Log level 4 will output any log entries with a level in the range 4-6)
- 5 - FATAL (Log level 5 will output any log entries with a level in the range 5-6)

- 6 - LOG (Log level 6 will output any log entries with a level of 6; this is the least verbose level)

Log levels can be changed dynamically on the server by using the **Config** panel (see "Realm Configuration" on page 49). The log file has a maximum size associated with it. When the maximum file size is reached, the log file will automatically roll, and rename the old log file to `_old` and create a new log file . The maximum size for a log file is set to 10000000 bytes (approximately 10MB). This value can be changed within the `Server_Common.conf` file in the `server/<InstanceName>/bin` directory of your installation, where *<InstanceName>* is the name of the Universal Messaging realm. You need to modify the `-DLOGSIZE` property within this file to change the size.

Other Logging Frameworks

By default, Universal Messaging uses a built in logging framework, but there is also the capability to use third party open source frameworks. Currently, we support the Logback (http://logback.qos.ch/) and Log4J2 (http://logging.apache.org/log4j/2.x/) frameworks.

To configure Universal Messaging to use one of these frameworks, you can pass a `-DLOG_FRAMEWORK` parameter with the values LOGBACK or LOG4J2. See the section *Server Parameters* in the Concepts guide for further information.

These frameworks are configured using XML configuration files loaded from the classpath. The Universal Messaging installation provides default versions of these configuration files in the `lib` directory. These files can be modified in order to produce the desired logging output. For more information on configuration see the official documentation of the relevant framework.

Note:
When Universal Messaging is configured to use Logback as the logging framework, the majority of the server startup messages in the server's `nirvana.log` file will be written with status ERROR. This happens due to a limitation in Logback that does not provide usage of custom log levels. Therefore, Universal Messaging messages logged with LOG level are translated to ERROR level when Logback is used.

The Log Manager

Universal Messaging has 3 different log managers for archiving old log files. When a log file reaches its maximum size, the log manager will attempt to archive it, and a new log file will become active. Options such as the number of log files to keep, and the maximum size of a log file are configurable through the logging section of the **Config** panel (see "Realm Configuration" on page 49). When a log file is archived and a new log file created, realm specific information such as Universal Messaging version number will be printed to the start of the new log in a similar way to when a realm is started. Each log manager uses a different method to store log files once they are not the active logs for the realm.

- ROLLING_OLD : This log manager uses 2 log files. The active log file is stored with the default log name, and the most recently rolled log file is stored with `_old` appended to the log name. e.g. `nirvana.log` and `nirvana.log_old`
- ROLLING_DATE : The rolling date manager stores a configurable number of log files (RolledLogFileDepth). Rolled log files are stored with the date they were rolled appended to the active log file name. e.g. `nirvana.logWed-Sep-14-02-31-40-117-BST-2011`.

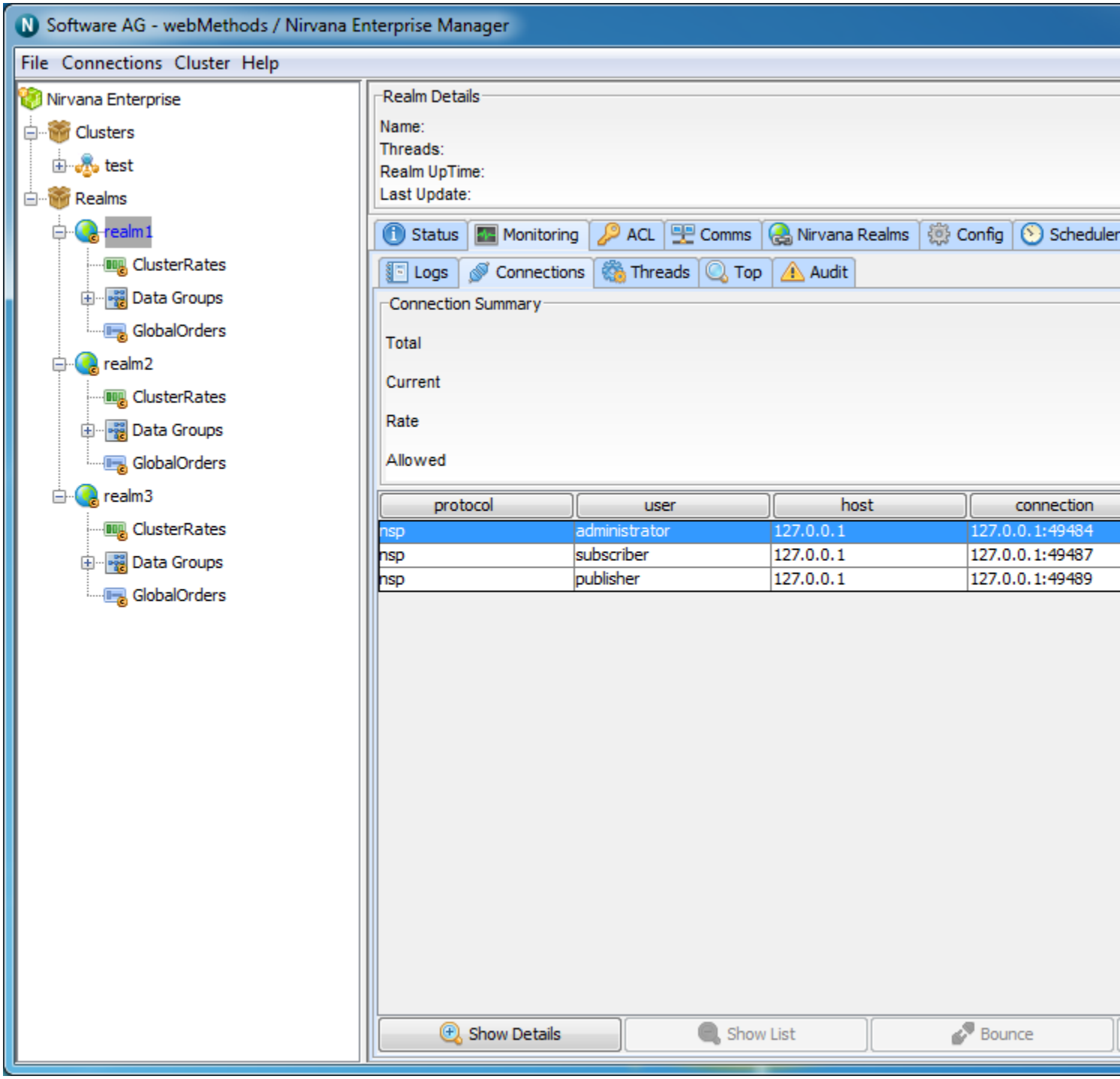
- ROLLING_NUMBER : The numbered log manager stores a configurable number of log files (RolledLogFileDepth). Rolled log files are stored with a numbered index appended to the file name e.g. `nirvana.log3` is the 3rd oldest log file

Realm Connections

When a Universal Messaging client connects to a Realm Server, the server maintains information on the connection (see "Connection Information" on page 607) that is available through the Universal Messaging Administration API. The API also provides mechanisms for receiving notification when connections are added and deleted (see the code example "Connection Watch" for an illustration of using this in the Administration API).

The Universal Messaging Enterprise Manager allows you to view the connections on a realm as well as drilldown and view specific information about each connection, such as the last event sent or received, and the rate of events sent and received from each connection.

To view the current realm connections, simply select a realm node from the namespace, and select the 'Connections' tab from within the 'Monitoring' tab of the selected realm node. This will display a panel containing a table of connections, as shown in the image below.



The connections table contains the following information:

- **protocol** - The protocol used in the connection.
- **user** - The name of the connected user.
- **host** - The host machine from which the user is connecting.
- **connection** - The local connection ID, defined as *hostname:local_port*.

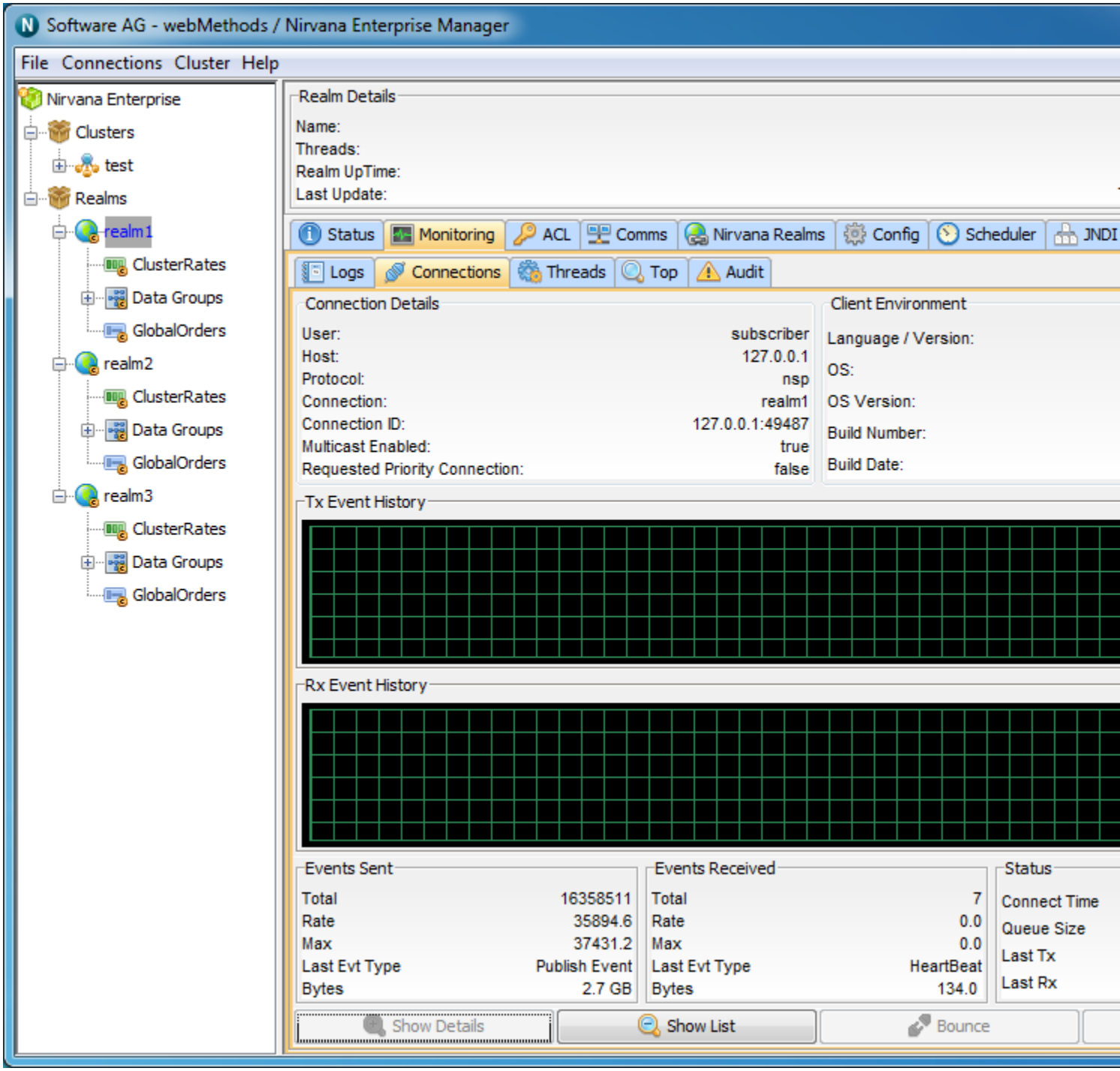
- **language** - The language that the client application is using.
- **name** - The name of the operating system on which the client application is running.
- **build number** - the build number of the client API.

The highlighted connection above shows that the user has connection using the nhp protocol, to localhost. In this example, the nhp interface is running on port 80, so the RNAME of this connection was nhp://localhost:80/

When a connection is highlighted, there a number of things that can be shown for a the connection.

Firstly, connections can be disconnected by clicking the 'Bounce' button.

Secondly, by double-clicking on a connection from the table, or by clicking the 'Show Details' button, you are presented with a panel that contains a more detailed look at the activity for the selected connection. The connection details panel is shown in the image below.



Connection Details

You will see that there are 2 separate information panels above the graphs once you have drilled down into a connection. The first of which is labelled Connection Details. This information contains information about the user connection, such as user name, host protocol.

Client Environment

Next to this you will see a panel that shows details regarding the client environment for this user. These includes API language / Platform, Host OS and Universal Messaging build number

The two graphs, labeled 'Tx Event History' and 'Rx Event History' show the total (yellow) and rates (red) for events received from the server (TX) and sent to the server (RX) for the selected connection.

The bottom of the connection details panel shows 3 sections of information for the selected connection, 'Events Sent', 'Events Received' and 'Status'. Each of these are described below.

Events Sent

The Events Sent section contains the values:

- **Total** - The total number of events sent by the realm server to this connection
- **Rate** - The rate at which events are being sent by the realm server to this connection
- **Max** - The maximum rate at which events have been sent by the realm server to this connection
- **Last Event Type** - The type of the last event sent from the realm server
- **Bytes** - Total bytes sent by the realm server to this connection

Events Received

The Events Received section contains the following values:

- **Total** - The total number of events sent by this connection to the realm server
- **Rate** - The rate at which events are being sent by connection to the realm server
- **Max** - The maximum rate at which events have been sent by this connection to the realm server
- **Last Event Type** - The type of the last event sent from the connection to the realm server
- **Bytes** - Total bytes sent by this connection to the realm server

Status

The Events Sent section contains the following values:

- **Connect Time** - The amount of time this connection has been connected to the realm server
- **Queue Size** - The number of events in the outbound queue of this connection (i.e. events waiting to be sent to the realm server)
- **Last Tx** - The time since the last event was received by this connection from the realm server
- **Last Rx** - The time since the last event was sent to the server from this connection

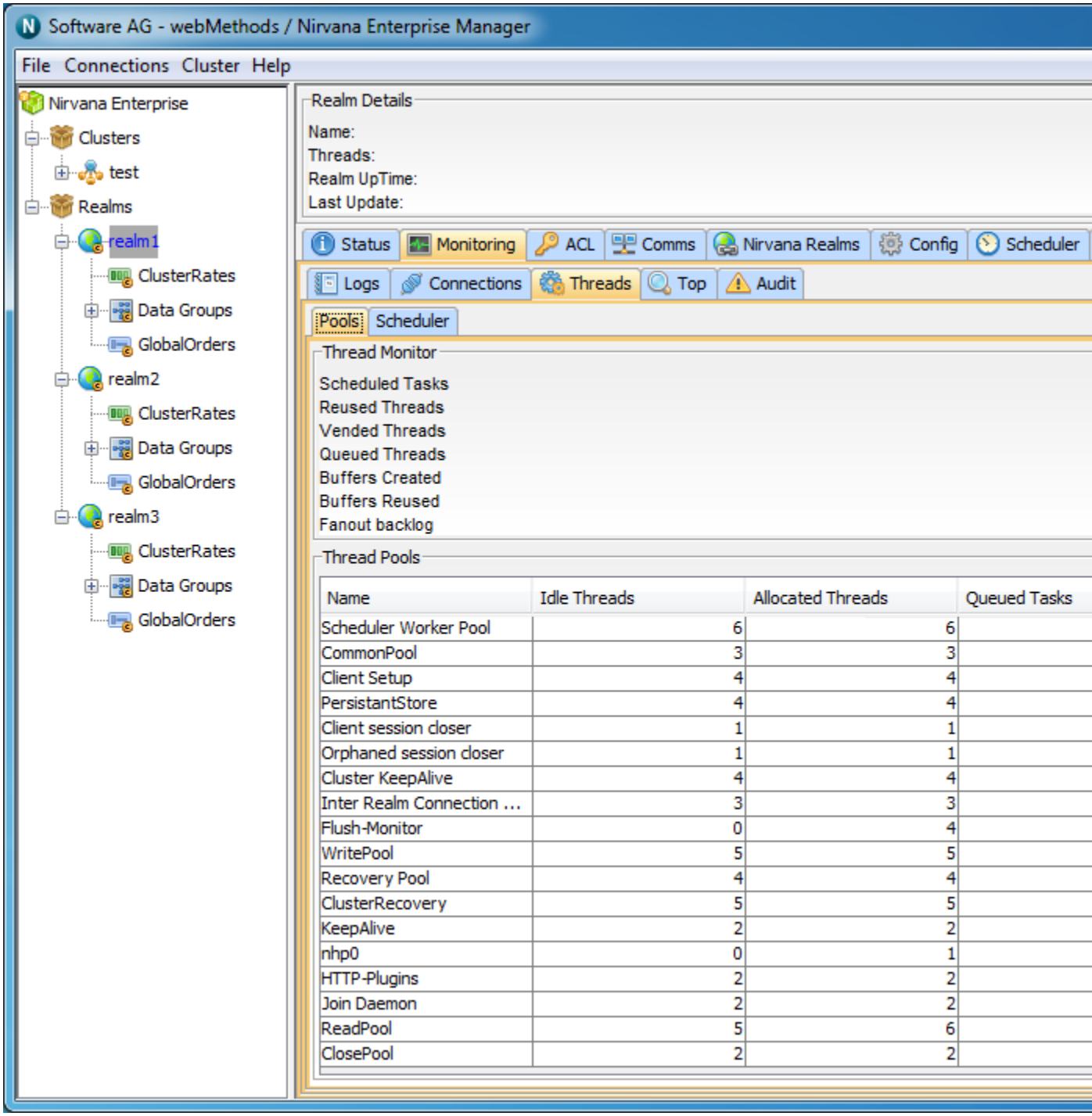
Clicking on the 'Show List' button will take you back to the connections table.

Threads Status

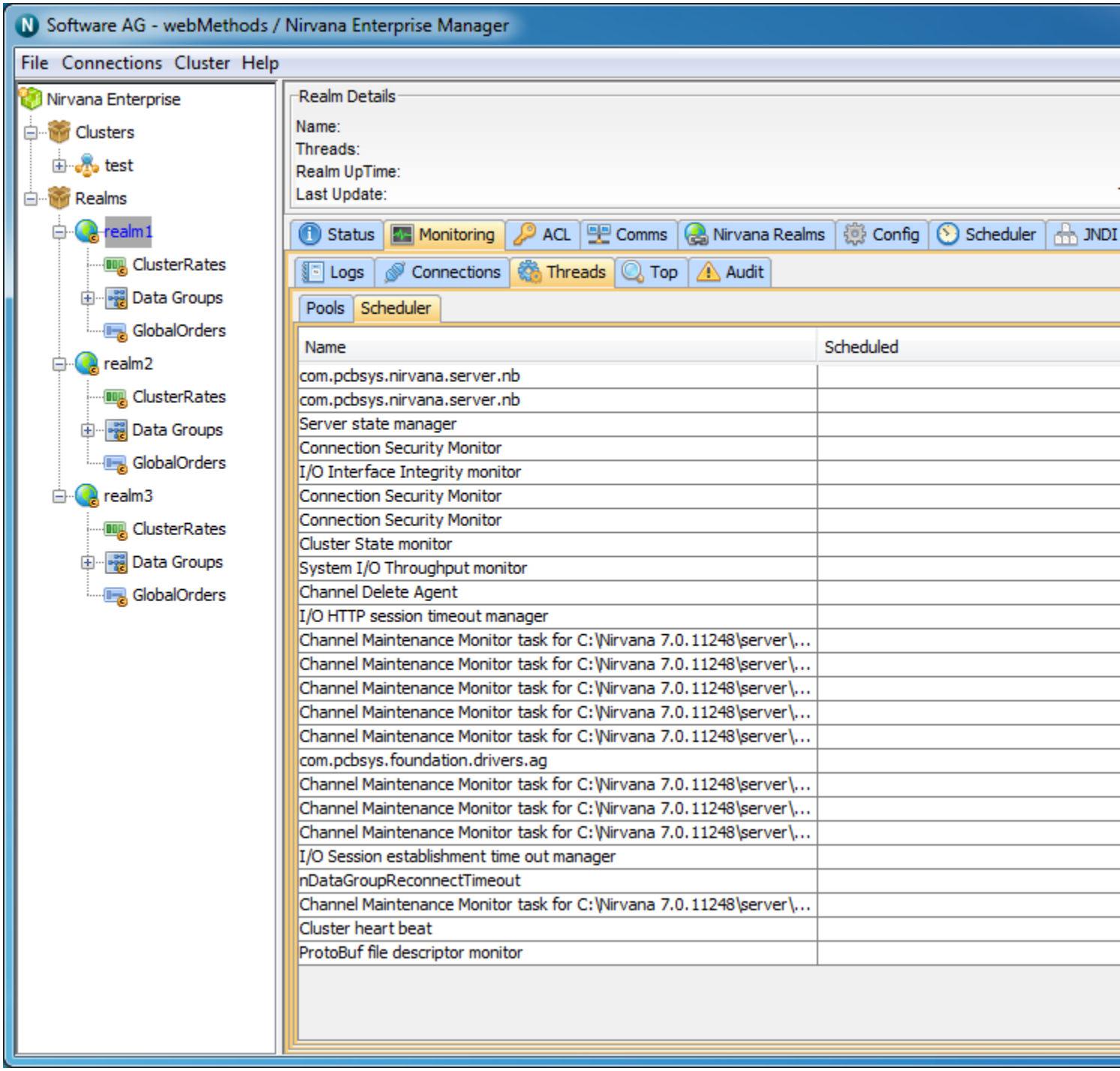
The threads tab found within the Enterprise Manager offers 2 statistical views, thread pools and scheduler tasks.

The thread pool display shows the number of idle and active threads per thread pool as well as the task queue size per thread pool and a total number of executed tasks for the respective thread pool

The Scheduler provides information pertaining to the number of scheduled operations each task has within the system.



Name	Idle Threads	Allocated Threads	Queued Tasks
Scheduler Worker Pool	6	6	
CommonPool	3	3	
Client Setup	4	4	
PersistantStore	4	4	
Client session closer	1	1	
Orphaned session closer	1	1	
Cluster KeepAlive	4	4	
Inter Realm Connection ...	3	3	
Flush-Monitor	0	4	
WritePool	5	5	
Recovery Pool	4	4	
ClusterRecovery	5	5	
KeepAlive	2	2	
nhp0	0	1	
HTTP-Plugins	2	2	
Join Daemon	2	2	
ReadPool	5	6	
ClosePool	2	2	



Top

Within the 'Monitoring' panel of a selected Realm node you will find a panel called 'Top'. This provides a view not unlike 'top' for unix systems or task manager for windows based systems. Its main purpose is to present the user with a high level view of realm usage, both from a connection perspective and also from a channel perspective.

The Top panel comprises 2 sections. The top most section contains 2 real time graphs illustrating the realm memory usage in the same way the Realm Status panel (see "Viewing a Realm" on page 16) displays memory usage, as well as displaying JVM GC stats. This section also contains a summary showing the number of mounted realms, the number of resources and the number of services.

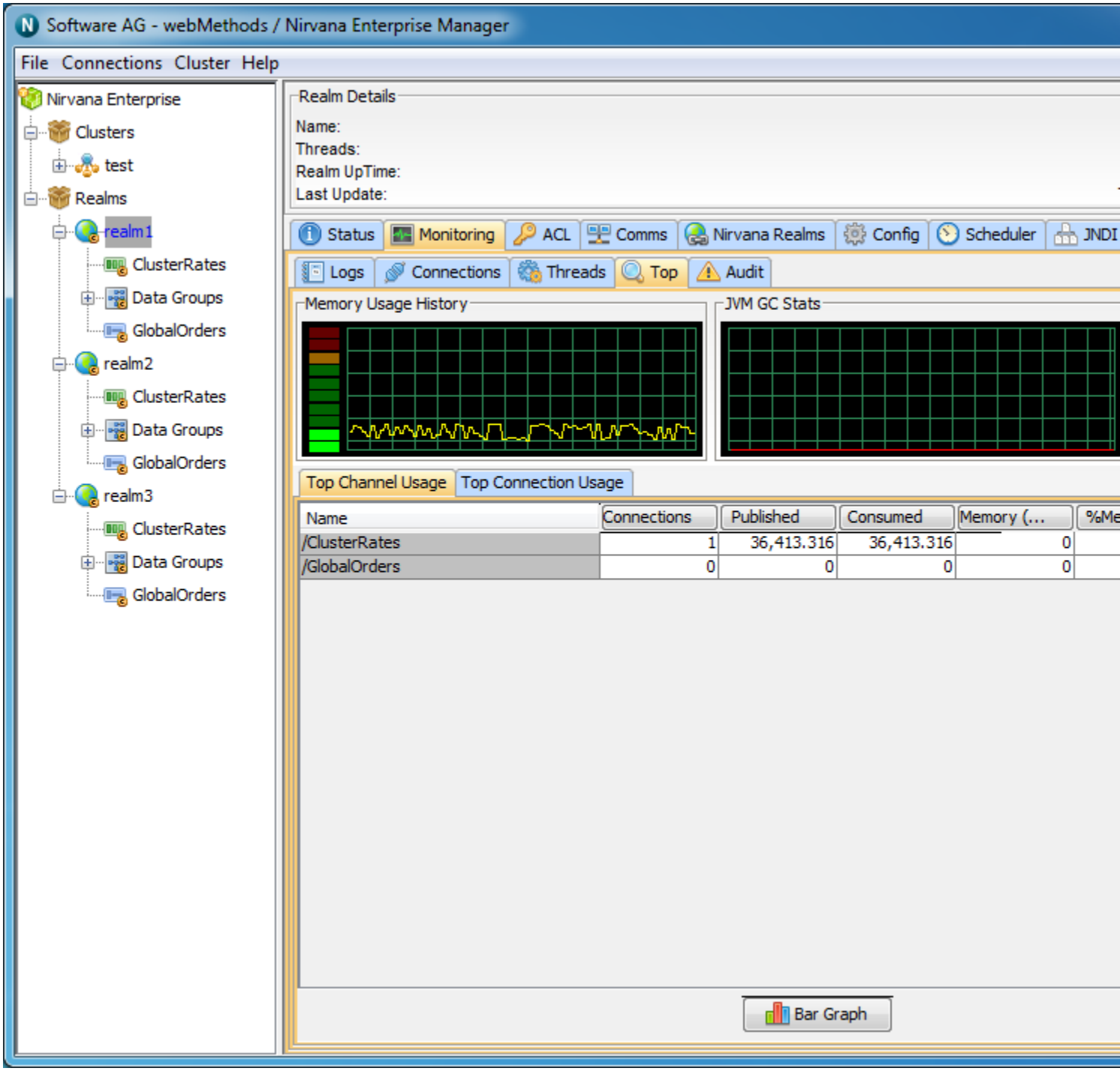
The bottom section of the Monitor panel displays a series of tabs, showing channel and connection usage throughout the realm.

Channel Usage

The middle section of the Monitor panel displays a table showing multiple columns and rows. This table represents channel usage throughout the realm. Each row in the table represents a channel. Channel usage can be measured a number of ways. Each measurement corresponds to a column within the table. By clicking on one of the column headers, all known channels will be sorted according to their value for the selected column. For example, one of the columns is 'Connections', i.e. the number of current consumers on the channel. By clicking on the column header labelled 'Connections', the table will be sorted according to the number of consumers each channel has. The channel with the most number of consumers will appear at the top of the table.

Channel usage measurements are described below:

- **Connections** - The number of consumers the channel has
- **Published** - The rate of events published per status interval
- **Consumed** - The rate of events consumed per status interval
- **Memory (bytes)** - The number of bytes the channel uses from the JVM memory
- **% Memory** - The percentage of overall JVM memory used by this channel
- **Disk** - The amount of disk space used by this channel, only relevant for persistent / mixed channels



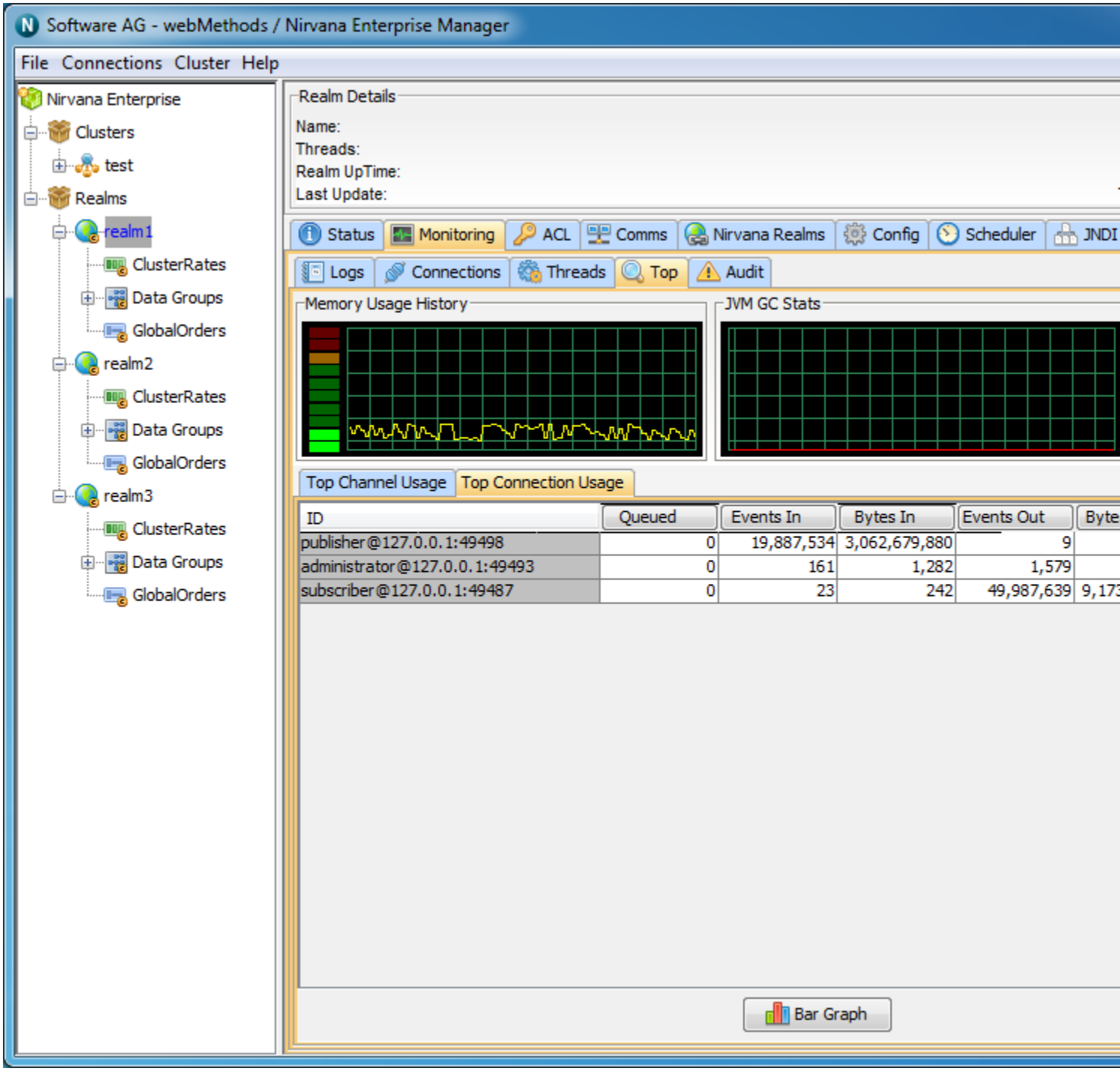
Connection Usage

The bottom section of the monitor panel shows a similar table to that of the channel usage table described above, except that this table represents connection usage. Each row represents a connection. A connection corresponds to the physical aspect of a Universal Messaging Session. Connection usage, like channel usage can be measured in a number of different ways. Each column in the table represents a type of measurement for a realm connection. Clicking on one of the column

headers will cause the table of connections to be sorted according to the value of the selected column. For example, one of the columns is 'Events In', i.e. the number of events sent to the server by the connection. By clicking on the column header labeled 'Events In', the table will be sorted according to the number of events each connection has sent to the server. The connection with the most 'Events In' count will appear at the top of the table.

Connection usage measurements are described below:

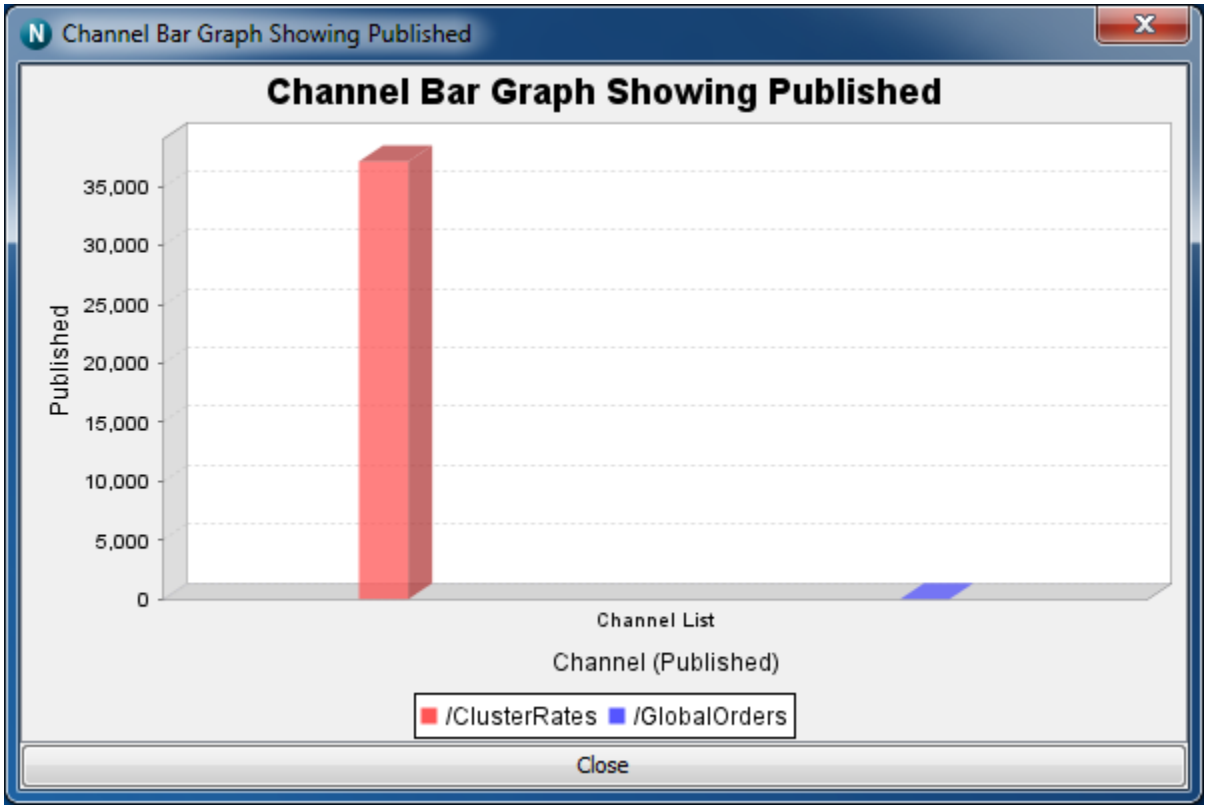
- **Queued**- The number of event in the connections outbound queue
- **Events In** - The rate of events sent by the connection to the realm server
- **Bytes In** - The rate of bytes sent by the connection to the realm server
- **Events Out** - The rate of events consumed by the connection from the realm server
- **Bytes Out** - The rate of bytes consumed by the connection from the realm server
- **Latency** - The measured time it takes the connection to consume events from the server, i.e. time taken between leaving the realm server and being consumed by the connection.



Monitor Graphs

The monitor panel provides a method of graphing both channel and connection usage. It uses a 3D graph package from sourceforge (http://sourceforge.net/projects/jfreechart/) to display the items in each table as columns in a 3D vertical bar chart. The bar charts can be update live as the values in the tables are updated. Once a column is selected, simply click on the button labeled 'Bar

Graph' under either the channel or connections table and a graph panel will appear, as shown in the image below showing a graph of the number of events published for channels within a realm..



Right-clicking anywhere within the graph will show a pop-up menu of options. One of the options is labeled 'Start Live Update', which will ensure the graph consumes updates as and when they occur to the table. Once the live update is started, you can also stop the live update by once again right clicking on the graph and selecting 'Stop Live Update'.

You can also print the graph, and save the graph image as a '.png' file, as well as alter the properties of the graph and its axis.

Audit Panel

Universal Messaging Realm Servers log administration operations performed on the realm to a file. These events are called Audit Events and are stored in a local file called NirvanaAudit.mem. These audit events are useful for tracking historical information about the realm and who performed what operation and when. The Universal Messaging Administration API provides the ability to consume the audit file entries from an nRealmNodeM. See the code example "Monitor the Remote Realm Log and Audit File" for an illustration of usage.

The Universal Messaging Enterprise Manager provides an Audit Panel that displays the contents of the remote audit file and receives real time updates as and when audit events are generated. The audit events that are written to the audit file are determined by the configuration specified in the Config Panel (see "Realm Configuration" on page 49) of the Universal Messaging Enterprise Manager.

Audit Events

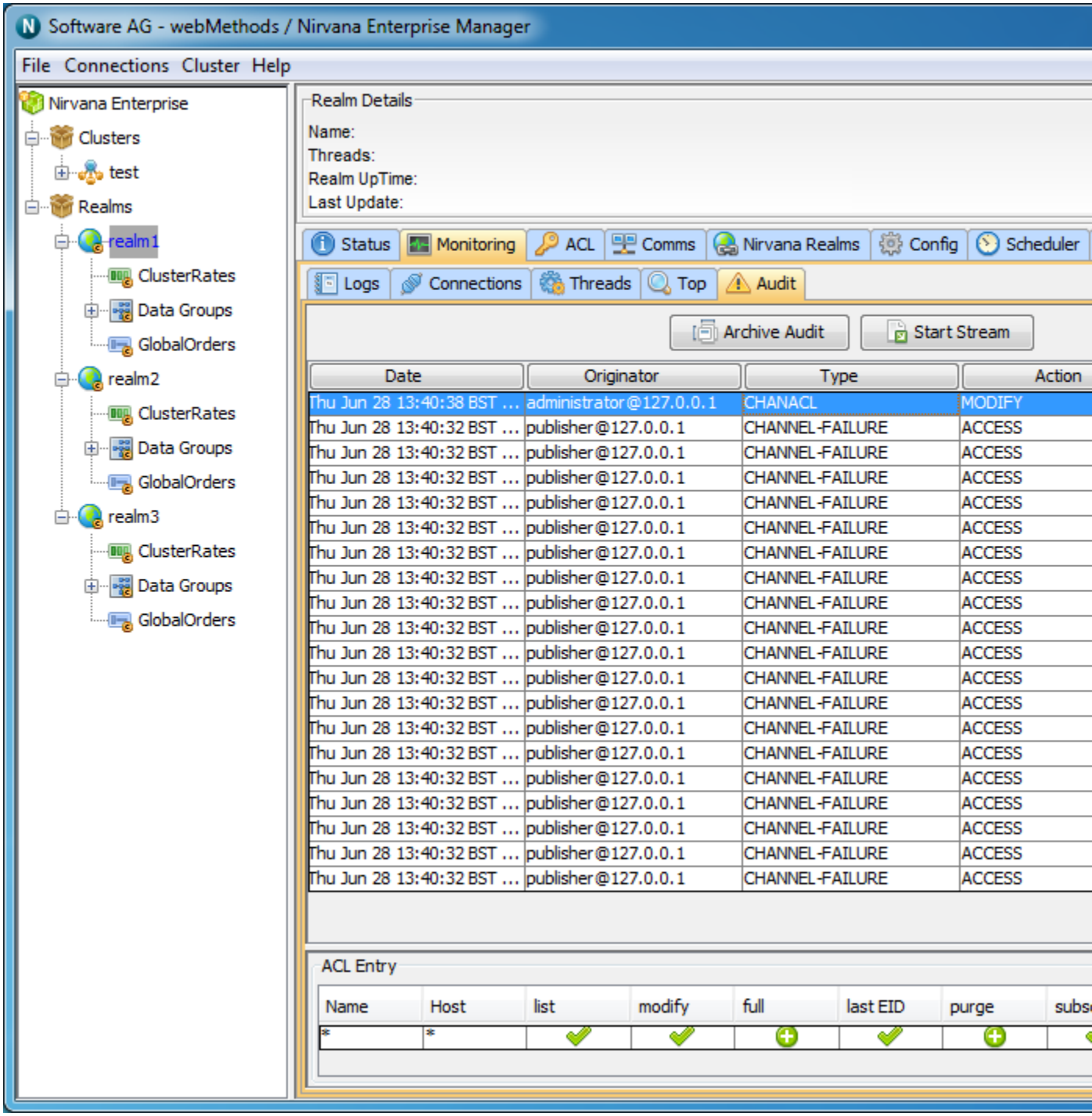
Each audit event corresponds to an operation performed on an object within a realm. The audit event contains the date on which it occurred, the object and the operation that was performed on the object.

The list below shows the objects that audit events correspond to as well as the operations performed on them which are logged to the audit file:

- **Realm** - CREATE, DELETE, ACCESS
- **Interfaces** - CREATE, DELETE, MODIFY, START, STOP
- **Channels** - CREATE, DELETE, MODIFY
- **Queues** - CREATE, DELETE, MODIFY
- **Services** - CREATE, DELETE
- **Joins** - CREATE, DELETE
- **Realm**
- **ACL** - CREATE, DELETE, MODIFY
- **Channel ACL** - CREATE, DELETE, MODIFY
- **Queue ACL** - CREATE, DELETE, MODIFY
- **Service ACL** - CREATE, DELETE, MODIFY

Audit Panel

The audit panel displays audit events for a realm server. You can view the audit panel by clicking on the realm you wish to view the audit file for within the namespace and selecting the panel labeled 'Audit' from within the 'Monitoring' panel of the selected realm. The image below shows an example of the audit panel for a Universal Messaging Realm.



When you first connect to a realm, the audit panel will display the last 20 audit events from its history. Audit files can become quite large over time on a heavily utilised realm, so the initial load is limited to just the last 20. After that all subsequent audit events will be shown in the audit panel.

Each audit event is shown as a row in a table. The table has 5 columns:

- **Date** - The time at which the audit event occurred on the server
- **Originator** - Who performed the operation

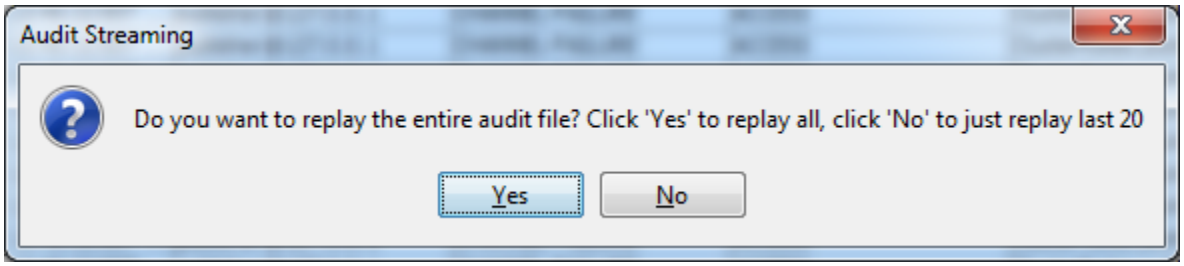
- **Type** - What type of object was the action performed on
- **Action** - What action was performed
- **Object** - The name of the object

If the object type is an ACL for either realm, resource or service, selecting the entry from the table will also display the ACL changes in the bottom section of the audit panel. For modified ACLs, each acl permission that has been granted or removed will be displayed as a green '+', or a red '-' respectively.

Audit Stream

The audit panel provides a button that enables you to stream the remote audit events from the realm to a local file. This also provides you with the option of replaying the entire audit file.

Clicking on the 'Start Stream' button will prompt you with a file chooser dialog to select the location and name of the file that the audit events will be streamed to. Once you have selected this file, you will be prompted whether you wish to replay the entire audit file into the stream or just the last 20 audit entries. The image below shows this dialog:



The text below is an exert from a sample audit file than has been streamed from a server. Each entry that relates to a modified ACL shows the permissions that have been changed, and the permissions that are granted by either a + or -. For permissions that have remained the same, the letter 'N' for not change will be placed after the permission.

```
Fri Jan 21 15:43:40 GMT 2005,CHANACL,/customer/sales:*@*,MODIFY,paul weiss@localhost,
  Full(-), Last Eid(N),Purge(-),Subscribe(N),Publish(-),Named Sub(N),Modify Acls(-),
  List Acls(-),
Fri Jan 21 15:43:40 GMT 2005,QUEUEACL,/partner/queries:*@*,MODIFY,
  paul weiss@localhost,Full(-),Purge(-), Peek(N),Push(-),Pop(-),Modify Acls(-),
  List Acls(-),
Fri Jan 21 15:43:40 GMT 2005,QUEUEACL,/partner/queries:paul weiss@localhost,MODIFY,
  paul weiss@localhost, Full(N),Purge(N),Peek(N),Push(N),Pop(N),Modify Acls(N),
  List Acls(N),
Fri Jan 21 16:13:10 GMT 2005,INTERFACE,nhp0,CREATE,paul weiss@localhost,
Fri Jan 21 16:15:31 GMT 2005,INTERFACE,nhp0,MODIFY,paul weiss@localhost,
```

Archive Audit

The audit panel provides a button that enables you to archive the audit file. As mentioned before, depending on what is being logged to the audit file, the file can grow quite large. As it's an audit and provides historical data, there is no automatic maintenance of the file it is down to the realm

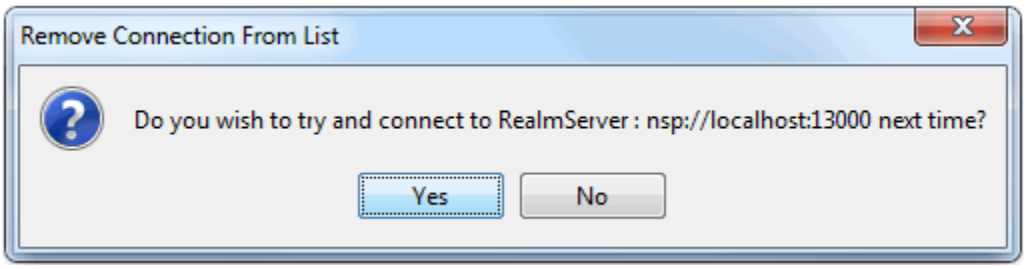
administrators when the file is archived. The 'Archive Audit' button when clicked will simply rename the existing audit file to a name with the current date, and start a new audit file.

Editing Connection Information

As mentioned in previous sections, Universal Messaging Enterprise manager can connect to multiple Universal Messaging realms at the same time and allows saving connection information in a configuration file. This configuration file can change in one of 3 ways:

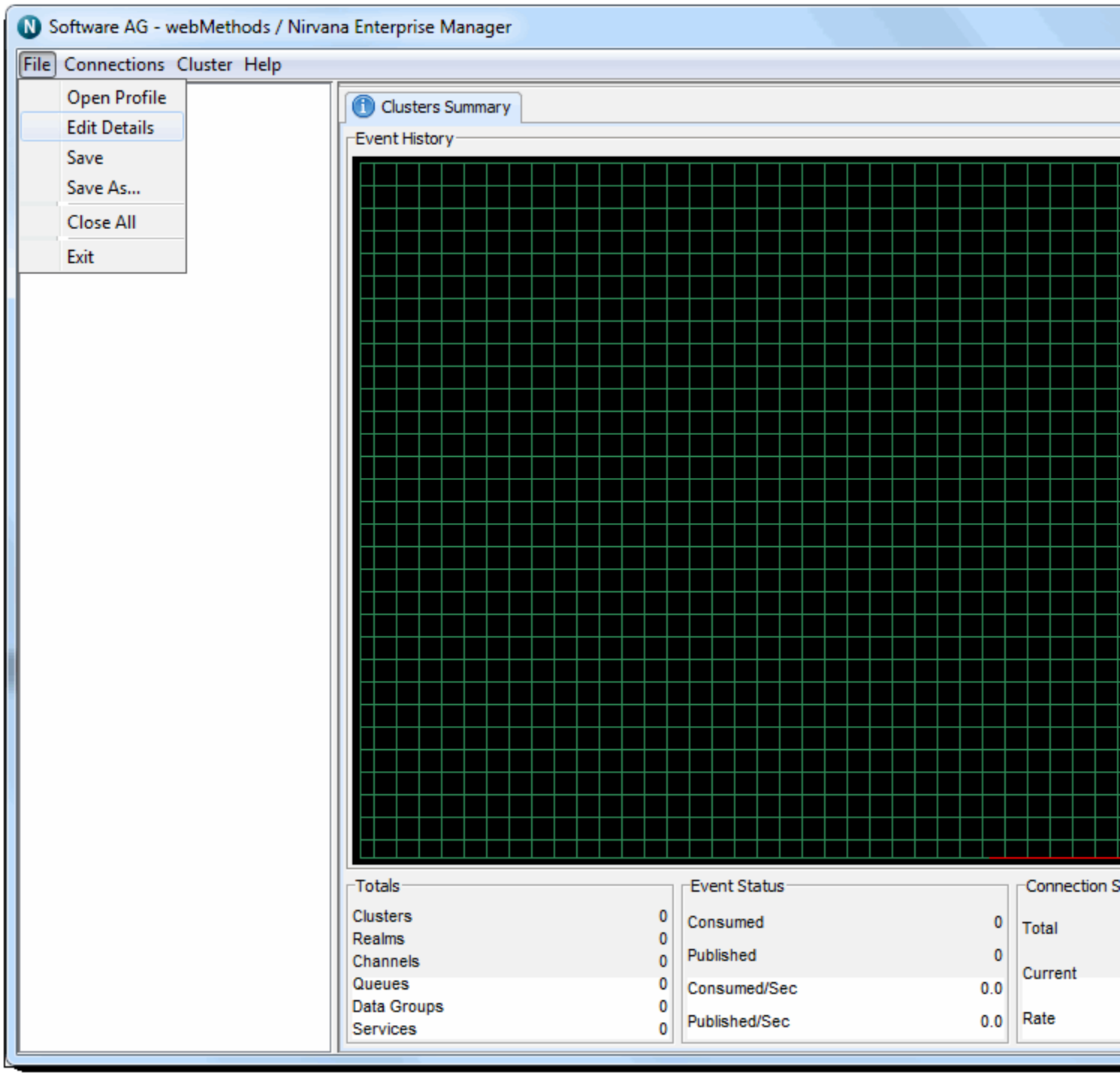
1. By selecting the Save Connection Info menu option (see "Realm Profiles" on page 43) which replaces the configuration file contents with the list of current connections.

2. When running the Enterprise manager, if a connection to a configured realm fails and the user chooses not to retry, a second dialog appears that looks like the example in the figure below:

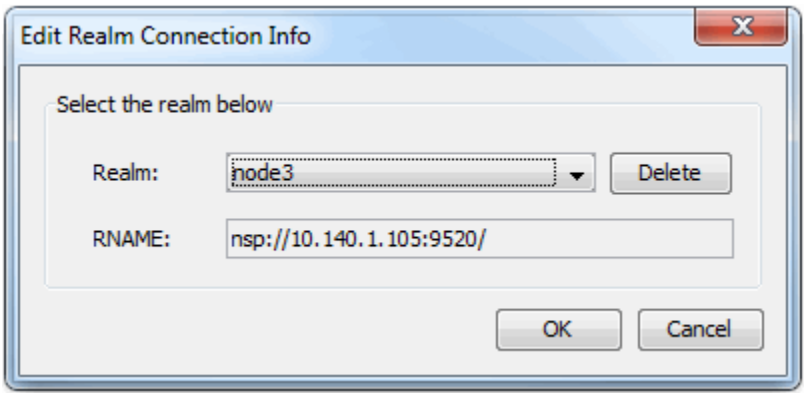


If the user clicks Yes, then the configuration file remains the same. However if the user chooses no, the failed connection is removed from the configuration file without any further action required. The Enterprise manager will never try to connect to that Universal Messaging realm again during startup.

3. By using the Edit Connection Info menu option, located under the File menu as shown in the figure below:



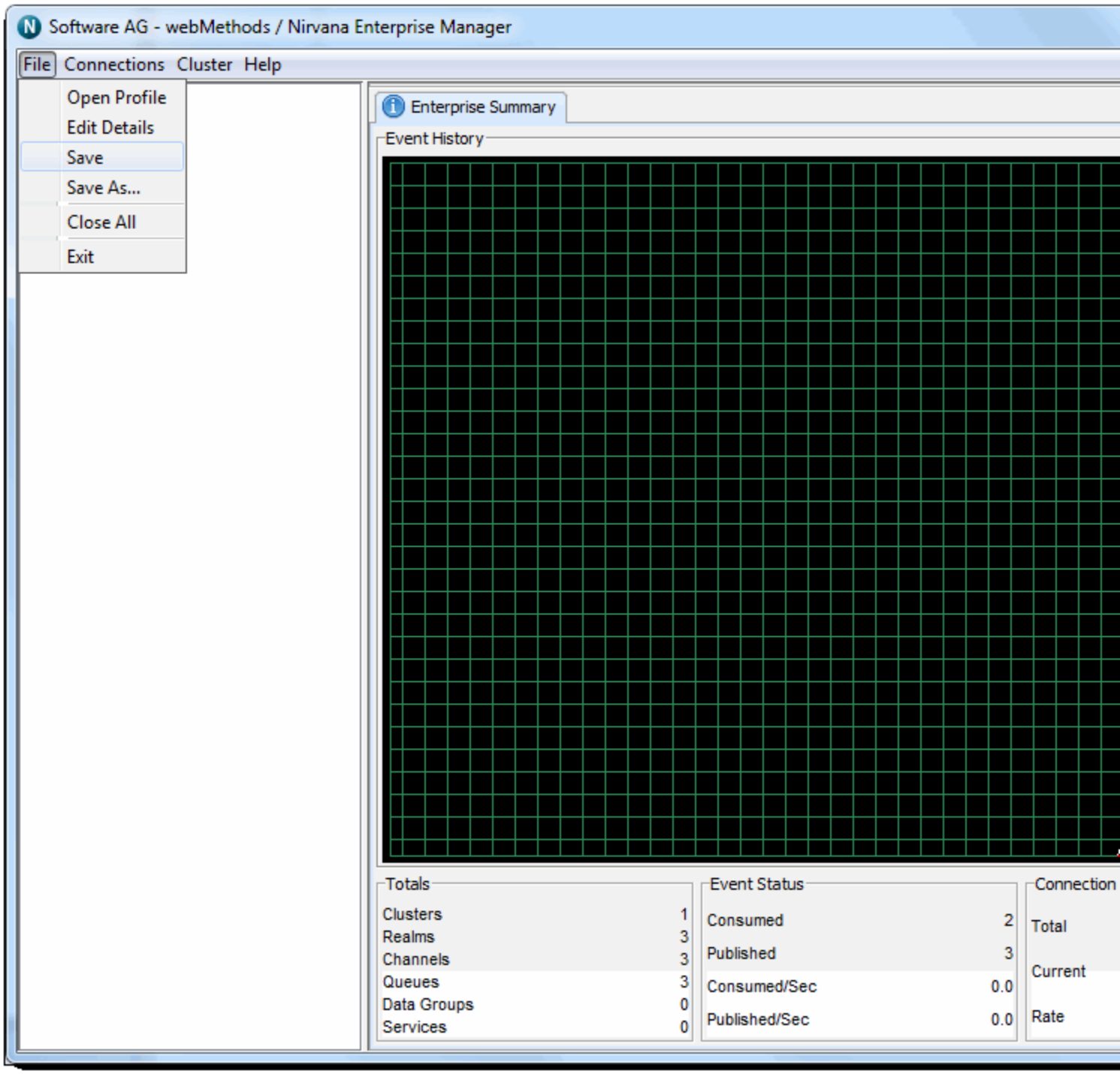
This causes the following dialog to appear:



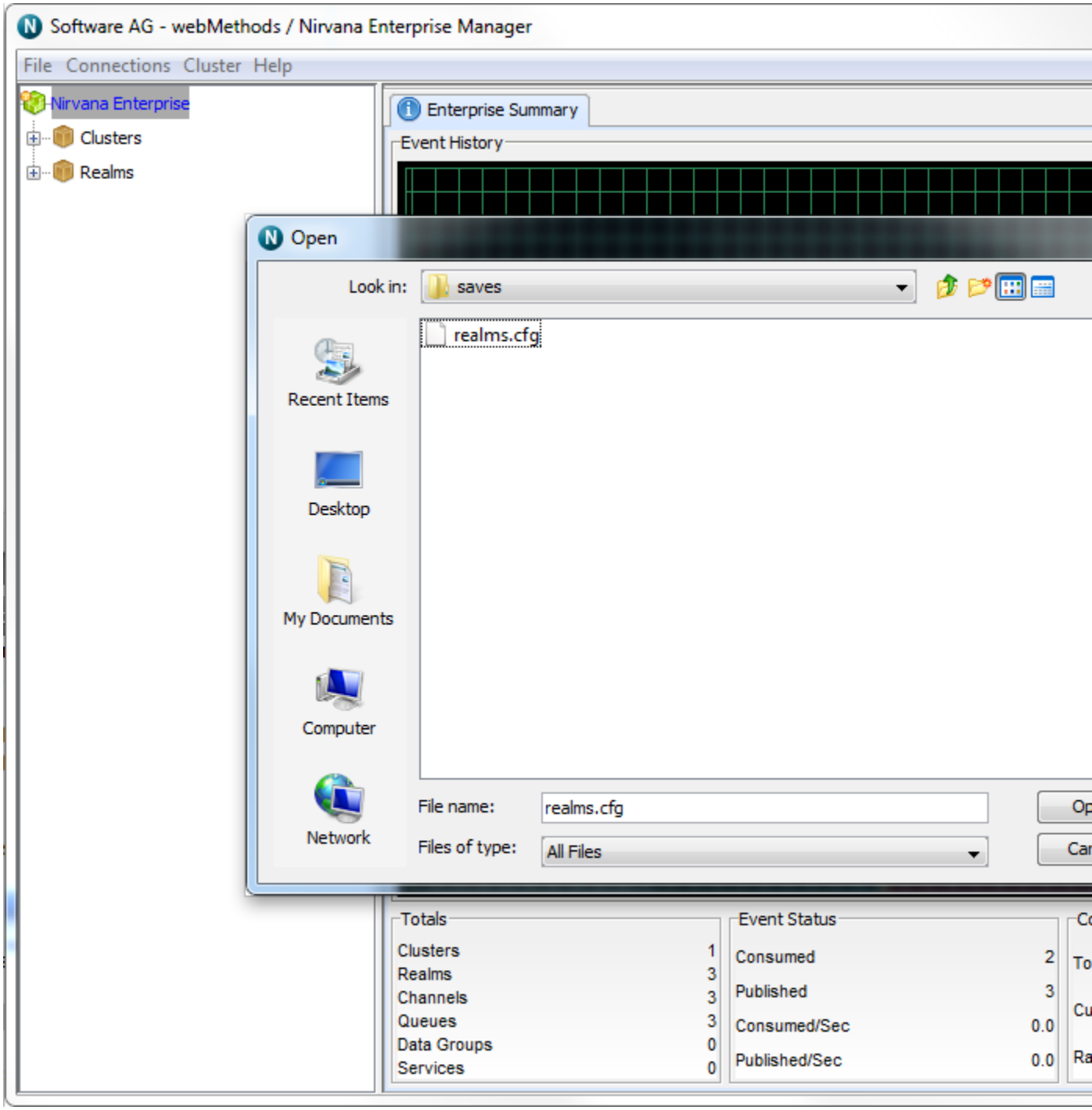
The Realm name combo box contains the complete list of configured Universal Messaging realms that had been connected during the last Save Connection Info operation. If you have connected to additional realms that had not been saved, these will not be included in this list. By selecting a particular Realm name, you can also see the connection RNAME value containing the RNAME that Enterprise manager uses to connect to it. Clicking on the delete button will remove the currently selected realm from the connection info file and this can be repeated many times until only the desired realms are present in the list. When this is done, click on the Save button to recreate the connection info file.

Realm Profiles

The Universal Messaging Enterprise Manager enables administrators to group realms and their respective connections into profiles for easy management and accessibility. Any number of realms can be saved as part of a profile.



When profiles are reloaded the Universal Messaging Enterprise Manager automatically connects to all realms defined within the loaded profile.



Realm Federation

As well as clustering technology, Universal Messaging supports the concept of a federated namespace which enables realm servers that are in different physical locations to be viewed within one logical namespace.

Note:

Clustering and Realm Federation are mutually exclusive. If a realm is a member of a cluster, you cannot use the realm for federation. Similarly, if a realm is part of a federation, the realm cannot be used for clustering.

If you consider that a Universal Messaging namespace consists of a logical representation of the objects contained within the realm, such as resources and services: a federated namespace is an extension to the namespace that allows remote realms to be visible within the namespace of other realms.

For example, if we had a realm located in the UK (United Kingdom), and 2 other realms located in the US (United States) and DE (Germany), we can view the realms located in DE and US within the namespace of the UK realm. Federation allows us to access the objects within the DE and US realms from within the namespace of the UK realm.

It is possible to add realms to a Universal Messaging namespace using the Universal Messaging Administration API or by using the Enterprise Manger as described below.

Adding Realms

The first step in order to provide federation is to add the realms. Adding a realm to another realm can be achieved in 2 ways. The first way simply makes a communication connection from one realm to another, so the realms are aware of each other and can communicate. This allows you to create a channel join between these realms.

Note:
For a description of the general principles involved in creating channel joins, see the section *Creating Channel Joins*. The description details the usage based on the Enterprise Manager, but the same general principles apply if you are using the API.

The second option also makes a new communication connection, but if you specify a '*mount point*', the realm you add will also be visible within the namespace of the realm you added it to.

Mount Points

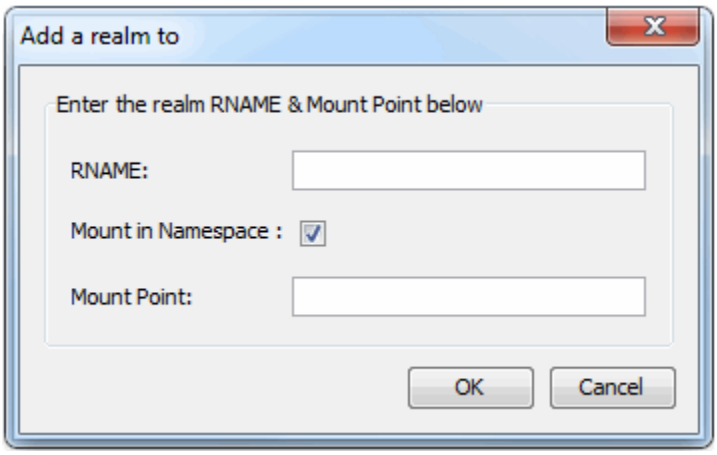
Providing a mount point for added realms is similar to the mount point used by file systems when you mount a remote file system into another. It specifies a logical name that can be used to access the resources within the mounted realm. The mount point is therefore the entry point (or reference) within the namespace for the realm's resources and services.

For example, if I have a realm in the UK, an wish to add to it a realm in the US, I could provide a mount point of '/us' when adding the US realm to the UK realm. Using the mount point of '/us', I can then access the channels within the US realm from my session with the UK realm. For example, if I wanted to find a channel from my session with the UK realm, and provided the channel name '/us/customer/sales', I would be able to get a local channel reference to the '/customer/sales' channel within the US realm.

Using the Enterprise Manager to add realms

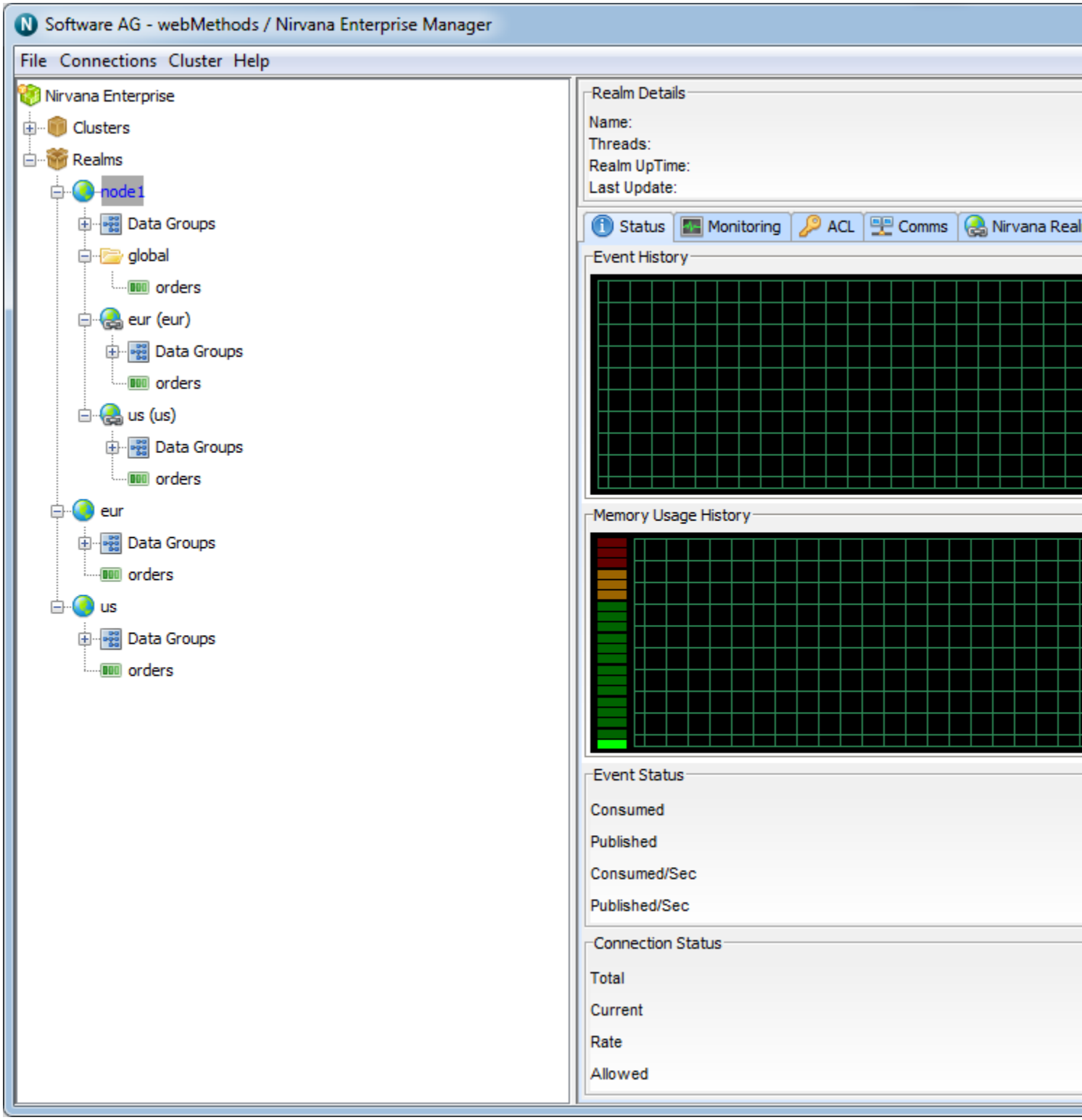
In order to add a realm to another realm, first of all you need to select the realm node from the namespace that you wish to add the realm to. Then, right-click on the realm node to display the

menu options available for a realm node. One of the menu options is labelled 'Add Realm to Namespace', clicking on this menu option will display a dialog that allows you to enter the RNAME of the realm you wish to add and an optional mountpoint. This dialog is shown in the image below.



The RNAME value in the dialog corresponds to the realm interface you wish the 2 realms to communicate using. The mount point corresponds to the point within the namespace that the realm will be referenceable.

The image below shows the namespace for a realm that has had 2 realms mounted within its namespace, called 'eur' and 'us' respectively. As you can see the resources within both the mounted realms are also displayed as part of the namespace of the 'node1' realm.



Sessions connected to the 'node1' realm now have access to three channels. These are :

- '/global/orders' which is a local channel

- '/eur/orders' which is actually a channel on another Universal Messaging Realm which has been added to this namespace under the mountpoint '/eur'
- '/us/orders' which is actually a channel on another Universal Messaging Realm which has been added to this namespace under the mountpoint '/us'

Example Use of Federation : Remote Joins

Once you have added the realms to one another, it is possible to create remote joins between the channels of the realms. This is very useful when considering the physical distance and communications available between the different realms. For example, if you wish all events published to the /customer/sales channel in the UK realm to be available on the /customer/sales channel in the US realm, one would create a join from the /customer/sales channel in the UK to the /customer/sales channel on the US realm, so all events published onto the uk channel would be sent to the us channel.

Federation and remote joins provide a huge benefit for your organization. Firstly, any consumers wishing to consume events from the uk channel would not need to do so over a WAN link, but simply subscribe to their local sales channel in the us. This reduces the required bandwidth between the us and uk for your organization, since the data is only sent by the source realm once to the joined channel in the us, as opposed to 1...n times where n is the number of consumers in the us. Remote joins are much more efficient in this respect, and ensure the data is available as close (physically) to the consumers as possible.

Note:
For a description of the general principles involved in creating channel joins, see the section *Creating Channel Joins*. The description details the usage based on the Enterprise Manager, but the same general principles apply if you are using the API.

Realm Configuration

Universal Messaging Realms can be configured based on a number of properties that are accessible both through the Universal Messaging Administration API as well as the Universal Messaging Enterprise Manager. Any changes made to the configuration properties for a Universal Messaging realm are automatically sent to the realm and implemented. This functionality offers major benefits to Administrators, since realms can be configured remotely, without the need to be anywhere near the actual realm itself. More importantly, multiple realms and clustered realms can also be automatically configured remotely.

Note:
Some Universal Messaging realm properties, such as the AMQP Message Transformation setting, are applied on a per-connection basis, meaning that clients must re-connect to pick up a change in the realm-wide value.

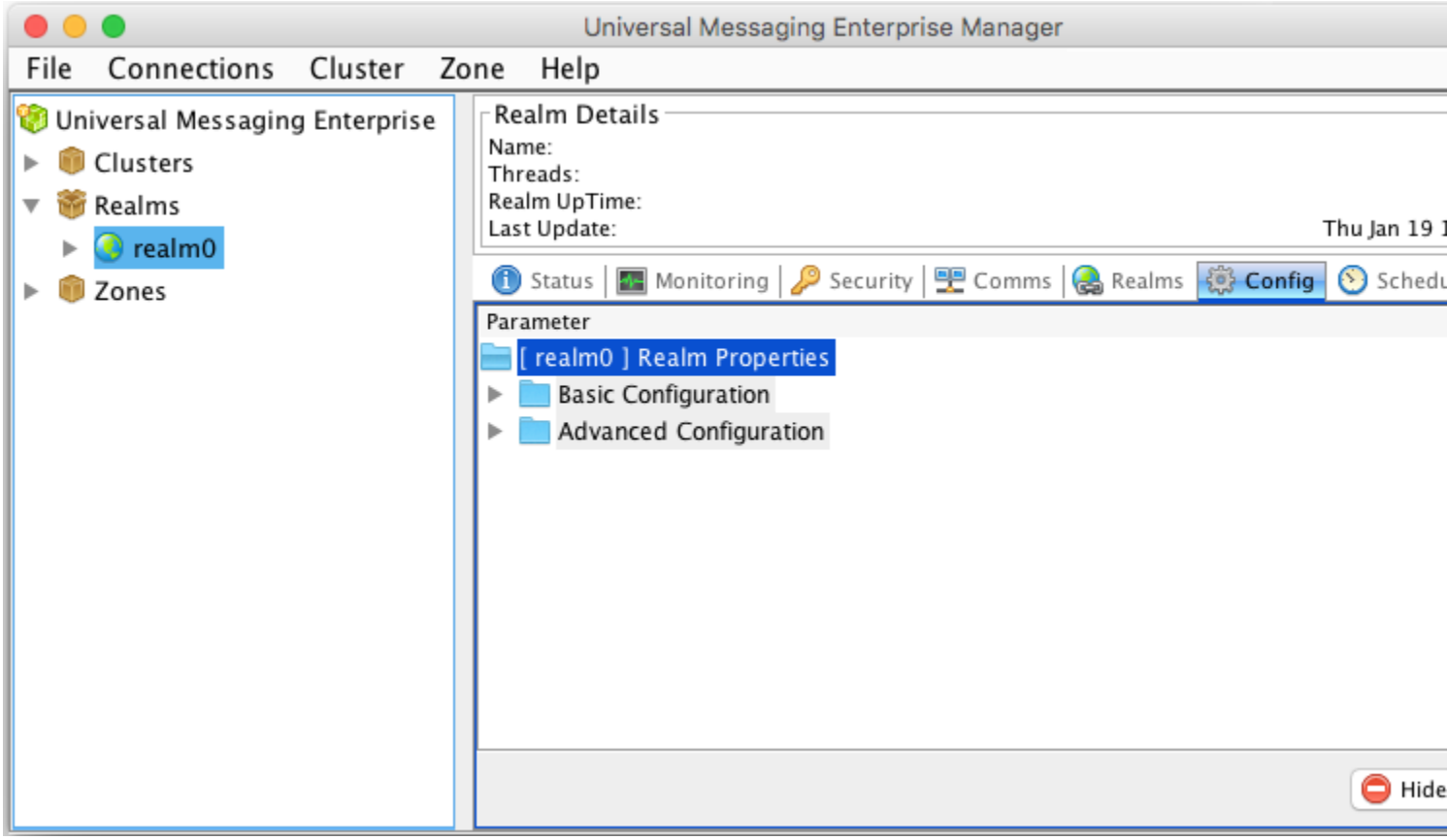
This section describes the different configuration properties that are available using the Universal Messaging Enterprise Manager.

When you select a realm from the namespace, one of the available panels in the Enterprise Manager is labelled 'Config'. Selecting this panel displays various groups of configuration properties, with

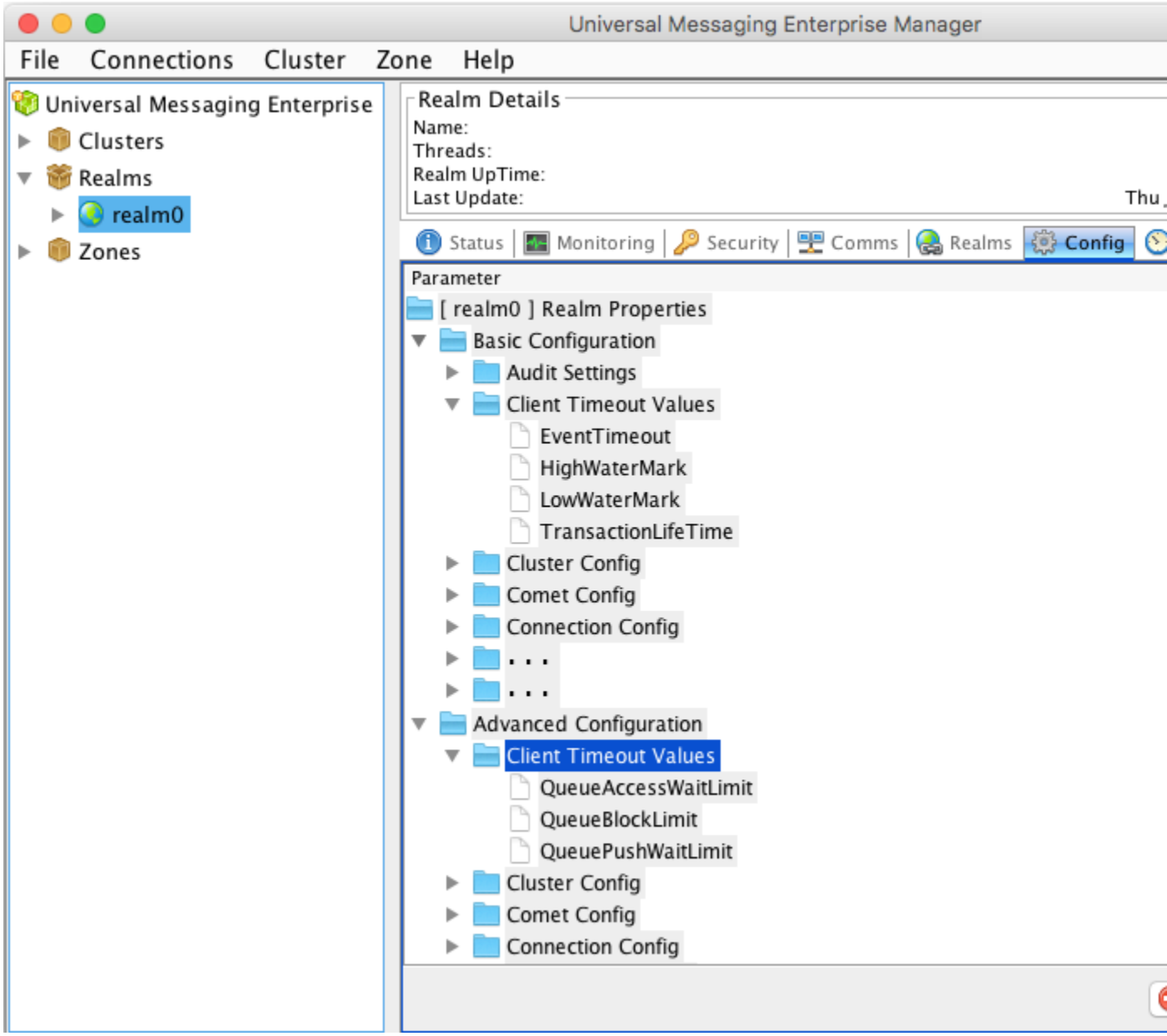
each group of properties relating to a specific area within the Universal Messaging Realm. Each group of properties contains different values for specific items.

Basic and Advanced Properties

There are currently a large number of configuration properties, and they are divided into two categories, namely Basic and Advanced. The properties in the Basic category are the most commonly used ones. The properties in the Advanced category will probably be less frequently used, and are intended for special cases or expert users.



When the Basic and Advanced categories are expanded, you will see a display of the configuration properties. Properties that have a similar effect are arranged into groups; for example, properties that determine when a client times out are contained in the group "Client Timeout Values":



Note that in the example shown, the group "Client Timeout Values" appears in both the Basic and the Advanced category. However, the properties "EventTimeout", "HighWaterMark" etc. belonging to this group appear only under the Basic category, whereas the properties "QueueAccessWaitLimit" etc. belonging to the same group appear only under the Advanced category. The properties in the Basic category are the ones which you will probably find most useful for your day-to-day work.

Configuration Groups

The configuration groups are :

1. **Audit Settings** - Values relating to what information is stored by the audit process
2. **Client Timeout Values** - Values relating to client / server interaction

3. **Cluster Config** - Values specific to the clustering engine
4. **Comet Config** - Values relating to the configuration of Comet
5. **Connection Config** - Values relating to the client server connection
6. **Data Stream Config** - Values relating to the configuration of Data Streams
7. **DurableConfig** - Values relating to usage of durables
8. **Environment Config** - Read only configuration values that relate to the system environment. These cannot be changed.
9. **Event Storage** - Values specific to how events are stored and retrieved on the server
10. **Fanout Values** - Values specific to the delivery of events to clients
11. **Global Values** - Values specific to the realm process itself
12. **Inter-Realm Comms Config** - Values relating to Inter-Realm communication
13. **JVM Management** - Values relating to the JVM the Realm Server is using
14. **Join Config** - Values specific to channel join management
15. **Logging Config** - Values specific to logging
16. **Metric Config** - Values relating to metric management
17. **Plugin Config** - Values relating to Realm Plugins
18. **Protobuf Config** - Values relating to Protocol Buffers
19. **Protocol AMQP Config** - Values relating to the use of AMQP connections
20. **Protocol MQTT Config** - Values relating to the use of MQTT connections
21. **RecoveryDaemon** - Values relating to clients that are in recovery (i.e. replaying large numbers of events)
22. **Server Protection** - Values specific to server protection
23. **Thread Pool Config** - Values specific to the servers thread pools.
24. **TransactionManager** - Values specific to the transaction engine of the RealmServer

The table below describes the properties that are available within each configuration group. It also shows valid ranges of values for the properties and a description of what each value represents. The “Adv. ” column shows "Y" if the property is in the Advanced category, whereas no entry indicates that the property is in the Basic category.

Configuration Group/Property	Valid values	Description	Adv.
Audit Settings			

Configuration Group/Property	Valid values	Description	Adv.
ChannelACL	True or False	Log to the audit file any unsuccessful channel ACL interactions. Default is `true`.	
ChannelFailure	True or False	Log to the audit file any unsuccessful realm interactions. Default is `true`.	
ChannelMaintenance	True or False	Log to the audit file any channel maintenance activity. Default is `false`.	
ChannelSuccess	True or False	Log to the audit file any successful channel interactions. Default is `false`.	
DataGroup	True or False	Log to the audit file any changes to Data Group structure	
DataGroupFailure	True or False	Log to the audit file any failed attempts to Data Group structure	
DataStream	True or False	Log to the audit file Data Stream add and removes	
Group	True or False	Log to the audit file any added or removed security groups	
GroupMembers	True or False	Log to the audit file any changes in group membership	
InterfaceManagement	True or False	Log to the audit file any interface management activity. Default is `true`.	
JoinFailure	True or False	Log to the audit file any unsuccessful join interactions. Default is `true`.	
JoinMaintenance	True or False	Log to the audit file any join maintenance activity. Default is `true`.	
JoinSuccess	True or False	Log to the audit file any successful join interactions. Default is `false`.	
QueueACL	True or False	Log to the audit file any unsuccessful queue ACL interactions. Default is `true`.	
QueueFailure	True or False	Log to the audit file any unsuccessful queue interactions. Default is `true`.	
QueueMaintenance	True or False	Log to the audit file any queue maintenance activity. Default is `false`.	

Configuration Group/Property	Valid values	Description	Adv.
QueueSuccess	True or False	Log to the audit file any successful queue interactions. Default is `false`.	
RealmACL	True or False	Log to the audit file any unsuccessful realm ACL interactions. Default is `true`.	
RealmFailure	True or False	Log to the audit file any unsuccessful realm interactions. Default is `true`.	
RealmMaintenance	True or False	Log to the audit file any realm maintenance activity. Default is `true`.	
RealmSuccess	True or False	Log to the audit file any successful realm interactions. Default is `false`.	
SnoopStream	True or False	Log to the audit file Snoop stream add and removes	
Client Timeout Values			
EventTimeout	5000 to No Max	The amount of ms the client will wait for a response from the server. Small values may cause clients to abandon waiting for responses and disconnect prematurely. Large values may cause clients to take an unusually long amount of time waiting for a response before disconnecting. Default is `60000`.	
HighWaterMark	2 to No Max	The high water mark for the connection internal queue. When this value is reached the internal queue is temporarily suspended and unable to send events to the server. This provides flow control between publisher and server. Default is `3000`.	
LowWaterMark	1 to No Max	The low water mark for the connection internal queue. When this value is reached the outbound internal queue will again be ready to push event to the server. Default is `1000`.	
QueueAccessWaitLimit	200 to No Max	The maximum number of milliseconds it should take to gain access to an internal connection queue to push events. Once this time has elapsed the	Y

Configuration Group/Property	Valid values	Description	Adv.
		client session will inform any listeners registered on the session which monitor these connection queues. Small values may result in an excessive number of notifications. Default is `200`.	
QueueBlockLimit	500 to No Max	The maximum number of milliseconds an internal connection queue will wait before notifying listeners after it has reached the `HighWaterMark`. Small values may result in excessive notifications. Default is `500`.	Y
QueuePushWaitLimit	200 to No Max	The maximum number of milliseconds it should take to gain access to an internal connection queue and to push events before notifying listeners. Small values may result in excessive notifications. Default is `200`.	Y
TransactionLifeTime	1000 to No Max	The default amount of time a transaction is valid before being removed from the tx store. Default is `20000`.	
Cluster Config			
ClientQueueSize	10 to 10000	Size of the client request queue. If this queue is small then the clients will wait longer and performance may drop. If too large then client requests are queued but not processed.	
ClientQueueWindow	1 to 1000	The value used when an async consumer of type queue or durables of types shared queue, shared, and serial do not set an explicit value for the window size. Default is `100`. **Important:** A small number will reduce performance.	

Configuration Group/Property	Valid values	Description	Adv.
ClientStateDelay	0 to 120000	The number of seconds to delay the cluster processing client requests when a cluster state change occurs. A large number will delay client requests longer than required.	
DisableHTTPConnections	True or False	Disable HTTP(s) connections between cluster nodes. If true then the server will only use nsp(s) connections between realm nodes, and any nhp(s) rnames will be switched to using nsp(s).	
DisconnectWait	1000 to 120000	Time to wait for the node to form in the cluster. Once this time has expired the behavior is defined by the `DisconnectWhenNotReady` flag.	
DisconnectWhenNotReady	True or False	Disconnects the client if the node has not formed in the cluster. If set to `true`, all non-Admin client sessions are disconnected, but the Admin sessions are not disconnected. If set to `false`, the user request will be queued.	
EnableMulticast	True or False	Enables cluster requests broadcast to realms to be send through the reliable multicast mechanism within Universal Messaging. This setting only takes effect if a multicast interface is configured for all nodes within the cluster.	
EnableStoreRecoveryRetry	True or False	Enables/Disables the ability for the slave to re-attempt a recovery of a store if it detects changes to the store during recovery. If `true` the slave will continue to attempt a cluster recovery of a store which may be changing due to TTL or capacity on the store attributes.	Y
EnginePipelineSize	1 to 32	Number of concurrent pipeline threads running within the cluster engine. If set to `1`, then all requests are pipelined through one thread, else topics/queues are bound to specific pipelines.	Y

Configuration Group/Property	Valid values	Description	Adv.
FormationTimeout	60000 to 300000	The time to wait for the state to move from recovery to slave or master. If this value is too small then recovering a large number of events will result in the realms dropping out of the cluster.	
HeartBeatInterval	1000 to 120000	Heart Beat interval in milliseconds. Default is `120000`. A small value here will cause excessive messages being generated between realms.	
InitialConnectionTimeout	5000 to 240000	The number of milliseconds that the server will wait while trying to establish a connection to a peer. A small value may reduce the chance of a connection in busy networks, while a large number may delay cluster formation.	
IsCommittedDelay	1000 to 30000	When a slave processes an IsCommitted request and it is still recovering the Transaction store, it will block the clients request for this timeout period. If this is set to a large value, clients may experience a substantial delay in response.	Y
MasterRequestTimeout	1000 to 900000	Specifies the amount of time in milliseconds that the master is going to wait for a slave to respond to a single request before disconnecting it. This timeout will prevent a slave from being reconnected if it fails to respond to a master request.	
MasterVoteDelay	1000 to 60000	When a node has requested to be master it will wait this timeout period in milliseconds for the peers to agree. If this number is too high the cluster formation may take some time.	
MasterWaitTimeout	1000 to 600000	When the master is lost from the cluster and the remaining peers detect that the master has the latest state they will wait for this time period for the master to reconnect. If the master fails to	

Configuration Group/Property	Valid values	Description	Adv.
		reconnect in this time period a new master is elected.	
PublishQueueEnabled	True or False	If enabled the slaves will queue publish requests prior to committing them to the cluster. If enabled and a slave is killed, any outstanding publish events will be lost.	Y
QueueSize	100 to 1000	Number of events outstanding to be processed by the clusters internal queue before sending flow control requests back. Increased size increases the memory usage.	
StateChangeScan	10000 to No Max	When a realm loses master or slave state then after this timeout all cluster based connections will be disconnected. If the realm reenters the cluster then the disconnect timeout is aborted. If this value is too low, all clients will be bounced while the cluster is forming.	
SyncPingSize	100 to 10000	Number of events sent before a cluster sync occurs. A small number will effect overall performance, a large number may result in a cluster being to far out of sync.	Y
Comet Config			
BufferSize	1024 to 102400	The buffer size for Comet requests. Large sizes will cause the realm to consume more memory when reading data from Comet clients. Small sizes may introduce delays in the time taken to read requests.	Y
EnableLogging	True or False	Enables logging of all comet queries, will impact server performance.	
Timeout	10000 to No Max	The timeout for a Comet connection. Small sizes may cause Comet-based connections to time out prematurely. Large sizes may increase the time a	

Configuration Group/Property	Valid values	Description	Adv.
		server holds a disconnected Comet connection open.	
Connection Config			
AllowBufferReuse	True or False	If set to `true` then buffers will be allocated from the buffer pool and once finished with returned to the pool. If set to `false` then buffers are allocated on the fly and then left for the system to free them. It is best to leave this set to `true`. For object creation limitation it is best to set this to `true`.	Y
BufferManagerCount	1 to 256	The number of Buffer Managers that the server will allocate. This is used during startup to size and manage the network buffers. This does not need to be large, but a rule of thumb is `1` per core.	Y
BufferPoolSize	100 to 10000	The underlying Universal Messaging IO utilizes buffers from a pool. By default we pre-load the pool with this number of buffers. As the reads/writes require buffers they are allocated from this pool, then once used are cleared and returned. If the size is too small we end up creating and destroying buffers, and the server may spend time creating them when needed. If the size is too large we have a pool of buffers which are not used taking up memory.	Y
BufferSize	1024 to 1048576	This specifies the default size of the network buffers that Universal Messaging uses for its NIO. If small, then Universal Messaging will require more buffers (up to the maximum specified by `BufferPoolSize`) to send an event. If too large, then memory may be wasted on large, unused buffers. These buffers are reused automatically by the server, and are used to transfer data from the upper application layer	

Configuration Group/Property	Valid values	Description	Adv.
		to the network. So, for example, the server might use all `BufferPoolSize` buffers to stream from 1 application level buffer (depending on the relative sizes of the buffers). An efficient size would be about 40% more than the average client event, or 5K (whichever is largest). If too small, the server will send many small buffers.	
CometReadTimeout	1000 to 120000	Specifies the time the server will wait for a client to complete sending the data	
ConnectionDelay	10 to 60000	When the server has exceeded the connection count, how long to hold on to the connection before disconnecting. If this is too low, the server will be busy with reconnection attempts. Default is `60000`.	Y
IdleDriverTimeout	120000 to No Max	Specifies the time in milliseconds that a communications driver can be idle before being deemed as inactive. When this happens the server will automatically close and remove the driver. This must be greater than the keep alive timeout else all connections will be closed due to inactivity.	
IdleSessionTimeout	10000 to No Max	If there has been no communication from a client for the configured number of milliseconds, the client is deemed idle and is disconnected. This typically occurs when there are network issues between a client and the server. If the value is too low, the chance of disconnecting a valid session is high.	
KeepAlive	5000 to No Max	The number of milliseconds the server will wait before sending a heartbeat. A small number will cause undue network traffic. Default is `60000`.	
MaxBufferSize	1024 to No Max	The maximum buffer size in bytes that the server will accept. Default is `20971520` (20MB).	

Configuration Group/Property	Valid values	Description	Adv.
		Rather than using larger buffers, it is recommended that you compress if possible to save bandwidth and memory on the server. This value exists to stop a user from accidentally or maliciously overloading the server and causing excessive memory consumption.	
MaxNoOfConnections	-1 to No Max	Sets the maximum concurrent connections to the server, `-1` indicates no restriction, default is `-1`. Reducing this to a small number may cause client connections to be rejected.	
MaxWriteCount	5 to 100	When writing many events to a client the write pool thread may continue to send the events before returning to the pool to process other clients requests. So, for example if it is set to `5`, then the thread will send 5 events from the clients queue to the client before returning to the pool to process another request. If this number is small it creates additional CPU overhead.	
NetworkMonitorThreads	2 to 100	The number of threads to allocate to flushing client data, Please note this will only take effect after a restart. Depending on the number of concurrent clients the latencies during load my be higher then expected	
PriorityQueueCount	2 to 10	Sets the number of queues to divide priority levels between, up to a maximum of `10` queues.	
PriorityReadSpinLockMaxConnections	0 to 8	Maximum number of clients allowed to allocate high priority spin locks. This property is deprecated and will be removed in a future product release.	
PriorityReadSpinLockTime	1 to 10000	The time interval (in milliseconds), during which the thread spin read handler will continuously try reading	Y

Configuration Group/Property	Valid values	Description	Adv.
		events. The setting has effect only when `PriorityReadType` is set to `Thread Spin`. Default value is `500` milliseconds. This property is deprecated and will be removed in a future product release.	
PriorityReadType	0 to 2	If enabled then high priority sessions will be enabled to run spin locks waiting to read. This property is deprecated and will be removed in a future product release.	
QueueHighWaterMark	100 to No Max	The number of events in a client output queue before the server stops sending events. A small number will cause undue work on the server. Default is `3000`.	
QueueLowWaterMark	50 to No Max	The number of events in the clients queue before the server resumes sending events. Must be less than the high water mark. Default is `1000`.	
ReadCount	1 to 20	Number of times the thread will loop around waiting for an event to be delivered before returning. Large values may cause read threads to be held for long periods of time, but avoid context switching for delivering events.	
UseDirectBuffering	True or False	If true the server will allocate `DirectByteBuffers` to use for network I/O, else the server will use `HeapByteBuffers`. The main difference is where the JVM will allocate memory for the buffers the `DirectByteBuffers` perform better. For the best performance the `DirectByteBuffers` are generally better.	Y
WriteHandlerType	1 to 5	Specifies the type of write handler to use	
whEventThresholdCount	1 to 2000	Number of events to exceed in the `whEventThresholdTime` to detect a peak.	

Configuration Group/Property	Valid values	Description	Adv.
		This number should be small enough to trigger peaks.	
whEventThresholdTime	1 to 2000	Number of milliseconds to sample the event rate to detect peaks	
whMaxEventsBeforeFlush	1 to 10000	Total number of events that can be sent before a flush must be done. If this number is too small then too many flushes will result.	
whMaxEventsPerSecond	No Min to No Max	Specifies the total number of events per second that a realm will send to clients before switching modes into peak mode. If this number is small then the server will go into peak mode too soon and latencies will start to increase.	
whMaxTimeBetweenFlush	1 to 1000	Total number of milliseconds to wait before a flush is done. If this number is too large then latencies will increase.	
Data Stream Config			
MonitorTimer	1000 to 120000	Time interval in milliseconds to scan the data group configuration looking for idle / completed streams. Large values may cause idle and inactive datastreams to remain on datagroups for long periods of time. Small values may cause transient disconnections to trigger datagroup removals for datastreams - requiring them to be added back into the datagroup.	Y
OffloadMulticastWrite	True or False	If true then all multicast writes will be performed by the parallel fanout engine.	Y
SendInitialMapping	True or False	When any stream registered client connect sends the entire Data Group Name to ID mapping	Y
DurableConfig			

Configuration Group/Property	Valid values	Description	Adv.
QueuedExtendedException	True or False	If `true`, then if the selector on a queued durable changes, the selector is added to the exception string.	Y
Environment Config			
AvailableProcessors	READ ONLY	Number of CPUs available	
Embedded	READ ONLY	If true, this specifies that the server is running as an embedded server	
InterRealmProtocolVersion	READ ONLY	Universal Messaging Server Inter-Realm Protocol Version	
JavaVendor	READ ONLY	Vendor of Java Virtual Machine	
JavaVersion	READ ONLY	Virtual Machine Version	
NanosecondSupport	READ ONLY	Nanosecond support available through JVM on Native OS	
OSArchitecture	READ ONLY	Operating System Architecture	
OSName	READ ONLY	Operating System Name	
OSVersion	READ ONLY	Operating System Version	
ProcessId	READ ONLY	Process ID	
ServerBuildDate	READ ONLY	Universal Messaging Server Build Date	
ServerBuildNumber	READ ONLY	Universal Messaging Server Build Number	
ServerReleaseDetails	READ ONLY	Universal Messaging Release Details	
ServerVersion	READ ONLY	Universal Messaging Server Build Version	

Configuration Group/Property	Valid values	Description	Adv.
TimerAdjustment	READ ONLY	The size of the Operating System's time quantum.	
Event Storage			
ActiveDelay	100 to No Max	The time in milliseconds that an active channel will delay between scans. The smaller the number, the more active the server. Default is `1000`.	Y
AutoDeleteScan	1000 to 500000	Specifies the number of milliseconds between scans on `AutoDelete` stores to see if they should be deleted. The larger this time frame, the more `AutoDelete` stores will potentially not be deleted on the server.	
AutoMaintenanceThreshold	0 to 100	Sets the percentage free before the server should run maintenance on the internal stores. It is by default `50`. This means maintenance will be performed when 50% of the number of the events in the file are marked as dead – already consumed and acknowledged so they can be deleted.	
CacheAge	1000 to No Max	The length of time in milliseconds that cached events will be kept in memory. The larger the value, the more memory will be utilized.	Y
EnableStoreCaching	True or False	If `true` the server will try to cache events in memory after they have been written/read. Please note the server will need to be rebooted for this to take effect	
IdleDelay	5000 to No Max	The time in milliseconds that an idle channel will delay between scans. The smaller the number, the more active the server. Default is `10000`.	Y
JMSEngineAutoPurgeTime	5000 to 600000	Defines the interval between clean up of events on a JMS Engine Resource. A large interval may result in topics with	Y

Configuration Group/Property	Valid values	Description	Adv.
		large numbers of events waiting to be purged.	
MaintenanceFileSizeThreshold	1024000	Sets the file size in bytes that will trigger the maintenance. With small file sizes, the maintenance will be run more often. With large file sizes, the maintenance may take longer to run. There are no performance issues with a big file except on startup when the stores need to be reloaded.	
MaintenanceMemoryThreshold	1048576	Maximum size in memory for any topic or queue to reach before maintenance of the in-memory cache is run.	
PageSize	10 to 100000	The page size to use for the event store. This value sets the number of events/page.	
QueueDeliveryPersistencePolicy	(Values as listed in next column)	Sets the Queue Delivery Persistence Policy. The policy is a combination of (a) making the disk storage location persistent, i.e. recoverable after a server restart, or non-persistent, i.e. erased at a server restart, and (b) writing events to the disk storage location synchronously or asynchronously. If you choose a policy that uses a non-persistent storage location, unacknowledged but delivered queue events will be stored elsewhere until they are acknowledged or rolled back. The available policies are: ■ `No persistent/No sync`: The storage location is not persistent, and writing events to disk is asynchronous. ■ `Persistent/No sync`: The storage location is persistent, and writing events to disk is asynchronous.	Y

Configuration Group/Property	Valid values	Description	Adv.
		■ `Persistent/Sync`: The storage location is persistent, and writing events to disk is synchronous. `Persistent/No sync` is the recommended value for production environments.	
StoreReadBufferSize	1024 to 3000000	Size of the buffer to use during reads from the store. Note that the server will need to be restarted for this to take effect.	Y
SyncBatchSize	1 to 1000	Specifies the maximum size before the sync call is made. The lower this value, the more sync calls made and the more overhead incurred.	Y
SyncServerFiles	True or False	If `true` the server will sync each file operation for its internal files. If `true`, this adds additional overhead to the server machines and can reduce overall performance.	
SyncTimeLimit	1 to 1000	Specifies the maximum time in milliseconds that will be allowed before the sync is called. The lower this value, the more file sync calls and the more overhead incurred.	Y
ThreadPoolSize	1 to 4	The number of threads allocated to perform the management task on the channels. The more channels a server has, the larger this number should be.	
Fanout Values			
ConnectionGrouping	True or False	If `true` allows the server to group connections with the same selector providing improved performance. This allows the server to optimize the way it processes events being delivered to the clients. This requires a server restart to take effect.	Y

Configuration Group/Property	Valid values	Description	Adv.
DelayPublishOnCapacity	True or False	Delays the publisher thread when the store capacity is exceeded. If this is not set, an exception is passed back to the client.	
HonourSharedDurableCapacity	True or False	If `true`, the channel will check any shared durables for capacity before accepting a published event. If any of these durables are over capacity, the server will respond as if the parent channel is over capacity. If `false`, the event will be published regardless of the number of events on its shared durables.	Y
IteratorWindowSize	1 to No Max	Specifies the number of events delivered to each Channel Iterator in a pre fetch. This allows the client to perform much faster by pre fetching events on fast moving topics requiring less client to server communication. The default is `100`.	
JMSQueueMaxMultiplier	1 to 10	The multiplier used on the High Water mark when processing events from a JMS Engine Queue/Topic. If this value is too high the server will consume vast amounts of memory.	Y
ParallelThreadPoolSize	2 to 64	Specifies the number of threads to use within the thread pool. If this number is small then there maybe adverse overheads. This value required a restart to take effect.	
PeakPublishDelay	0 to No Max	When clients start to hit high water mark, this specifies how long to delay the publisher to allow the client time to catch up. If this is too small the publisher can overwhelm the server.	Y
PublishDelay	0 to No Max	How long to delay the publisher when the subscriber's queue start to fill, in milliseconds. If this number is `0` then no delay.	Y

Configuration Group/Property	Valid values	Description	Adv.
PublishExpiredEvents	True or False	Specifies whether to publish expired events at server startup. Default is `true`.	
SendEndOfChannelAlways	True or False	Specifies whether to always send an End Of Channel, even if we find no matches within the topic. If set, the subscriber will always be informed that the subscription request has completed the recovery of the topic.	
SendPubEventsImmediately	True or False	Specify whether to send publish events immediately. If `true`, then the server will send all publish events to clients immediately, if `false` the server is allowed to collect events before publishing.	
SyncQueueDelay	10 to 3600000	Maximum number of milliseconds the queue publisher will be delayed. This can be used to slow down the queue publishers.	
SyncQueuePublisher	True or False	If true then the queue publisher will be synchronized with the queue consumers. This allows flow control of queue publishers. If `false` then the value of `SyncQueueDelay` is not used.	
Global Values			
AllowRealmAdminFullAccess	True or False	If `true` then any user with the full realm access will have access to all channels and queues.	Y
CacheJoinInfoKeys	True or False	If enabled we cache join key information between events passed over joins. This reduces the number of objects created. If this parameter is set to false then the server will create a new byte[] and string for each joined event.	Y
DisableExplicitGC	True or False	If enabled the server will call the Garbage Collector at regular intervals to keep memory usage down. If this is disabled then the garbage collection will be done solely by the JVM.	

Configuration Group/Property	Valid values	Description	Adv.
EnableCaching	True or False	If `EnableCaching` is set to `true`, the channel storage properties **Cache On Reload** and **Enable Caching** are set to the values specified by the client. If `EnableCaching` is set to `false`, then the channel storage properties **Cache On Reload** and **Enable Caching** are set to False, regardless of the values set by the client for these storage properties.	
EnableDNSLookups	True or False	If enabled the server will attempt to perform a DNS lookup when a client connects to resolve the IP address to a hostname. In some instances this may slow down the initial client connections.	
EnableWeakReferenceCleanup	True or False	If enabled then the server will hook into the JVM's garbage collection and release cached items when the JVM needs memory. By enabling this, the number of cached events stored will be reduced but memory will be maintained.	Y
ExtendedMessageSelector	True or False	If `true`, allows the server to use the extended message selector syntax (enabling string to numeric conversions within the message selector). Default is true.	
HTTPCookieSize	14 to 100	The size in bytes to be used by nhp(s) cookies	Y
NHPScanTime	5000 to No Max	The number of milliseconds that the server will wait before scanning for client timeouts. Default is `5000`, i.e. 5 seconds.	Y
NHPTimeout	2000 to No Max	The number of milliseconds the server will wait for client authentication. If this number is too large, the server may have unwanted connections. Default is `120000`, i.e. 2 minutes.	Y

Configuration Group/Property	Valid values	Description	Adv.
OverrideEveryoneUser	True or False	Override the *@* permission for channels / queues with explicit ACL entry permissions. Default is `false`.	
PauseServerPublishing	True or False	If `true`, the Pause Publishing feature is activated. Default is `false`. This feature causes the server to block all attempts by clients to publish events, and such clients will receive an `nPublishPausedException`. However, events that already exist in the publishing client queues on the server continue to be consumed by the subscribing clients until the queues are emptied. You can use the Pause Publishing feature when it is necessary to clear the client event queues on the realm server. This could be, for example, before performing maintenance tasks such as increasing buffer storage or performing a backup, or before changing the server configuration.	Y
SendRealmSummaryStats	True or False	If `true` sends the realm's status summary updates every second. Default is `false`.	
StampDictionary	True or False	Place Universal Messaging details into the dictionary. The default is `true`.	
StampHost	True or False	Stamps the header with the publishing host (true/false). If `true` adds additional overhead to the server/client.	
StampTime	True or False	Stamps the header with the current time (true/false). If `true`, adds additional overhead to the server/client.	
StampTimeUseHPT	True or False	If this is set to `true`, then the server will use an accurate millisecond clock, if available, to stamp the dictionary. This may impact overall performance when delivering events when latency is important.	

Configuration Group/Property	Valid values	Description	Adv.
StampTimeUseHPTScale	0 to 2	This has 3 values, `milli`, `micro` or `nano` accuracy	
StampUser	True or False	Stamps the header with the publishing user (true/false). If true, adds additional overhead to the server/client.	
StatusBroadcast	2000 to No Max	This property has two purposes: ■ The number of milliseconds between status events being published to any clients using Admin API or Enterprise Manager. A small value increases the server load. ■ The number of milliseconds between status messages being written to the server log, when periodic status logging has been activated via the `EnableStatusLog` property. Remember that if you change the value of this property, it will affect the time interval for both status events and status log intervals. The default is `5000`, i.e. every 5 seconds.	
Inter-Realm Comms Config			
EstablishmentTime	10000 to 120000	Time for an inter-realm link to be initially established. This value should reflect the latency between nodes.	Y
KeepAliveInterval	1000 to 120000	Time interval where if nothing is sent a Keep Alive event is sent. This can be used to detect if remote members are still up and functioning.	Y
KeepAliveResetTime	10000 to 180000	If nothing has been received for this time the connection is deemed closed. This value must be larger than the `KeepAliveInterval`.	Y

Configuration Group/Property	Valid values	Description	Adv.
MaximumReconnectTime	1000 to 50000	The maximum number of milliseconds to wait before trying to re-establish a connection. If this value is too large, cluster formation will be delayed. The reconnect will be attempted at a random amount of time between `MinimumReconnectTime` and `MaximumReconnectTime`.	Y
MinimumReconnectTime	100 to 10000	The minimum time to wait before trying to re-establish a connection. If this number is too high then it may impact the network during outages. The reconnect will be attempted at a random amount of time between `MinimumReconnectTime` and `MaximumReconnectTime`.	Y
Timeout	60000 to 180000	If no events are received within this time limit, the link is assumed dead and will be closed. If this limit is less than the keep alive time then the link will be closed.	Y
WriteDelayTimeout	1000 to 60000	The maximum time to wait on a write if the link has dropped. If a realm disconnects when we are able to write to it, we wait for a set amount of time for the link to come back before abandoning the write and resetting altogether. This insulates the cluster against some transitive network conditions.	Y
JVM Management			
EmergencyThreshold	50 to 99	The memory threshold when the server starts to aggressively scan for objects to release. If this value is too large the server may run out of memory. Default is `94`, i.e. 94%	
EnableJMX	True or False	Enable JMX beans within the server. If enabled the server will present JMX MBeans so it can be monitored by any JMX client.	

Configuration Group/Property	Valid values	Description	Adv.
ExitOnDiskIOError	True or False	If `true`, the server will exit if it gets an I/O Exception. Setting this to false may result in lost events if the server runs out of disk space. Default is true	Y
ExitOnInterfaceFailure	True or False	If `true` and for any reason an interface cannot be started when the realm initializes, the realm will shut down.	Y
IORetryCount	2 to 100	Number of times a file I/O operation will be attempted before aborting	Y
IOSleepTime	100 to 60000	Time between disk I/O operations if an I/O operation fails. If this time is large then the server may become unresponsive for this time.	Y
JMXRMIServerURLString	String	JNDI Lookup URL for the JMX Server to use.	
MemoryMonitoring	60 to 30000	Number of milliseconds between monitoring memory usage on the realm. If this value is too large then the realm will be slow to handle memory usage. Default is 2000.	
ThrottleAllPublishersAtThreshold	True or False	Defines if publishers will be throttled back when the memory emergency threshold is reached.	Y
WarningThreshold	40 to 95	The memory threshold when the server starts to scan for objects to release. If this value is small then the server will release objects too soon, resulting in a lower performing realm. Default is `85`, i.e. 85%.	
Join Config			
ActiveThreadPoolSize	1 to No Max	The number of threads to be assigned for the join recovery. Default is `2`.	
IdleThreadPoolSize	1 to No Max	The number of threads to manage the idle and reconnection to remote servers. This number should be kept small. Default is `1`.	

Configuration Group/Property	Valid values	Description	Adv.
MaxEventsPerSchedule	1 to No Max	Number of events that will be sent to the remote server in one run. A low number will increase the time to recover the remote server, a large number will impact other joins which are also in recovery. Default is `50`.	Y
MaxQueueSizeToUse	1 to No Max	The maximum events that will be queued on behalf of the remote server. A low number increases the time for the remote server to recover, a large number increases the memory used for this server. Default is `100`.	Y
RemoteJoinAckBatchSize		Events received through remote joins are acknowledged in batches. This property configures the batch size.	Y
RemoteJoinAckInterval		In addition to the batch acknowledgement, remote join events get acknowledged every n milliseconds. This property configures this interval.	Y
UseQueuedLocalJoinHandler	True or False	Specifies whether to use a queued join event handler. `True` will enable source channels and destination channels to be process events independently.	Y
Logging Config			
DefaultLogSize	100 to No Max	The default size of the log in bytes	
DisplayCurrentThread	True or False	If enabled will display the current thread in the log message. Default: `true`	
EnableLog4J	True or False	If enabled will intercept log messages and pass to Log4J as well. This requires a restart before it will take effect.	
EmbedTag	True or False	Used to control if the message tag is displayed in log messages. Default: `false`	

Configuration Group/Property	Valid values	Description	Adv.
EnableStatusLog	True or False	If true, periodic logging of the Universal Messaging server status is activated. The messages will be logged at time intervals given by the `StatusBroadcast` configuration property described in the *Global Values* section. The default is `true`.	
fLoggerLevel	0 to 6	The server logging level, between `0` and `6`, with `0` indicating very verbose, and `6` indicating very quiet. The more logging requested, the more overhead on the server. Default is `4`.	
LogManager	0 to 2	The Log manager to use. 0 = `ROLLING_OLD`, 1 = `ROLLING_DATE`, 2 = `ROLLING_NUMBER` Default: `ROLLING_DATE`	
RolledLogFileDepth	No Min to No Max	The number of log files to keep on disk when using log rolling. Oldest log files will be deleted when new files are created.	
Note: For further information about using the log file, see "Universal Messaging Enterprise Manager : Logs Panel" on page 20.			
Metric Config			
EnableEventMemoryMonitoring	True or False	If this is set to `true`, the server will make available memory usage.	
EnableMetrics	True or False	If this is set to `true`, the server will make available system metrics (e.g. memory usage).	
Plugin Config			

Configuration Group/Property	Valid values	Description	Adv.
EnableAccessLog	True or False	Defines if plugin access log is produced	
EnableErrorLog	True or False	Defines if plugin error log is produced	
EnablePluginLog	True or False	Defines if plugin status log is produced	
MaxNumberOfPluginThreads	10 to 10000	Maximum number of threads to allocate to the plugin manager	Y
PluginTimeout	1000 to 30000	Time in milliseconds that the plugin will read from a client. If too small, the plugin may not load all of the clients requests	Y
Protobuf Config			
CacheEventFilter	True or False	Hold the Protocol Buffer filter cache in memory. Default `true`.	Y
Protocol AMQP Config			
AnonymousUser	String	The user name to use for anonymous users	
BufferSize	1000 to 60000	The size of the buffer that will be used to read/write on the AMQP connection	
DefaultNodeMode	0 to 1	The type of node if it is not able to detect it. 0=`Queue`, 1=`Topic`	
Enable	0 to No Max	If true the server will accept incoming AMQP connections	
EnableWriteThread	True or False	Enables the off loading of the physical write to a thread pool	Y
EngineLoopCount	4 to 100	How many times the AMQP state engine will cycle per thread pool allocation	Y
MaxFrameSize	10000 to No Max	Maximum size of an AMQP frame	Y

Configuration Group/Property	Valid values	Description	Adv.
MaxThreadPoolSize	2 to 100	Largest number of threads the pool can have.	
MinThreadPoolSize	1 to 10	Smallest number of threads for the dedicated AMQP thread pool	
QueuePrefix	String	The address prefix for specifying topic nodes as required by some clients	
SASL_Anonymous	True or False	Enable Anonymous SASL	
SASL_CRAM-MD5	True or False	Enable CRAM-MD5 SASL	
SASL_DIGEST-MD5	True or False	Enable DIGEST-MD5 SASL	
SASL_Plain	True or False	Enable Plain SASL	
SubscriberCredit	100 to No Max	Sets the subscriber (receiver) credit	
Timeout	10000 to 300000	Sets the network timeout	Y
TopicPrefix	String	The address prefix for specifying topic nodes as required by some clients	
TransformToUse	0 to 3	Selects the type of transformation to use from AMQP style events to native UM events. 0 - `No transformation`, 1 - `Basic Transformation`, 2 - `Complete Transformation`, 3 - `User Configurable`	
Protocol MQTT Config			
DisconnectClientsOnPublishFailure	True or False	Defines whether the server should disconnect clients to inform them that publishing has failed. The default is `true`.	
Enable	True or False	If `true`, the server will accept incoming MQTT connections. The default is true	

Configuration Group/Property	Valid values	Description	Adv.
		if this feature is enabled in the product licence.	
EnableAutoCreateTopics	True or False	If `true`, the server will auto-generate Topics for MQTT clients for subscriptions and publishing. The default is `true`. **Note:** This property is only applicable for client IDs with no wildcard.	
EnforceAlphaNumericClientID	True or False	If `true`, the Client ID must consist solely of alphanumeric characters. The default is `false`. **Note:** This property is only applicable for MQTT 3.1.1.	Y
IgnoreClientIDLength	True or False	If `true`, ignore the standard Client ID maximum length check of 23 characters. The default is `true`. **Note:** This property is only applicable for MQTT 3.1.1.	Y
MaxOutstanding	100 to 64000	Sets the maximum number of events that the server will send before waiting for the client to acknowledge them (QoS:1 and above). The default is `64000`.	
QoS0AsTransient	True or False	If `true`, the server will not recover publish events with QoS greater than or equal to 0. The default is `false`. **Note:** This property is only applicable for MQTT 3.1.1.	Y
SessionStateTTL	0 to No Max	The number of milliseconds the state of a Client ID is kept between connections. The default value is set to `3` days. Setting this value to `0` will store the Client ID state until a clean session is received.	Y

Configuration Group/Property	Valid values	Description	Adv.
SupportZeroLength	True or False	If `true`, the server will auto-generate the Client ID if it has zero length. The default is `true`. **Note:** This property is only applicable for MQTT 3.1.1.	Y
Strict	True or False	To be compliant with MQTT, stores must use the JMS Engine. This flag enforces this check. The default is `true`.	Y
RecoveryDaemon			
EventsPerBlock	1 to No Max	The number of events to send in one block to a recovering connection. Small values may slow down the overall speed of recovery, however large values may saturate the recovery thread and keep it busy from performing recovery tasks for other stores and connections.	Y
ThreadPool	1 to No Max	Number of threads to use for client recovery	
Server Protection			
EnableFlowControl	True or False	Enables flow control of producer connections. Default is `true`.	
FlowControlWaitTimeOne	0 to 120000	The time in milliseconds to hold a producing connection before processing its events. This is the longest level of waiting.	Y
FlowControlWaitTimeTwo	0 to 120000	The time in milliseconds to hold a producing connection before processing its events. This is the second level of waiting.	Y
FlowControlWaitTimeThree	0 to 120000	The time in milliseconds to hold a producing connection before processing its events. This is the first level of waiting and the shortest wait time.	Y

Configuration Group/Property	Valid values	Description	Adv.
Thread Pool Config			
CommonPoolThreadSize	5 to 1000	Maximum number of threads to allocate to the common thread pool	
ConnectionThreadPoolMaxSize	10 to No Max	The maximum number of threads allocated to establish client connections. If this number is too small then connections may be left waiting for a thread to process it.	
ConnectionThreadPoolMinSize	4 to 100	The minimum number of threads allocated to establish client connections. If too large then the server will have many idle threads.	
ConnectionThreadWaitTime	10000 to 300000	The time for the thread to wait for the client to finalize the connection. If too low then slow linked clients may not be able to establish a connection.	Y
EnableConnectionThreadPooling	True or False	If `true` then if NIO is available it will be available for interfaces to use it and then all reads/writes will be done via the Read/Write thread pools. If NIO is not available then a limited used write thread pool is used. This requires a realm restart before it takes effect.	Y
MaxUnauthorisedCount	10 to 10000	The maximum outstanding unauthorized connections per hostname (or IP address if host name is unavailable)	Y
MultiplexReadThreadPoolMaxSize	4 to No Max	The maximum number of threads to allocate to the multiplex thread pool to read multiplex sessions. Default is `100`.	
MultiplexReadThreadPoolMinSize	4 to No Max	The minimum number of threads to allocate to the multiplex thread pool to read multiplex sessions. Default is `4`.	
PendingTaskWarningThreshold	100 to 100000	The threshold at which the server starts to warn about the number of pending tasks. When the number of pending tasks is below the threshold, but over	Y

Configuration Group/Property	Valid values	Description	Adv.
		100, the server logs a WARNING message. When the number is above the threshold, the server logs an ERROR message. When the server does not find available threads, it logs a message that the thread pool is exhausted. Default is 1000.	
ReadThreadPoolMaxSize	4 to No Max	The maximum number of threads that will be allocated to the read pool. If NIO is not available this should be set to the maximum number of clients that are expected to connect. If NIO is available then it's best to keep this number under 20.	
ReadThreadPoolMinSize	4 to No Max	This is the number of threads that will always be present in the read thread pool. If this is too small then the thread pool will be requesting new threads from the idle queue more often. If too large then the server will have many idle threads.	
SchedulerPoolSize	10 to 100	The number of threads assigned to the scheduler, default is 10.	
SlowTaskWarningTime	1000 to 30000	The time in milliseconds before reporting a slow-running task. The server logs the information at the WARNING log level and generates a thread dump. Default is 5000.	Y
StalledTaskWarningTime	10000 to 60000	The time in milliseconds before reporting a stalled task. The system writes the information at the WARNING log level and generates a thread dump. When you change this configuration, the thread pool monitor interval is updated to monitor at the same time interval as the value you specify for this property. Default is 60000.	Y
ThreadDumpInterval	1000 to 600000	The interval in milliseconds at which a thread dump is generated when the system reports slow or stalled tasks, or	Y

Configuration Group/Property	Valid values	Description	Adv.
		when the number of pending tasks exceeds the value of `PendingTaskWarningThreshold`. The thread dump interval applies across all thread pools in the JVM instance. Default is `60000`.	
ThreadDumpOnSlowTask	True or False	Whether to generate a thread dump when the system reports a slow task. Default is `false`.	Y
ThreadIdleQueueSize	5 to 50	When threads are released from various pools since they no longer need them they end up in the idle queue. If this idle queue exceeds this number the threads are destroyed. Specify this number to be large enough to accommodate enough idle threads, so that if any thread pool requires to expand then it can be reused. If the number is too large then the server may have many idle threads.	
WriteThreadPoolMaxSize	5 to No Max	The maximum number of threads that will be allocated to the write pool. If NIO is not available this should be set to the maximum number of clients that are expected to connect. If NIO is available then it's best to keep this number under `20`.	
WriteThreadPoolMinSize	5 to No Max	This is the number of threads that will always be present in the write thread pool. If this is too small then the thread pool will be requesting new threads from the idle queue more often. If too large then the server may have many idle threads.	
TransactionManager			
MaxEventsPerTransaction	0 to No Max	The maximum number of events per transaction, a `0` indicates no limit.	
MaxTransactionTime	1000 to No Max	Time in milliseconds that a transaction will be kept active. A large number will	

Configuration Group/Property	Valid values	Description	Adv.
		cause the server to retain these transactions in memory.	
TTLThreshold	1000 to 60000	The minimum time in milliseconds, below which the server will not store the Transaction ID.	Y

Double-clicking on the property you wish to modify in the configuration group will provide you with a dialog window where the new value can be entered. The values of configuration properties will be validated to check whether they are within the correct range of values. If you enter an incorrect value you will be notified.

Connecting to Multiple Realms

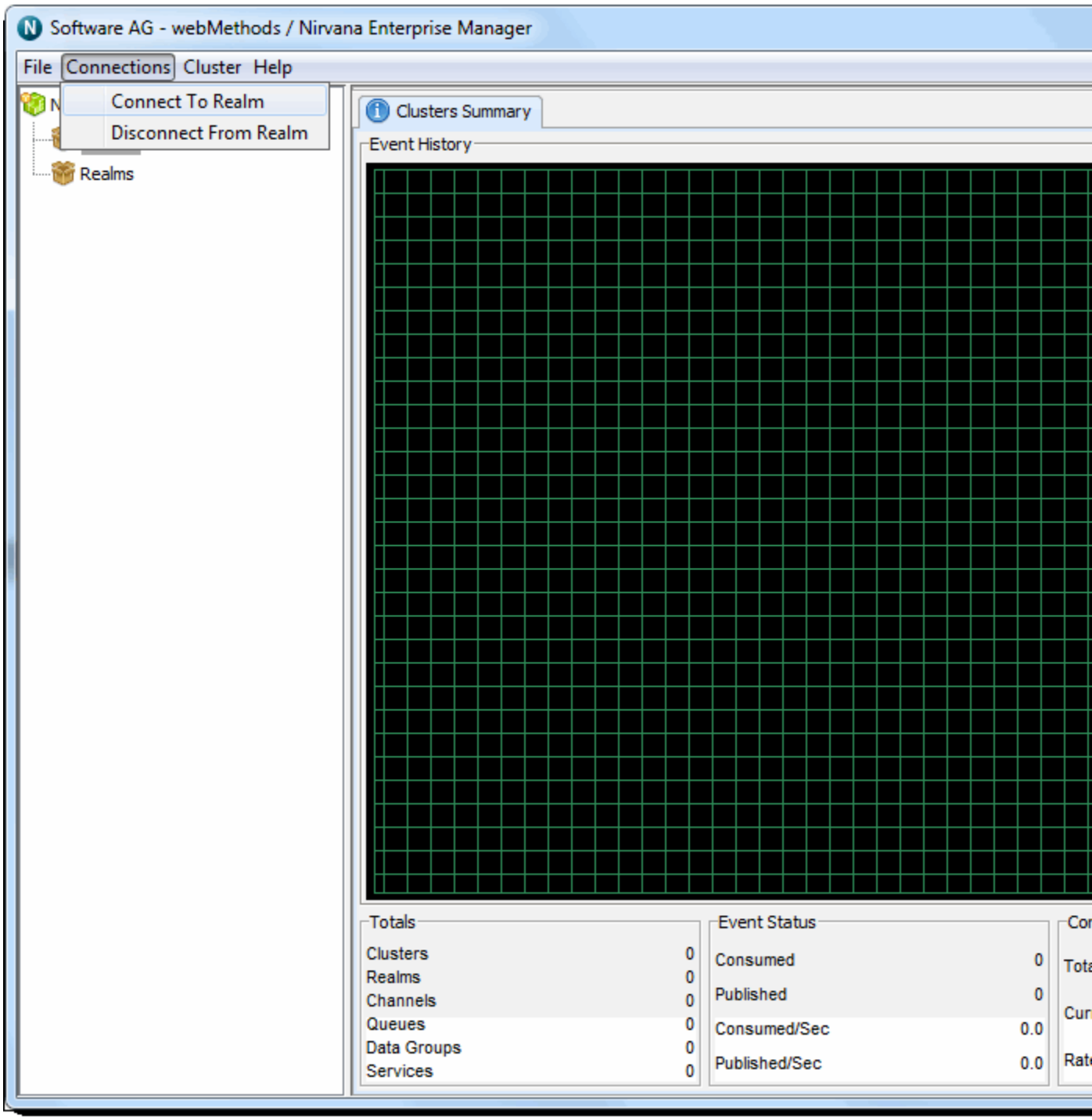
An Enterprise Manager has the ability to connect to multiple Universal Messaging realms at the same time. These realms can be standalone or clustered so developers and administrators can now manage and monitor the whole Universal Messaging enterprise infrastructure from a single instance of Enterprise Manager. Once connected to a set of Universal Messaging realms, it is possible to save (see "Realm Profiles" on page 43) the connection information so that Enterprise Manager automatically connects to all those realms each time it starts.

A bootstrap RNAME environment variable is needed the very first time you run Enterprise Manager or if your connection info file is empty. If you use the shortcut / link created by the installation process this will be automatically set to point to the locally installed realm's bootstrap interface so you don't need to take additional action. If however you open a client command prompt and you wish to initially connect to a realm other than the local one, then you need to change your RNAME environment variable.

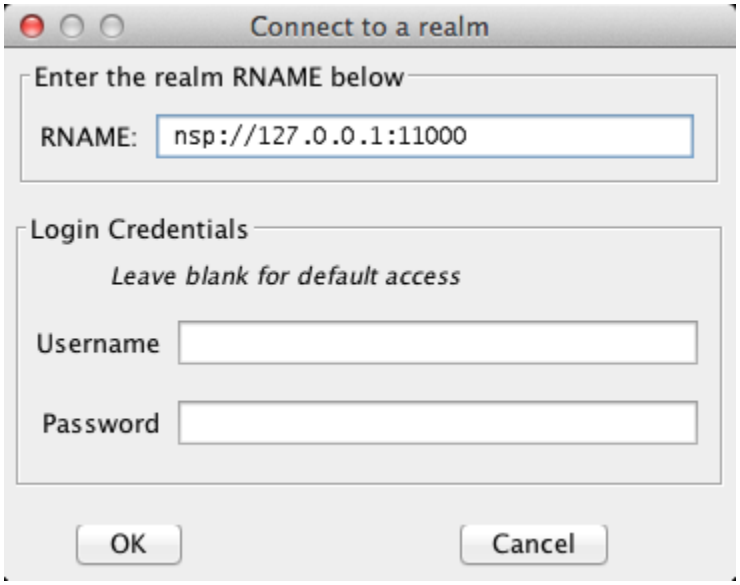
For more information on how to set the RNAME variable, see the section *Communication Protocols and RNAMEs* in the Developer Guide.

Please also note that once your realm connection information is saved, the RNAME environment variable will be ignored.

Once your Enterprise manager is up and running, you have to select the Connect to Realm menu option from the Connections menu, as shown in the figure below:



The menu option will open the Connect To Realm dialog as shown in the figure below:

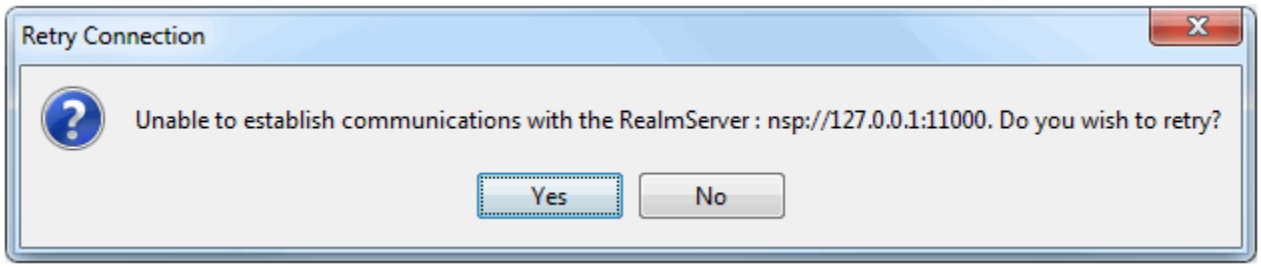


Simply fill in the RNAME that points to the interface of the Universal Messaging realm you wish to connect to and click on the OK button. The Enterprise Manager status bar will display a message informing you where it is trying to connect to. If the connection is successful, a new realm node will be rendered on the tree with the unique name of that realm. You can manage and monitor the new realm by selecting that newly rendered tree node.

The name displayed for the realm uses the syntax: `realmname(host:port)`, for example `realm1(MyHost:11010)`.

When you connect to a realm server that is part of a cluster or zone, the Enterprise Manager automatically connects to and displays the other realms in the cluster or zone.

If you enter an incorrect RNAME, if that realm is not running or if it is running but the particular interface is not up the connection will fail. In that case a retry dialog will appear like this one below:

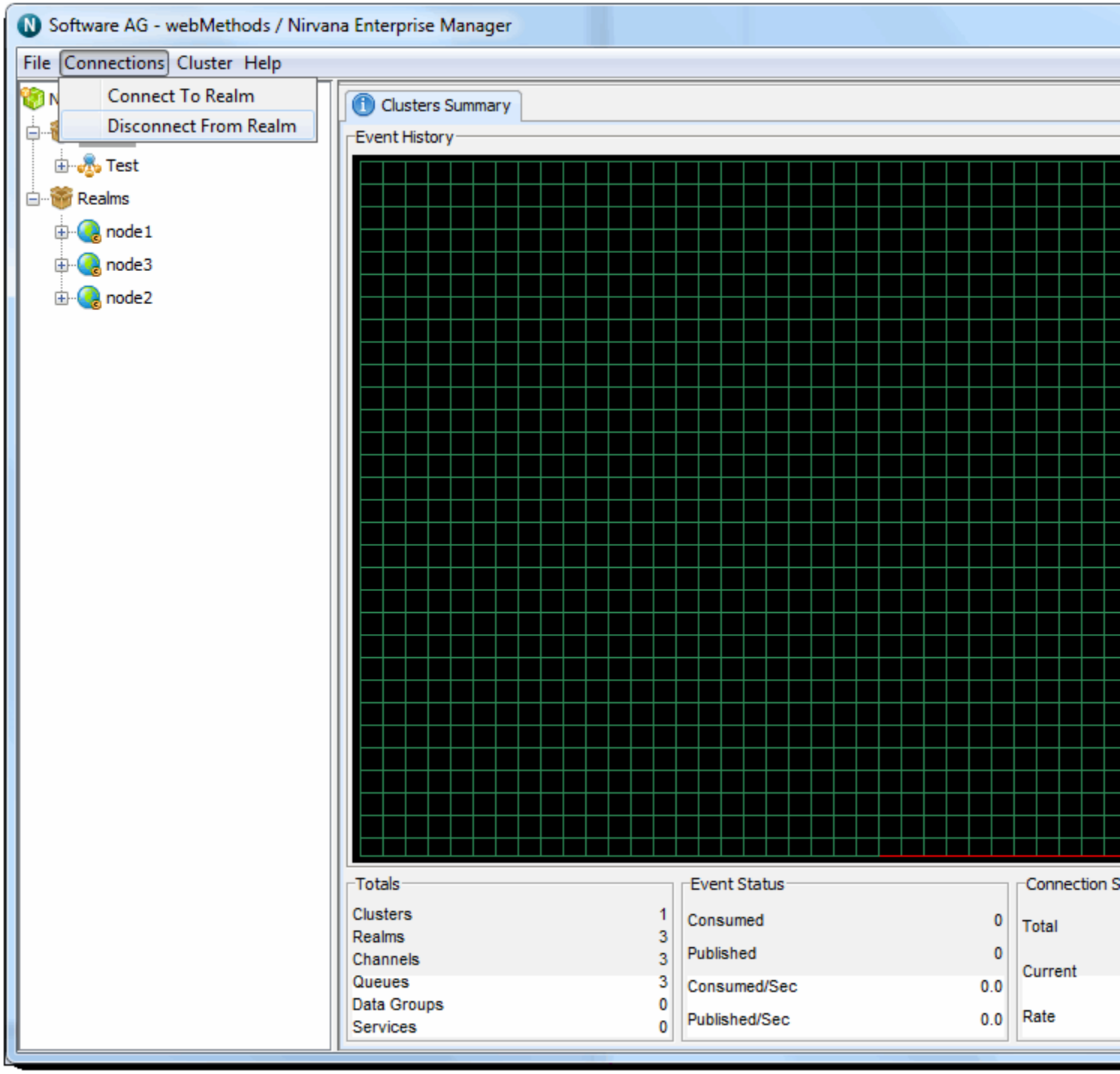


If you had typed the correct RNAME this gives you the opportunity to start the Universal Messaging realm or interface needed and click yes to retry the connection without entering the information again. If however the RNAME entered was wrong or you do not wish to retry then clicking no will close the dialog. Finally don't forget that to make this connection get attempted each time you start Enterprise Manager you need to save your connection information.

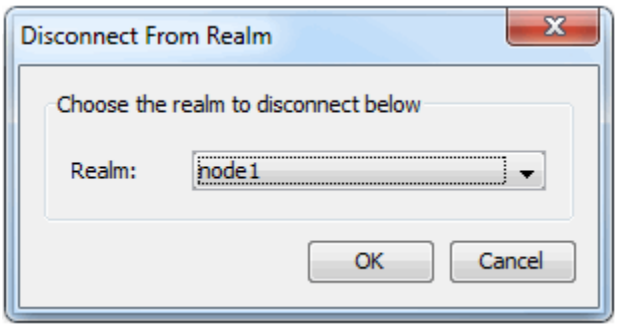
Disconnecting from Realms

Using the multiple realm connection functionality, the startup time of the Enterprise manager is slightly increased each time you add a Universal Messaging realm to your connection list. If you connect from a different location or network, if the development phase of a Universal Messaging application completes or if you simply wish to have faster startup times for Enterprise Manager, you may want to stop connecting to one or more of your Universal Messaging realms.

This section explains how it is possible to disconnect from one of multiple realms that your Universal Messaging Enterprise manager may be connected to. To do so, simply select the Disconnect from Realm menu option in the Connections menu as shown in the figure below:



This causes a disconnection dialog to appear like the one shown below:



The dialog lists the names of the currently connected Universal Messaging realms. Select the realm you wish to disconnect from and click OK. The disconnected realm node and any other realm nodes that were added by it when the node was created disappear from the namespace tree.

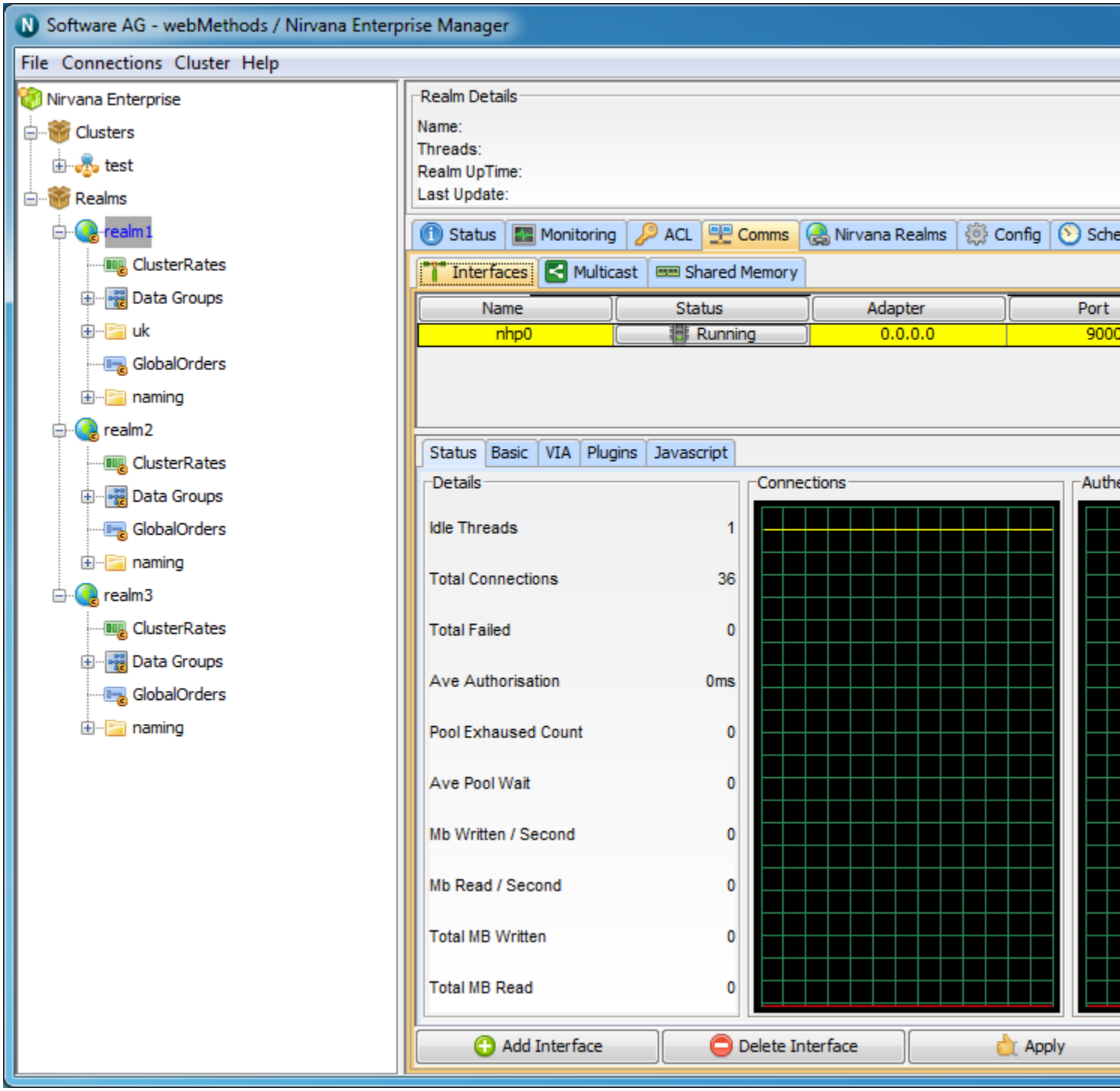
Disconnecting from a realm is not necessarily a permanent operation. If you disconnect from a realm that was listed in your connection information, then the disconnect is applicable for this Enterprise Manager session only, next time you start up the connection will be attempted again. In order to make the disconnect permanent, please save (see "Realm Profiles" on page 43) your connection information after you disconnect.

Interface Status

Universal Messaging interfaces (see "TCP Interfaces, IP Multicast and Shared Memory" on page 277) allow users to connect to a Realm using various protocols and ports on specific physical Network interfaces on the host machine. Interfaces are also available to users through the Universal Messaging Administration API and can provide useful status information regarding user connections.

The Enterprise manager provides a summary of this status information for each interface. This section will describe the status information available for each interface.

To view status information for an interface, you must first select the 'Comms' tab for the Realm you want to view. This tab contains the interface configurations as well as Multicast and Shared Memory configurations Select the interface you wish to view from the list of interfaces in the 'Interfaces'. By selecting the desired interface, you will be presented with a number of panels, one of which is labeled 'Status'. This panel is shown in the image below.



The interface status panel has a section that describes the details of the interface status information. The status information contains 6 values, each of which is described below.

Details Panel

- **Idle Threads**- The number of idle threads, calculated as the total threads from the interface accept threads pool - the number of threads from the pool currently accepting connections. Corresponds to available threads
- **Total Connections** - The total number of successful connections made to this interface
- **Total Failed** - The total number of failed connection attempts made to this interface
- **Ave Authorisation** - The average time it takes a connection to authenticate with the realm server
- **Pool Exhausted Count** - The number of times that the interface thread pool has had no threads left to service incoming connection requests. When this count increases, you should increase the number of accept threads (see "Interface Configuration" on page 282) for the interface
- **Ave Pool Wait** - The average time that a client connection has to wait for the accept thread pool to provide an available thread. Like the Pool Exhausted count, this is a good indicator that the number of accept threads for an interface is too low and needs to be increased

The status panel also shows 2 graphs that depict connection attempts (successful connections are shown in yellow, failed connection attempts are shown in red) and authentication times (average authentication times are shown in yellow, and the last authentication time is shown in red).

Zone Administration

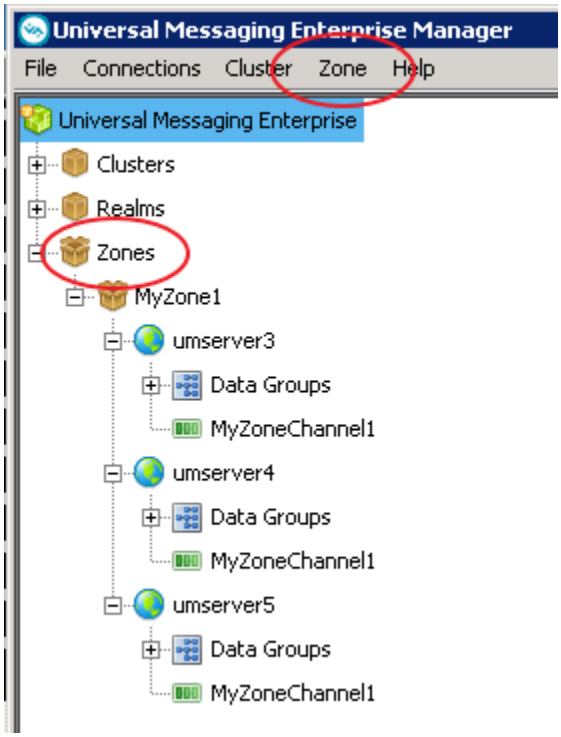
Overview of Zone Administration

The Enterprise Manager provides menu items for performing the administrative functions on zones. In a zone, messages that are published to a channel on one realm are automatically forwarded to a channel of the same name on other realms in the zone.

Note:
Messages on queues are not forwarded between realms in a zone; the zone functionality applies only to channels.

For general information about using zones, refer to the *Architecture* section of the *Universal Messaging Concepts* guide.

Zone administrative functionality is offered in the Enterprise Manager menu bar and in the navigation tree:



The **Zone** tab in the menu bar allows you to perform operations on zones, such as creating and deleting zones.

The **Zones** node in the navigation tree is the parent node of any zones you create.

The zone administration operations that you can perform are described in the following sections.

Creating a Zone

To create a zone and define it with an initial set of realms or clusters, proceed as follows:

1. Open the dialog for creating a zone.

   You can do this in one of the following ways:

   - In the Menu bar, select **Zone > Create Zone**, or
   - In the navigation tree, select the Zones node, and from the context menu choose **Create Zone**.

2. In the dialog, specify a name that will be assigned to the zone.
3. Add realms or clusters to the zone.

   If you select the radio button for realms, you see all of the realms that you can add to the zone. If you select the radio button for clusters, you see all of the clusters that you can add to the zone.

   Specify the realms or clusters you want to add to the zone, then click **Add**.

4. Click **OK** to create the zone and close the dialog.

The newly created zone is now displayed under the **Zones** node in the navigation tree.

If you expand the node of the new zone, you will see the realms that belong to the zone.

Note:

1. A zone can contain either realms or clusters, but not a mixture of realms and clusters.
2. A zone cannot be empty; it must contain at least one realm or cluster.

Modifying the set of realms or clusters in a zone

To modify the set of realms or clusters in a zone, proceed as follows:

1. Under the **Zones** node in the navigation tree, select the node representing the required zone. In the context menu, select **Modify Zone Members**.

   This displays the realms/clusters that are currently members of the zone, and also the realms/clusters that are currently not members but which are available to become members.

2. As required, add realms/clusters to the zone's existing members, or remove existing members.

3. Click **OK** to save the modified zone and close the dialog.

Deleting a zone

To delete a zone, proceed as follows:

1. Select the **Zones** node in the navigation tree, then in the context menu, select **Delete Zone**.

   Alternatively, select **Zone > Delete Zone** from the menu bar.

2. Select the required zone from the displayed list and click **OK** to delete the zone.

Creating a channel in a zone

You can create a channel for a zone, and the channel will be automatically created on all realms/clusters in the zone.

To create a channel in a zone, proceed as follows:

1. Select the node for the zone in the navigation pane. Then, in the context menu of the node, select **Create Channel**.

2. In the **Add Channel** dialog, specify the attributes of the channel that you wish to create.

3. Click **OK** to complete the dialog and create the channel.

The Enterprise Manager now creates the channel on all realms or channels in the zone.

Modifying a channel in a zone

If you wish to modify the attributes of a channel that was created in a zone via **Create Channel**, you must modify the attributes for the channel in each of the zone members (realms, clusters) individually.

Note:
Any changes you make to the channel definition for a realm/cluster in a zone are NOT propagated automatically to the other zone members. If you wish to keep all zone members in sync, you have to update the other zone members individually.

To modify a channel on one realm/cluster in a zone, proceed as follows:

1. Select the node for the channel under the node for the realm/cluster on which the channel is defined.
2. In the context menu of the channel, select **Edit Channel**.
3. In the **Modify Channel** dialog, make your changes and click **OK** to complete the dialog.

Channel interface attributes for use in zones

For the message forwarding mechanism between realms in a zone, Universal Messaging requires each affected realm to use an interface that has the attribute **Allow for InterRealm** activated. See the section "Interface Configuration" on page 282 for a description of this attribute.

General notes on using zones

This section summarizes some operational aspects of using zones.

- If a zone member (a realm or cluster) is not active (e.g. the server is down), no Enterprise Manager operations will be allowed on the zone until all zone members are available again.
- Any given realm or cluster cannot be a member of more than one zone at the same time.

Cluster Administration

Overview

This section describes the process of creating and managing a Universal Messaging cluster. Universal Messaging Clusters enables the replication of resources across the cluster. The state of a clustered resource is maintained across all realms within the cluster. For example if an event is popped from a clustered queue it is popped from all nodes within the cluster.

For more information on how to use the Enterprise Manager to manage Universal Messaging Clusters please see:

- "Creating Realm Clusters" on page 102
- "Deleting Realm Clusters" on page 106

- "Modifying Cluster Members" on page 108
- "Creating Cluster Channels" on page 110
- "Creating Cluster Queues" on page 115
- "Viewing and Monitoring Cluster Information" on page 97
- "Manage Inter-Cluster Connections" on page 123

For more information on how to use the Enterprise Manager to manage Universal Messaging Clusters with Sites please see:

- "Creating and Managing Clusters with Sites" on page 126

Viewing the Available Clusters

The **Clusters** view is designed to provide an overview of the characteristics as well as current status of the set of Universal Messaging clustered realms that enterprise manager is aware of.

This section describes the type of status information that you can observe from the Clusters Summary view.

The top of the screen displays a large real time graph illustrating the total number of events published (yellow) and consumed (red) across all Universal Messaging clusters.

The bottom of the screen displays 3 panels named **Totals**, **Event Status** and **Connection Status**. These panels and the information displayed are described below.

Totals

The Totals section describes the following :

- **Clusters**- The number of clusters defined within the enterprise manager and its realm nodes
- **Realms**- The number of realms known by the enterprise manager
- **Channels**- The number of channels that exist across all known realms
- **Queues**- The number of queues that exist across all known realms
- **Data Groups**- The number of data groups that exist across all known realms
- **Services**- Total number of services that exist across all known realms

Event Status

The Event Status section describes the following :

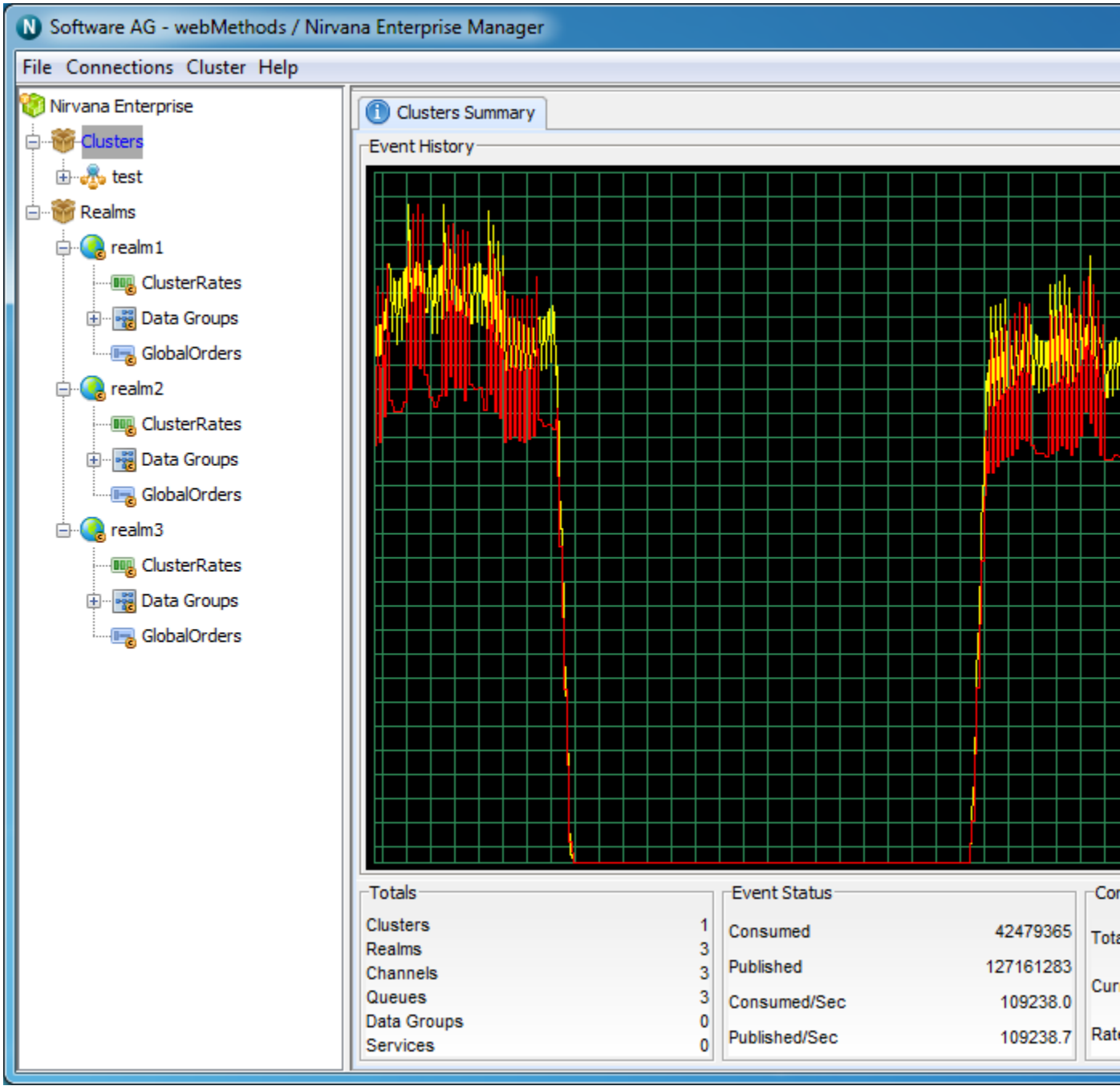
- **Published** - The total number of events published to all channels, queues and services across all realms within known clusters
- **Consumed** - The total number of events consumed from all channels, queues and services across all realms within known clusters

- **Published/Sec** - The number of events published to all channels, queues and services, per second across all realms within known clusters
- **Consumed/Sec** - The number of events consumed from all channels, queues and services, per second across all realms within known clusters

Connection Status

The Connection Status section describes the following :

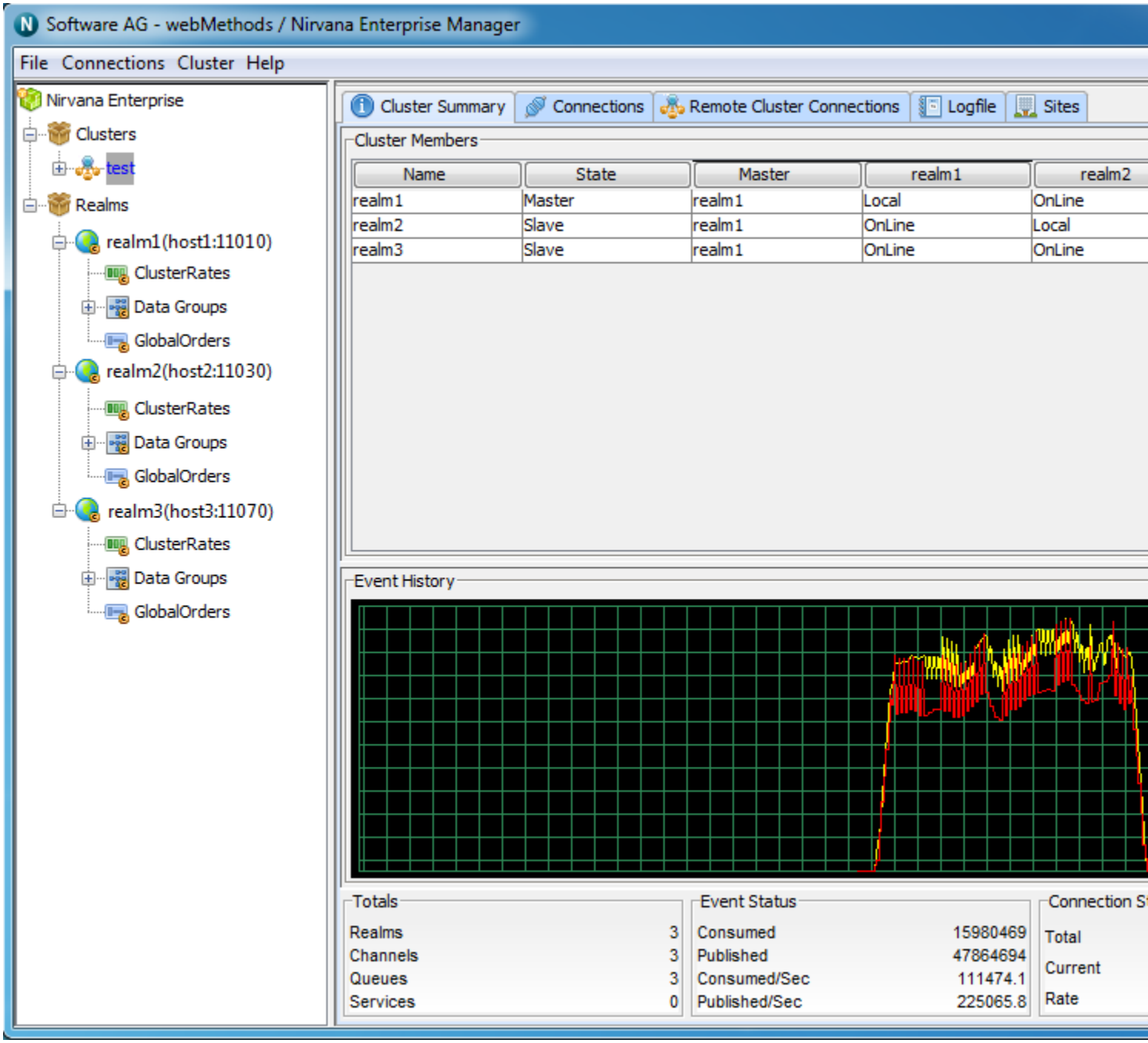
- **Total** - The total number of connections made to all realms within known clusters
- **Current** - The current number of events across all realms within known clusters
- **Rate** - The number of connections being made per second across all realms within known clusters



Viewing Information for a Cluster

You can view information for an individual cluster by expanding the **Clusters** node in the navigation pane and selecting the node for the required cluster. The view provides an overview of the characteristics as well as current status of a selected cluster.

View of a Cluster



The top of the view shows the realms that have been defined for the cluster.

The view includes a real time graph illustrating the total number of events published (yellow) and consumed (red) across all realms in the cluster.

The bottom of the screen displays 3 panels named **Totals**, **Event Status** and **Connection Status**. These panels and the information displayed are described below.

Totals

The Totals section describes the following :

- **Realms**- The number of realms within the cluster
- **Channels**- The number of channels that exist across all realms within the cluster
- **Queues**- The number of queues that exist across all realms within the cluster
- **Services**- Total number of services that exist across all realms within the cluster

Event Status

The Event Status section describes the following :

- **Published** - The total number of events published to all channels, queues and services across all realms within the cluster
- **Consumed** - The total number of events consumed from all channels, queues and services across all realms within the cluster
- **Published/Sec** - The number of events published to all channels, queues and services, per second across all realms within the cluster
- **Consumed/Sec** - The number of events consumed from all channels, queues and services, per second across all realms within the cluster

Connection Status

The Connection Status section describes the following :

- **Total** - The total number of connections made to all realms within the cluster
- **Current** - The current number of events across all realms within the cluster
- **Rate** - The number of connections being made per second across all realms within the cluster

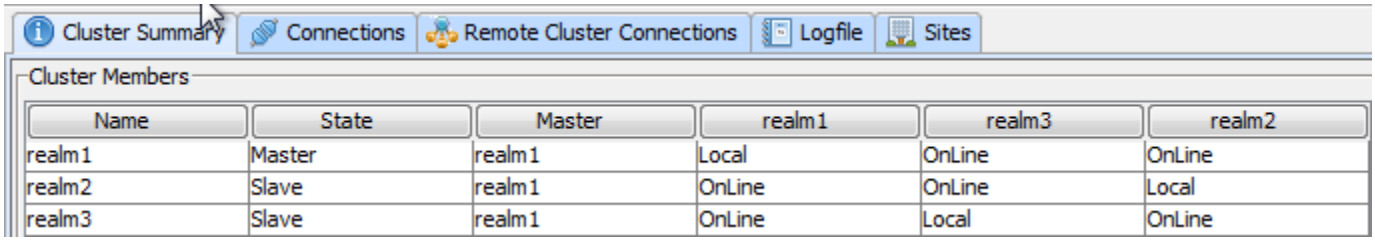
The Enterprise Manager provides information for the cluster through the following tabs:

- Cluster Summary
- Connections
- Remote Cluster Connections
- Logfile
- Sites

Cluster Summary

The Cluster Summary tab provides an overview of all realms in the Cluster. It identifies the current Master realm, and also shows each realm's perception of the state of all other realms.

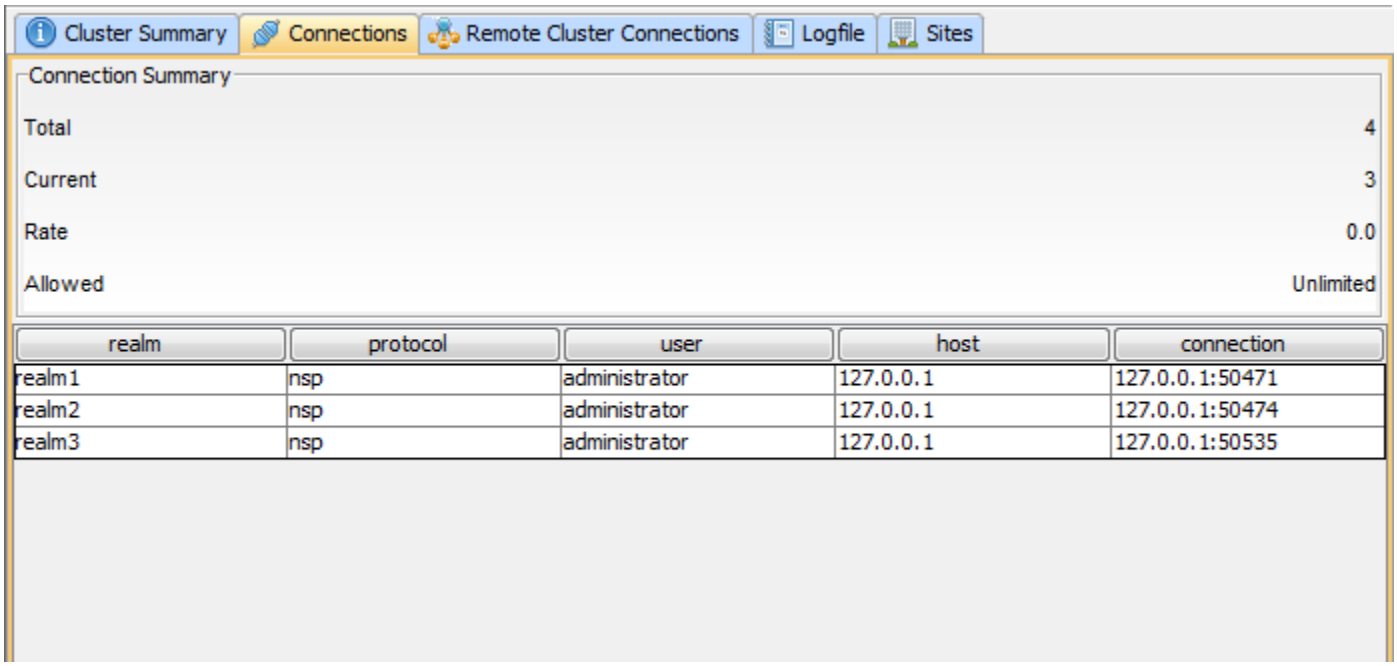
The Cluster Summary Tab



Connections

The Connections tab shows all connections to realms in the Cluster. In this example, it shows a single user connected to three realms in the Cluster:

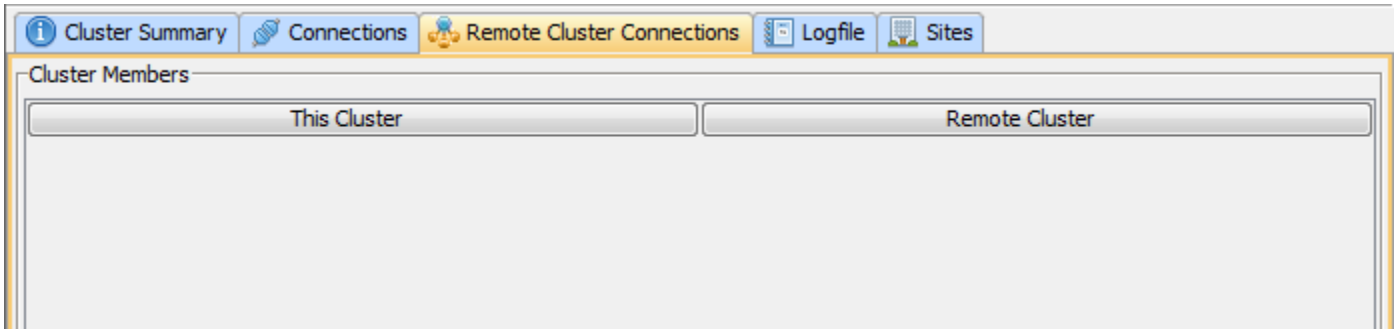
The Cluster Connections Tab



Remote Cluster Connections

The Remote Cluster Connections tab shows all remote cluster connections for this Cluster. Clusters can be remotely connected together providing the ability to create joins between channels in different clusters:

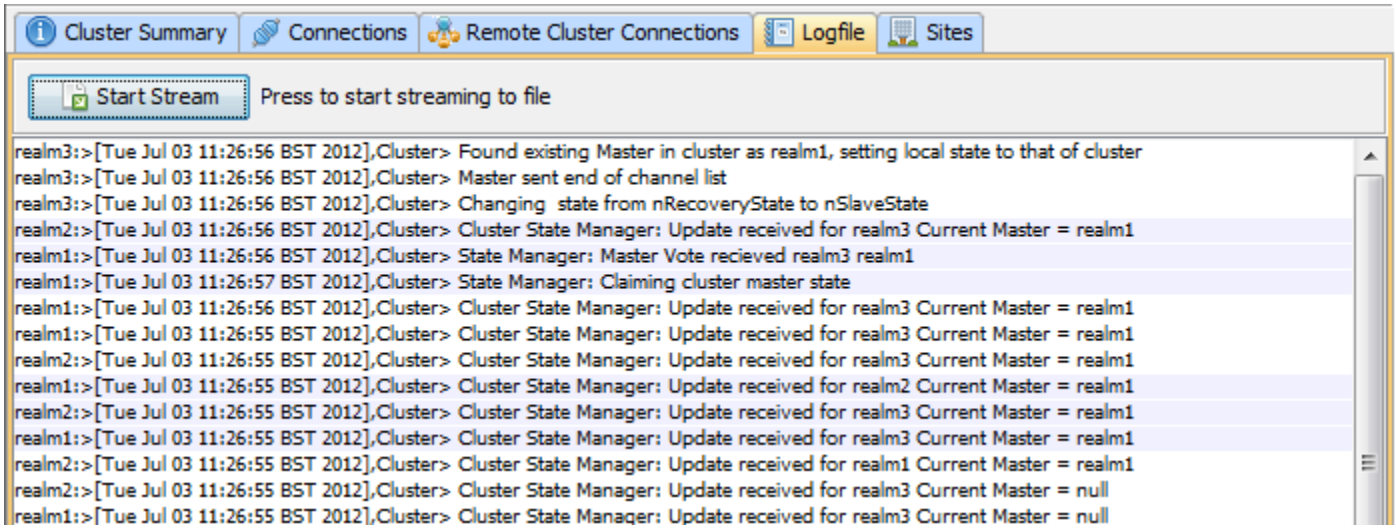
The Remote Cluster Connections Tab



Logfile

The Logfile tab shows a real-time Cluster-specific log, and provides the option to stream the log output to a file:

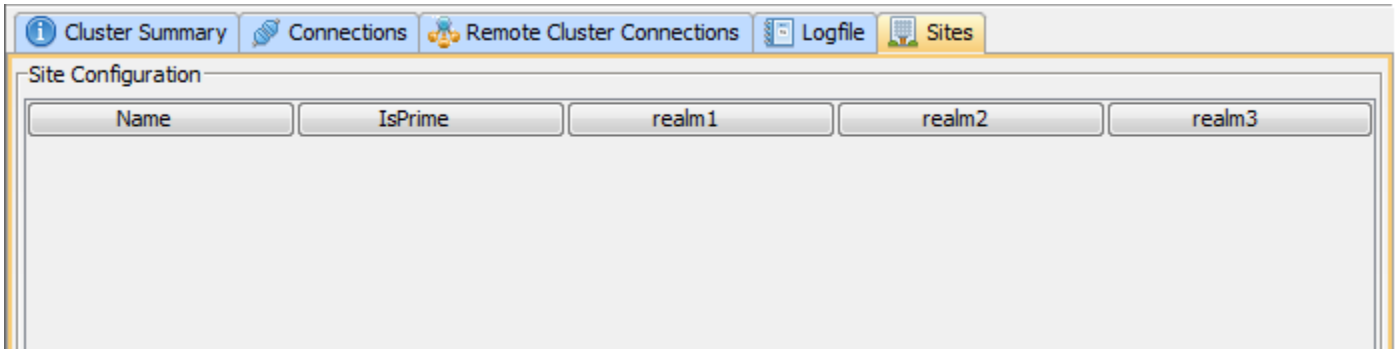
The Cluster Logfile Tab



Sites

The Sites tab shows any site configurations (see "Creating and Managing Clusters with Sites" on page 126) for the current cluster. Clusters that have Site configurations are known as *Universal Messaging Clusters with Sites*. (Those without are known as *Universal Messaging Clusters*):

The Cluster Sites Tab



Creating a Cluster

This section describes the process of creating a Universal Messaging Cluster.

Tip:
Since the underlying purpose of a cluster is to provide resilience and high availability, we advise against running all the servers in a cluster on a single physical or virtual machine in a production environment.

Important:
The Enterprise Manager does not support working with two clusters that have the same name.

Viewing Clusters in Enterprise Manager

The Enterprise Manager's top level view shows a tree node labelled **Universal Messaging Enterprise**. One level below this is a tree node labelled **Clusters**, which contains any known clusters.

If you use the Enterprise Manager to connect to a realm which is a member of an existing cluster, then the cluster will automatically be displayed under the above-mentioned **Clusters** tree node. When a cluster node is found, the Enterprise Manager will also automatically connect to all of the cluster member realms (if not already connected by default as a result of having loaded realm connection information in a custom Enterprise Manager Connection Profile). See "Connecting to Multiple Realms" on page 84 for related information.

Preparing to Create a Cluster

Firstly, before a cluster can be created, the Enterprise Manager needs to connect to those realms (see "Connecting to Multiple Realms" on page 84) that will form the cluster. If any realms cannot be connected to, or you receive a 'Security Alert' message when you click on the realm node, you may want to check that the realm is running, and check the permissions (see "Realm Entitlements" on page 214) on the realm. If the realms you are connecting to are running on different machines, you need to ensure that all realm machines are given full permissions to connect to the other realms in the cluster. Each realm communicates with the other cluster realms via its own connection. The subject of each connection is as follows:

```
realm-realmname@ip_address
```

For example, in the following example, there are 3 realms that will form part of a cluster, each realm subject needs to exists in the ACLs of the other realms. The following realm subjects need to be added to the ACL for each realm in our example:

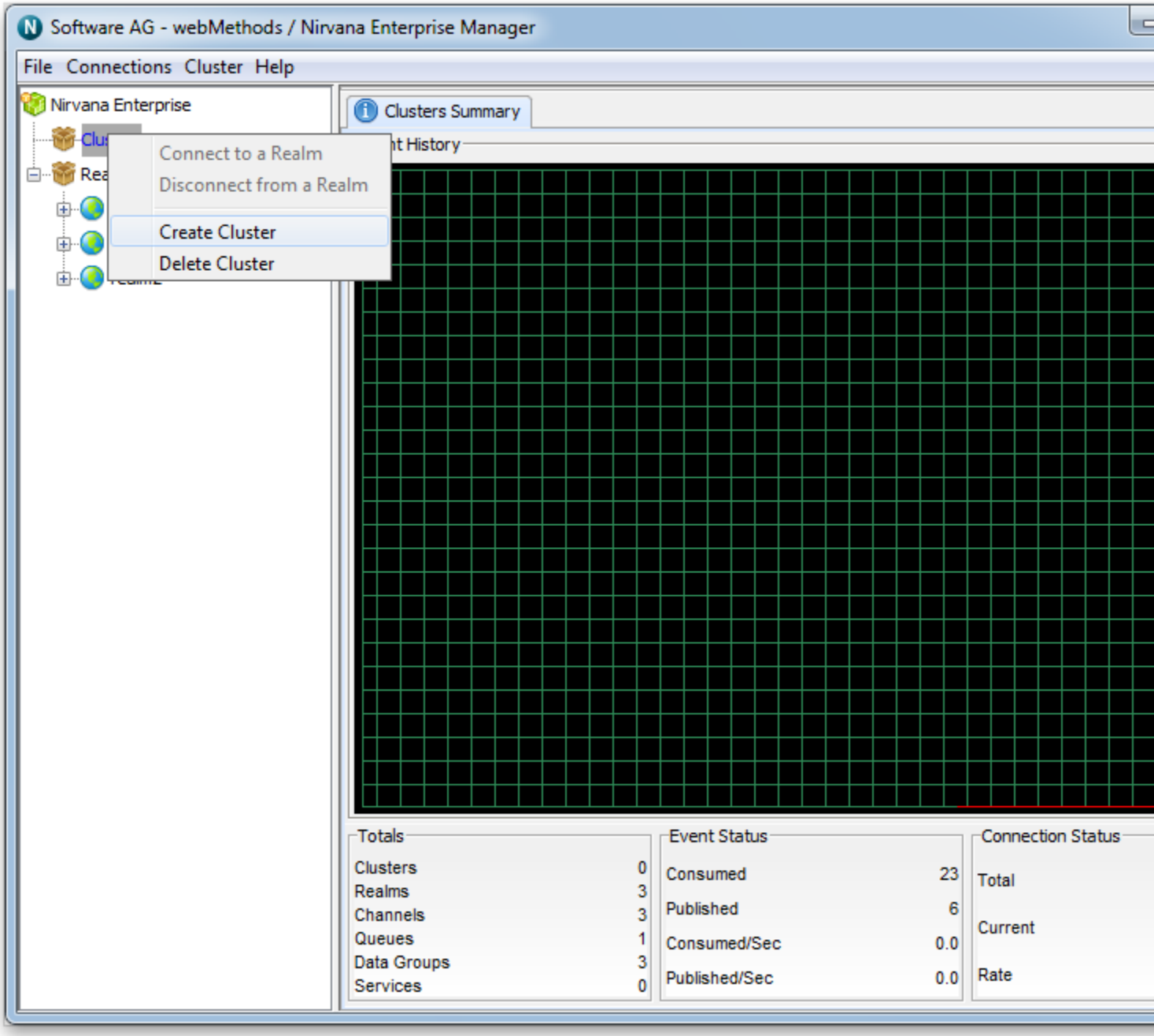
```
realm-realm1@10.140.1.1 realm-realm2@10.140.1.2 realm-realm3@10.140.1.3
```

The permissions given for each realm need to be 'Access Realm'. As well as this, each realm must have a valid entry for the user@host that corresponds to the user that will create the cluster using the Enterprise Manager. The permissions for this user must be sufficient in order to create the cluster object. Temporarily it is often better to provide the *@* default subject 'Full' privileges to facilitate setting up a realm and clusters.

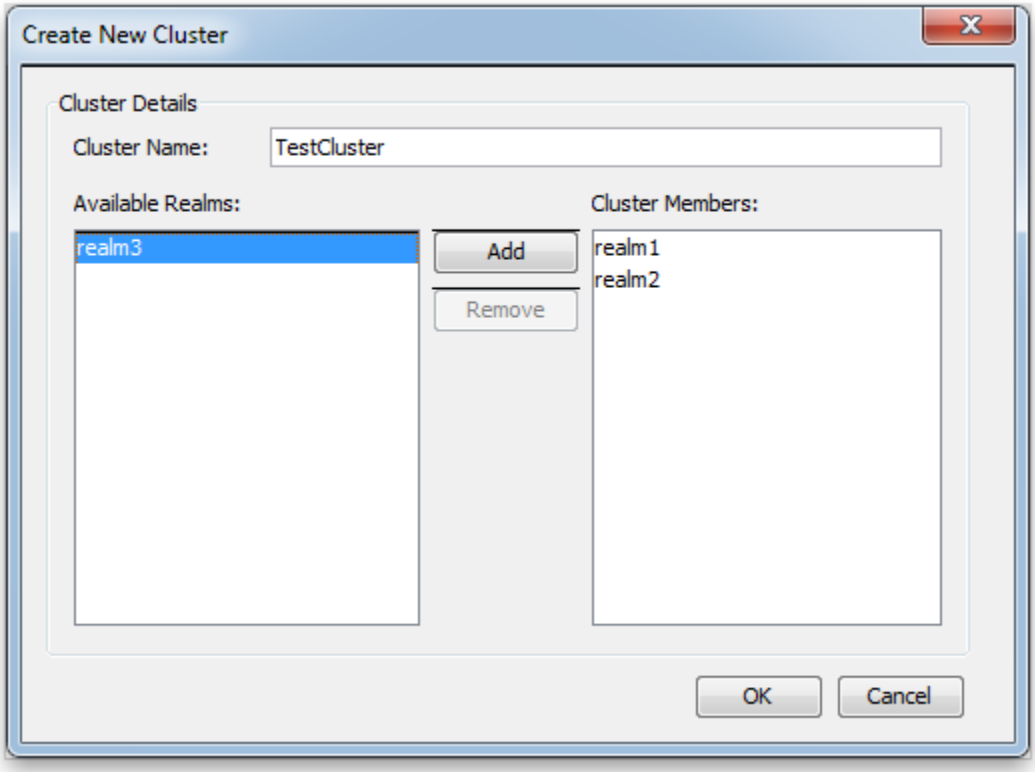
Creating a cluster

Important:
The Enterprise Manager does not support working with two clusters that have the same name.

To create a cluster, select the 'Clusters' node under the 'Universal Messaging Enterprise' node.



When you select the 'Create Cluster' menu option, you are presented with the cluster dialog. The cluster dialog allows you to select which of the realms that the Enterprise Manager is connected to will become members of the cluster. One of the selected realms will become the master during the cluster creation. The master realm will control synchronizing the state between the other realms, and acts as the authoritative source for this information.



The cluster dialog contains a text box for you to input the name of the cluster. Below the name are the details of the cluster members. The available realms are shown on the left hand side of the dialog. The right hand side shows those realms that are members. When you double-click on a realm name in the **Available Realms** list, or click on a realm name and click on the **Add** button, the realm will be added to the **Cluster Members** list. You can remove any realm from the **Cluster Members** list by either double clicking on the realm from the list or by selecting the realm name, and clicking on the **Remove** Button.

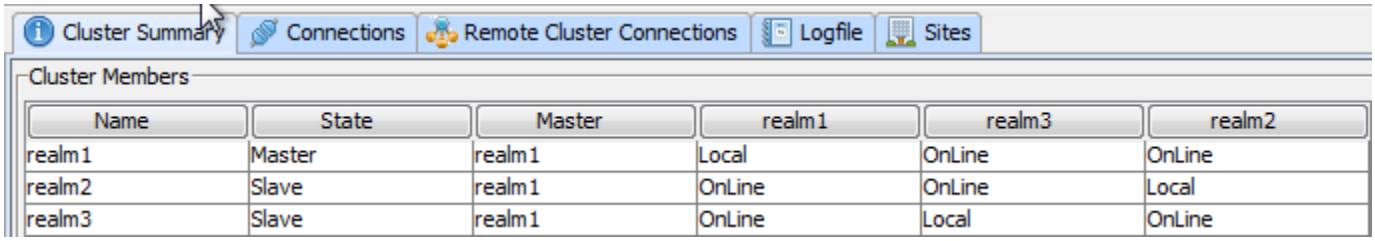
When you have finished selecting your cluster members, clicking on the 'OK' button presents you with the local store migration dialog. This dialog allows you to select whether to migrate local stores to cluster-wide stores. Selecting the **Yes** button will convert any local stores (channels, queues, etc.) on the realms being added to the cluster into cluster-wide stores. These stores will then be present on all realms in the cluster when cluster creation completes. If the name of a local store is the same as the name of an existing cluster store, clicking the **Yes** button will cause the cluster creation to fail due to a name clash. Clicking the **No** button will keep these stores local to the realms, and they will not be present on other realms in the cluster when creation is completed.

If the cluster creation is successful, a new cluster node will appear under the **Clusters** node, and the realms that have been selected as members will be shown beneath the cluster.

Checking the cluster state

When a cluster has been created, you can monitor its state by selecting the cluster node. The 'Cluster Summary' tab will show the state of all cluster members, and which realm is current cluster master.

The image below shows the state of a cluster when it has been created and all realms within the cluster are fully online.

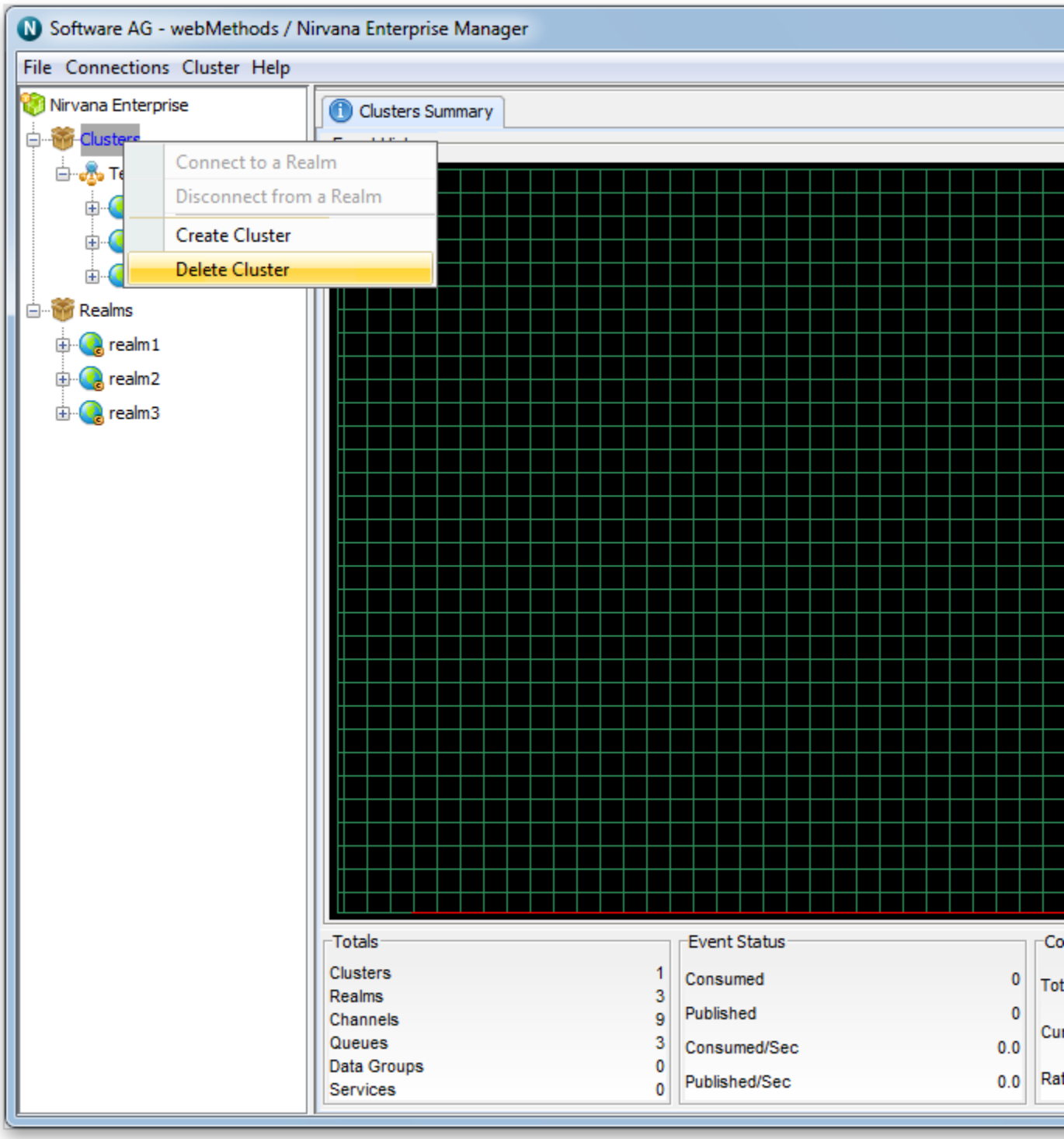


Creating cluster channels (see "Cluster Channel Administration" on page 110) and cluster queues (see "Cluster Queue Administration" on page 115) is not permitted if any of the cluster realms are offline.

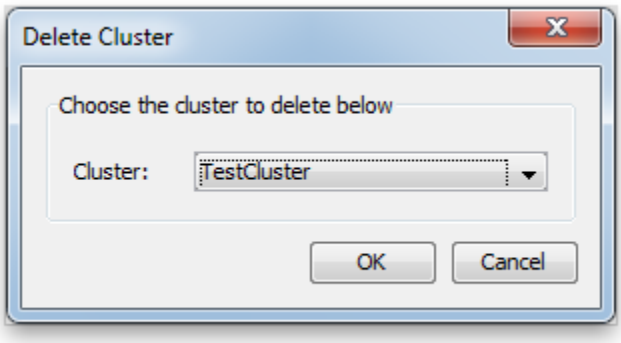
Deleting Clusters

When a Universal Messaging cluster needs to be deleted, all cluster resources that exist in all cluster member realms will also be deleted. Removal of a cluster is a simple operation that can be achieved using the Enterprise Manager. This section will describe the process of removing a cluster.

In order to remove a Universal Messaging Realm cluster, you must first of all select the 'Clusters' node from the Enterprise Manager. Right-clicking on this node will present a pop up menu, as shown in the image below.



Selecting the menu option 'Delete Cluster' will prompt you with a dialog that asks you to select a cluster node from a list. This list will contain all known clusters within the realms you have connected to. This dialog is shown in the image below.



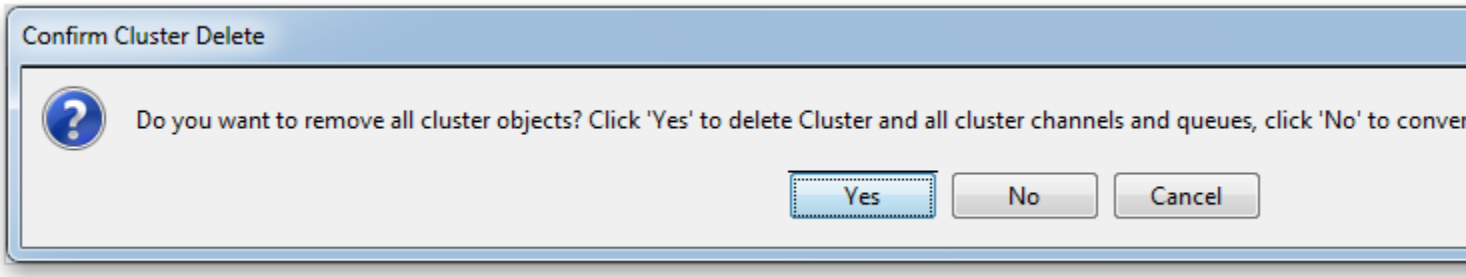
Clicking on the 'OK' button once you have selected the cluster you wish to delete will prompt you to answer a question. This question gives you 2 choices with regard to the cluster resources that may exist within the cluster. These are:

- Delete all cluster wide resources from each cluster realm
- Convert all cluster wide resources to local within each realm

Choosing to delete all cluster resources will not remove any locally created channels, only those created for the cluster.

Choosing to convert each one to local, will keep any data that may be contained with the resources .

This dialog is shown in the image below. Choosing 'Yes' will remove the cluster objects, 'No' will make them all local, 'Cancel' will take no action at all.



Modifying Clusters

The Universal Messaging Enterprise Manager enables you to modify clusters. By 'Modify' we mean adding new realms to the cluster or removing existing cluster members.

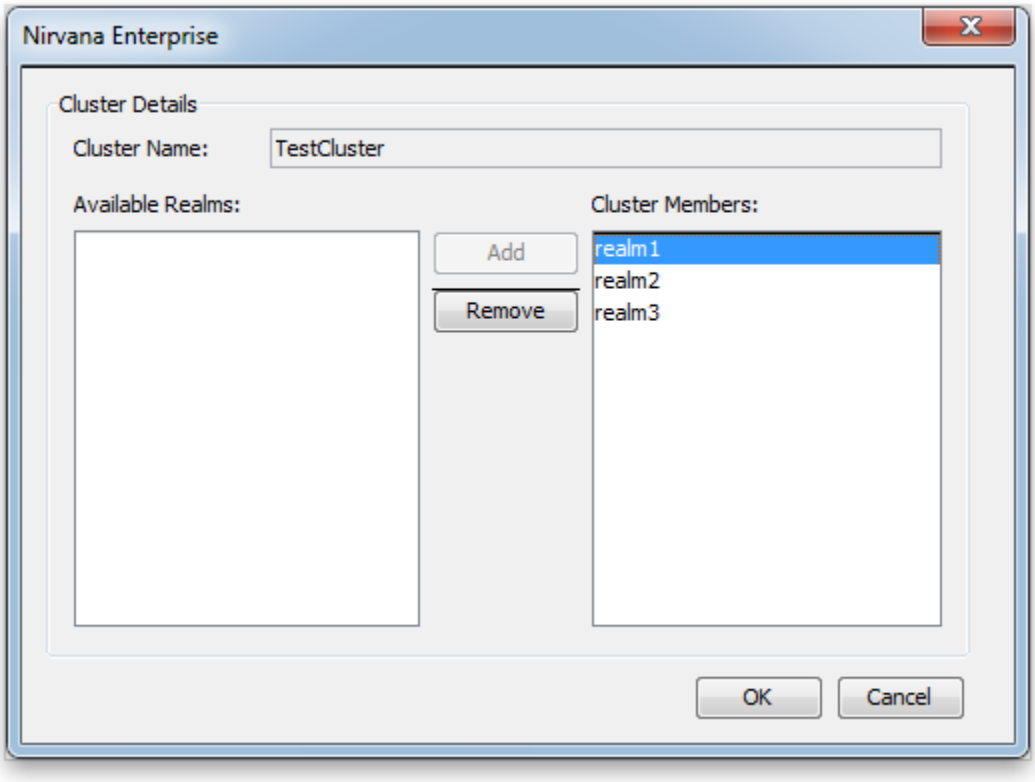
To add a new realm to a cluster, you must first of all ensure that you have connected to (see "Connecting to Multiple Realms" on page 84) the realm you wish to add. Removing realms is accomplished by selecting the realm you wish to remove from the cluster. This will be discussed in more detail further on.

If for example, you have a cluster with two realms, and wish to add a third realm to the cluster, it is possible to do so. Similarly, if you have a cluster with three (or any number of realms) and wish to remove a realm from the cluster, this is also possible.

Adding Cluster Members

In order to add a realm to a cluster, you must first ensure that you have created a cluster (see "Creating a Cluster" on page 102). Once you have a cluster, then also ensure you have connected to (see "Connecting to Multiple Realms" on page 84) the realm you wish to add to your cluster. Select the cluster node from the namespace and right-click on the node. This will present you with a pop-up menu. Select the menu item labelled 'Modify Cluster Members'.

The dialog this presents you displays the current members of your cluster as well as any realms you are not connected to that are not cluster members. This dialog is shown below:



The dialog shows the name of the cluster, a list of realms which are not currently members of the cluster (shown as a list on the left hand side), and a list of current cluster members (on the right hand side).

As you can see from the above example, currently there are three realms within the cluster 'TestCluster'.

Double-clicking on any non-member realm, or selecting it from the list and clicking the 'Add' button will enable you to add the realm as a member.

Note:

If the realm has local stores with names matching any store on the cluster, the realm will not be added to the cluster. This is to prevent naming clashes on the cluster.

When you have added the realms you wish to add as cluster members, click on the button labelled 'OK'. This will add all realms in the right-hand list to the cluster. All cluster resources will also be created on the newly added realms once the realms have successfully been added to the cluster.

Removing Cluster Members

Removing cluster realms is achieved by again selecting the cluster node, right-clicking on the node and choosing the 'Modify Cluster Members' menu item. This presents the same dialog as shown above.

To remove a realm, double-click on the realm from the 'Cluster Members' list or select the realm and click the 'Remove' button. This will remove the realm from the list and add it back into the non-members list.

Clicking on the button labelled 'OK' will then prompt you to answer a question. This question allows you to select one of 2 options:

- Delete all cluster wide resources from each the removed realm members
- Convert all cluster wide resources to local within the removed realm members

Choosing to delete all cluster resources will not remove any locally created channels, only those created for the cluster within the realms you are removing.

Choosing to convert each one to local, will keep any data that may be contained within the cluster resources for the realms you wish to remove.

Adding and Removing Cluster Members

Cluster members can be added and removed in the same operation. For example, if you have a cluster with 'realm1' and 'realm2' but want to remove 'realm2' and add 'realm3', you would simply remove 'realm2' and add 'realm3' from the 'Cluster Members' list in the 'Modify Cluster' dialog. The Enterprise Manager will work out which realms to add and which to remove for you and perform the necessary channel conversion and deletions you choose.

Cluster Channel Administration

This section describes the process of creating channels on a Universal Messaging realm cluster. Channels are the logical rendezvous point for data that is published and subscribed. Each channel that is created in a cluster consists of a physical object within each Universal Messaging realm within the cluster as well as its logical reference within each realm's namespace.

Note:

Simple and Transient channels are not supported across realms in a cluster.

Creating channels using the Enterprise Manager creates the physical object within each cluster realm. Once created, references to the cluster channels can be obtained using the Universal

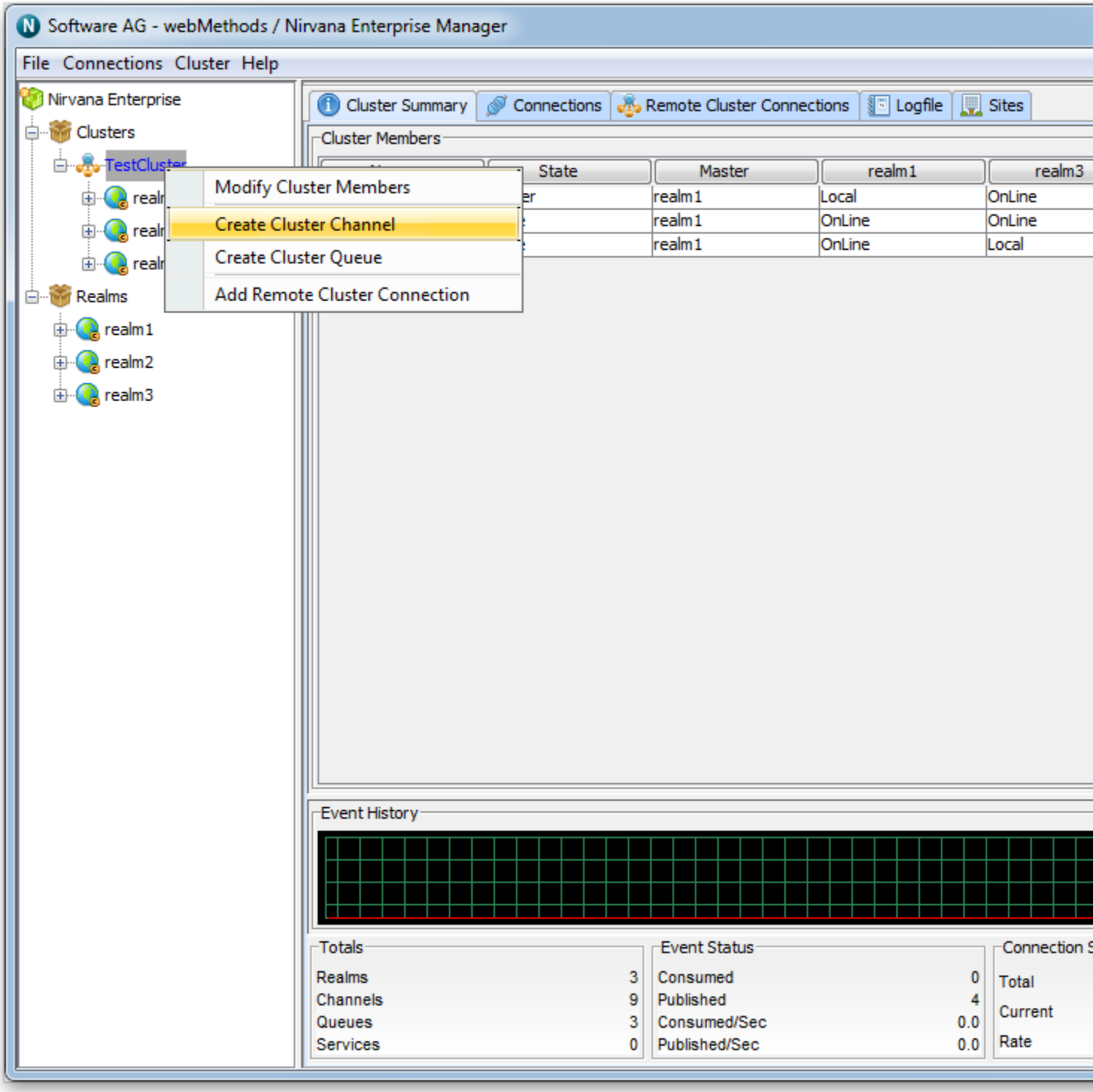
Messaging Client and Admin APIs, as you would with normal channels that are not cluster wide channels. Clustered channels can also be monitored and managed using the Enterprise Manager.

When you connect to a Universal Messaging realm in the Enterprise Manager, all resources and services found within each realm's namespace are displayed in a tree structure under each realm node. Each cluster node also displays the member realms that make up the cluster.

Creating Cluster Channels

To create new cluster channels, you must first create a cluster (see "Creating a Cluster" on page 102) if one does not already exist.

Secondly, in order to create a cluster channel, you must select the cluster node from the namespace tree where the channel will be created. For example, if there is a cluster called 'TestCluster', which contains 3 realms called 'realm1', 'realm2' and 'realm3' and you want to create a channel called '/eur/gbp' within that cluster of realms, you would need to first of all click on the cluster node called 'TestCluster'. Then, by right-clicking on the cluster node a pop-up menu will be displayed that shows a number of menu items (as shown in the image below).

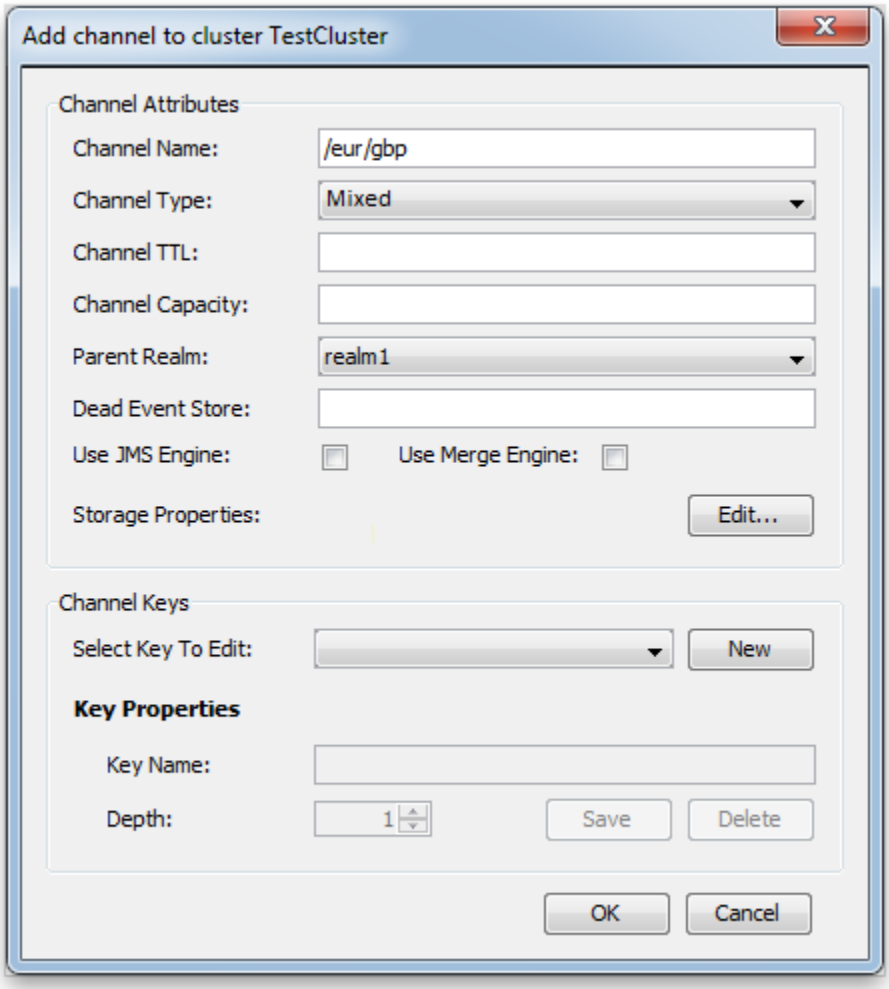


By clicking on the menu item 'Create Cluster Channel', you will be prompted with a dialog box that allows you to enter the details of the cluster channel you wish to create.

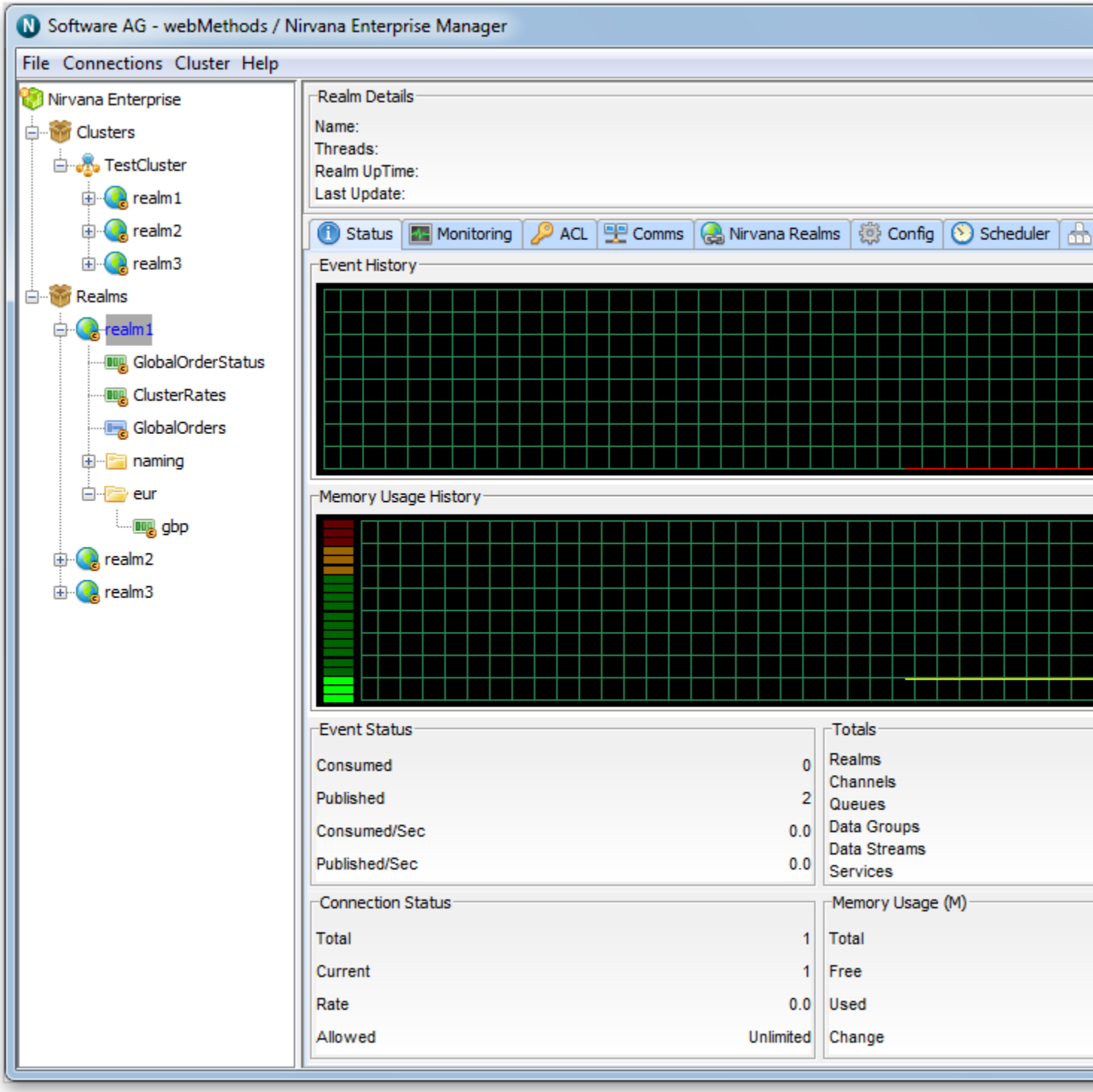
Cluster channels have exactly the same set of attributes assigned to them as normal channels when they are created.

The create channel dialog for cluster channels allows you to input values for each of these attributes. The only difference is that the channel will be created across all of the realms within the cluster and the same state will be maintained between all instances of that channel by the cluster realms. This means, for example, that if an event is published to a clustered channel it becomes available simultaneously on all realms in the cluster.

In order to create a Mixed cluster channel called '/eur/gbp' the following settings would be configured:



Clicking on the 'OK' button will create the channel '/eur/gbp' across all realms within the cluster 'TestCluster' and render the channel object in the namespace tree of the Enterprise Manager. The image below shows how the namespace tree looks after the cluster channel has been created.



As you can see from the image above, each realm node now contains the channel node in its namespace tree under a folder (which we call a container node) called '/eur'. The icon used for a cluster channel is different from that of normal channel and is denoted by the small letter 'c' in the icon, whereas the normal channel icon does not contain the 'c'.

Cluster Queue Administration

This section describes the process of creating queues on a cluster of Universal Messaging realm servers. Each cluster queue that is created consists of a physical object within each Universal Messaging realm within the cluster as well as its logical reference within each realm's namespace.

Note:

Simple and Transient queues are not supported across realms in a cluster.

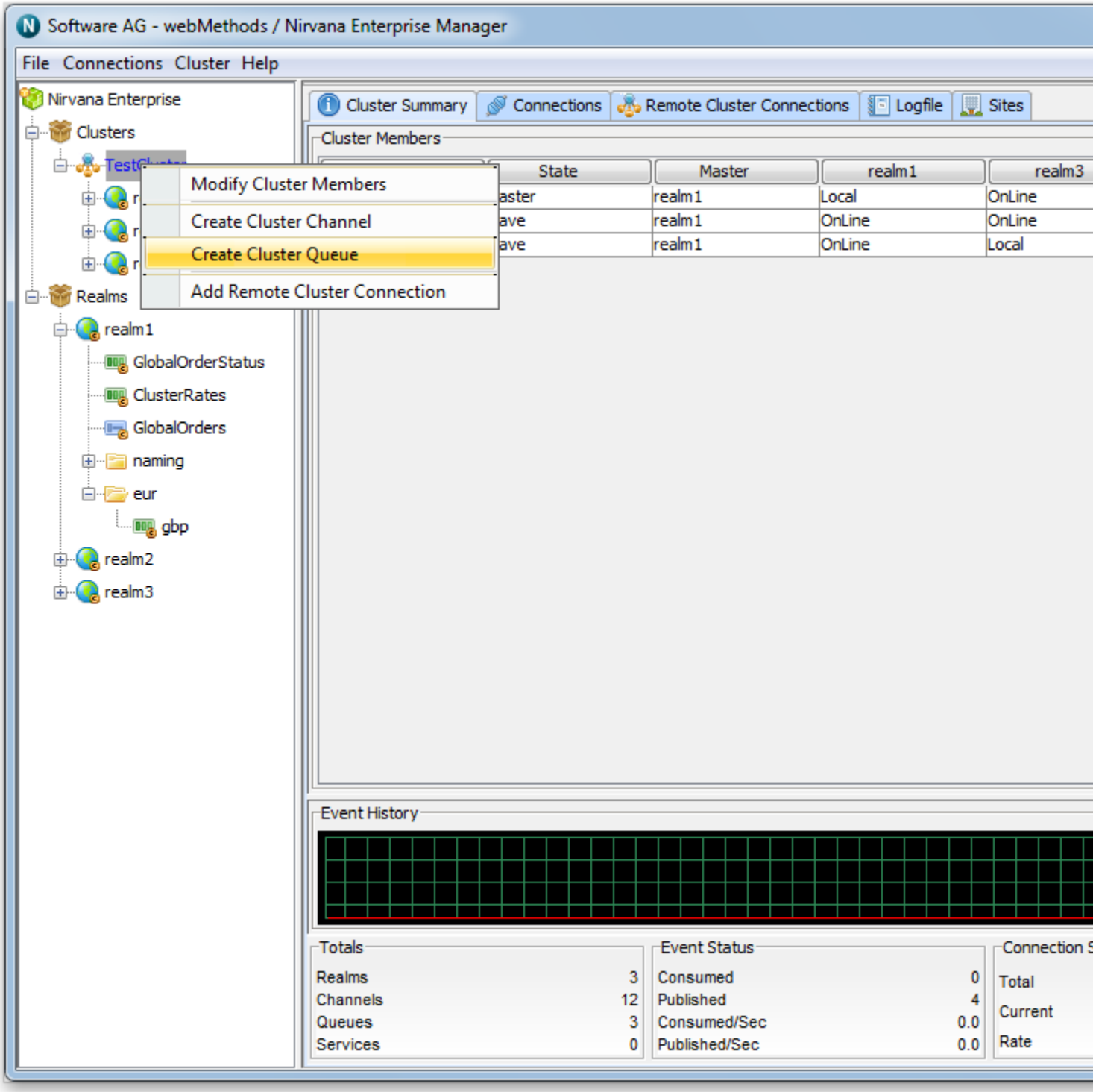
Creating queues using the Enterprise Manager creates the physical object within each cluster realm. Once created, references to the cluster queues can be obtained using the Universal Messaging Client and Admin APIs, as you would with normal queues that are not cluster wide queues. Clustered queues can also be monitored and managed using the Enterprise Manager.

When you connect to a Universal Messaging realm in the Enterprise Manager, all resources and services found within each realm's namespace are displayed in a tree structure under each realm node. Each cluster node also displays the member realms that make up the cluster.

Creating Cluster Queues

To create new cluster queues, you must first create a cluster (see "Creating a Cluster" on page 102) if you have not already done so.

Secondly, in order to create a cluster queue, you must select the cluster node from the namespace tree where the queue will be created. For example, if there is a cluster called 'TestCluster', which contains 3 realms called 'realm1', 'realm2' and 'realm3' and you want to create a queue called "/eur/orders" within that cluster of realms, you would need to first of all click on the cluster node called 'TestCluster'. Then, by right-clicking on the cluster node a pop-up menu will be displayed that shows a number of menu items (as shown in the image below).

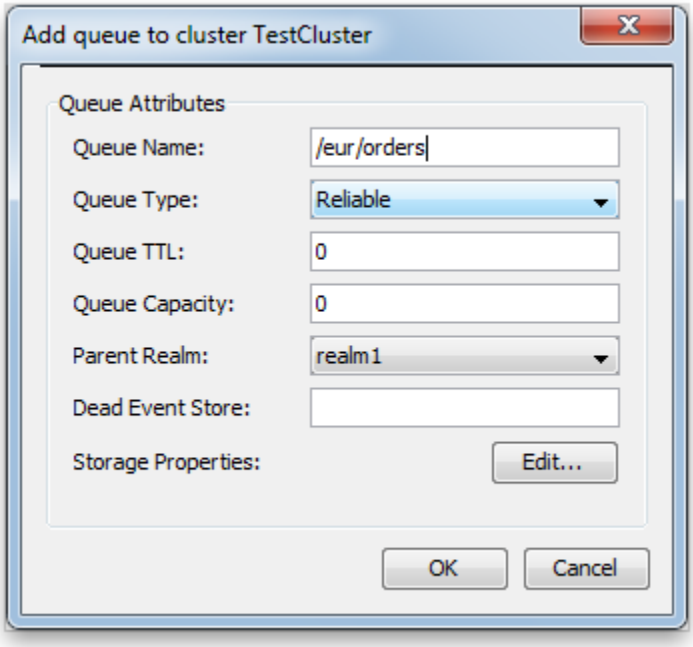


By clicking on the menu item 'Create Cluster Queue', you will be prompted with a dialog box that allows you to enter the details of the cluster queue you wish to create.

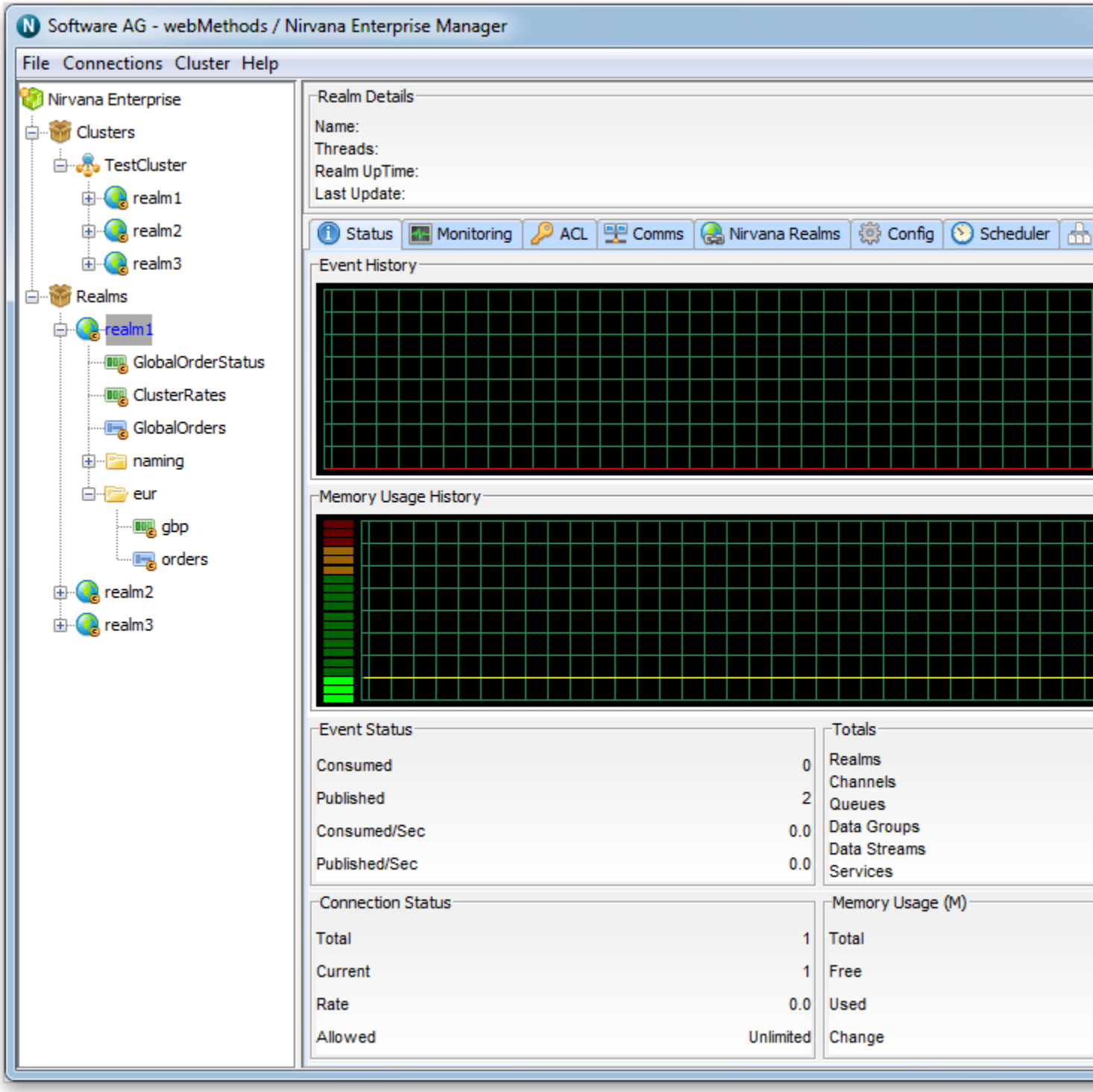
Cluster queues have exactly the same set of attributes assigned to them as normal queues when they are created.

The create queue dialog for cluster queues allows you to input values for each of these attributes. The only difference will be that the queue will be created across all of the realms within the cluster and the same state will be maintained between all instances of that queue by the cluster realms.

In order to create a cluster queue called '/eur/orders' attributes you would add the attributes as shown below:



Clicking on the 'OK' button will create the queue '/eur/orders' across all realms within the cluster 'TestCluster' and render the queue object in the namespace tree of the Enterprise Manager, both under each realm in the cluster as well as each under each realm underneath the Realms container node. The image below shows how the namespace tree looks after the cluster queue has been created.



As you can see from the image above, each realm node now contains the queue node in its namespace tree under a folder (which we call a container node) called '/eur'. The icon used for a cluster queue is different from that of normal queue and is denoted by a 'c' in the icon, whereas the normal queue icon does not have a 'c'.

Setting up Inter-Realm Communication

Communication between realms can occur in various configurations:

- between realms in the same cluster.
- between realms in a zone.
- between realms in connected clusters.

The communication between realms can be secure (encrypted) or non-secure (non-encrypted). The communication is implemented by defining one or more *interfaces* on each realm. The required setup of the interfaces is the same, regardless of which of the above configurations you use for the communication between realms. For example, the attribute **Allow for InterRealm** must be activated on the interface that you use, otherwise the communication between realms is not possible.

The following description uses some examples from working with a cluster, but the principles apply to all configurations.

Since all realms in a cluster are required to have the same configuration (so that for example if the master realm goes offline, one of the other realms can become the new master), you must ensure that any interface definitions on one realm match the interface definitions on all other realms in the cluster.

For non-encrypted inter-realm communication, you can set up the interfaces to use either NSP (Socket Protocol) or NHP (HTTP Protocol). In general, we recommend you to use NSP rather than NHP for non-encrypted inter-realm communication.

For encrypted inter-realm communication, you can set up the interfaces to use either NSPS (Secure Socket Protocol) or NHPS (Secure HTTP Protocol). In general, we recommend you to use NSPS rather than NHPS for encrypted inter-realm communication.

Information about using the Enterprise Manager to manage a cluster is contained in the section *Cluster Administration*. Information about managing realm interfaces is contained in the section *TCP Interfaces, IP Multicast and Shared Memory*. Managing zones is described in the section *Zone Administration*. Setting up an inter-cluster connection is described in the section *Interconnecting Two Clusters*, and conceptual details are provided in the section *Data Routing using Channel Joins* in the Concepts Guide.

Setting up non-encrypted inter-realm communication

Each realm contains by default one predefined interface, and this interface uses the NSP protocol (i.e. socket protocol without encryption). Also by default, the interface is configured to be usable for inter-realm communication as well as for communication between realm and clients.

If you do not define any additional interfaces on the realm, all communication between the realm and other realms, and between the realm and clients, will use this interface. You can set up a cluster consisting of multiple realms, each of them having just this one default interface defined.

However, in general we recommend you to set up two NSP interfaces on each realm for non-encrypted communication, namely one interface for only inter-realm communication and one

interface for only client communication to the realm. The options for setting up this configuration are available in the Enterprise Manager, under the **Comms > Interfaces > Basic** tab of each realm.

On the interface that you will use for inter-realm communication, use the following settings:

- **Allow for InterRealm**: yes
- **Allow Client Connections**: no

Similarly, on the interface that you will use for client communication with the realm, use the following settings:

- **Allow for InterRealm**: no
- **Allow Client Connections**: yes

After you make these changes, restart all of the realms, to ensure that the new interfaces are activated.

When you form the cluster, communication between realms in the cluster will use the NSP interface that you have configured for inter-realm communication.

Setting up encrypted inter-realm communication

The assumed starting point in this scenario is that there is no cluster formed yet. All of the realms that will later form the cluster need to be configured.

The steps required are as follows:

1. If you intend to use self-signed certificates, or if you intend to use a custom truststore (which contains the public certificates associated with each Universal Messaging realm's private certificate), the keystore and the truststore must be added to the Universal Messaging JVM process.

   In the file `Server_Common.conf` on each realm, provide details of the truststore and keystore, according to the following pattern:

   ```
   wrapper.java.additional.7="-Djavax.net.ssl.trustStore=<TRUSTSTORE>
   wrapper.java.additional.8=-Djavax.net.ssl.trustStorePassword=<TRUSTSTORE_PWD>
   wrapper.java.additional.9="-Djavax.net.ssl.keyStore=<KEYSTORE>
   wrapper.java.additional.10=-Djavax.net.ssl.keyStorePassword=<KEYSTORE_PWD>
   ```

   for example

   ```
   wrapper.java.additional.7="-Djavax.net.ssl.trustStore=
      /webmethods/truststores/um_truststore.jks"
   wrapper.java.additional.8=-Djavax.net.ssl.trustStorePassword=nirvana
   wrapper.java.additional.9="-Djavax.net.ssl.keyStore=
      /webmethods/keystores/um_keystore.jks"
   wrapper.java.additional.10=-Djavax.net.ssl.keyStorePassword=nirvana
   ```

   See the section *Server Parameters* in the *Concepts* guide for general information about setting up such parameters.

2. On each realm in the cluster, add two secure interfaces:

a. Add one interface using the NSPS protocol, to be used only for inter-realm communication.

> **Note:**
> The demo certificates generated by the Universal Messaging Certificate Generator tool (see the section "How to generate certificates for use" on page 301) are only valid for the loopback interface (localhost / 127.0.0.1). Therefore, if you use these demo certificates, ensure that the adapter that you add is bound only on the loopback interface.

For this interface, set the following options (in the Enterprise Manager, they are located under the **Basic** or **Certificates** tabs of the interface definition screen):

- **Allow for InterRealm**: yes
- **Allow Client Connections**: no
- **Enable client certificate validation**: no

  The reason for disabling client certificate validation is because Universal Messaging does a certificate exchange between realms already when constructing a cluster, so doing another certificate exchange at the SSL layer would be redundant.
- Specify Certificates and Truststore on the interface as you would normally.
- If you want to use a certain level of SSL / TLS (eg. TLS 1.2)

  1. Pick the right algorithms for that interface.
  2. Enforce the SSL level in the realm (using a JVM argument in `Server_Common.conf`). Example: to enforce TLS1.2 globally on the Universal Messaging server, set:

     ```
     wrapper.java.additional.XX=-DSSLProtocols=TLSv1.2
     ```

b. Add one more interface using the NSPS protocol, to be used only by clients for communication with the realm. For this interface, set the following options:

- **Allow for InterRealm**: no
- **Allow Client Connections**: yes
- **Enable client certificate validation**: no

3. Disable the setting for inter-realm communication on the original, non-encrypted, interface.
4. Close and restart the Enterprise Manager.
5. Restart all Universal Messaging realms (to make sure all JVM arguments are activated).
6. Use the Enterprise Manager to form the cluster.

Switching from non-encrypted to encrypted inter-realm communication

The assumed starting point in this scenario is that there is already a cluster in which the inter-realm communication is not encrypted, i.e. the interface protocol is NHP or NSP, and you want to change this to encrypted communication, i.e. using the interface protocol NHPS or NSPS.

Here are the steps to follow to switch from non-encrypted to encrypted inter-realm communication in a Universal Messaging cluster:

1. Close the cluster and stop any running realms.
2. In the file `Server_Common.conf` on each realm, provide details of the truststore and keystore, as described in the previous section ("Setting up encrypted inter-realm communication" on page 120).
3. Restart all realms.
4. On each realm, create two NSPS interfaces, as described in the previous section.
5. Under the **Certificates** tab for each of the NSPS interfaces, add a reference to the custom truststore and the keystores containing the server signed certificates, for example:

   ```
   Key store path : /webmethods/keystores/um_keystore.jks
   Trust store path : /webmethods/truststores/um_truststore.jks
   ```

6. Close Enterprise Manager.
7. Set the environment variables CAKEYSTORE and CAKEYSTOREPWD for each realm to reference the truststore containing the CA root chain, and the truststore's password. You can set up these variables as follows:

   a. Open the file `Admin_Tools_Common.conf` that is located in `UniversalMessaging/java/<instanceName>/bin`, where `<instanceName>` is the name of the realm server.

   b. Locate the lines

   ```
   set.default.CAKEYSTORE=
   set.default.CAKEYSTOREPASSWD=
   ```

   c. Set these variables to the required values, for example:

   ```
   set.default.CAKEYSTORE=/webmethods/keystores/um_keystore.jks
   set.default.CAKEYSTOREPASSWD=nirvana
   ```

   Note that if these variables have already been assigned a value elsewhere in the session, for example in a startup script, the values defined here in `Admin_Tools_Common.conf` will be ignored.

8. Restart Enterprise Manager.

   By restarting the Enterprise Manager after setting values for CAKEYSTORE and CAKEYSTOREPASSWD, the Enterprise Manager will be able to connect over a secured interface.

9. Disable the inter-realm connection option on each realm's non-encrypted interfaces.
10. Form the cluster.

Note on public/private keys used for inter-realm handshake

When a Universal Messaging realm starts for the first time, it automatically generates a public/private key pair for encryption purposes and stores it in the internal keystore `server.jks`

file in the realm's `data/RealmSpecific` directory. The public keys of other nodes are also added to this file whenever the realms are added to form a cluster.

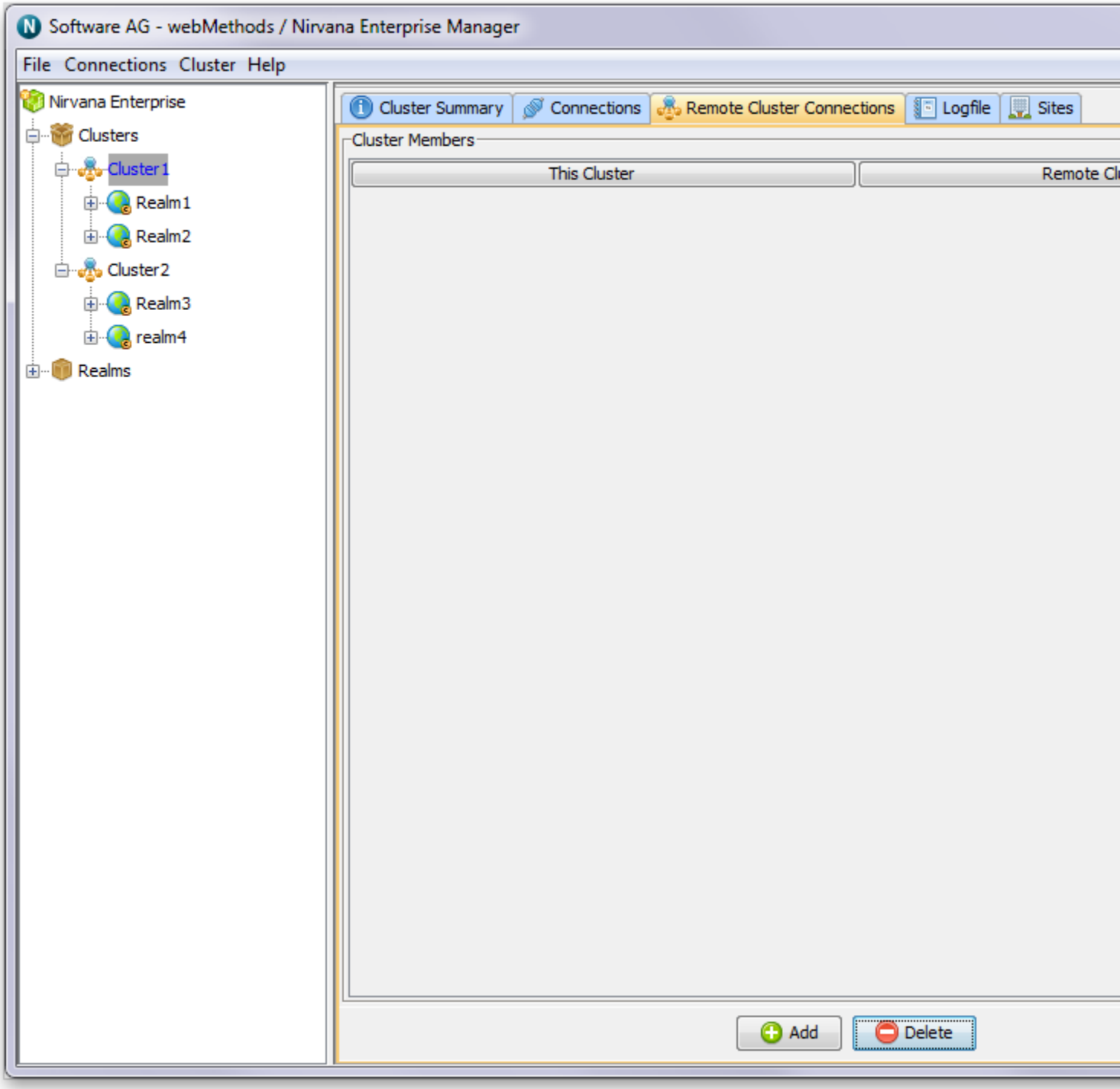
These auto-generated keys are used for server identification only; basically whenever two realms establish a connection, they will exchange a single signed message as part of the handshake routine, in order to confirm they know each other.

After this initial handshake has taken place, all encrypted communication between realms in a cluster uses separate keys and keystores, as stated in the section "Setting up encrypted inter-realm communication" on page 120 above.

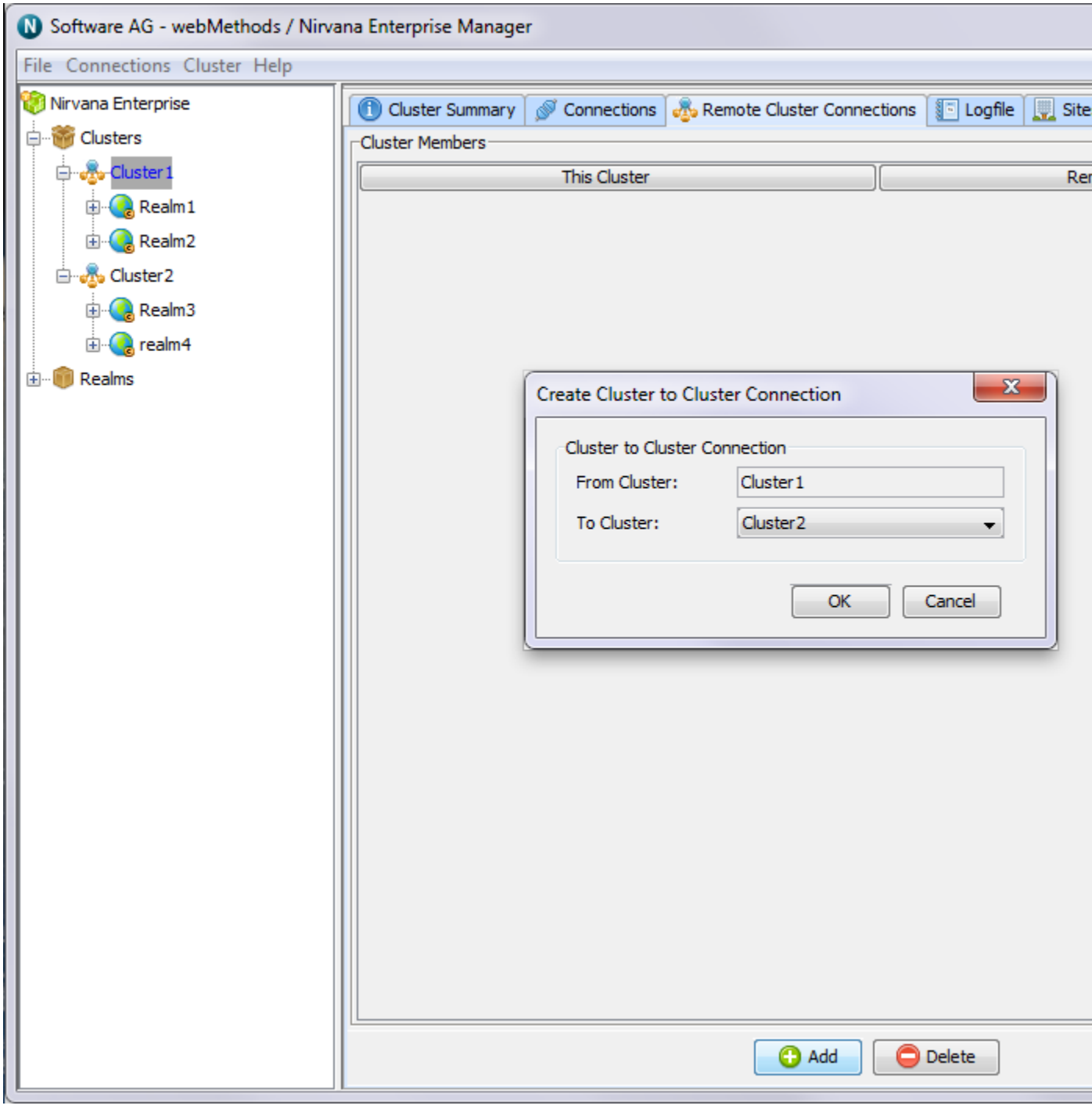
Managing Inter-Cluster Connections

Creating Inter-Cluster Connections

Inter cluster connections can be created through the Enterprise Manager. To do this, firstly connect to a realm in each cluster. Then, once both clusters are displayed in the Enterprise Manager, click on the "Inter-Cluster Connections" tab under one of the cluster panels.



Next, select "Add" and choose the remote cluster from the dropdown list in the popup dialog which will now appear:

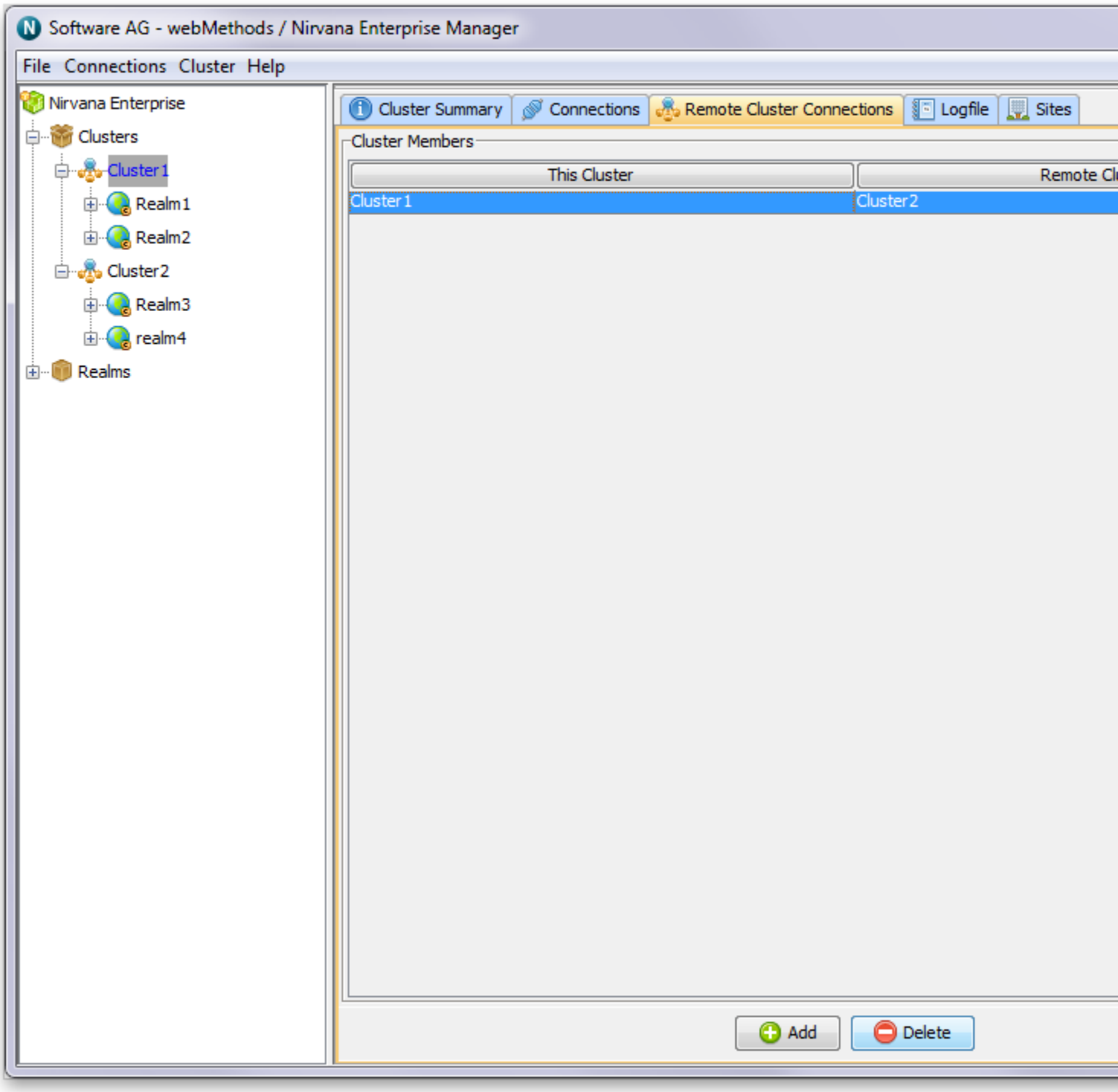


The inter-cluster connections should now be established, and inter-cluster joins can now be formed through the Enterprise Manager or programmatically.

Note:
For a description of the general principles involved in creating channel joins, see the section *Creating Channel Joins*. The description details the usage based on the Enterprise Manager, but the same general principles apply if you are using the API.

Deleting Inter-Cluster Connections

To delete an inter-cluster connection, simply select the connection from the list and click "Delete".



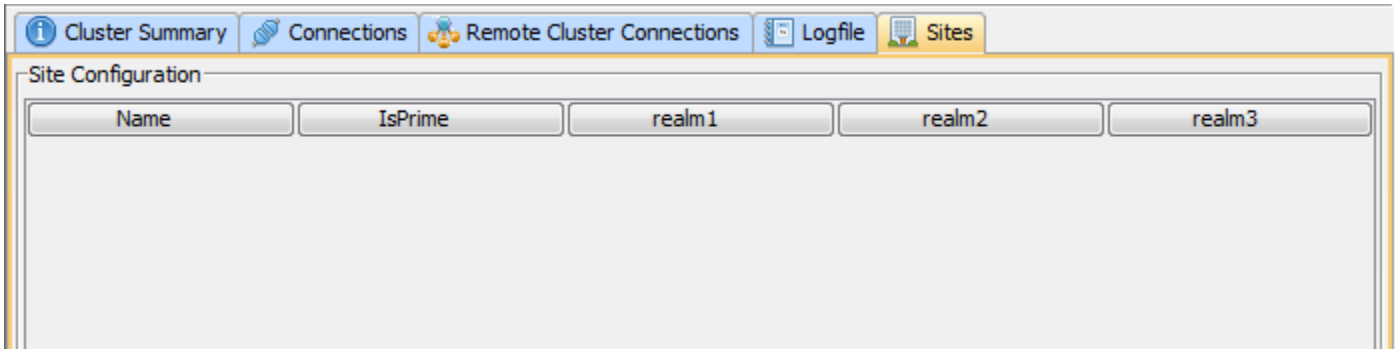
Creating and Managing Clusters with Sites

This section describes the process of modifying a Universal Messaging Cluster into a *Universal Messaging Cluster with Sites*. Clusters with Sites allow a standard Universal Messaging cluster to operate with as little as 50% of the active cluster members (as opposed to the standard 51% quorum in effect for clusters without sites), and provides administrators with a mechanism to prevent the split brain scenario that would otherwise be introduced when using exactly half of a cluster's realms.

Viewing Site Information in Enterprise Manager

The Enterprise Manager's top level view shows a tree node labelled **Universal Messaging Enterprise**. One level below this is a tree node labelled **Clusters**, which contains any known clusters.

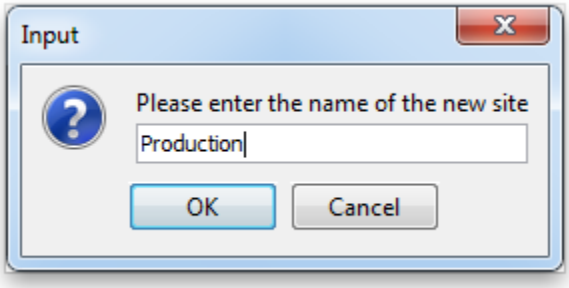
After creating your cluster (see "Creating a Cluster" on page 102) and selecting the cluster's icon in the Enterprise Manager, click the *Sites* tab:



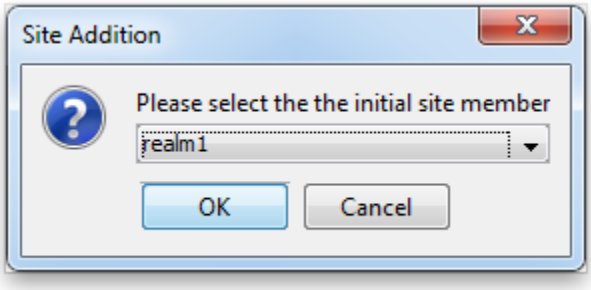
Sites Tab in Cluster View. No Sites have yet been created for the above cluster.

Creating a Primary Site for a Cluster

Click the *New* button to create the first site. We'll assume the site is named *Production*. Follow the prompts and pick a Realm to include in the site, for example realm1:

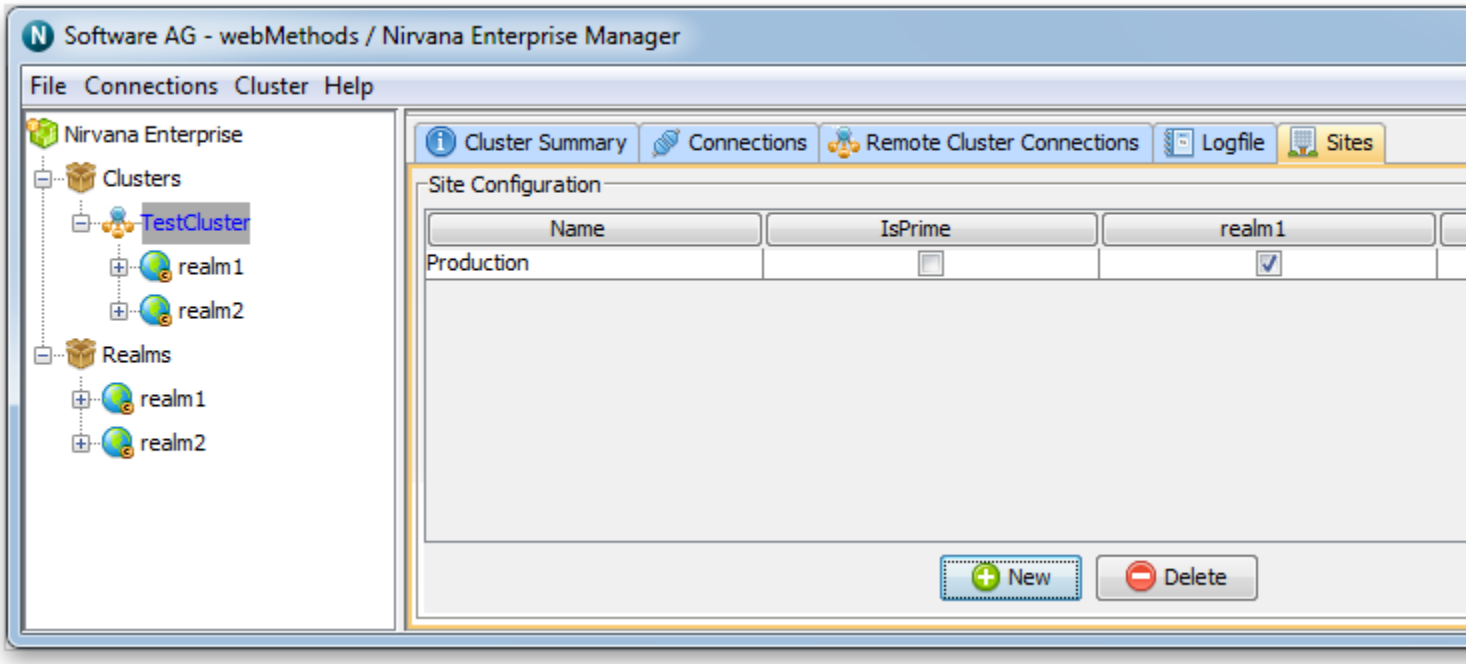


Creation of a "Production" Site.



Addition of an Initial Member to "Production" Site.

At this point, the new site will appear in the Sites tab:

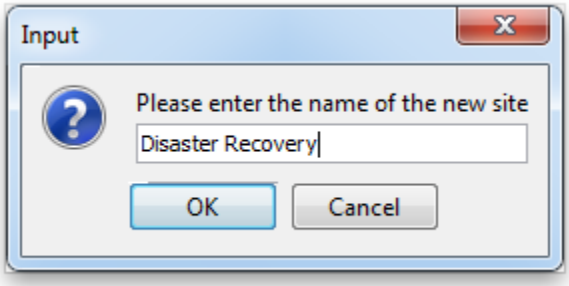


The "Production" Site with its initial member realm is shown.

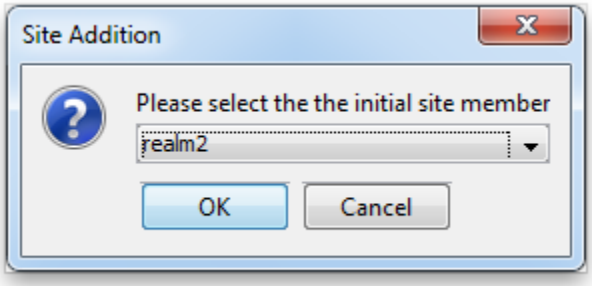
Note that the table will contain a column for all realms in the cluster. In this example we have only added one realm to the Production site. Checking and unchecking the appropriate checkboxes will add or remove clustered realms from the corresponding sites.

Creating a Backup Site for a Cluster

Next, follow the same steps to create the second site, which in this example we shall assume is named : **Disaster Recovery**

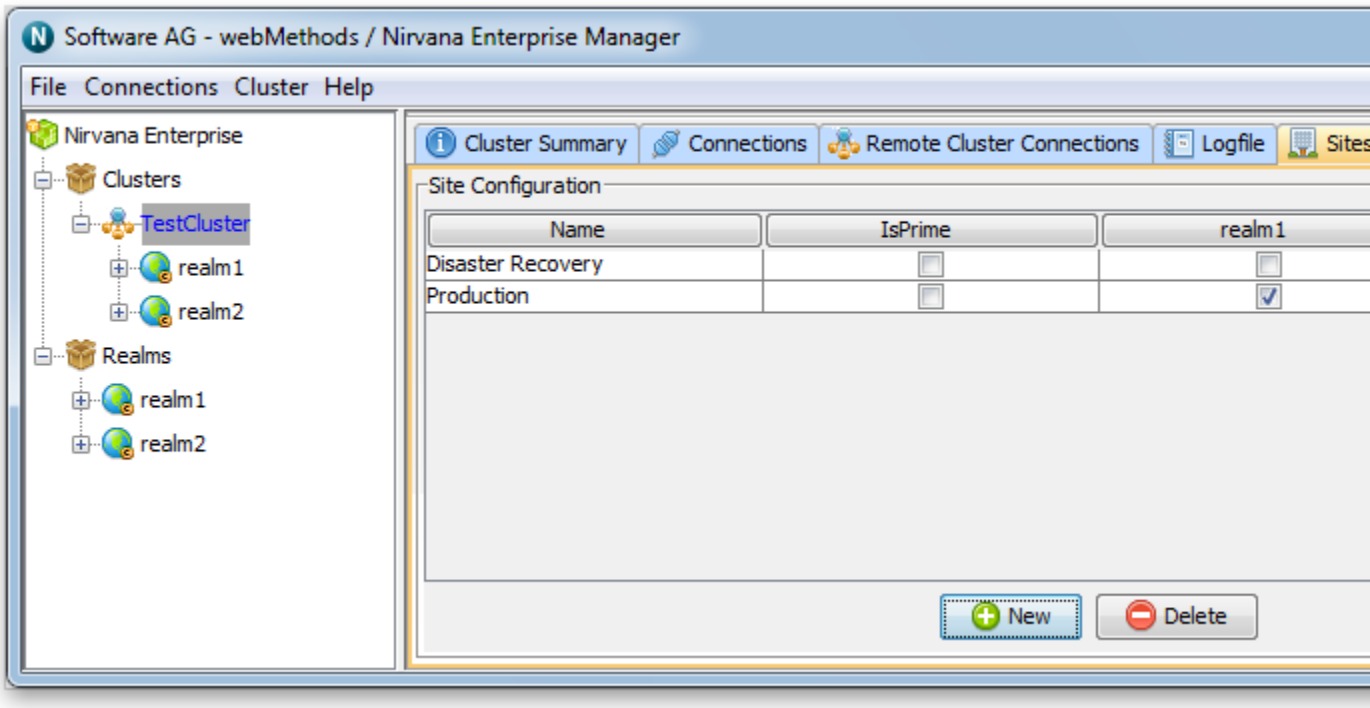


Creation of a "Disaster Recovery" Site and addition of an initial site member realm.



Addition of an Initial Member to "Disaster Recovery" Site.

The new Disaster Recovery site will now also appear in the Sites tab:

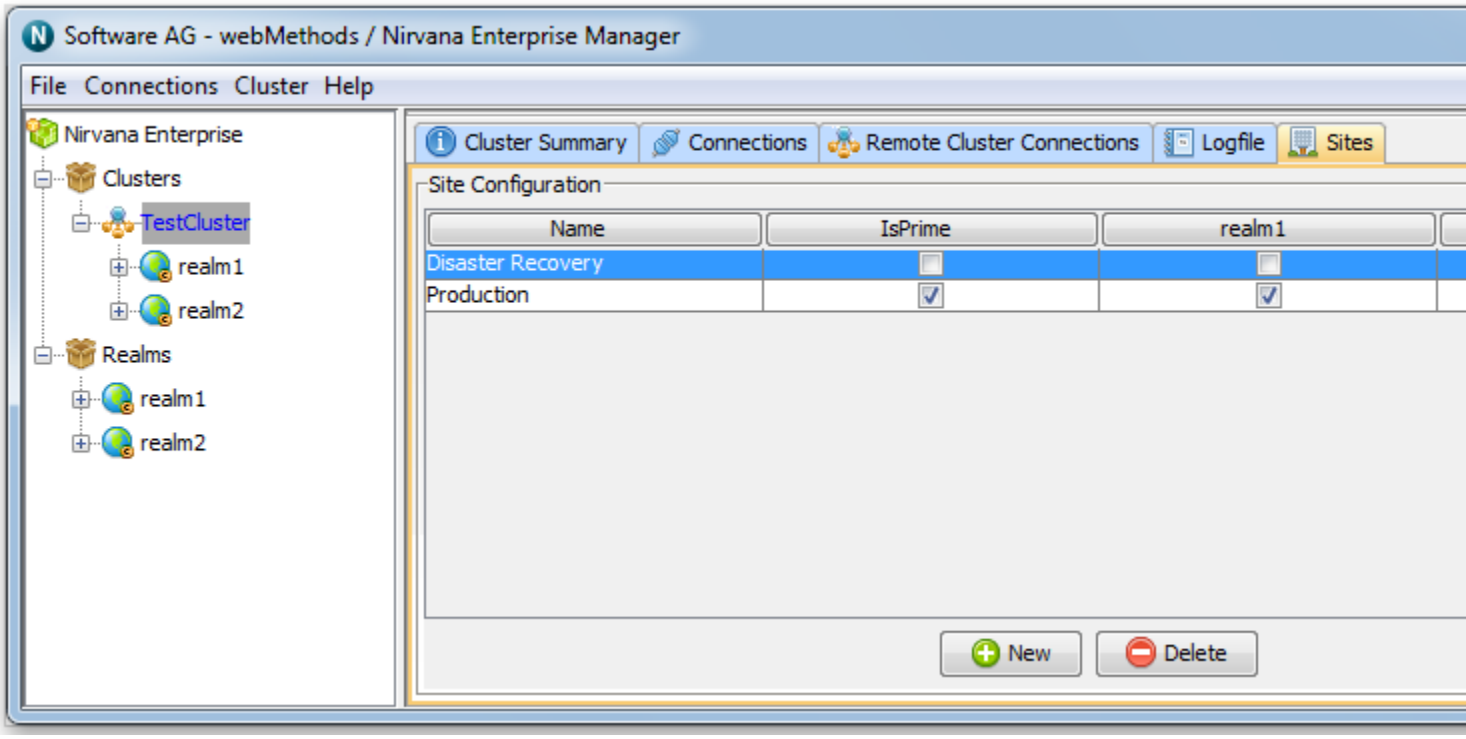


Sites Tab in Cluster View. The "Production" and "Disaster Recovery" Sites are both shown.

Setting a Site's IsPrime Flag

Administrators use a site's *isPrime* flag to determine the site that will contain a cluster's master realm.

We recommend you to make the production site the prime site. For a discussion on the pros and cons of whether to make the production site or the disaster recovery site the prime site, refer to the section *Clusters with Sites* in the *Concepts* guide..



Setting the Production Site to be Prime by checking its isPrime checkbox.

Channel Administration

Overview

The Enterprise Manager allows monitoring, administration and configuration of Universal Messaging channels.

Channel Status

The Universal Messaging Enterprise Manager allows monitoring of a channel's status in terms of publish & consume event totals / rates as well as connection total / rates and persistent store / memory.

Channel Access Control List (ACL)

Universal Messaging offers complete control over security policies. Universal Messaging stores security policies locally or can be driven by any external entitlements service. Universal Messaging's rich set of entitlements ensure that everything from a network connection through to a channel/queue creation can be controlled on a per user and/or host basis. For more information, see the Universal Messaging ACL's FAQ.

Channel Joins

Universal Messaging allows channels to be joined to other channels or queues creating server side routing tables with the possibility to apply filters based on message content on the local or a remote Universal Messaging realm.

Channel Connections

Channel subscribers are reported as channel connections and can be monitored or managed through the Universal Messaging Enterprise Manager.

Channel Durables

Channel subscribers can manage their subscription's event id manually or they can become a named subscriber and let that be managed by the Universal Messaging realm. The Universal Messaging Enterprise Manager allows complete management of channel durables.

Channel Event Snooping

The Universal Messaging Enterprise Manager provides the ability to inspect the contents of messages remotely using the Snoop panel.

Viewing the Channel Status

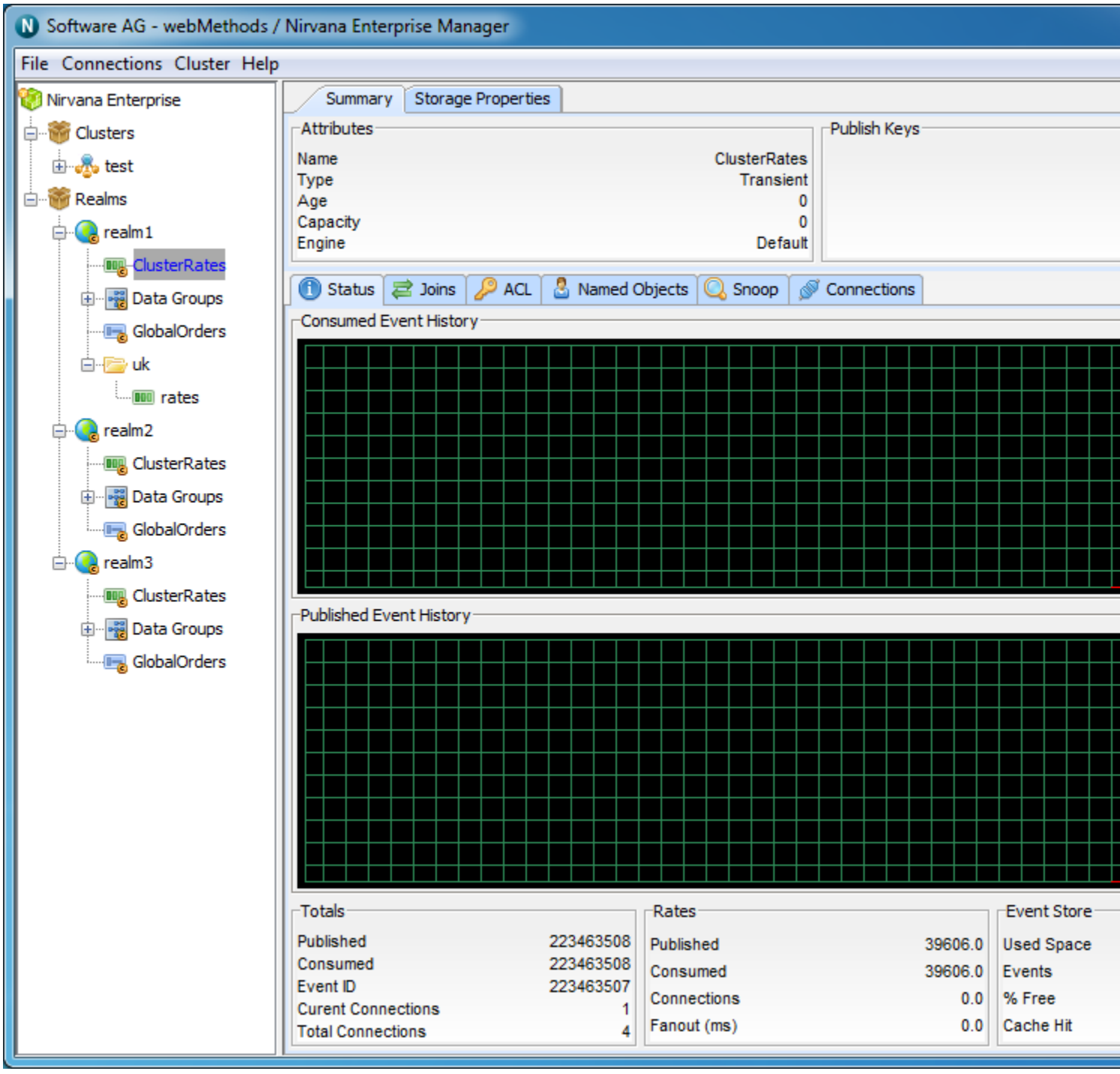
Introduction

When you select a channel object from the namespace, the first panel to be displayed on the right hand side of the Enterprise Manager panel is the 'Status' panel. Configuration information is always displayed at the top section of the Enterprise Manager when a channel is selected. This configuration information shows channel type, TTL (age), capacity as well as any channel key information available. The channel 'Status' tab shows real-time management information for the selected channel.

The status panel is split into 2 main sections. The top section shows real time graphs representing the events published and consumed on the channel, both in terms of rates (i.e. per status interval) as well as the totals.

The bottom section shows the actual values plotted in the graphs for events published and consumed, as well as information about the actual channel store at the server.

The image below shows the status panel for an active cluster channel.



The top most graph in the panel shows the event history for events consumed from the channel. The red line graphs the rates at which events are being consumed while the yellow line graphs the total events consumed from the channel.

The bottom graph shows the event history for events published to the channel. The red line graphs the rates at which events are being published while the yellow line graphs the total events published

to the channel. As the status events are consumed, and the channel (nLeafNode) is updated with the new values for events consumed and published, the status panel and its graphs will be updated.

The bottom section of the status panel shows 3 types of information: Totals, Rates and Event Store. These are discussed below.

Totals

The totals section shows 5 values:

- **Published** - The total number of events published to the channel when the last status events was consumed
- **Consumed** - The total number of events consumed from the channel when the last status event was consumed
- **Event ID** - The event id of the last event published to the channel
- **Current Connections** - The current number of consumers on the channel
- **Total Connections** - Total number of subscribers that have subscribed to the channel

Rates

The rates section shows 3 values:

- **Published** - The current rate of events published to the channel, calculated as (total - previous total) / (interval 1000 milliseconds)
- **Consumed** - The current rate of events consumed from the channel, calculated as (total - previous total) / (interval 1000 milliseconds)
- **Connections** - The current rate of subscriptions being made to the channel

Event Store

The event store section shows 4 values:

- **Used Space** - The amount of space in KB used by the channel on the server (either memory, or disk for persistent / mixed channels)
- **Events** - The current number of events on the channel
- **% Free** - The amount of free space in the channel store (calculated as (used space - (total space used by all purged or aged events))
- **Cache Hit** - The %age of events consumed from the channel event cache as opposed form the actual physical store if persistent or mixed

Creating Channels

This section describes the process of creating a Universal Messaging channel on Universal Messaging realm servers. Channels are the logical rendezvous point for data that is published and subscribed. If you are using Universal Messaging Provider for JMS then channels are the equivalent of JMS topics.

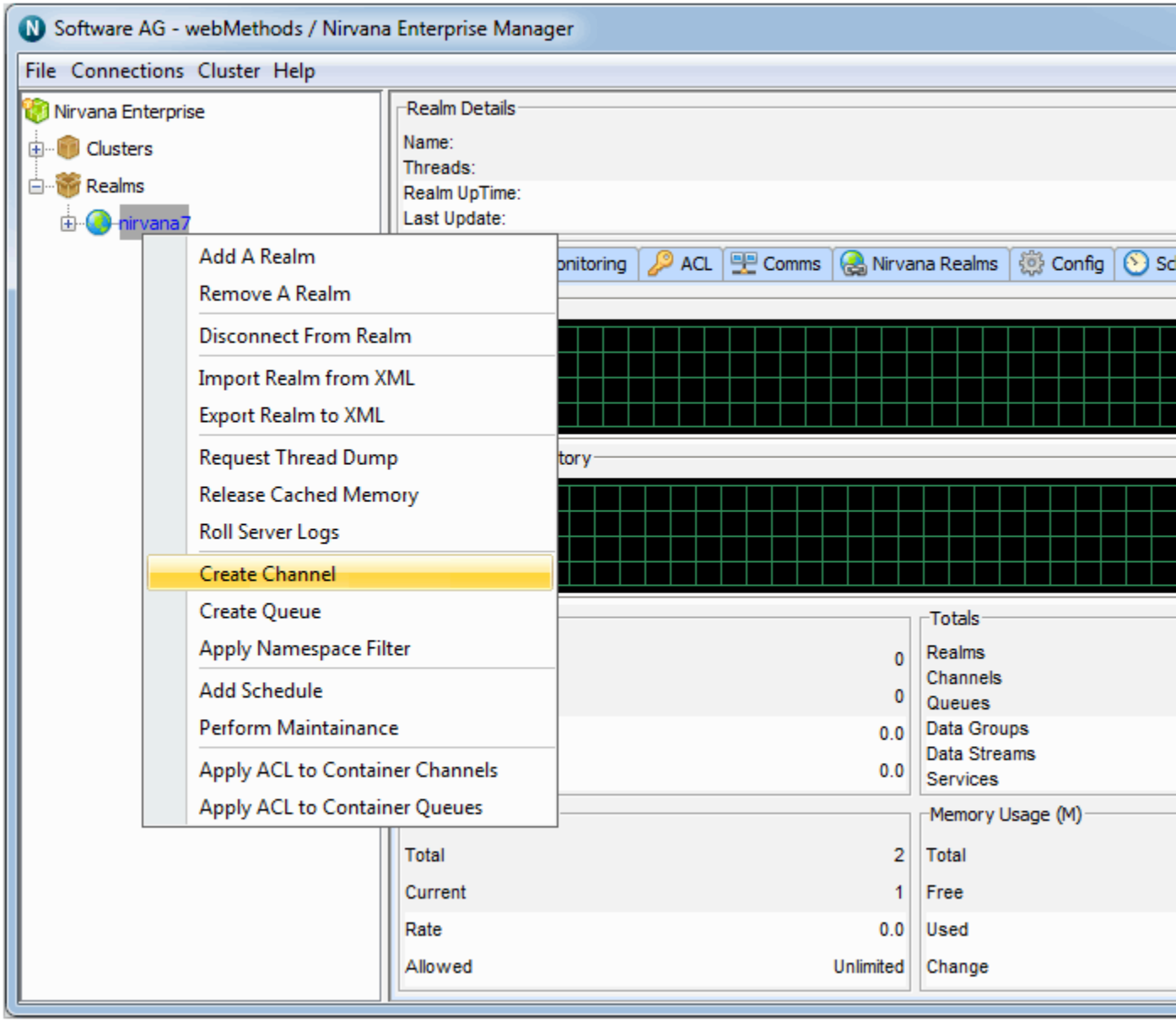
Each channel that is created consists of a physical channel within the Universal Messaging realm as well as its logical reference within a namespace that may be made up of resources that exist across multiple Universal Messaging realm servers. Creating channels using the Enterprise Manager creates the physical object within the realm. Once created, references to channels can be obtained using the Universal Messaging Client and Admin APIs. Channels can also be monitored and managed using the Enterprise Manager.

When you connect to a Universal Messaging realm in the Enterprise Manager, all resources and services found within the realm namespace are displayed in a tree structure under the realm node itself. It is possible to view multiple Universal Messaging realm servers from a single enterprise manager instance.

Creating Channels on a Universal Messaging Realm

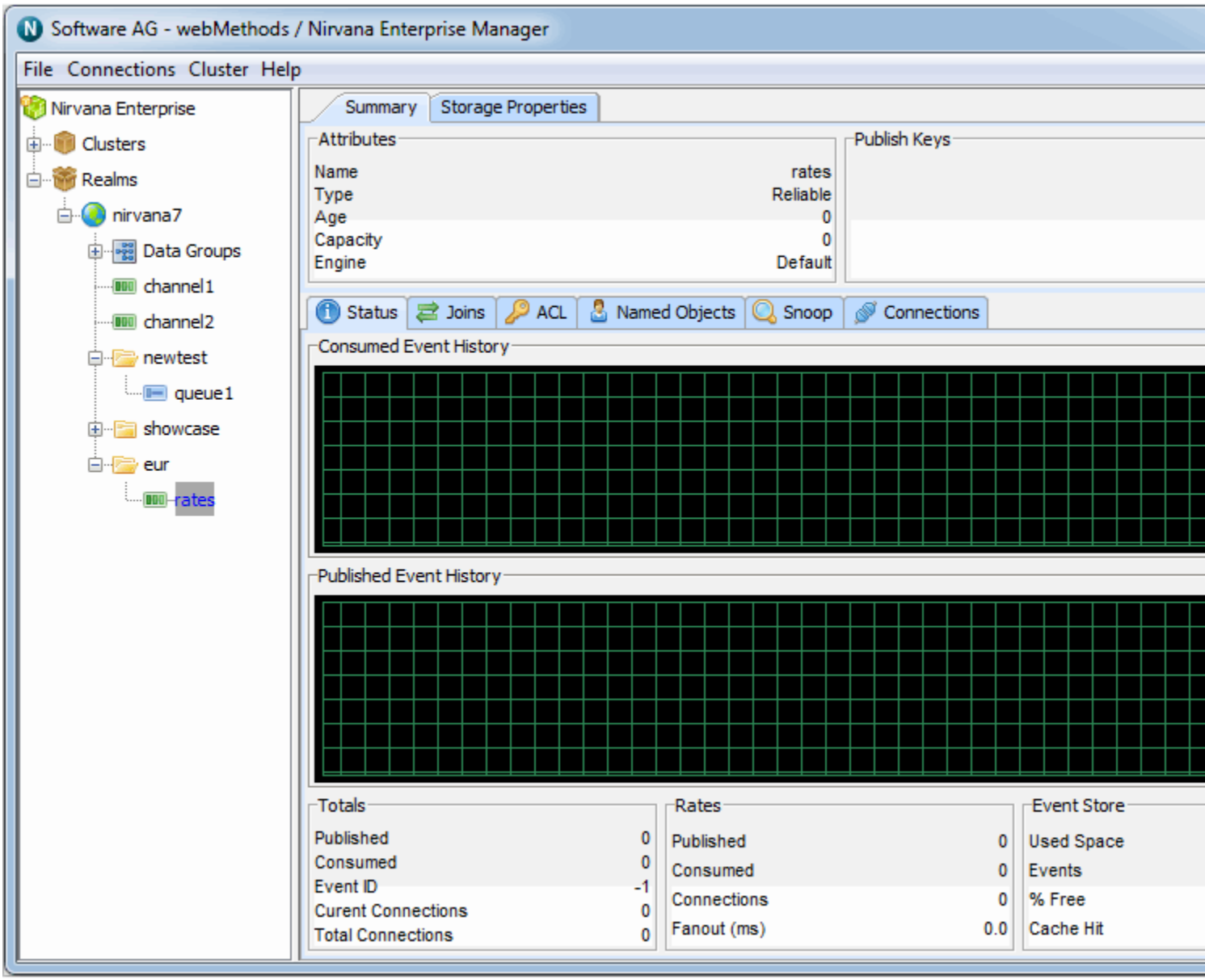
To create new Universal Messaging channels, the Enterprise Manager provides a number of options. Firstly, in order to create a channel, the branch where the channel will exist needs to be selected within the namespace tree.

For example, to create a channel called '/eur/rates' on a Universal Messaging realm called 'nirvana7' simply right-click on the realm node to display a pop-up menu which contains a 'Create Channel' option.



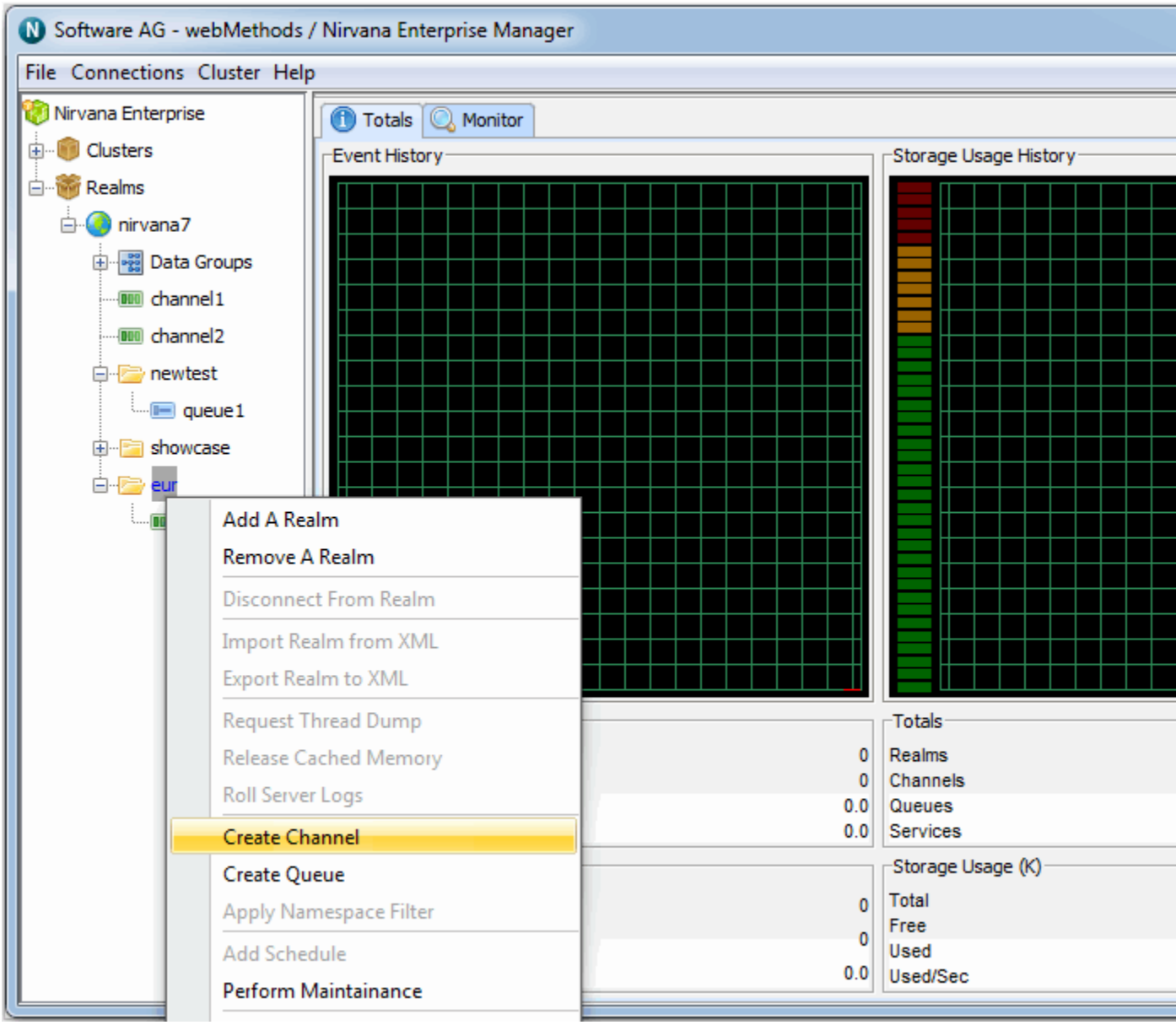
When you select this option, the resulting dialog allows you to configure the channel behavior. See the sections "Channel Attributes" on page 137, "Channel Storage Properties" on page 138 and "Google protobuf descriptors" on page 139 below for a description of the configuration options.

Clicking on the 'OK' button in the Add Channel dialog will create the channel with the name you specified (in this example, '/eur/rates') on the Universal Messaging realm 'nirvana7' and render the channel object in the namespace tree of the Enterprise Manager underneath the realm node. This is shown in the image below.



As you can see from the image above, the channel node in the tree has been created under a folder (which we call a container node) called '/eur' under the realm 'nirvana7'.

It is also possible to create channels directly underneath container nodes. For example, if we wished to create another channel called '/eur/trades', we could repeat the process described above using the full absolute name of the channel. This would again create a channel called trades under the container node /eur. Alternatively, we can select the /eur node and create the new channel using its relative name /trades. Selecting the container node and right-clicking on the node, shows another pop-up menu of options for container nodes. One of the options is 'Create Channel'. The image below shows this menu as it appears when the container is right-clicked.



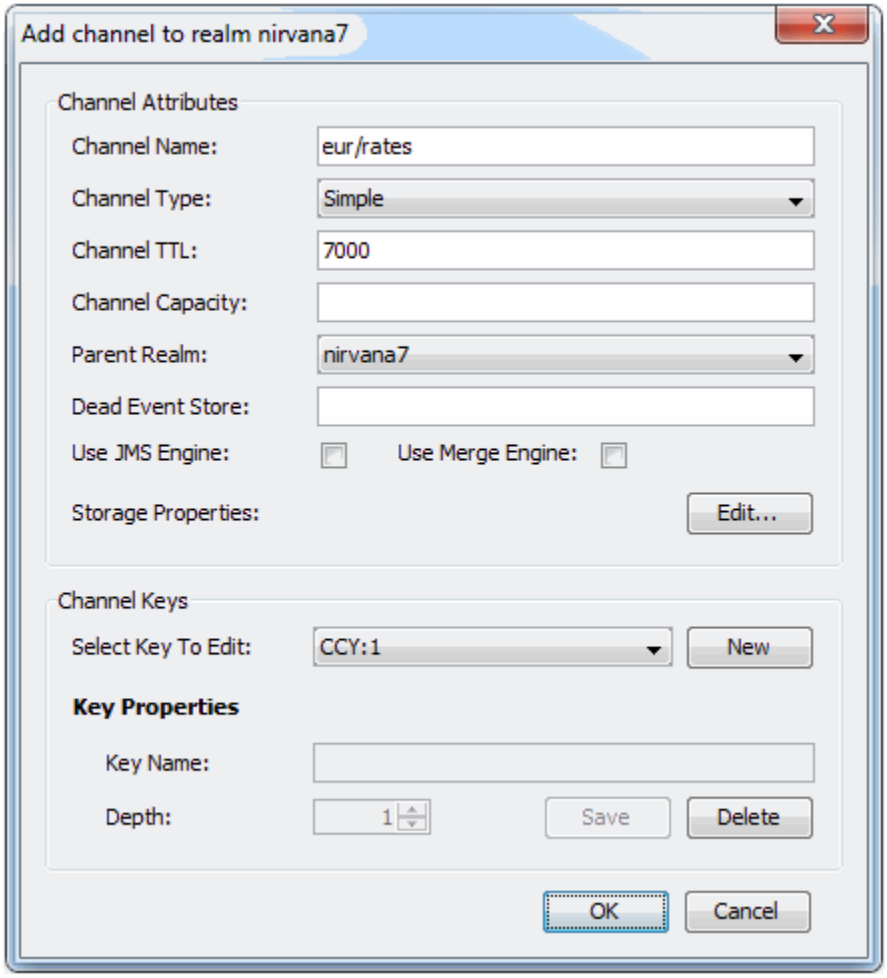
By selecting the menu item, 'Create Channel' from the container node, you are once again presented with the create channel dialog. This dialog looks like the dialog used previously, except the dialog shows that the channel will be created under the container /eur.

Channel Attributes

Channels have a set of attributes assigned to them when they are created. The Add Channel dialog allows you to input values for each of these attributes.

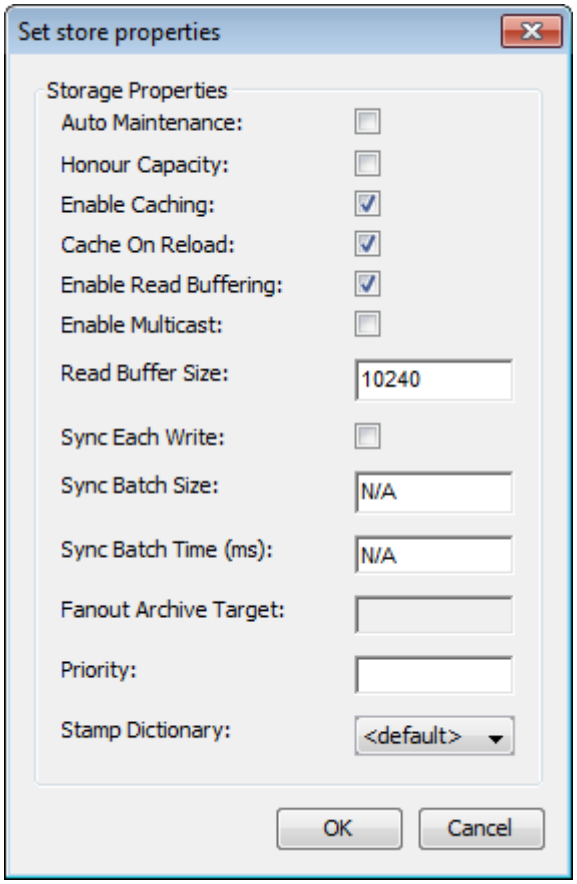
For information about the available channel attributes, such as Channel Type and Channel TTL (time-to-live), see the summary of Channel Attributes in the Commonly Used Features section of the Universal Messaging Concepts guide.

In the example below, the channel '/eur/rates' will be created with a channel type of Simple and a TTL of 7 seconds.



Channel Storage Properties

There are a number of Storage Properties associated with the channel. The storage properties allow you to configure the operational environment of the channel. The properties can be set by clicking the "Edit..." button to the right of "Storage Properties" in the Add Channel dialog.



For information about the available channel attributes, see the summary of Storage Properties in the Commonly Used Features section of the Universal Messaging Concepts guide.

Google protobuf descriptors

If you wish to activate Google protocol buffer (also called "protobuf") descriptors for the channel, click the "Set..." button to the right of "Protobuf Descriptor" in the Add Channel dialog. This allows you to specify a file containing the required protobuf descriptors.

For general information on protocol buffers, refer to the section *Google Protocol Buffers* in the *Concepts* guide.

Valid Channel and Queue Names

The channel and queue names can contain any of the following characters:

- All letters and digits
- Slash "/"
- Hyphen "-"
- Underscore "_"
- Hash symbol "#"

Certain character strings are replaced:

- Backslash "\" is replaced by slash "/"
- Two colons "::" are replaced by slash "/"
- Two slashes "//" are replaced by one slash "/"

Valid names have the following length restrictions:

- Channel and queue names have a maximum limit of 235 characters.
- Namespace names have a maximum limit of 255 characters.
- The full path containing the data directory folder path with a channel or queue path appended to it has a maximum limit of 4096 characters.

If your channel or queue name contains slash characters, for example "a/b/c", this is represented in the Enterprise Manager view as a hierarchy, with "a" being the top node, "b" being the child node of "a", and "c" being the child node of "b". This virtual hierarchy is just a visual aid to help you to keep track of your channels and queues, but the store itself is not divided internally into hierarchical parts and can only be referenced by the full name, which in this example is "a/b/c".

Note:
There is a restriction that a channel or queue name cannot be the same as an existing folder name. So if you have named a channel "a/b/c", you cannot name a different channel "a" or "a/b". This would lead to a display conflict in Enterprise Manager, since we would have a folder "a" as the root of the path "a/b/c", as well as a channel "a" at the same position in the display. Similarly, trying to assign the name "a/b" to a new channel would conflict in the display with the folder named "a/b". You can however name a different channel "a/c", since "a" is used here again as a virtual folder. Similarly, you can name another channel "a/b/d", since both "a" and "a/b" are used here as virtual folders.

Editing Channels

This section describes the process of editing the attributes of a Universal Messaging channel.

Editing channels using the Enterprise Manager enables you to change specific attributes of a channel, such as name, event time to live (TTL), capacity, channel keys or even the realm on which the channel exists.

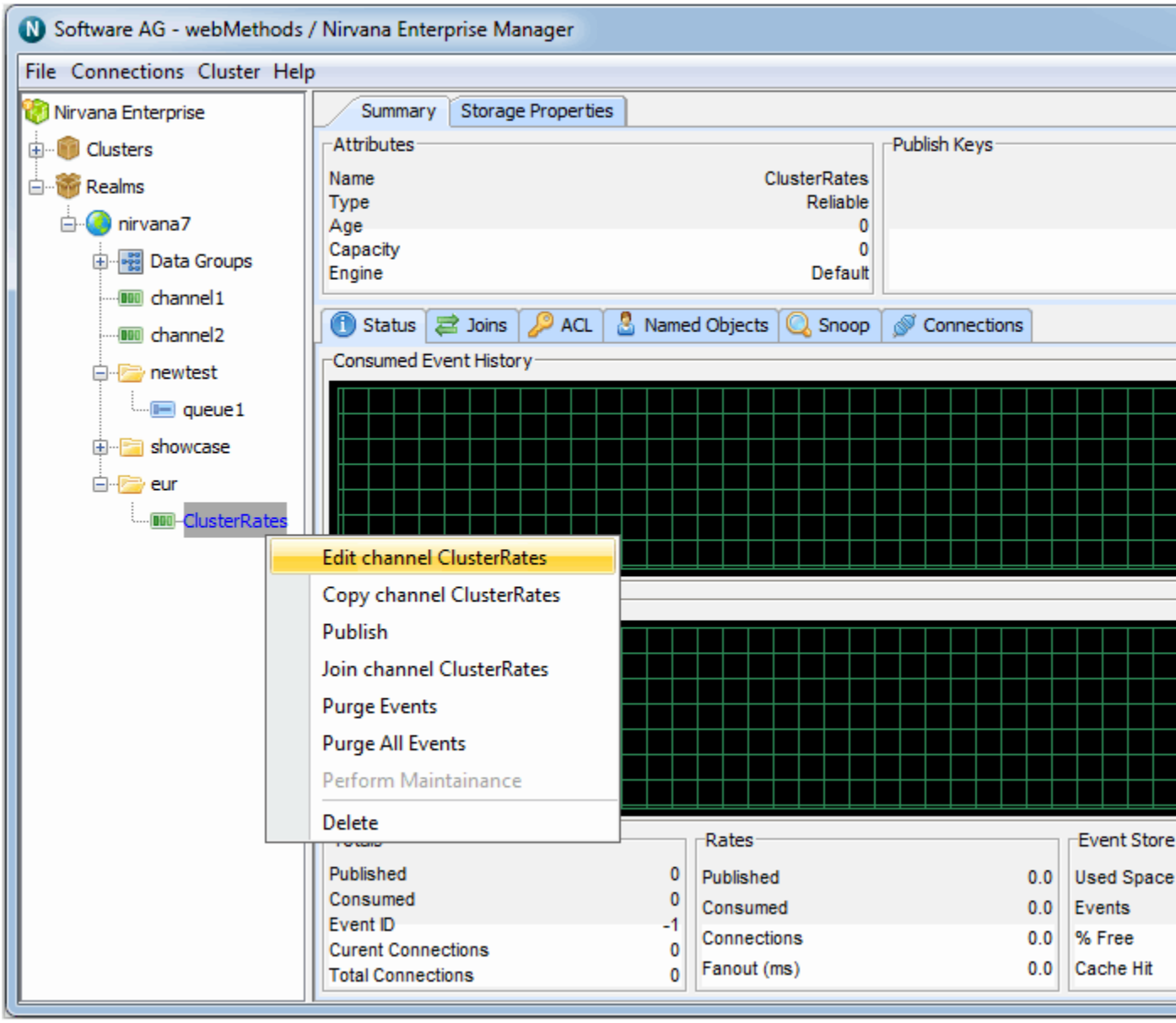
Note:

When a channel is edited, its attributes and any events found on the channel will be copied into a temporary channel, the old channel is then deleted and then the new channel is created. The original events are then copied from the temporary channel onto the new channel.

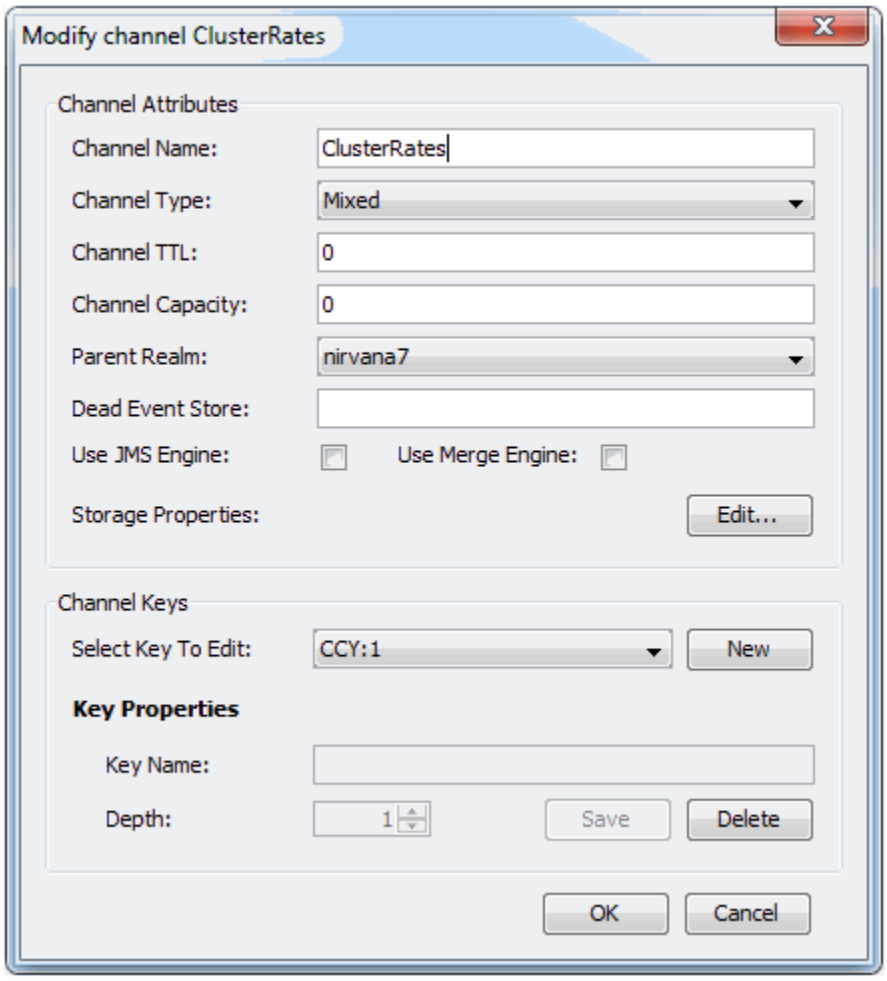
Since editing a channel involves deleting the old channel, certain activities and objects associated with the old channel will also be terminated. See the section "Deleting Channels and Queues" on page 162 for related information.

As far as possible, channel events are held in memory for performance reasons. The temporary channel is also held in memory, and requires the same amount of memory as the channel being edited. The realm server must be able to allocate sufficient memory to store the temporary copy, otherwise the channel edit operation will be aborted and an error will be logged. If such a situation occurs, you can resolve it by allocating additional heap size, so that the temporary copy can exist in memory at the same time as the channel being edited.

In order to edit a channel, select it in the namespace, and then after right-clicking on the node, a menu will be displayed with the various options for a channel node. The image below shows this menu.

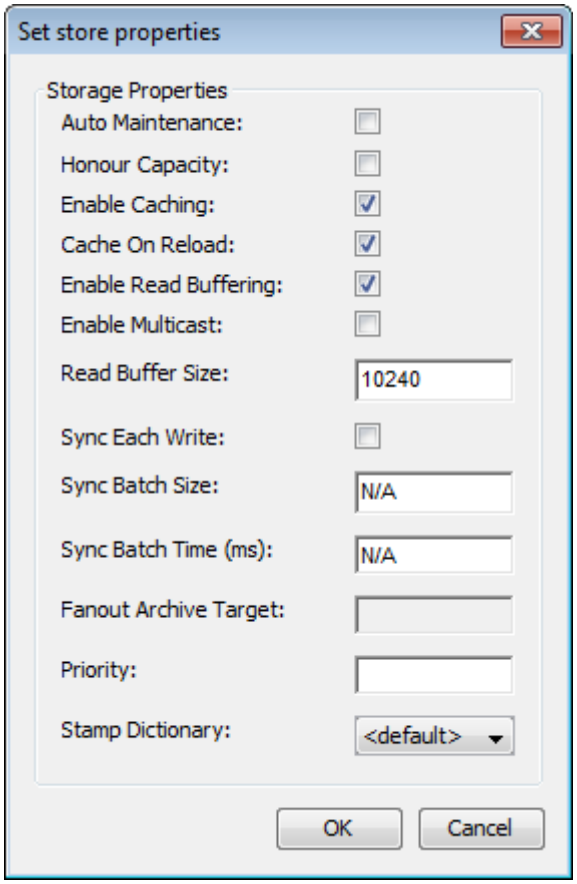


By selecting the 'Edit Channel' option, you will be presented with a dialog that allows you to modify the details of the channel. These details not only include the channel attributes, but also the realm to which the channel belongs. The image below shows the edit channel dialog.



The Parent Realm field shows a drop down list containing all the names of the realms that the enterprise manager is currently connected to. By selecting a realm name from the list, it is possible to move the selected channel to any of the available realms. Clicking on the 'OK' button will perform the edit operation on the channel.

There are also a number of Storage Properties associated with the channel which can be configured by clicking the "Edit..." button to the right of "Storage Properties".



Updating Protobuf Descriptors

The protocol buffer (Protobuf) definition files associated with a store (a channel or a queue) can be updated without requiring the store to be deleted and re-created. After you update the Protobuf descriptor, all filtering will be done with the new Protobuf definitions.

To update protocol buffer definitions in the Enterprise Manager

1. Select the store whose descriptors you want to update and right-click it.

2. From the drop-down menu, select **Update Protocol Buffers**.

3. Select the file or files that contain the descriptors you want to set on the store (multi-select is enabled for loading multiple file descriptor sets). Then click **Open**.

The new Protobuf definitions are applied to the store.

For information about updating protocol buffer definitions programmatically, for example in Java, see the section *Google Protocol Buffers* in the *Developer Guide*.

Exporting Protobuf Descriptors

You can export the protocol buffer (Protobuf) definitions associated with a store (a channel or a queue) to a folder.

To export protocol buffer definitions in the Enterprise Manager

1. Select the store whose Protobuf descriptors you want to export and right-click it.
2. From the drop-down menu, select **Export Protobuf Definitions**.
3. Select the folder where you want the descriptors from the store to be exported. Then click **Open**.

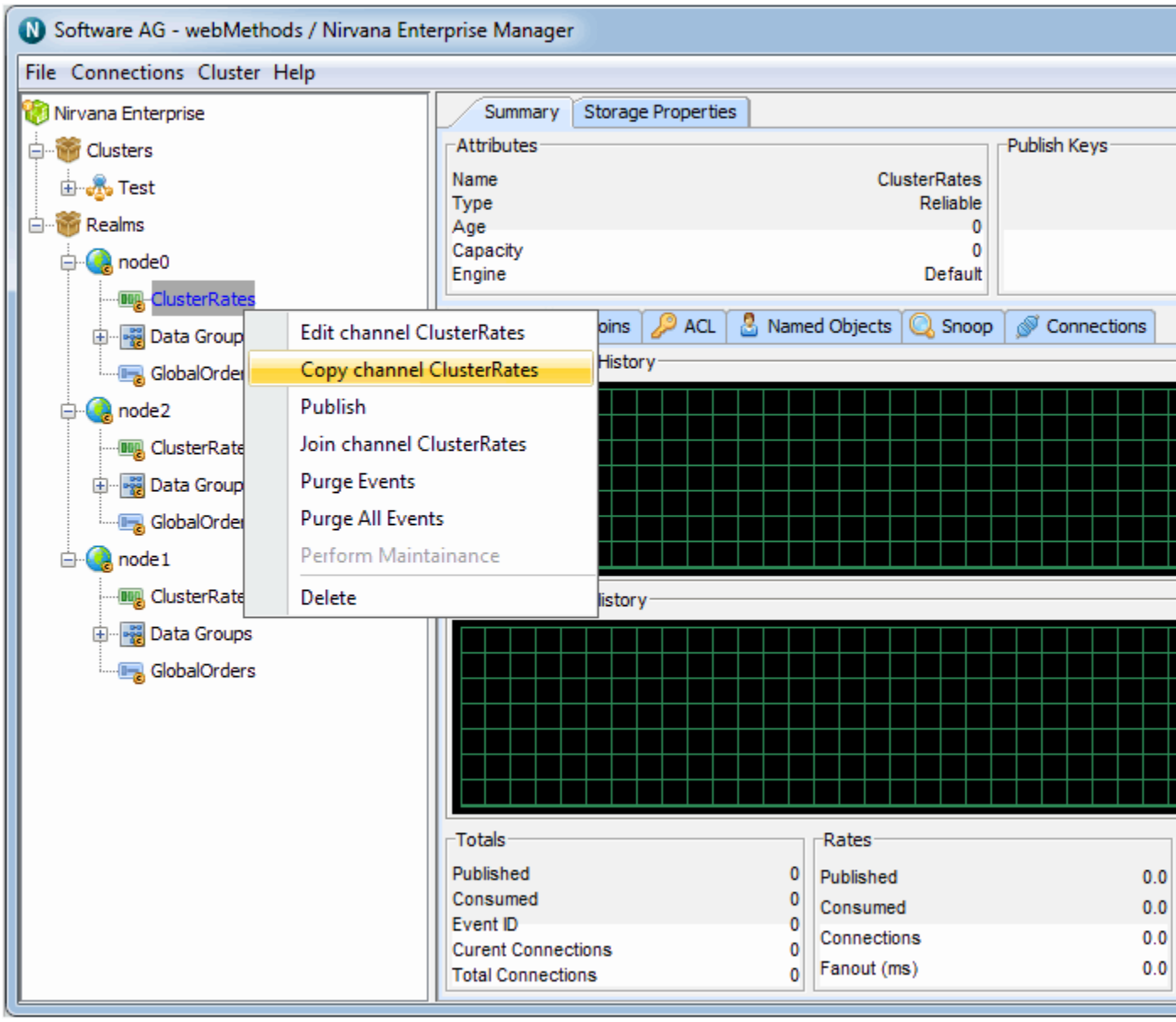
The Protobuf definitions are exported from the store.

Copying Channels

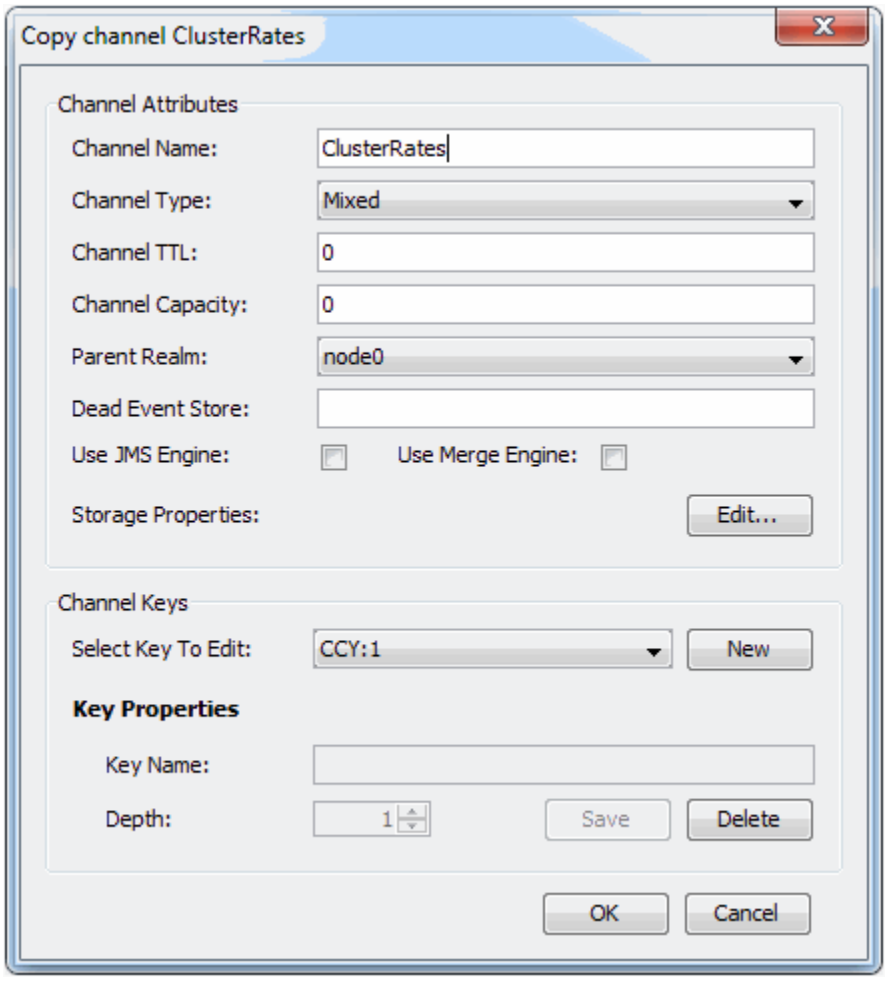
This section describes the process of copying channels in Universal Messaging realms.

Copying channels using the Enterprise Manager enables you to duplicate channels automatically across realms. When a channel is copied, its attributes and any events found on the channel will be copied over onto the new channel copy.

Firstly, by selecting the channel in the namespace that you wish to copy and right-clicking on the node, you will be presented with a menu that shows you the various options for a channel node. The image below shows this menu.



By selecting the 'Copy Channel' option, you will be presented with a dialog that allows you to input the details of the new channel copy. These details not only include the channel attributes, but also the realm to which the channel will be copied. The image below shows the copy channel dialog.



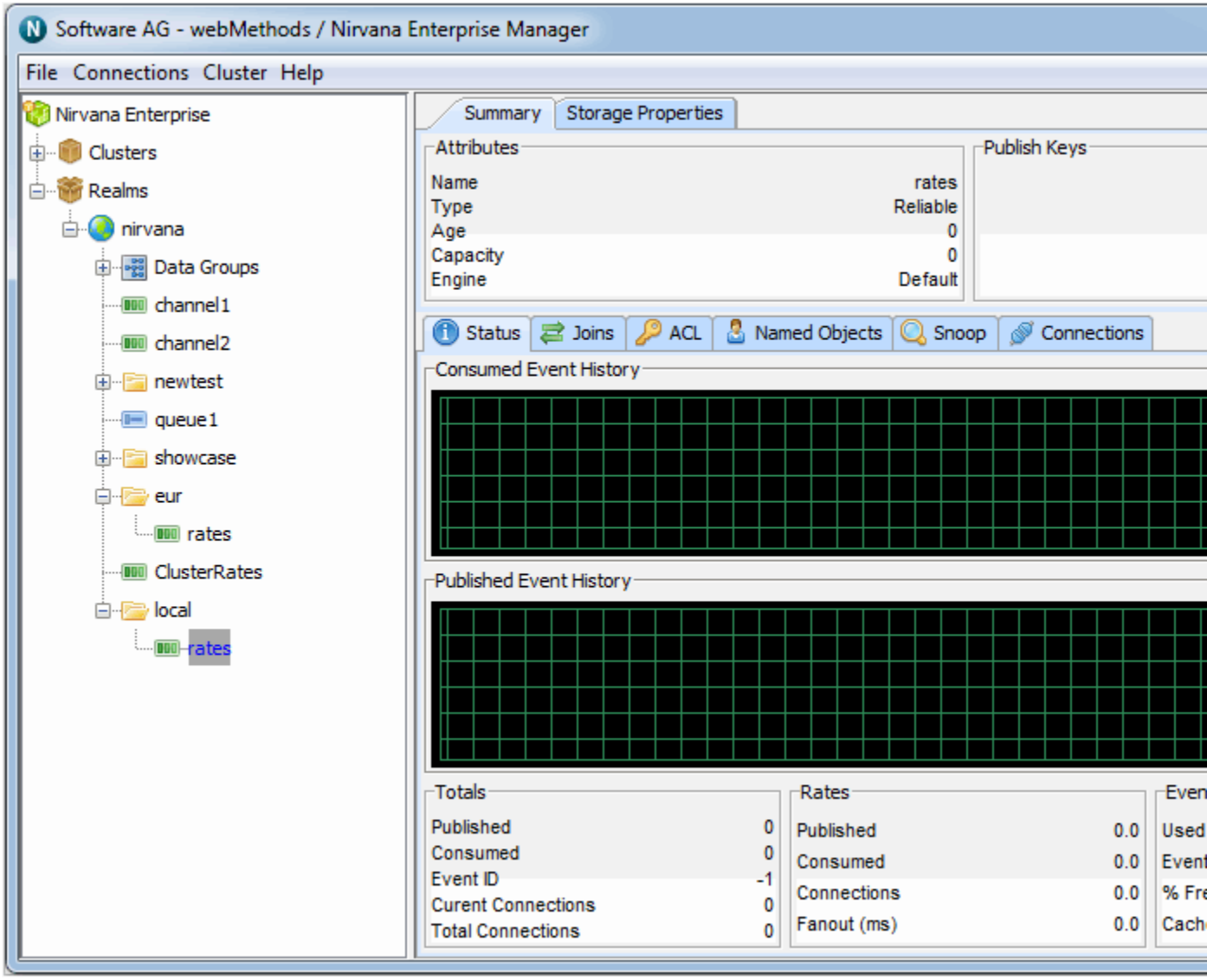
The Parent Realm field shows a drop down list containing all the names of the realms that the Enterprise Manager is currently connected to. By selecting a realm name from the list, it is possible to create a copy of the selected channel in that realm. Clicking on the 'OK' button will create the channel on the selected realm and the channel will then appear in the namespace tree.

Creating Channel Joins

This section describes the process of joining channels on Universal Messaging realms.

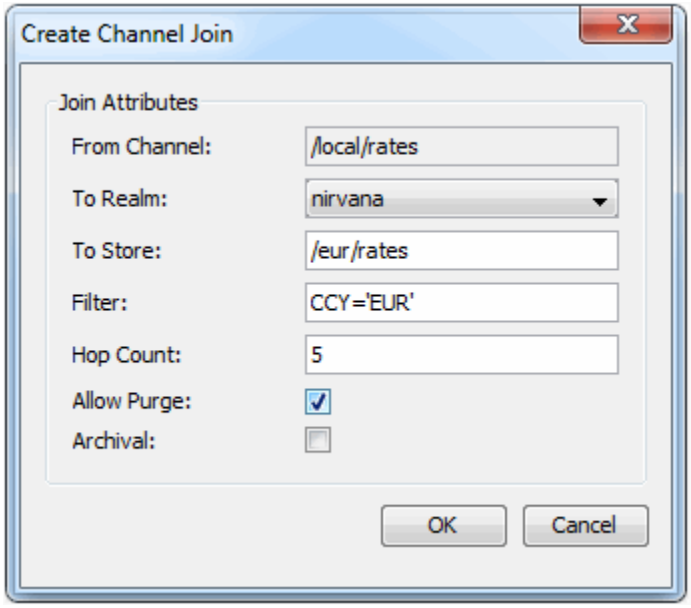
Channels can be joined programmatically or by using the Universal Messaging Enterprise Manager as described below.

The image below shows a realm that contains the cluster channels `/eur/rates` and `/local/rates`.



By selecting any of the cluster channels, then right-clicking, you can select the 'Join Channel' menu option. You will be presented with the join dialog for the selected cluster channel. This allows you to create a join from the channel to any other channel in any realm.

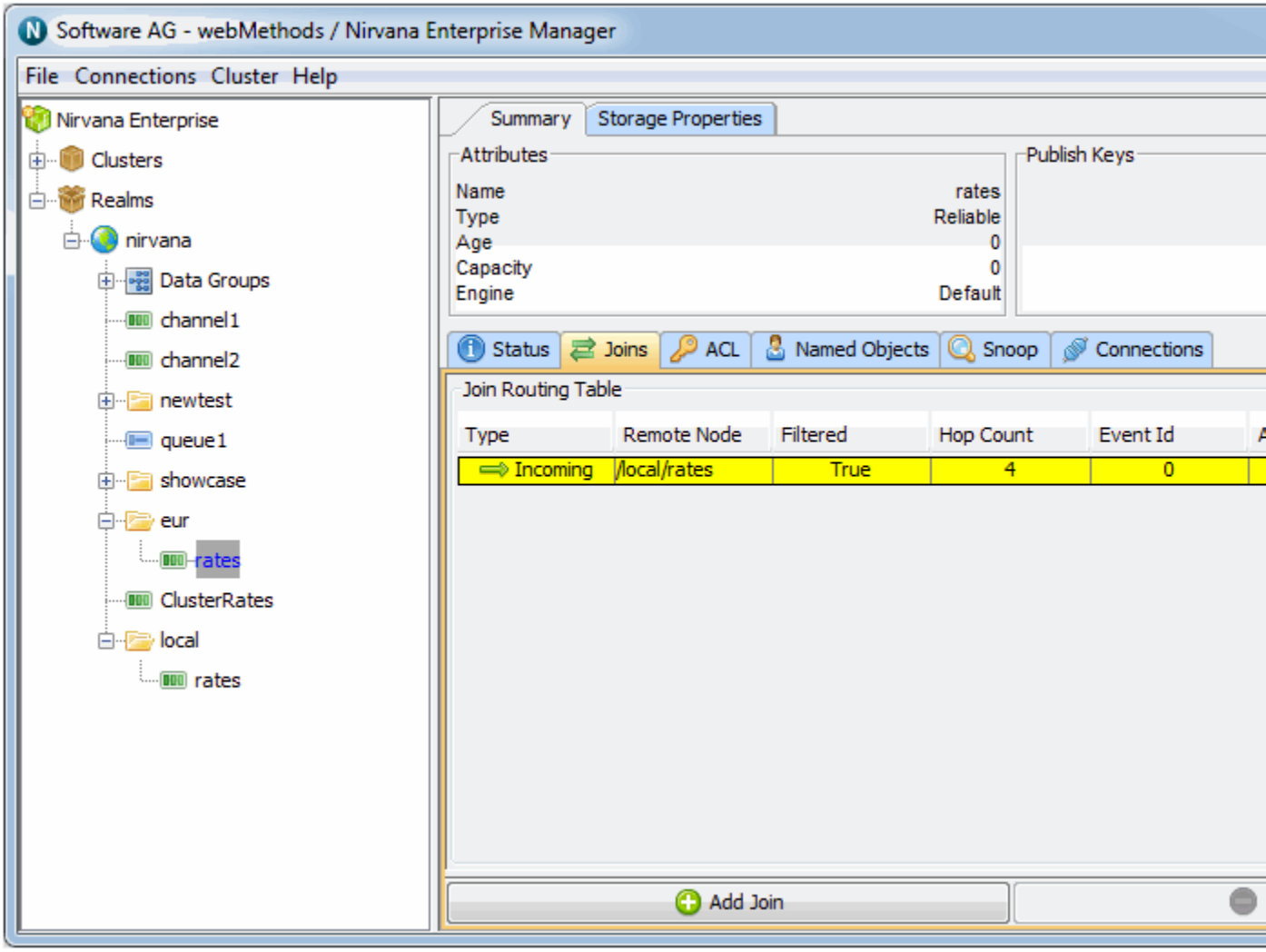
We can, for example, select `/local/rates` as the channel we wish to join to `/eur/rates`, as shown in the join dialog below, with a filter of CCY='EUR'. This will ensure that only those events with the event property CCY equal to 'EUR' occurring on `/local/rates` will be published to the `/eur/rates` channel.



The **From Channel** field represents the source channel and the **To Store** field represents the destination channel. The term *store* is a generic term for either a channel or a queue. By default, the **From Channel** field contains the name of the currently selected channel from which the join dialog was opened.

Clicking on the **OK** button will create the join.

By selecting the **Joins** tab panel for either the `/local/rates` channel or the `/eur/rates` channel, you will be presented with a panel that shows the join just created. The image below shows the display for the `/eur/rates` channel, which in this example is the destination channel. Selecting the newly created join will also show you any relevant filtering criteria that the join has been created with.



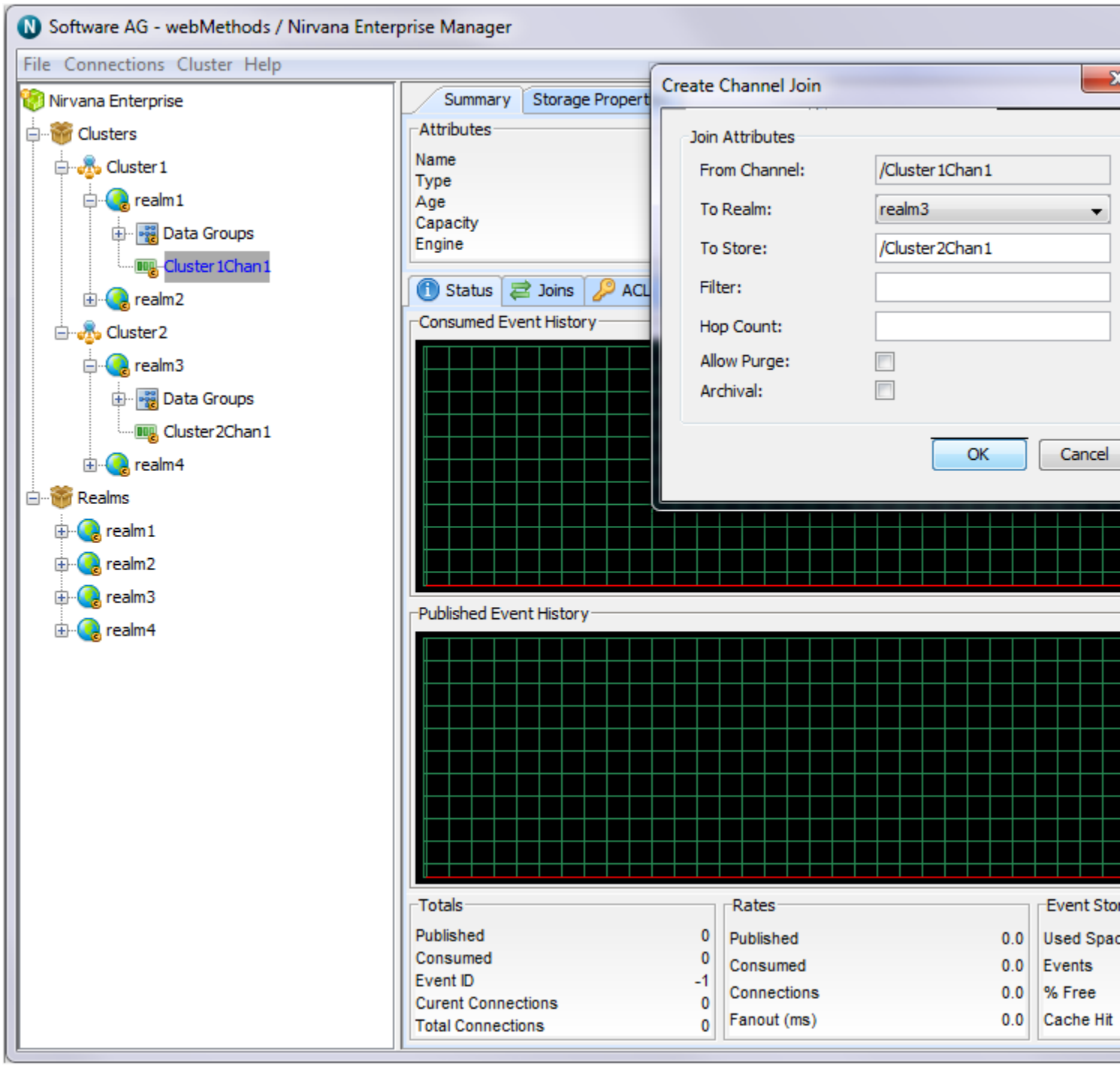
The table that shows the joins for a channel will indicate in the **Type** column whether the join is 'Outgoing' or 'Incoming'. An outgoing join indicates that the selected channel is the source for the join, whereas 'Incoming' indicates that the selected channel is the destination channel.

Deleting Joins

You can delete a join by selecting the join from the joins table, and clicking the **Delete Join** button. Joins can only be deleted in the Enterprise Manager from the source (outgoing) channel. If the destination channel (incoming) is selected, the **Delete Join** button will be disabled.

Inter-Cluster Joins

To add a join between channels on different clusters, first create an inter-cluster connection between the two clusters. Next, simply select a realm in the desired destination cluster as the join destination in the dropdown menu, as below:



As with non-inter-cluster joins, these joins can be specified with filters and hop-counts. If realms in the destination cluster go down, the join will failover and will continue to deliver events so long as the destination cluster is formed.

Inter-cluster joins can also be formed programmatically .

Allowing Purge

When a channel is purged, the events in the channel can also be purged. To activate this feature, mark the checkbox **Allow Purge**.

Archival Joins

An archival join is a specific type of join made between a channel and a queue, where events will not be checked for duplication. Events that are published to the source channel will appear in the destination queue. This may result in duplicate events in the queue if the queue has multiple sources.

The channel join dialog contains a checkbox labelled **Archival**. This checkbox is available when you create a join from a channel to a queue. When you mark the checkbox, the join is defined as an archival join.

If you use the optional **Filter** field, only the events matching the filter criteria will appear in the destination queue.

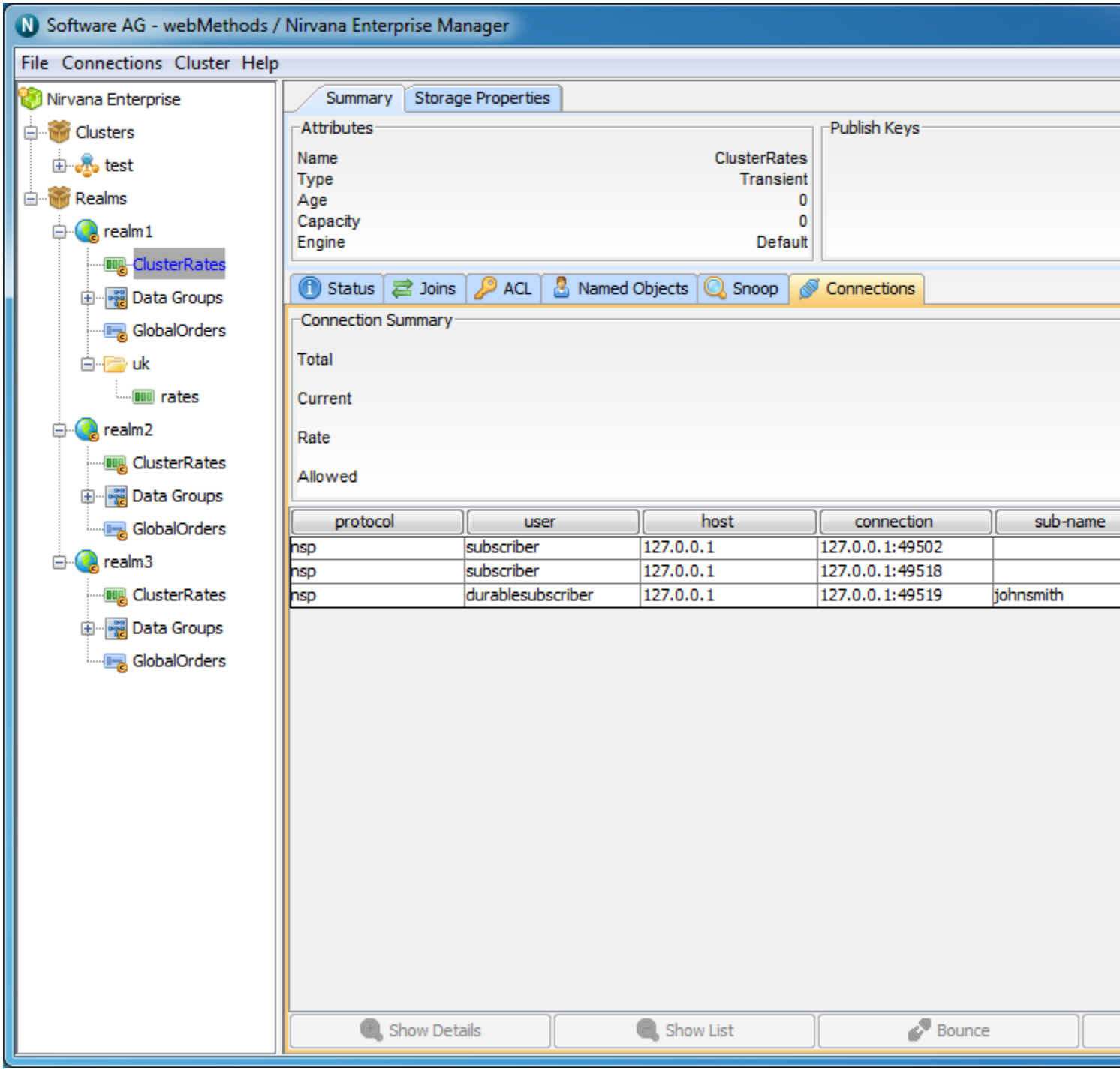
Channel Connections

When a Universal Messaging client connects to a Realm Server, the server maintains information on the connection (see "Connection Information" on page 607) that is available through the Universal Messaging Administration API. The API also provides mechanisms for receiving notification when connections are added and deleted (see the code example "Connection Watch" using the Administration API).

Connection information is also maintained when Universal Messaging clients subscribe to channels. This section guides you through channel connection information.

The Universal Messaging Enterprise Manager allows you to view the connections (channel subscriptions) on a realm and drilldown to view more detailed information about each connection, such as the last event sent or received, and the rate of events sent and received from each connection.

To view connections for a channel, select a channel node from the namespace, and select the 'Connections' tab. This will display a panel containing a table of connections, as shown in the image below.



Connections have the following attributes:

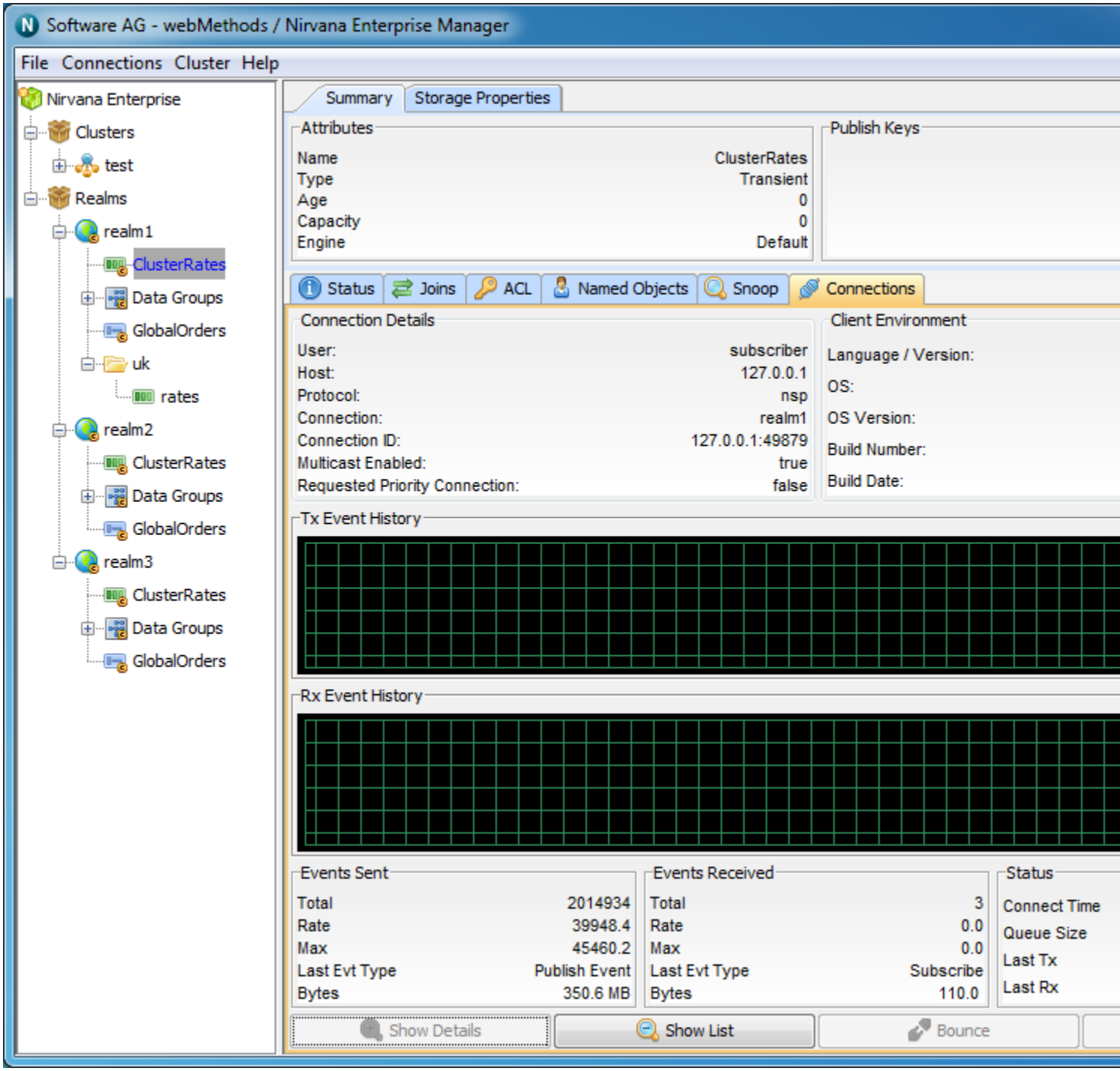
- **Protocol** - The protocol used in the connection
- **User** - The name of the user connected
- **Host** - The host machine that the user is connecting from
- **Connection** - The local connection id, defined as hostname:local_port

- **Sub-Name**- The durable reference if one is provided. For more information about channel durables, see " Viewing and Managing Durables for a Channel" on page 161.
- **Filter** - The filter string for the subscription if one has been provided

The highlighted connection above shows that the user has subscribed to the 'ClusterRates' channel using the nsp protocol, to localhost. The user has also provided a durable called 'johnsmith' and a filter of "region='UK'" which will ensure the user only consumes events with the value 'UK' in the 'region' property of the event properties.

When a connection is highlighted, there a number of things that can be shown for the connection.

By double-clicking on a connection from the table, or by clicking on the 'Show Details' button, you are presented with a panel that contains a more detailed look at the activity for the selected connection. The connection details panel is shown in the image below.



Connection Details

You will see that there are 2 separate information panels above the graphs once you have drilled down into a connection. The first of which is labelled Connection Details. This information contains information about the user connection, such as user name, host protocol.

Client Environment

Next to this you will see a panel that shows details regarding the client environment for this user. These includes API language / Platform, Host OS and Universal Messaging build number

The two graphs, labeled 'Tx Event History' and 'Rx Event History' show the total (yellow) and rates (red) for events received from the server (TX) and sent to the server (RX) for the selected connection.

The bottom of the connection details panel shows 3 sections of information for the selected connection, 'Events Sent', 'Events Received' and 'Status'. Each of these are described below.

Events Sent

The Events Sent section shows the following values:

- **Total** - The total number of events sent by the realm server to this connection
- **Rate** - The rate at which events are being sent by the realm server to this connection
- **Max** - The maximum rate at which events have been sent by the realm server to this connection
- **Last Event Type** - The type of the last event sent from the realm server
- **Bytes** - Total bytes sent by the realm server to this connection

Events Received

The Events Received section shows the following values:

- **Total** - The total number of events sent by this connection to the realm server
- **Rate** - The rate at which events are being sent by connection to the realm server
- **Max** - The maximum rate at which events have been sent by this connection to the realm server
- **Last Event Type** - The type of the last event sent from the connection to the realm server
- **Bytes** - Total bytes sent by this connection to the realm server

Status

The Events Sent section shows the following values:

- **Connect Time** - The amount of time this connection has been connected to the realm server
- **Queue Size** - The number of events in the outbound queue of this connection (i.e. events waiting to be sent to the realm server)
- **Last Tx** - The time since the last event was received by this connection from the realm server
- **Last Rx** - The time since the last event was sent to the server from this connection

Clicking on the 'Show List' button will take you back to the connections table.

Channel Snoop

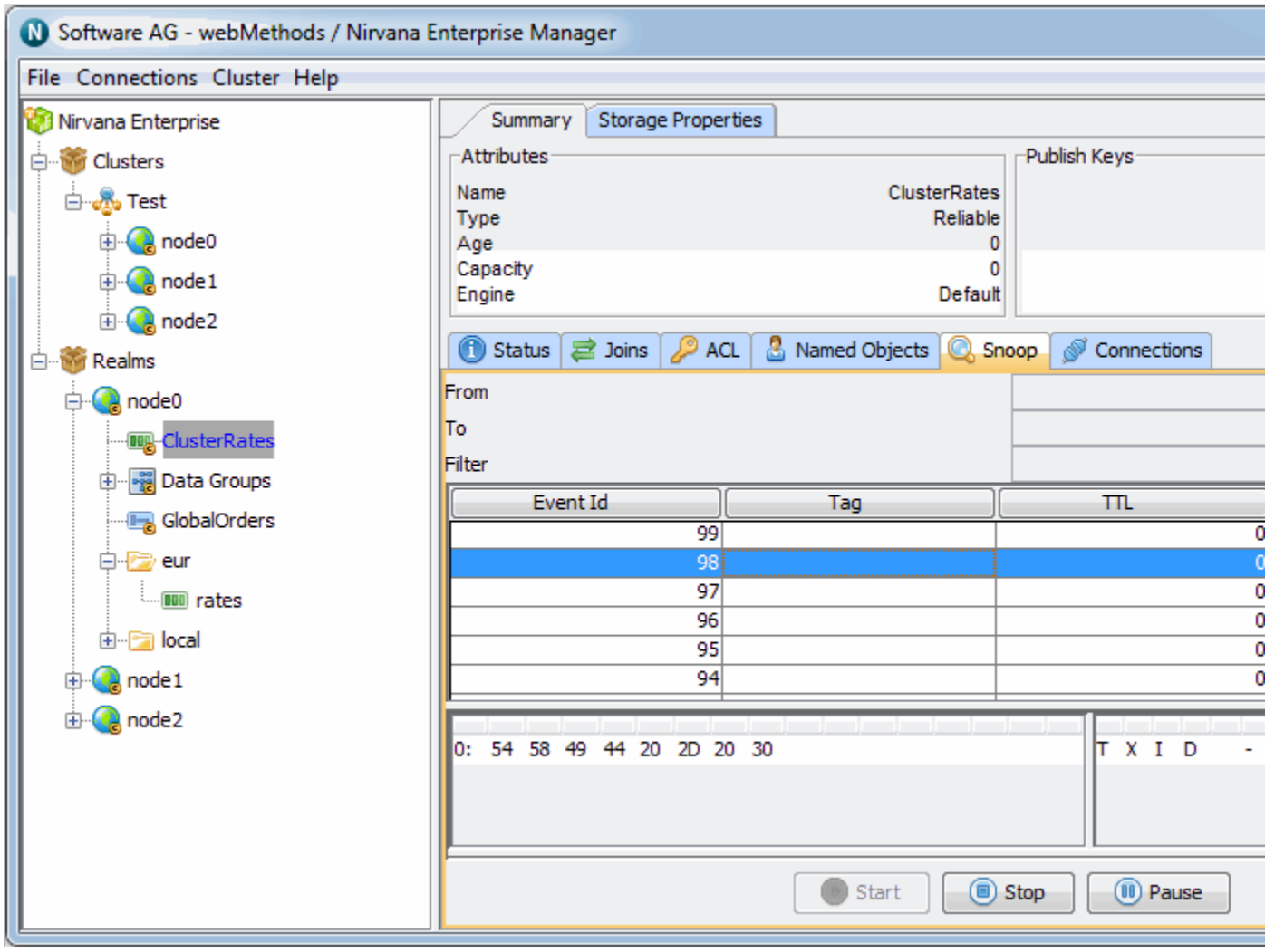
This section describes how to snoop a Universal Messaging channel. Channels are the logical rendezvous point for data that is published and subscribed.

Snooping a channel using the Enterprise Manager allows the display of the contents of events contained within that channel. Each channel node in the namespace tree of a Universal Messaging realm, when selected, displays a snoop panel that provides you with a means of subscribing to the channel so that the events' contents can be displayed in a graphical panel.

You can select where on the channel you wish to subscribe from and to, based on the event id, and you can also provide a filter that enables you to select specific events that match certain criteria.

First of all, by selecting the channel you wish to snoop in the namespace tree, the Enterprise Manager will display a number of panels in a tabbed pane. One of these tabs is labeled 'Snoop'. Selecting the snoop tab will display a panel like the one shown in the image below.

The snoop panel is split up into several sections. Firstly, the text fields **From**, **To** and **Filter** at the top of the panel allow you enter an event id range from and to, and a selector string that will be used to filter events being snooped on the channel. Clicking the **Start** button will begin the channel snoop, and start displaying any events that are published onto the channel using whatever values you have input into the text fields.



When events are published, they are added to the main table below the text field input. This main table shows 4 columns of summary information about each event: the event id, event tag, time to live, and whether the event is persistent. By clicking on any event shown as a row in the main table, more information on the event is shown in the bottom 3 panels.

The Top 2 remaining panels show a Hexadecimal view of the event data and an ASCII representation of the same event data. The panel below that shows the contents of the event properties (if one exists for the event) listed within a table. Each property is displayed as a row in the table. The table columns show the name of the property, the type, and the value.

The button labeled **Pause** will temporarily suspend receipt of any new message being received into the snoop panel for the selected channel. The **Stop** button will stop snooping events and clear all the panels and tables.

In order to snoop the contents of a Universal Messaging queue (see "Queue Snoop" on page 179) please see that section of the enterprise .

Using channel snoop to edit and republish events

Channel snoop can be used not only to view event data but also to edit event data. The edited data can then be published to the channel as a new event, and you can choose whether or not to purge the original event. See the section "Channel Publishing" on page 158 for related information.

Channel Publishing

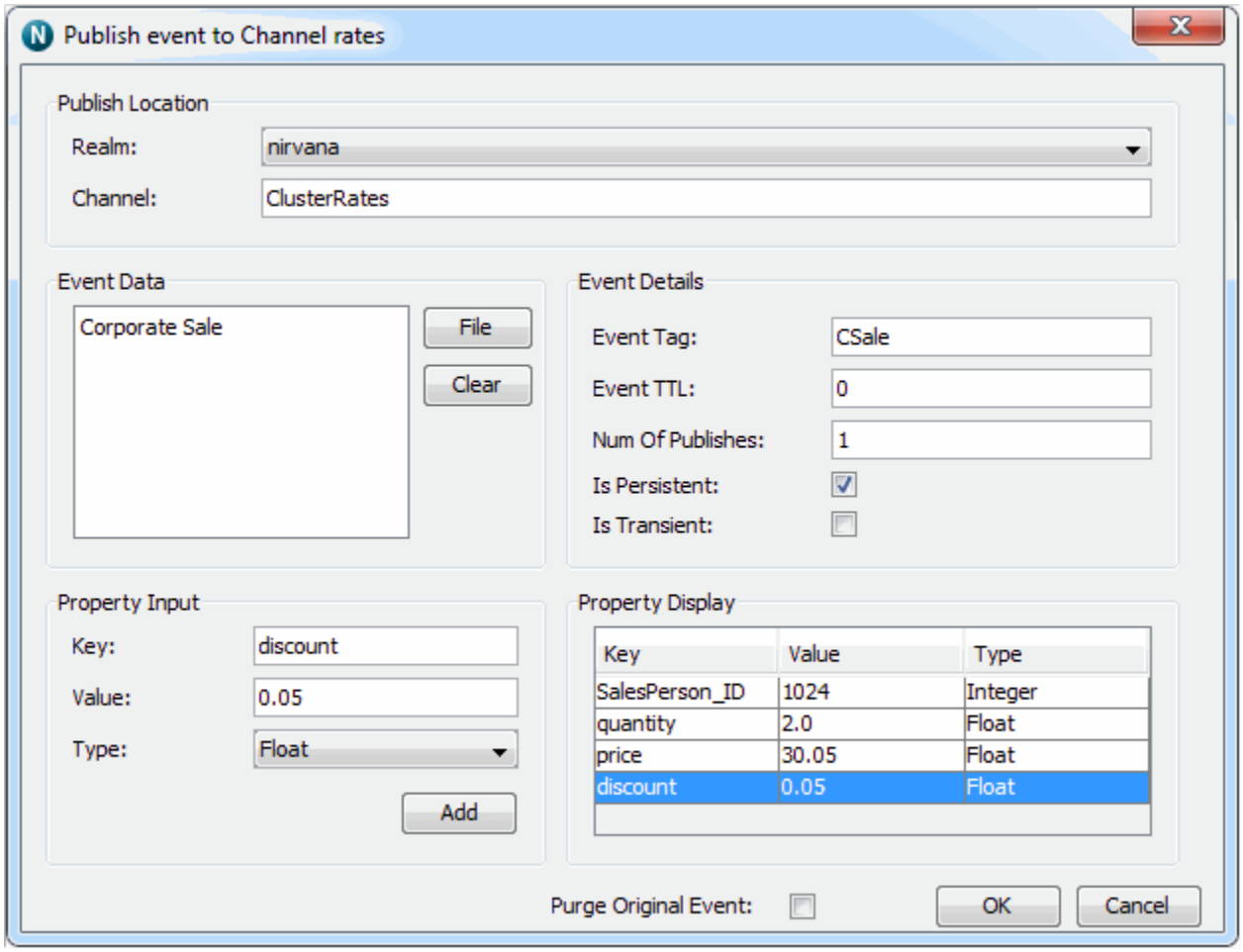
This section describes how to publish events to a Universal Messaging channel from the Enterprise Manager.

Events can be published either from scratch, or by duplicating events already published onto a channel from within the snoop (see "Channel Snoop" on page 156) panel view. Both options allow you to add and remove event properties, set the event TTL, the event persistence, the event tag and the number of times the event will be published to the channel.

The event data can be either manually input, obtained from an XML Document file or any other binary file.

Publishing an event from scratch

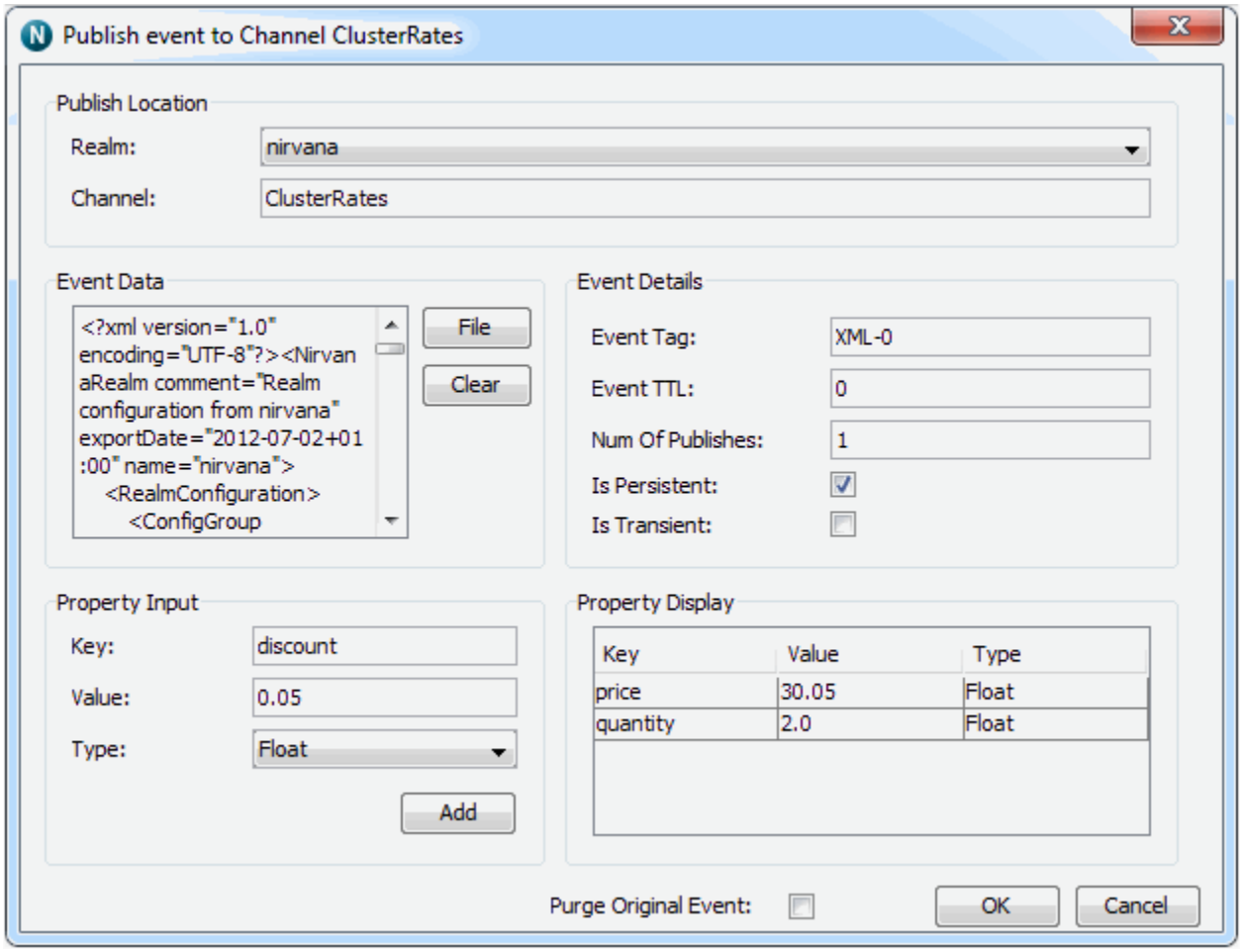
Firstly, to publish a new event from scratch, select the desired channel you wish to publish an event onto, and then right-click on the same node and choose the **Publish** menu option. This will display a dialog as shown below where you can construct the event.



In the image above, the data for the event is simply a string. The properties are added by entering a property key, a value and a type and then clicking on the **Add Property**. Once added, the properties are displayed in the table at the bottom of the panel. To remove a property entry, click on the property within the table view and select the **Remove Property** menu option. Properties can also be edited by clicking on the property from the table and double-clicking in the cell you wish to change. Once you hit return, the value will be updated in the table.

Publishing an event from a file

To add an XML document as the event data, click on the **File** button in the dialog, and choose your XML file from the file chooser. Once opened, the contents of the file will be displayed within the event data section of the dialog and will be non-editable. An example of this is shown in the image below.



To publish the contents of any other file within the event data, repeat the above steps for XML and select a non-XML file. The contents of the file will not be displayed, however the file will be read in binary format when the **OK** button is clicked and published to the channel. Once again, when a file is selected for publishing the event data section is non-editable.

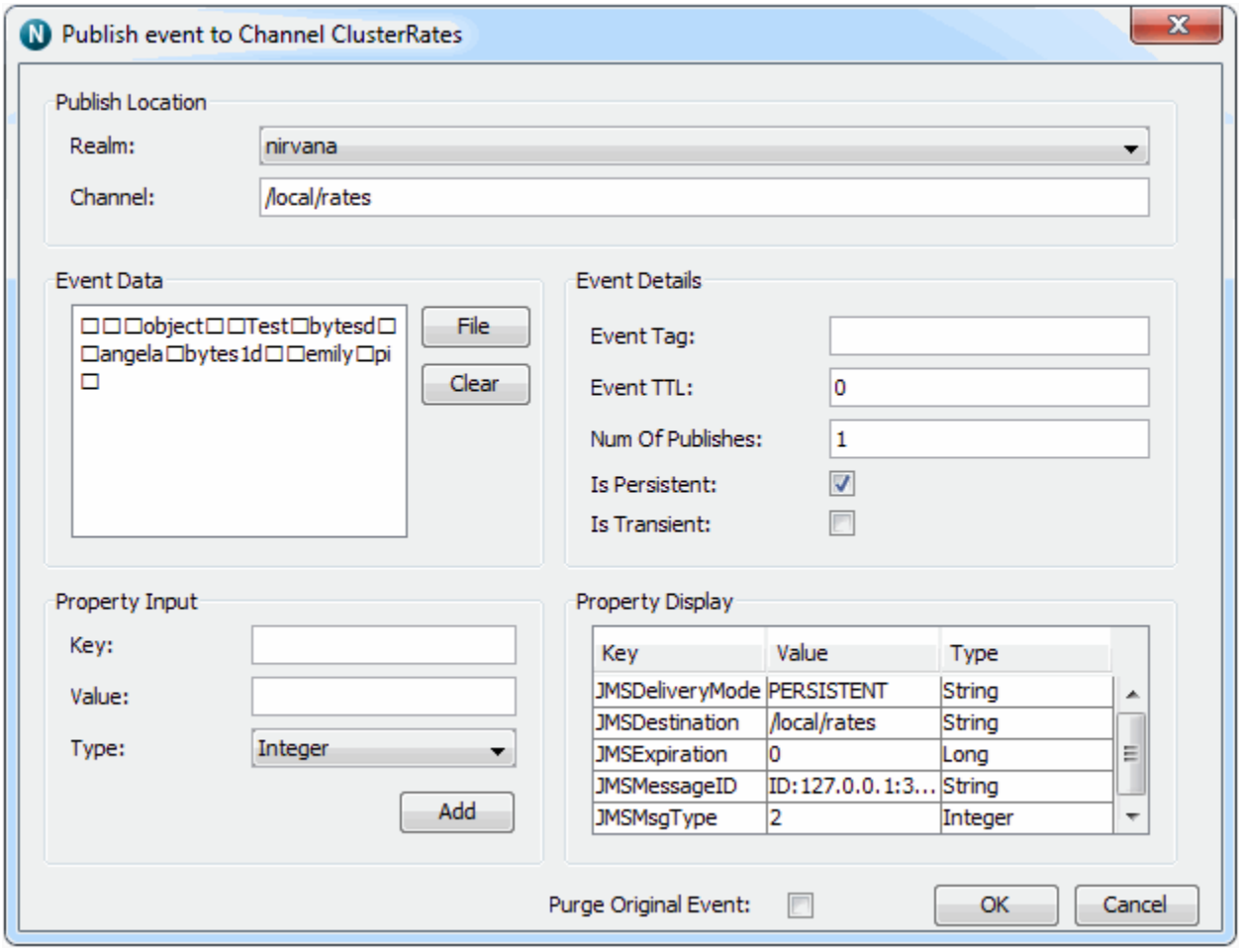
Clicking on the **Clear** button will cancel any file that has previously been selected and allow you to once again select a file or manually enter the event data.

Duplicating, editing and republishing an event

In order to duplicate or edit and republish an event that has already been published to a channel, you must first of all select the snoop panel (see "Channel Snoop" on page 156) for the channel in question, and snoop the channel. Once events are displayed in the snoop panel, select the event you wish to duplicate or edit and republish from the table of events. Right-clicking on the event will display a menu with 2 options. The first option **Purge Event** allows you to purge an individual event from a channel. The second menu option **Edit and Republish Event** opens the event publish dialog with the details of the event already filled in, including the event properties, TTL and persistence.

Properties can be added / removed from the duplicate event, and existing properties can be edited. To edit a property, double-click in the cell of the property you wish to change and then edit its contents. You can also choose to purge the original event from the channel by checking the **Purge**

Original Event checkbox. If you want to create multiple copies of the event, for example for test purposes, enter the required number of copies in the **Num Of Publishes** field. When you click the **OK** button, the event(s) will be published onto the channel. The image below shows the publish dialog for a JMS Message published onto a Universal Messaging channel.



The publish option is also available for queues. The republish option is also available from the snoop panel for queues.

Viewing and Managing Durables for a Channel

Durables (named objects) are channel objects stored by a realm server, which provide state information for durable consumers. Depending on its type, a durable can have one or more durable consumers connected to it. Each time a consumer connects to a durable, the consumer starts consuming events from the last event ID successfully consumed by the previous consumer connected to the durable. The consumed events include all events sent to the channel after the previous consumer disconnected and before the new consumer connected.

- To view durables for a channel in Enterprise Manager, select the channel and click the **Durables** tab.

The durables table lists all durables present on the channel. Each row of the table shows a separate durable. The columns of the table show the attributes of a durable, such as the name and current event ID, the number of outstanding events, whether the durable is cluster-wide and whether it is persistent, and what its type is.

When a durable is added or removed, or the attributes of a durable are changed, Enterprise Manager updates the table automatically.

Note:
When the attributes of a durable are changed, Enterprise Manager updates the durables table with a delay of several seconds.

- To delete a durable from the durables table, select the durable and click **Delete Durable**.

Deleting Channels and Queues

To delete a store (i.e., a channel or queue), proceed as follows:

1. Select the store in the namespace of the Enterprise Manager,
2. Select **Delete** in the context menu of the store.

Note:
Since *editing* a store involves deleting the existing store before creating the new store, all of the points mentioned below for deleting a store apply also for editing a store.

Upon deletion of a store, all assets dependent on it will be deleted and all content in the store will be deleted. All active subscriptions to the store will be terminated, as well as all shared durables attached to the store, along with the associated messages. Such subscriptions or shared durables need to be recreated by the original creator of those objects after you have finished deleting the channel.

Deleting a store which serves as a dead event store for another store will cause that reference to be removed, therefore the user should re-create the reference.

Any joins from or to this store will need to be recreated as they are now disabled.

Before you delete a store, we suggest that you observe the following procedure:

- Drain the store and its durable subscriptions, otherwise any messages in-flight within the store or related to the store will now be lost and transactions will not be deterministic.
- Prevent all publishing activity on the store while it is being deleted; see the section *Pause Publishing* in the *Concepts* guide for related information.

Queue Administration

Creating Queues

This section describes how to create queues on Universal Messaging realms. Each queue that is created consists of a physical object within the Universal Messaging realm as well as its logical reference within the namespace.

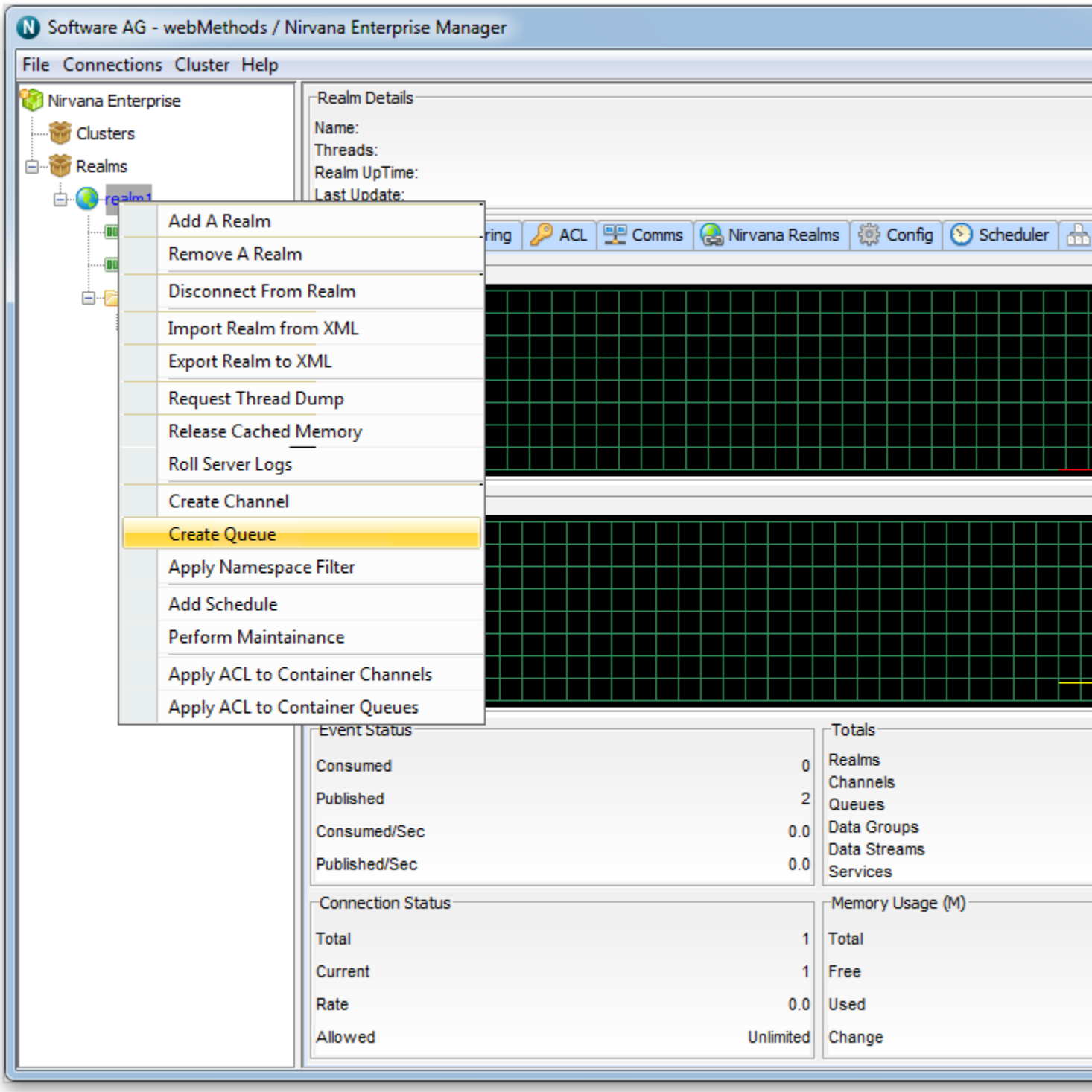
Creating queues using the Enterprise Manager creates the physical object within the realm. Once created, references to queues can be obtained using the Universal Messaging Client and Admin APIs. Queues can also be monitored and managed using the Enterprise Manager.

When you connect to a Universal Messaging realm in the Enterprise Manager, all resources and services found within the realm namespace are displayed in a tree structure under the realm node itself.

Creating Realm Queues

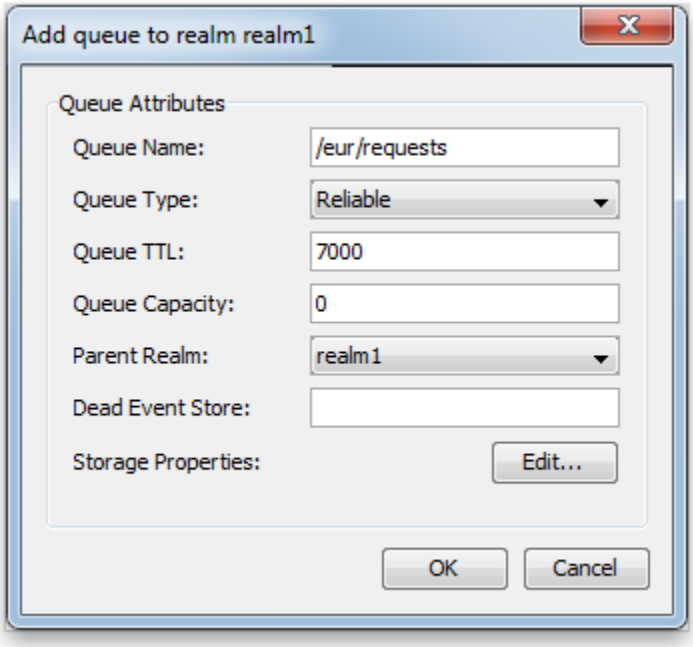
To create new realm queues, the Enterprise Manager provides you with a number of options.

In order to create a queue called '/eur/requests' on a realm called 'nirvana' simply right-click on the realm node called 'nirvana' to display a pop-up menu containing an option called 'Create queue' (as shown in the image below).

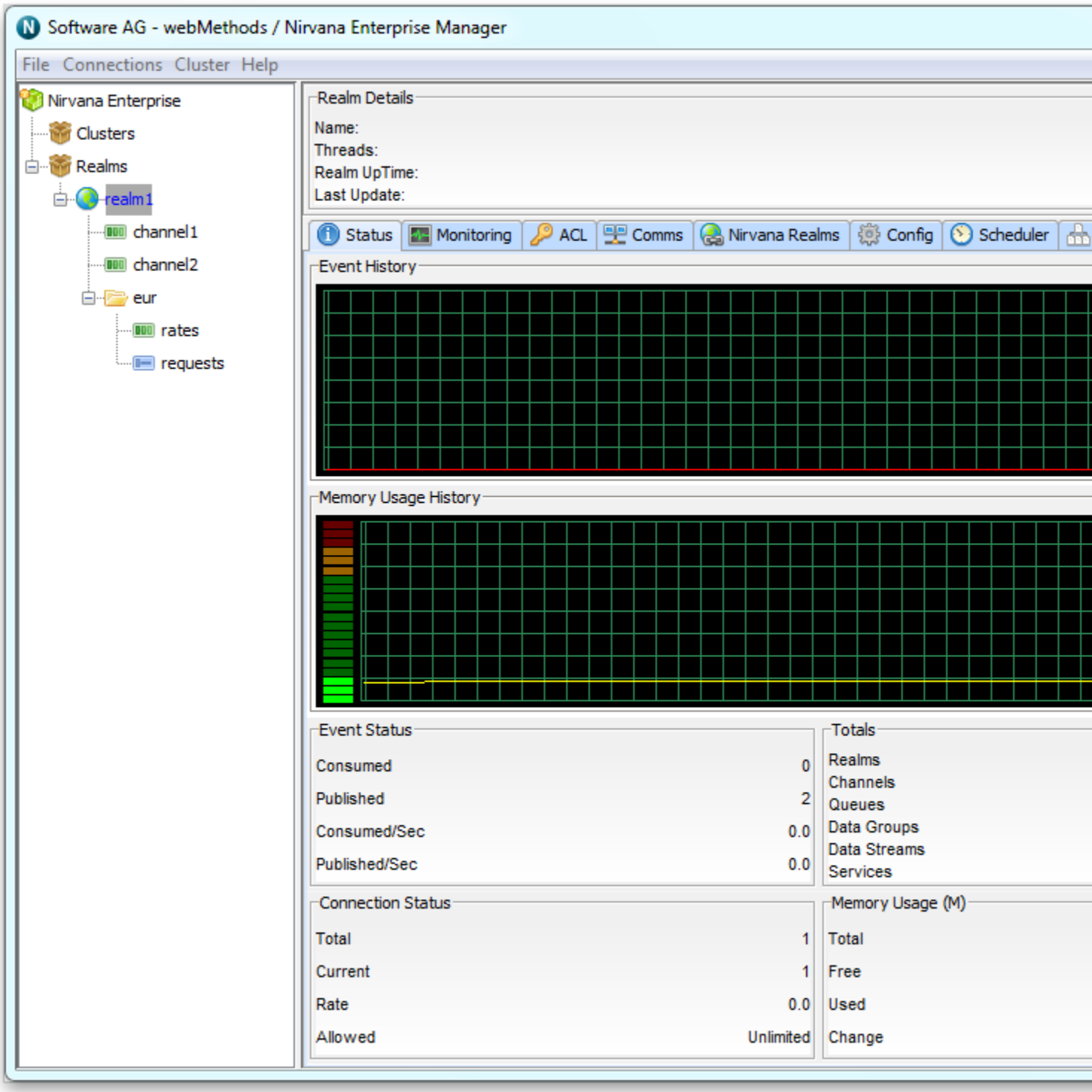


By clicking on the menu item 'Create Queue', you will be prompted with a dialog box that allows you to enter the queue attributes. Queues have a set of attributes assigned to them when they are created. The create queue dialog allows you to input values for each of these attributes.

The set of valid characters that you can use for queue names is the same as the valid character set for channel names. For more information, see "Valid Channel and Queue Names" on page 139.

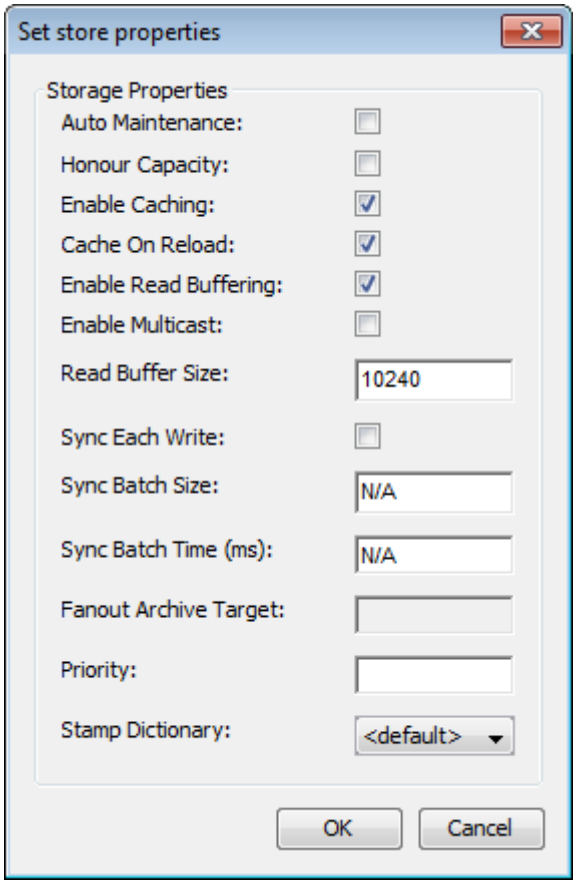


Clicking on the 'OK' button will create the queue '/eur/requests' on the Universal Messaging realm 'nirvana' and render the queue object in the namespace tree of the Enterprise Manager underneath the realm node. This is shown in the image below.



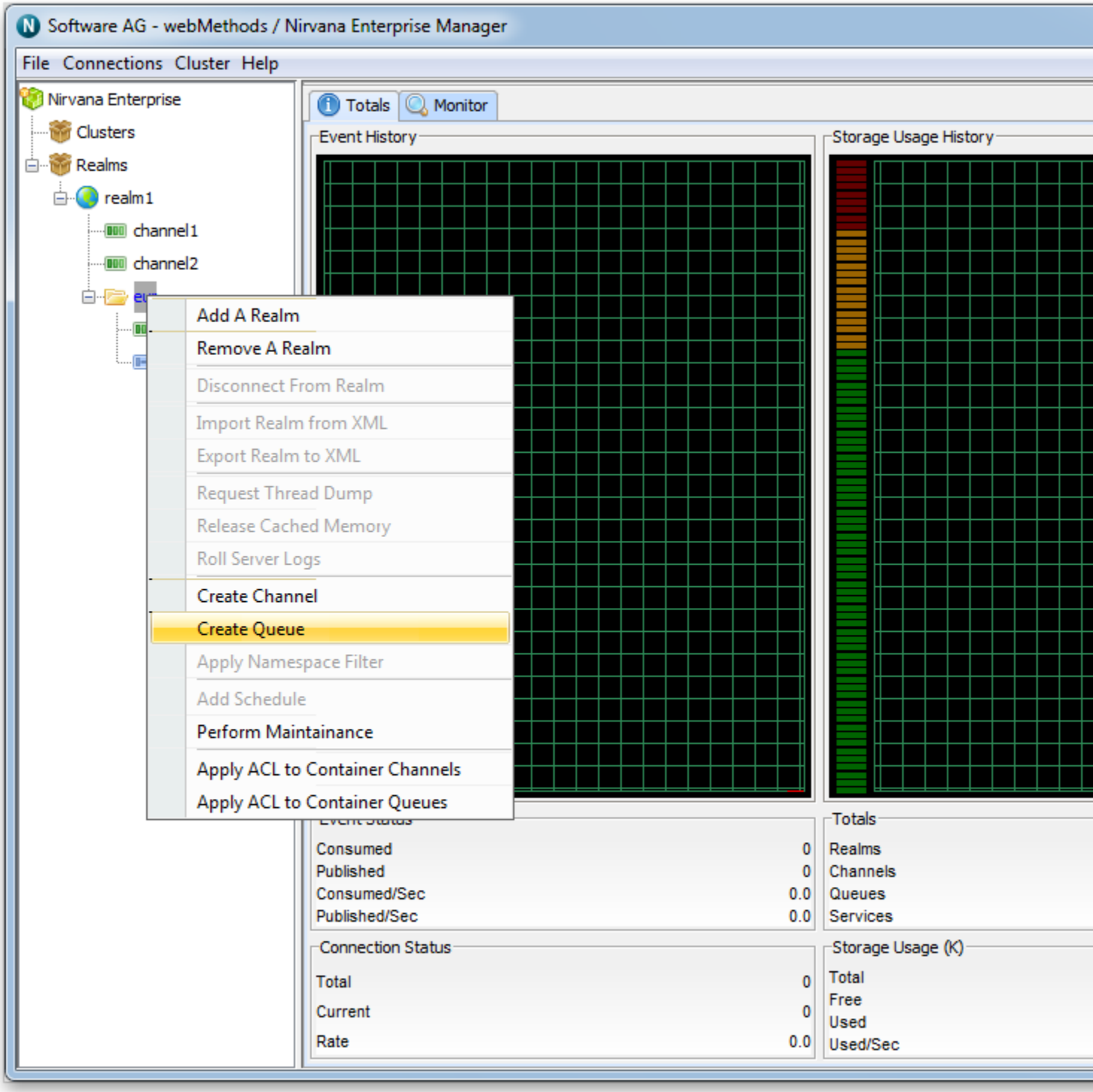
As you can see from the image above, the queue node in the tree has been created under a folder (container node) called '/eur' under the realm 'nirvana'.

There are also a number of Storage Properties associated with the queue which can be configured by clicking the "Edit..." button to the right of "Storage Properties".

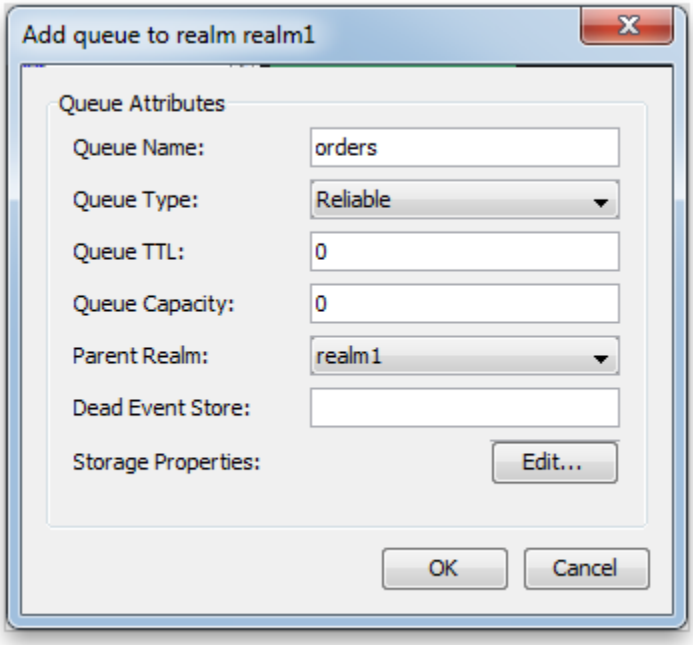


Queues can also be created from the context of a container nodes by specifying a relative queue name.

For example, to create another queue called '/eur/orders', you can select the '/eur' node and create the new queue using its relative name '/orders'. Selecting the container node and right-clicking on the node, shows another pop-up menu of options for container nodes. One of the menu is 'Create Queue'. The image below shows this menu as it appears when the container is right-clicked.



By selecting the menu item, 'Create Queue' from the container node, you are once again presented with the create queue dialog. This dialog looks like the dialog used previously, except the title of the dialog shows that the queue will be created under the container '/eur', as shown in the image below.



Viewing Queues

The Enterprise Manager allows you to view the status of queues, as well as ACL lists associated with queues.

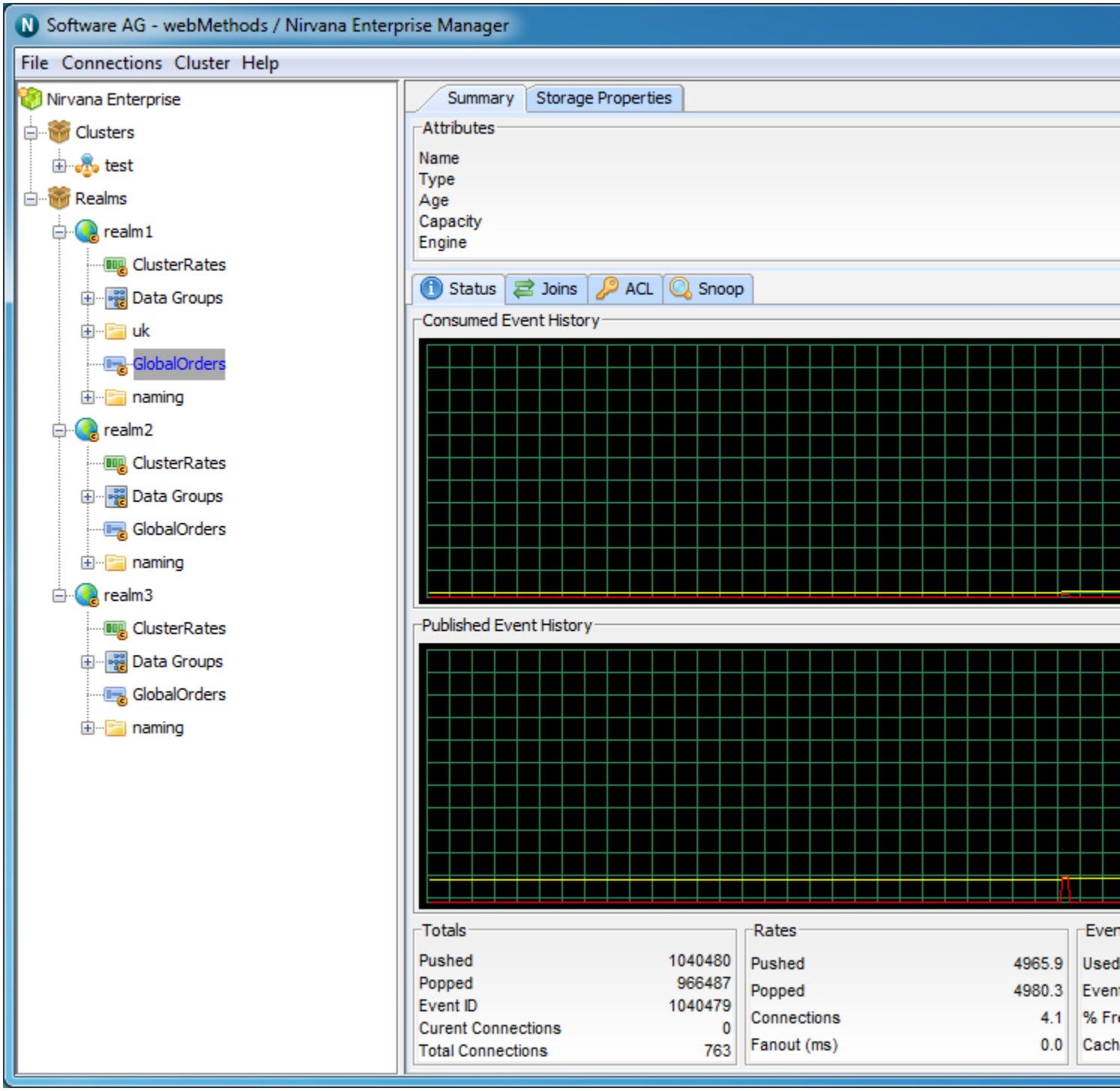
Queue Status

When you select a queue object from the namespace, the first panel to be displayed on the right hand side of the Enterprise Manager panel is the 'Status' panel. Configuration information is always displayed at the top section of the Enterprise Manager when a queue is selected. This configuration information shows queue type, TTL (age) and capacity. The queue 'Status' tab shows real-time management information for the selected queue.

The status panel is split into 2 main sections. The top section shows real time graphs representing the events pushed and popped from the queue, both in terms of rates (i.e. per status interval) as well as the totals.

The bottom section shows the values plotted in the graphs for events pushed and popped, as well as information about the actual queue store at the server.

The image below shows the status panel for an active queue.



The top most graph in the panel shows the event history for events popped from the queue. The red line graphs the rates at which events are being popped while the yellow line graphs the total events popped from the queue.

The bottom graph shows the event history for events pushed onto the queue. The red line graphs the rates at which events are being pushed while the yellow line graphs the total events pushed

to the queue. As the status events are consumed, and the queue nLeafNode () is updated with the new values for events popped and pushed, the status panel and its graphs will be updated.

The bottom section of the status panel shows 3 types of information : Totals, Rates and Event Store. These are discussed below.

Totals

The totals section contains the following values:

- **Published** - The total number of events pushed to the queue when the last status event was consumed
- **Consumed** - The total number of events popped from the queue when the last status event was consumed
- **Event ID** - The event id of the last event pushed to the queue
- **Current Connections** - The current number of asynchronous consumers on the queue
- **Total Connections** - Total number of asynchronous consumers that have subscribed to the queue

Rates

The rates section contains the following values:

- **Published** - The current rate of events pushed to the queue, calculated as (total - previous total) / (interval 1000 milliseconds)
- **Consumed** - The current rate of events popped from the queue, calculated as (total - previous total) / (interval 1000 milliseconds)
- **Connections** - The current rate of asynchronous subscriptions being made to the queue

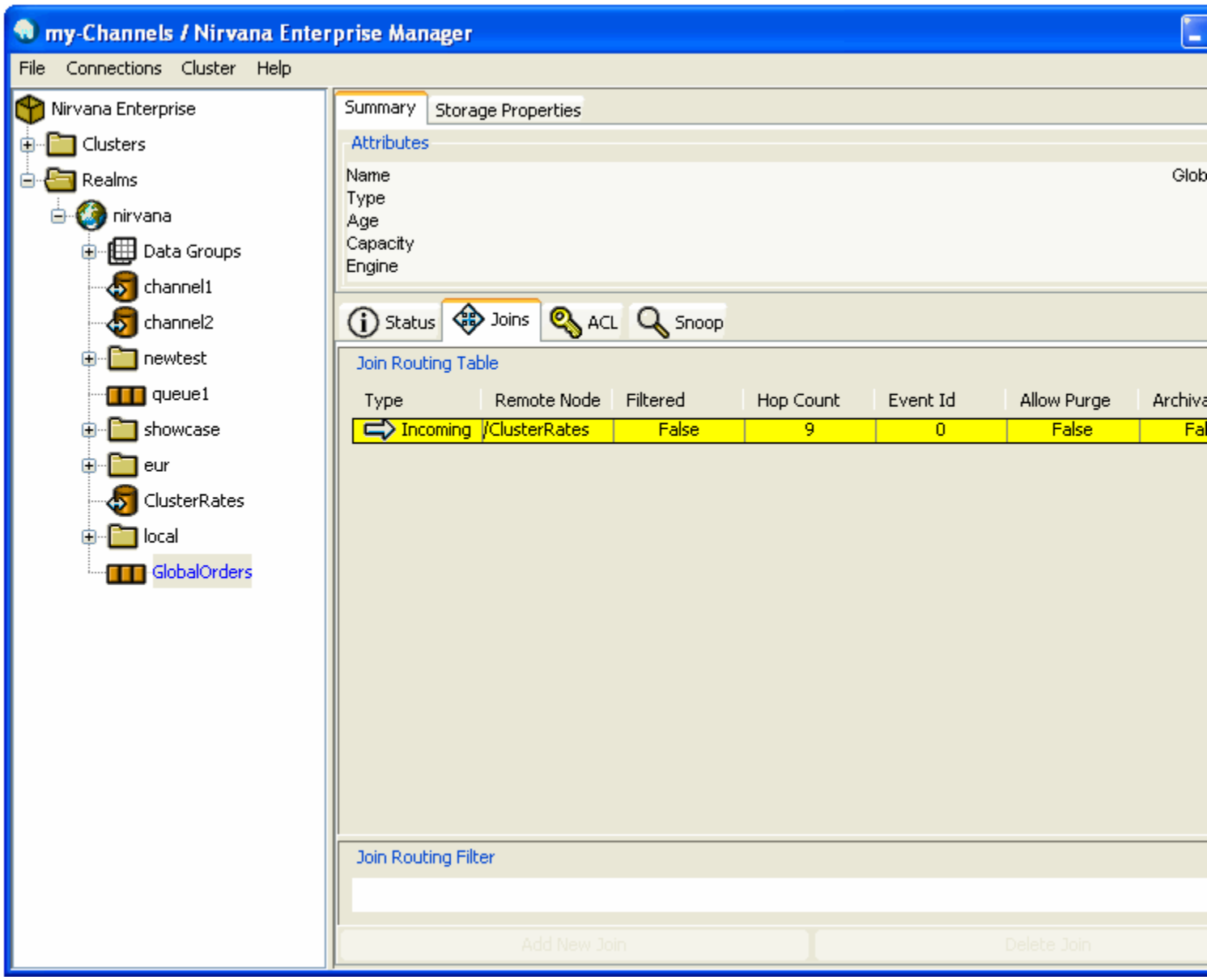
Event Store

The event store section contains the following values:

- **Used Space** - The amount of space in KB used by the queue on the server (either memory, or disk for persistent / mixed queues)
- **Events** - The current number of events on the queue
- **% Free** - The amount of free space in the queue store (calculated as (used space - (total space used by all purged or aged events))
- **Cache Hit** - The %age of events popped from the queue event cache as opposed form the actual physical store if persistent or mixed

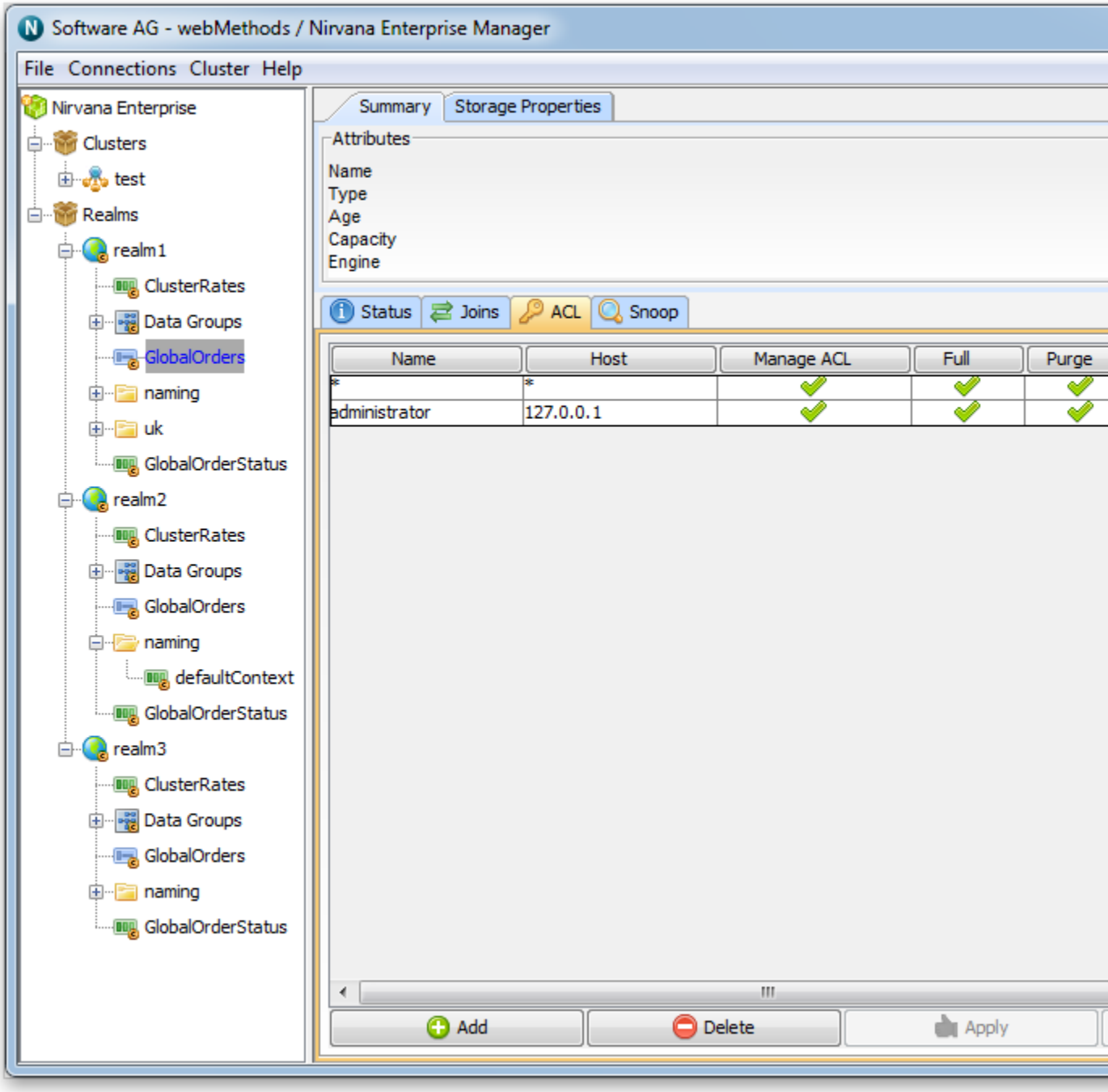
Queues as targets of Channel Joins

Universal Messaging allows channels to be joined to other channels or queues creating server side routing tables with the possibility to apply filters based on message content on the local or a remote Universal Messaging realm.



Queue Access Control List (ACL)

Universal Messaging offers complete control over security policies. Universal Messaging stores security policies locally or be driven by any external entitlements service. Universal Messaging's rich set of entitlements ensure that everything from a network connection through to a channel/queue creation can be controlled on a per user and/or host basis. For more information please see the Universal Messaging ACL's FAQ.



Editing Queues

This section describes the process of editing queues in Universal Messaging realms. Each queue that is created consists of a physical object within the Universal Messaging realm as well as its logical reference within the namespace.

Editing queues using the Enterprise Manager enables you to change specific attributes for a queue, such as name, TTL, capacity or even the realm on which the queue exists.

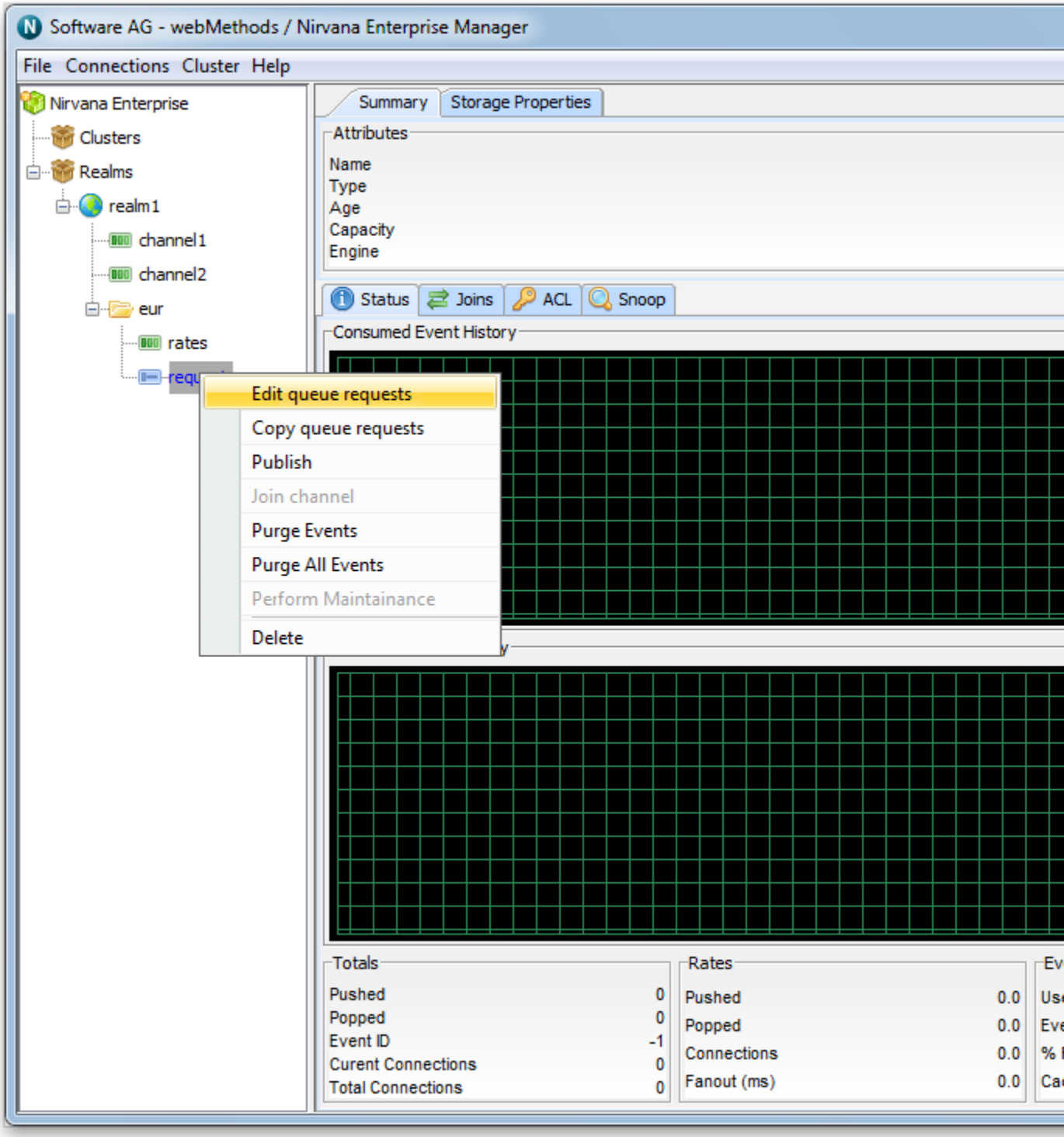
Note:

When a queue is edited, its attributes and any events found on the queue will be copied into a temporary queue, the old queue is then deleted and then the new queue is created and the events are then copied from the temporary queue onto the new queue.

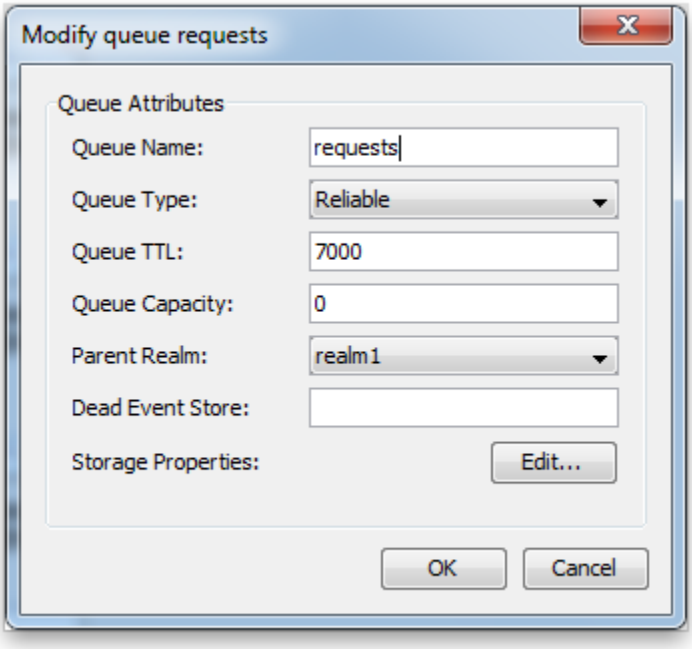
Since editing a queue involves deleting the old queue, certain activities and objects associated with the old queue are also terminated and may need to be recreated. For details, see the section "Deleting Channels and Queues" on page 162.

As far as possible, queue events are held in memory for performance reasons. The temporary queue is also held in memory, and requires the same amount of memory as the queue being edited. The realm server must be able to allocate sufficient memory to store the temporary copy, otherwise the queue edit operation will be aborted and an error will be logged. If such a situation occurs, you can resolve it by allocating additional heap size, so that the temporary copy can exist in memory at the same time as the queue being edited.

In order to edit a queue, select it in the namespace, and then after right-clicking on the node, a menu will be displayed with the various options for a queue node. The image below shows this menu.

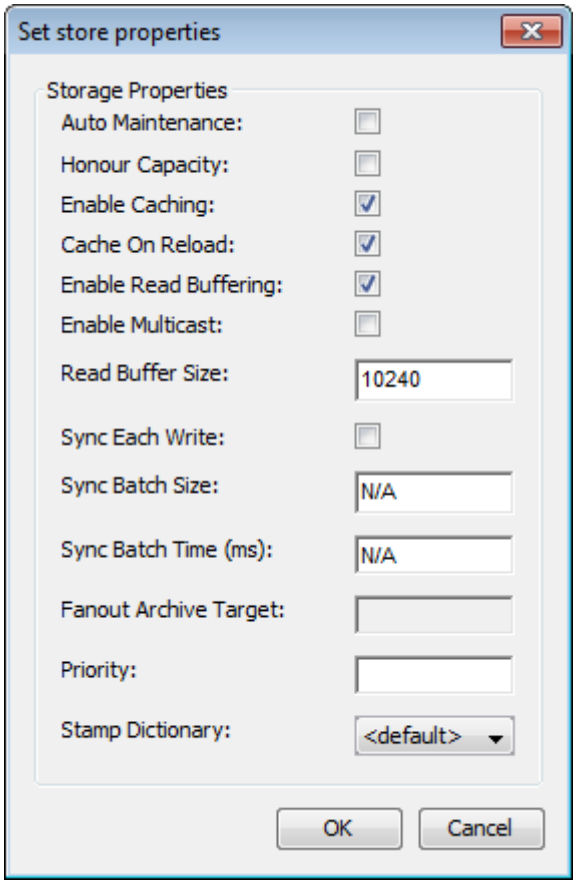


By selecting the 'Edit Queue' option, you will be presented with a dialog that allows you to modify the details of the queue. These details not only include the queue attributes, but also the realm to which the queue exists. The image below shows the edit queue dialog.



The image shows a drop down list containing all the names of the realms that the enterprise manager is currently connected to. By selecting a realm name from the list, it is possible to move the selected queue to any of the available realms. Clicking on the 'OK' button will perform the edit operation on the queue.

There are also a number of Storage Properties associated with the queue which can be configured by clicking the "Edit..." button to the right of "Storage Properties".

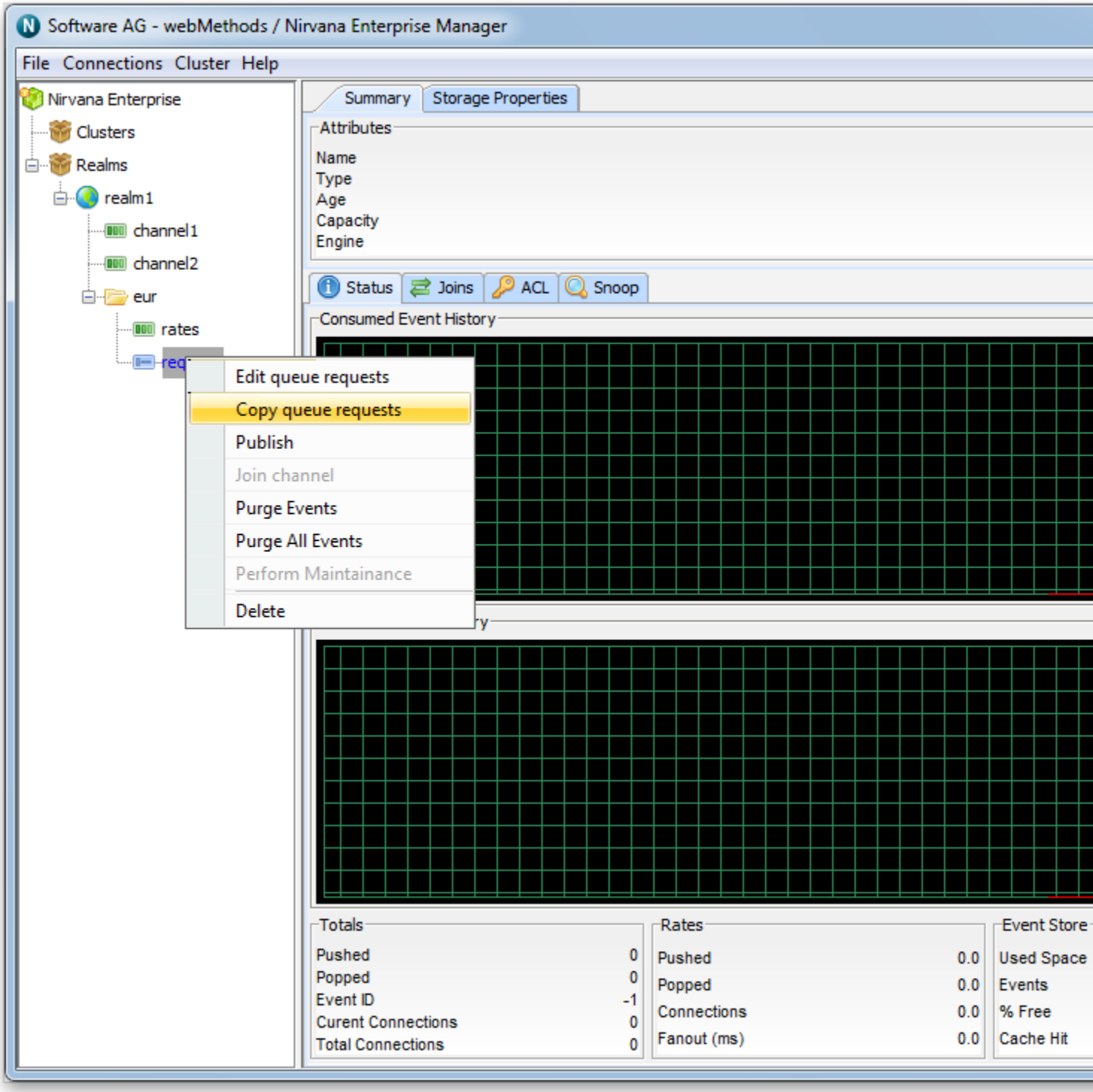


Copying Queues

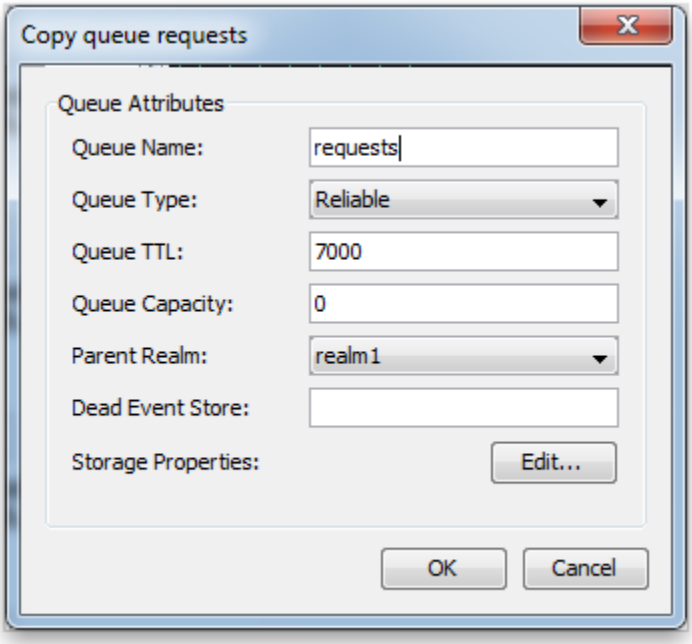
This section will describe the process of copying queues in Universal Messaging realms. Each queue that is created consists of a physical object within the Universal Messaging realm as well as its logical reference within the namespace.

Copying queues using the Enterprise Manager enables you to duplicate queues automatically across realms. When a queue is copied, its attributes and any events found on the queue will be copied over onto the new queue copy.

Firstly, by selecting the queue in the namespace that you wish to copy and right-clicking on the node, you will be presented with a menu that shows you the various options for a queue node. The image below shows this menu.



By selecting the 'Copy Queue' option, you will be presented with a dialog that allows you to input the details of the new queue copy. These details not only include the queue attributes, but also the realm to which the queue will be copied to. The image below shows the copy queue dialog.



The image shows a drop down list containing all the names of the realms that the enterprise manager is currently connected to. By selecting a realm name from the list, it is possible to create a copy of the selected queue in that realm. Clicking on the 'OK' button will create the queue on the selected realm and the queue will then appear in the namespace tree.

Queue Snoop

This section will describe how to snoop a Universal Messaging queue. Each queue that is created consists of a physical object within the Universal Messaging realm as well as its logical reference within the namespace.

Snooping a queue allows you to view the events contained on a queue. Each queue node in the namespace tree of a Universal Messaging realm, when selected, displays a snoop panel that provides you with a means of browsing the queue and present the events on the queue in a graphical panel.

You can also provide a filter that enables you to select specific events that match a certain criteria.

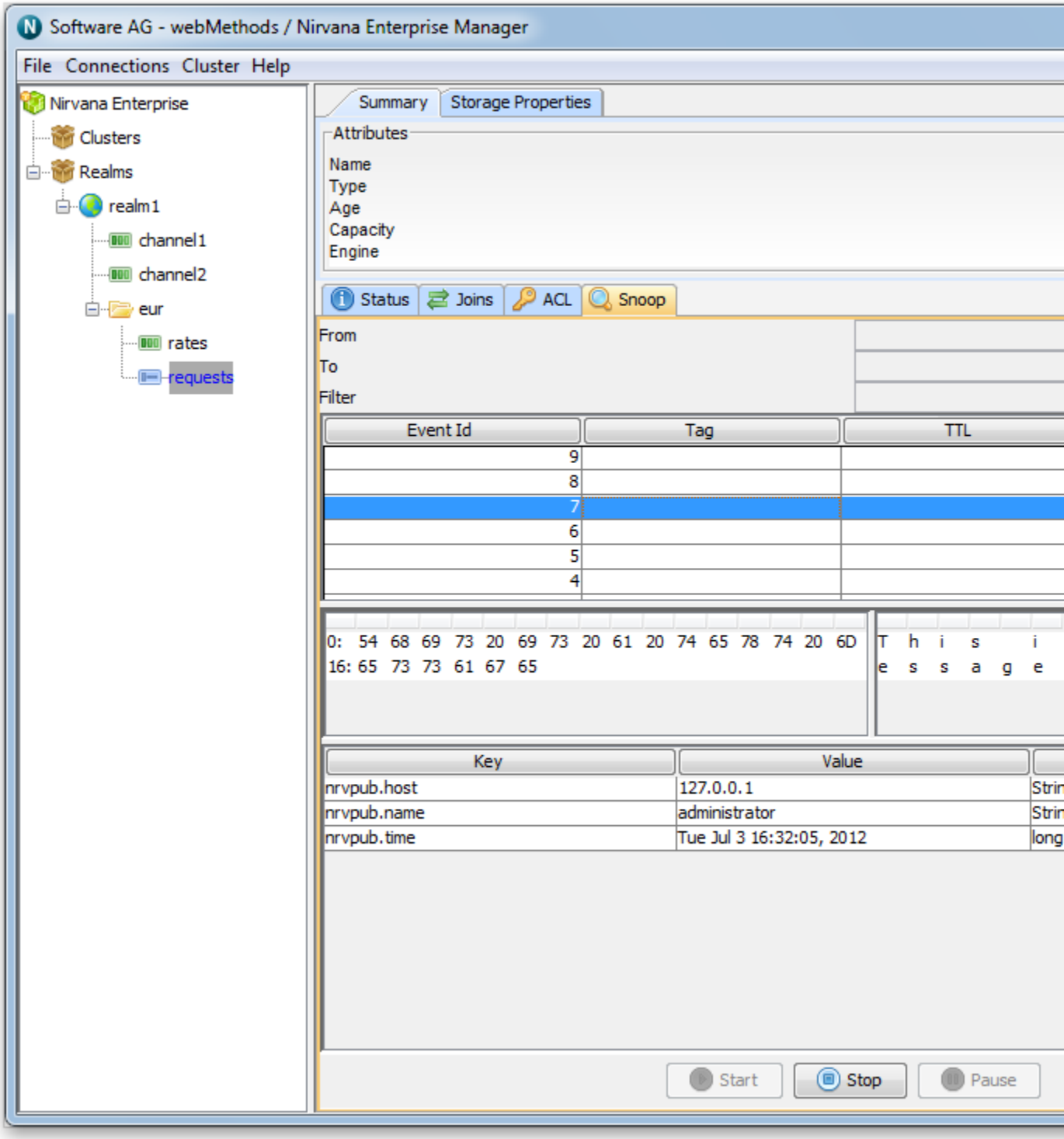
First of all, by selecting the queue you wish to snoop in the namespace tree, the Enterprise Manager will display a number of panels in a tabbed pane. One of these tabs is labelled 'Snoop'. Selecting the snoop tab will display a panel like the one shown in the image below.

The snoop panel is split up into a number of different sections. Firstly, the 3 text fields at the top of the panel allow you enter an event id range to and from, and a selector string that will be used to filter events being snooped on the queue. For queues, the event id ranges are disabled. Clicking the 'Start' button will begin the queue snoop, and start displaying any events that are published onto the queue using whatever values you have input into the text fields.

When events are published, they are added to the main table below the text field input. This main table shows 4 columns of basic information about each event: the event id, event tag, time to live,

and whether the event is persistent. By clicking on any event shown as a row in the main table, more information on the event is shown in the bottom 3 panels. As shown in the image below.

The top 2 remaining panels show a Hexidecimal view of the event data and an ASCII representation of the same event data. The panel below that shows the contents of the event properties (if one exists for the event) listed within a table. Each property is displayed as a row in the table. The table columns show the name of the property, the type, and the value.



The button labelled 'Pause' will temporarily suspend receipt of any new message being received into the snoop panel for the selected queue. The 'Stop' button will stop snooping events and clear all the panels and tables.

Channels can also be snooped using the snoop panel (see "Channel Snoop" on page 156).

Data Group Administration

Creating Data Groups

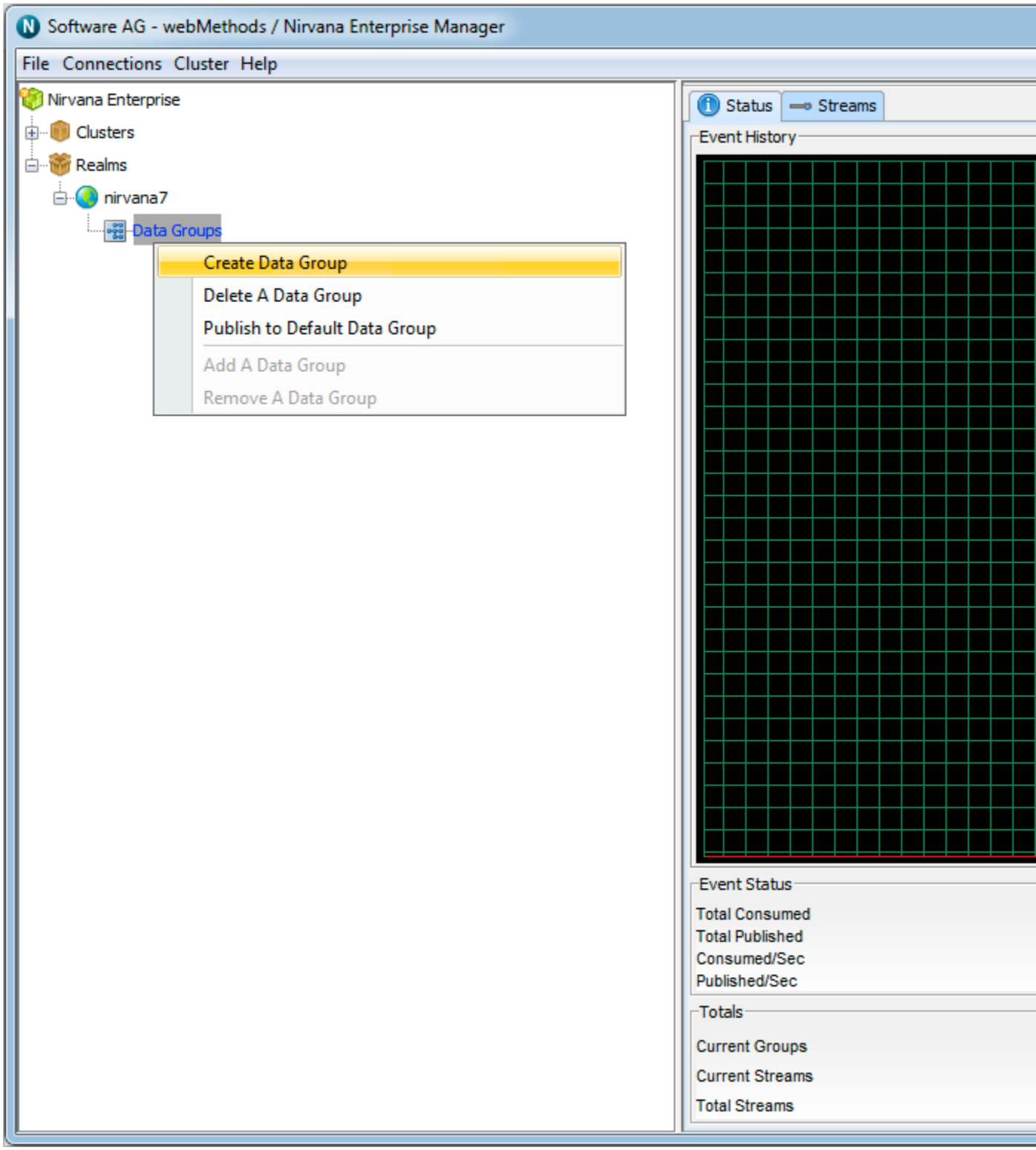
Universal Messaging data groups provide a very lightweight grouping structure that allows developers to manage user subscriptions remotely and transparently.

Each data group is a resource that exists within the Universal Messaging realm server, or within a cluster of multiple realm servers. Creating a data group - in this case using the Enterprise Manager - creates the physical object within the realm. Once created, references to the data group can be obtained using the Universal Messaging Client and Admin APIs. Data groups can also be monitored and managed using the Enterprise Manager.

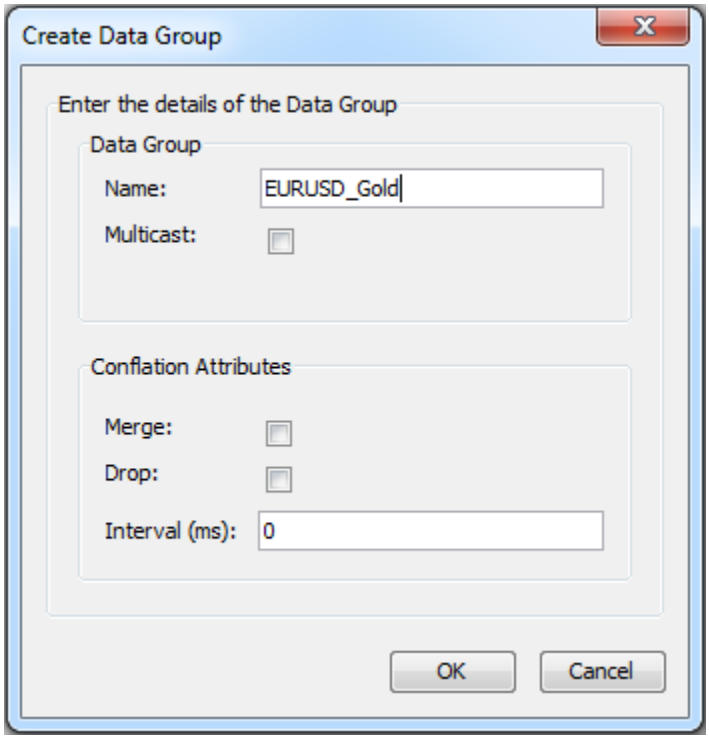
When you connect to a Universal Messaging realm in the Enterprise Manager, all resources and services found within the realm namespace are displayed in a tree structure under the realm node itself. It is possible to view multiple Universal Messaging realm servers from a single enterprise manager instance.

Creating a Data Group

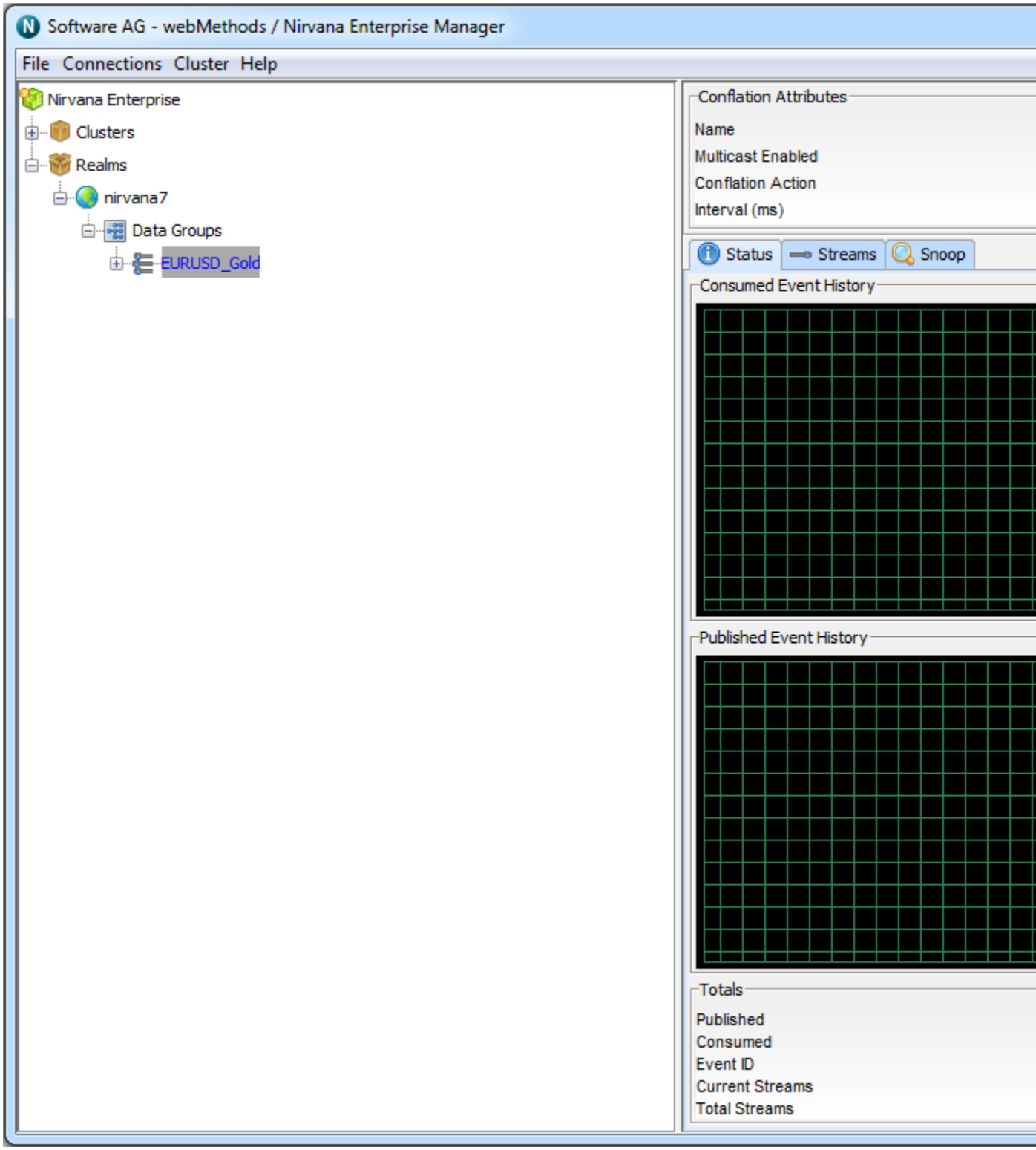
In Enterprise Manager, the **Data Groups** node exists within the realm node. Locate the Data Group Node, and right click on it to bring up a context menu:



Create the data group - in this example, we'll call it EURUSD_Gold :



The new data group can now be seen in Enterprise Manager:

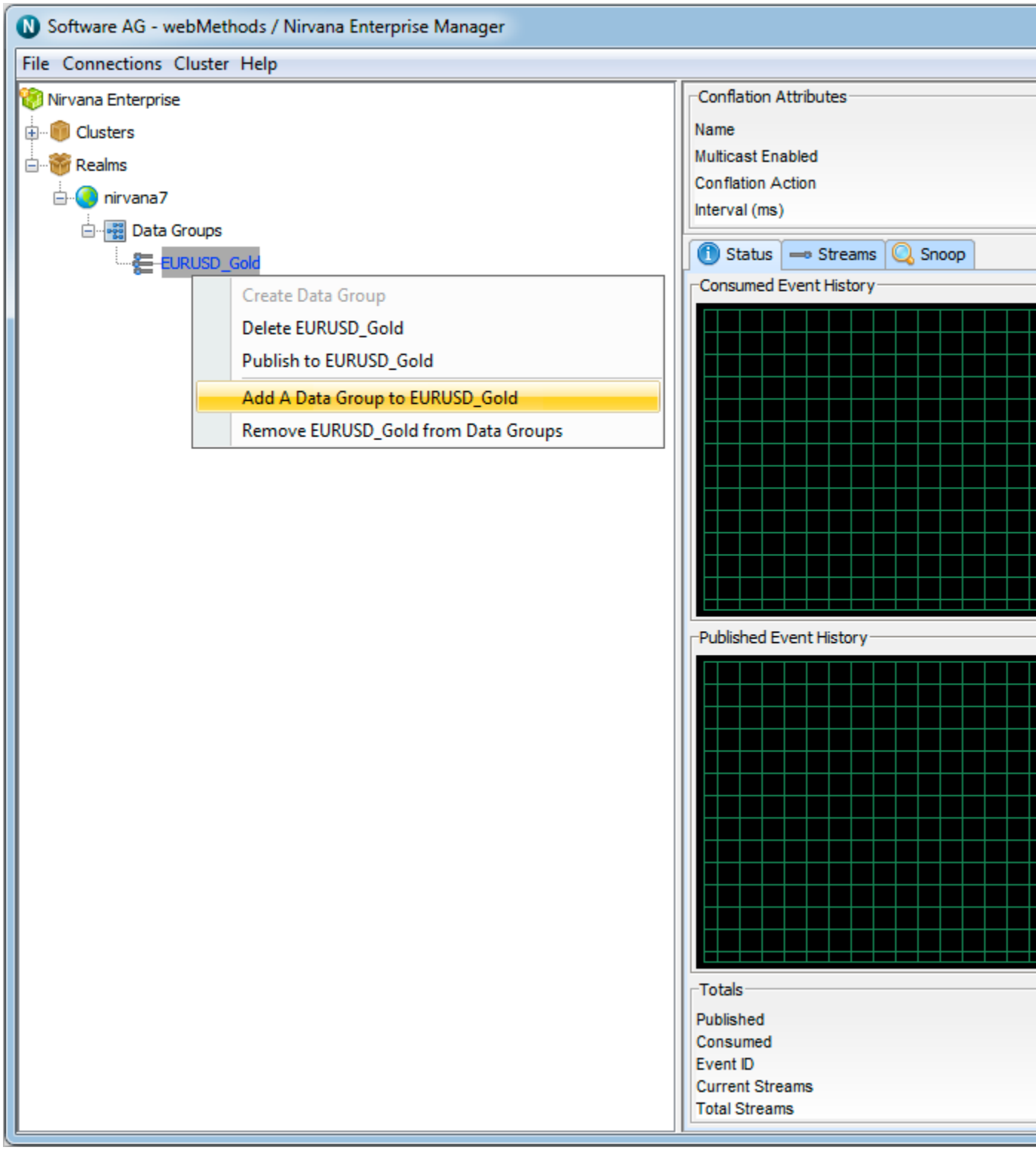


Publishers with the *Publish to DataGroups* ACL permission can now publish messages to the new data group programmatically.

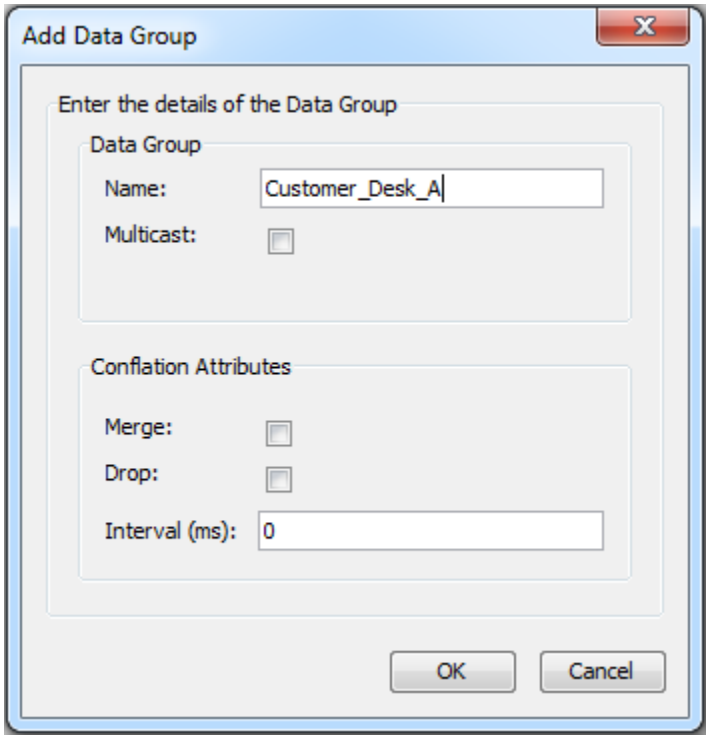
Creating a Nested Data Group

Choose the data group that is going to contain a new data group. In this example, we'll choose the EURUSD_Gold data group we created earlier.

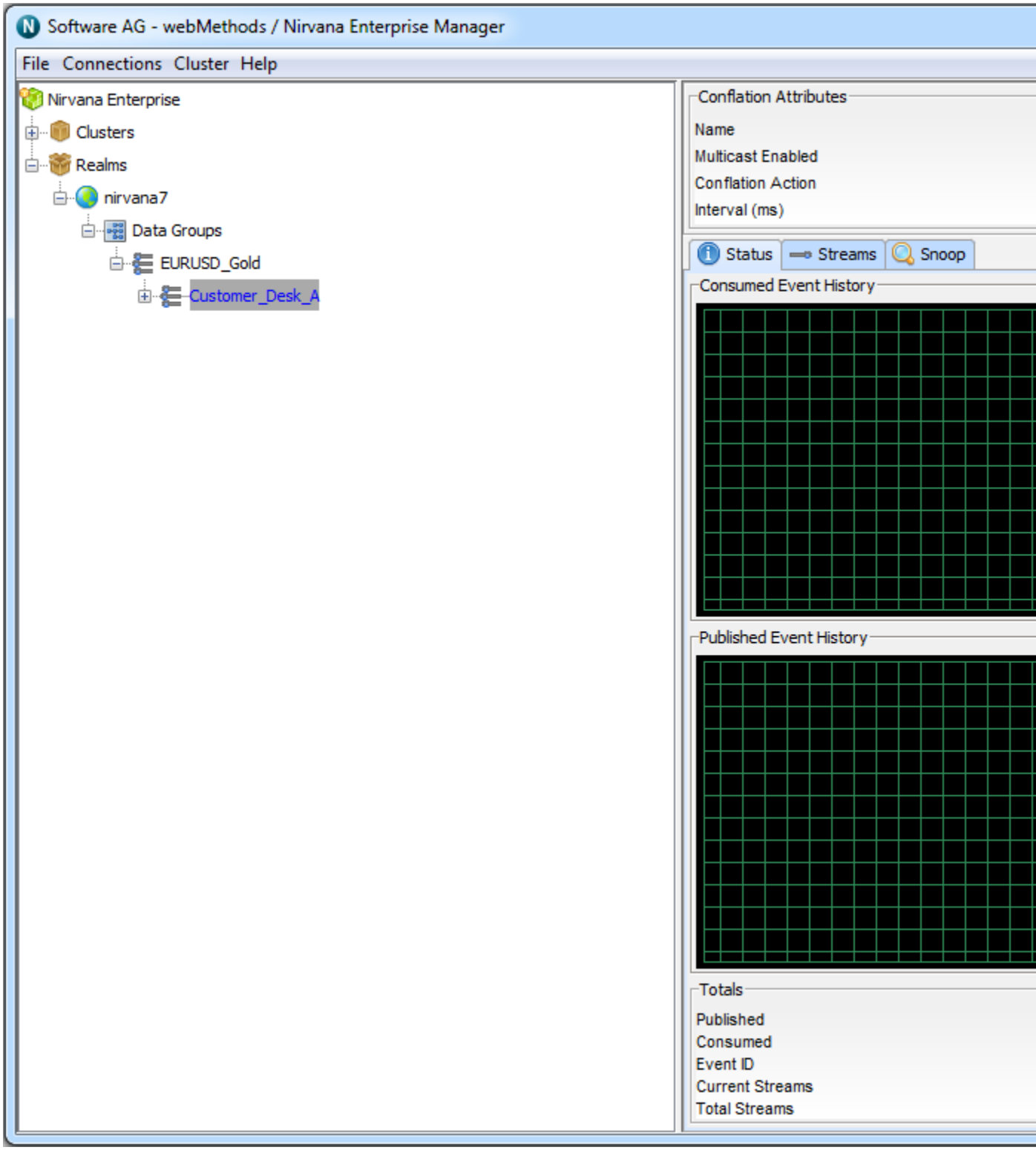
Right-click its icon, and the following context menu appears:



Add the nested data group:



The nested data group can now be seen in the tree:



Data Group Status

When you select the 'Data Groups' node from the tree, one of the available panels to select is labeled 'Status' panel.

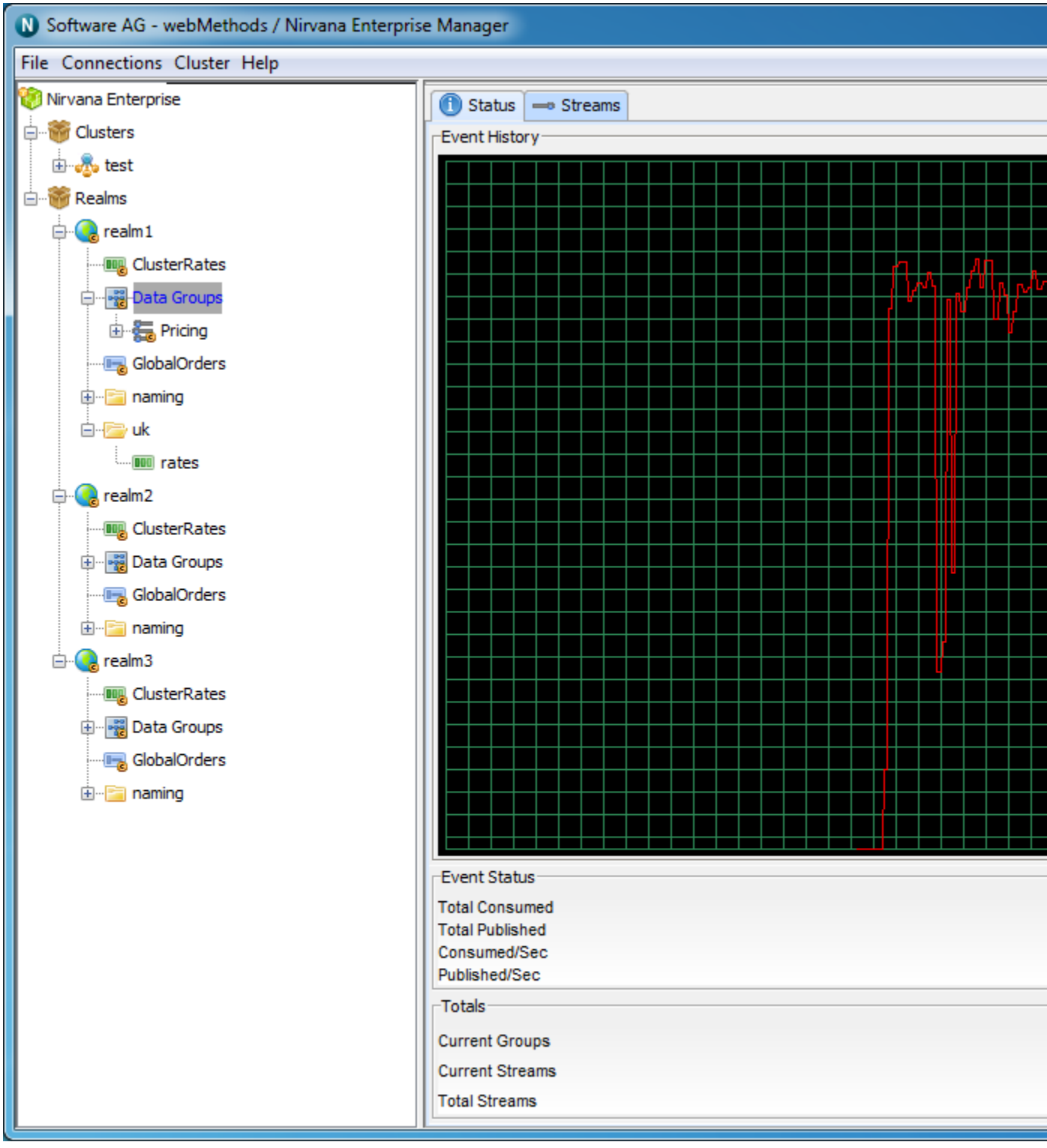
The Status panel for The 'Data Groups' node contains information regarding the publish and consumed events on Data Groups as well as the number of Data Groups and Data Streams currently connected

The status information shown within this panel is explained below.

The Status panel is split into 2 main sections. The top most section of this panel shows a graph that demonstrates **Event History**

The event history graph shows the rates that events are published (red) and consumed (yellow) across all data groups in the current realm.

This graph is updated every time a status event is received from the realm in which data groups are actively being used. The image below demonstrates the status graphs as described.



The bottom section of the panel displays 4 sections of information, **Event Status**, **Totals**, **Connection Status** and **Storage Usage** respectively. These panels and the information displayed are described below.

Event Status

The Event Status section describes the following :

- **Published** - The total number of events published to all channels, queues and services within the container
- **Consumed** - The total number of events consumed from all channels, queues and services within the container
- **Published/Sec** - The number of events published to all channels, queues and services, per second within the container
- **Consumed/Sec** - The number of events consumed from all channels, queues and services, per second within the container

Totals

The Totals section describes the following :

- **Realms**- The number of realms mounted within this container
- **Channels**- The number of channels that exist within this container
- **Queues**- The number of queues that exist within this container
- **Services**- Total number of services that exist within this container

Connection Status

The Connection Status section describes the following :

- **Total** - The total number of connections made to channels, queues and services within this container
- **Current** - The current number of connections made to channels, queues and services within this container
- **Rate** - The number of connections being made per second to channels, queues and services within this container

Storage Usage (KB)

The Memory Usage section describes 4 values :

- **Total** - The total amount of KB used by channels, queues and services found within this container
- **Free** - The free memory available in the Realm JVM
- **Used** - The amount of memory available in the Realm JVM

- **Change** - The amount of change in Realm JVM memory between newest update and previous update

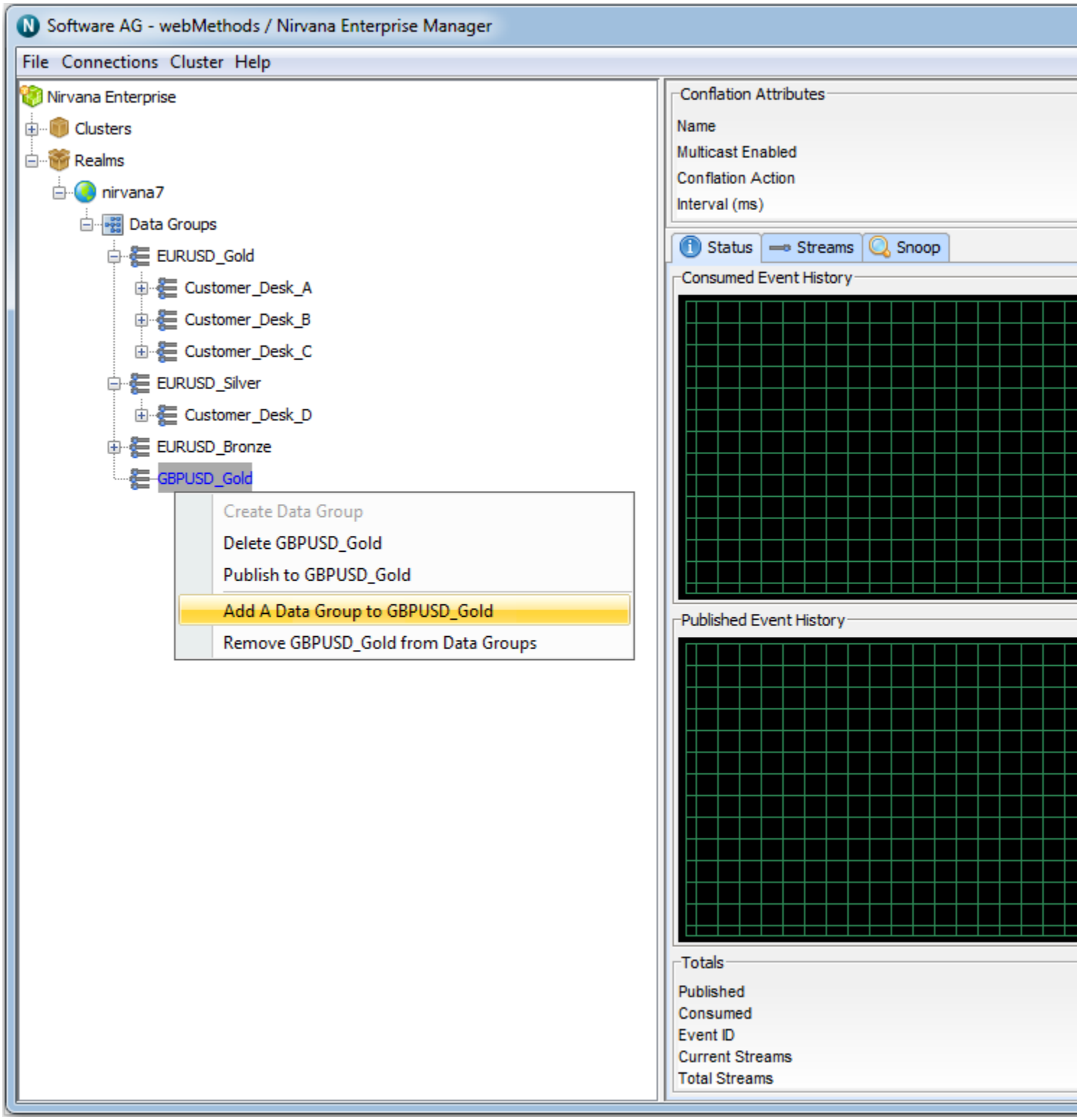
Adding Existing Data Groups to Data Groups

We've already seen how to use Enterprise Manager to create a new data group and add it to an existing data group (see "Creating Data Groups" on page 182). In this section, we will add an existing data group to a data group.

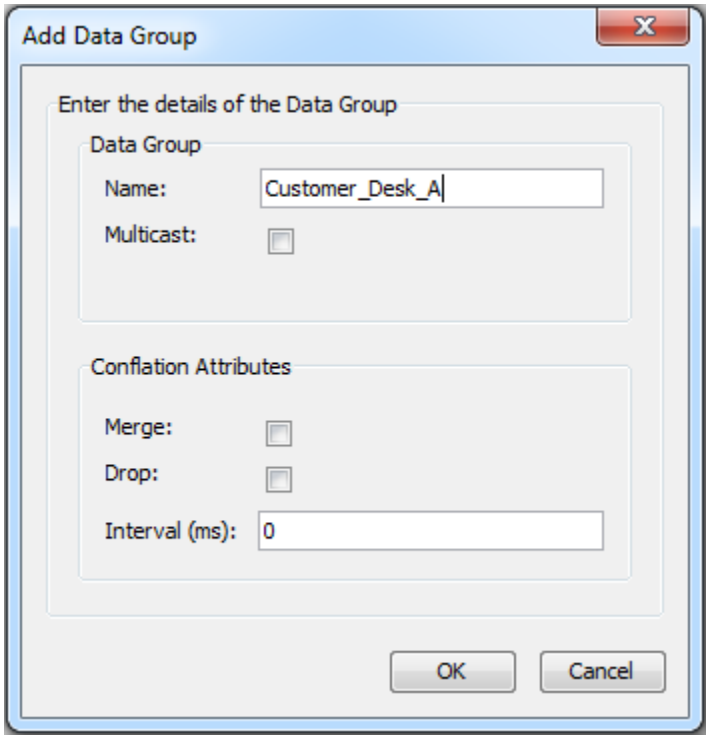
Assume that in the following example structure of data groups, we would like to add the existing **Customer_Desk_A** data group, which itself already a member of the **EURUSD_Gold** data group, to the **GBPUSD_Gold** data group too.

First, choose the data group that is going to contain a new data group. In this case, it's the **GBPUSD_Gold** data group we created earlier.

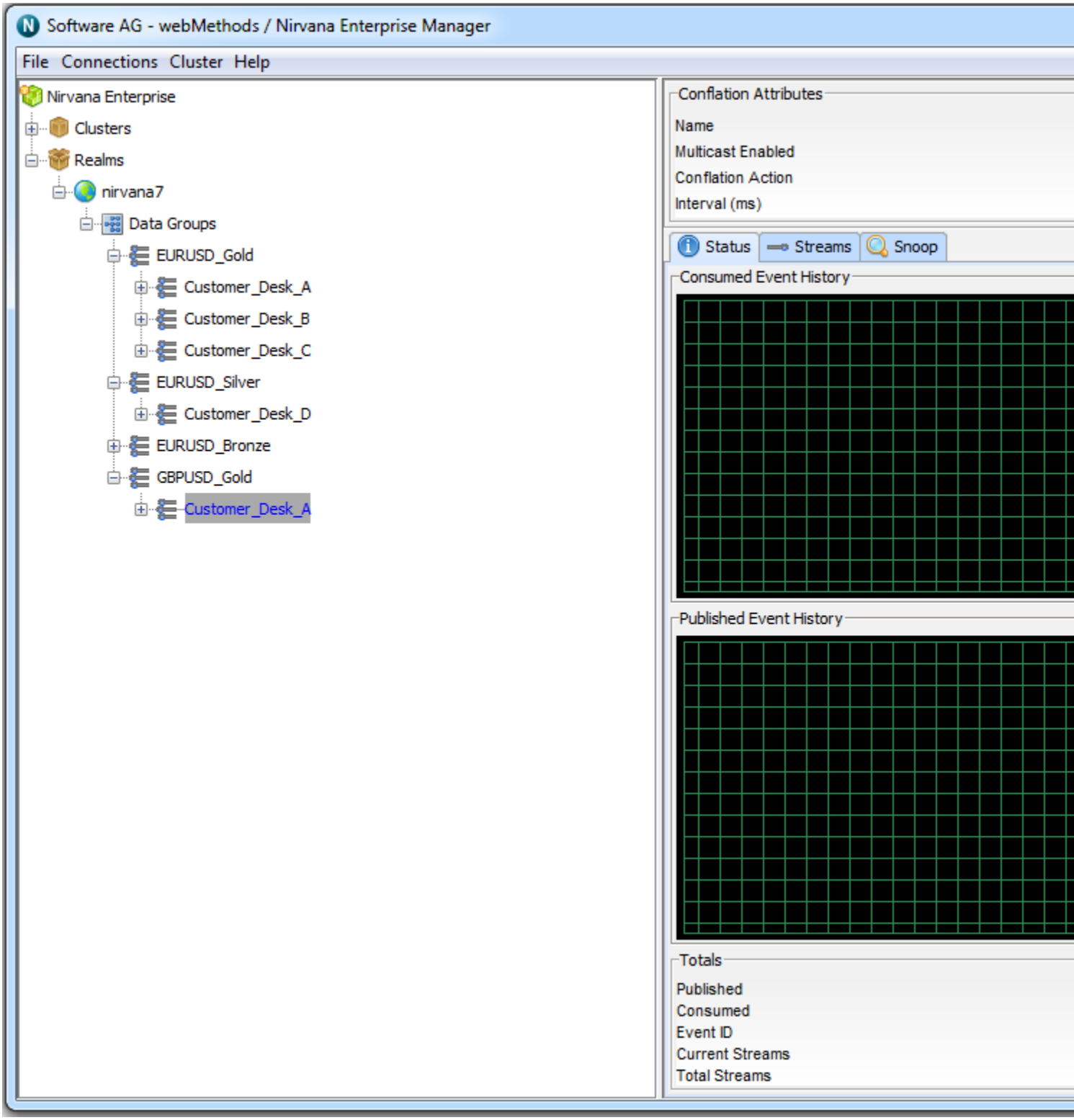
Right-click its icon, and the following context menu appears:



Click the "Add A Data Group to GBPUSD_Gold" context menu option, then type in the name of the data group we wish to add as a member - in this case, **Customer_Desk_A**:



Because this data group already exists (and was a member of **EURUSD_Gold**), it now appears as a member of two data groups (and thus appears twice in the tree of Data Group nodes):



Now, any events published to the data groups **EURUSD_Gold** or **GBPUSD_Gold**, or directly to the data group **Customer_Desk_A**, will be delivered to any data streams which are members of the **Customer_Desk_A** data group.

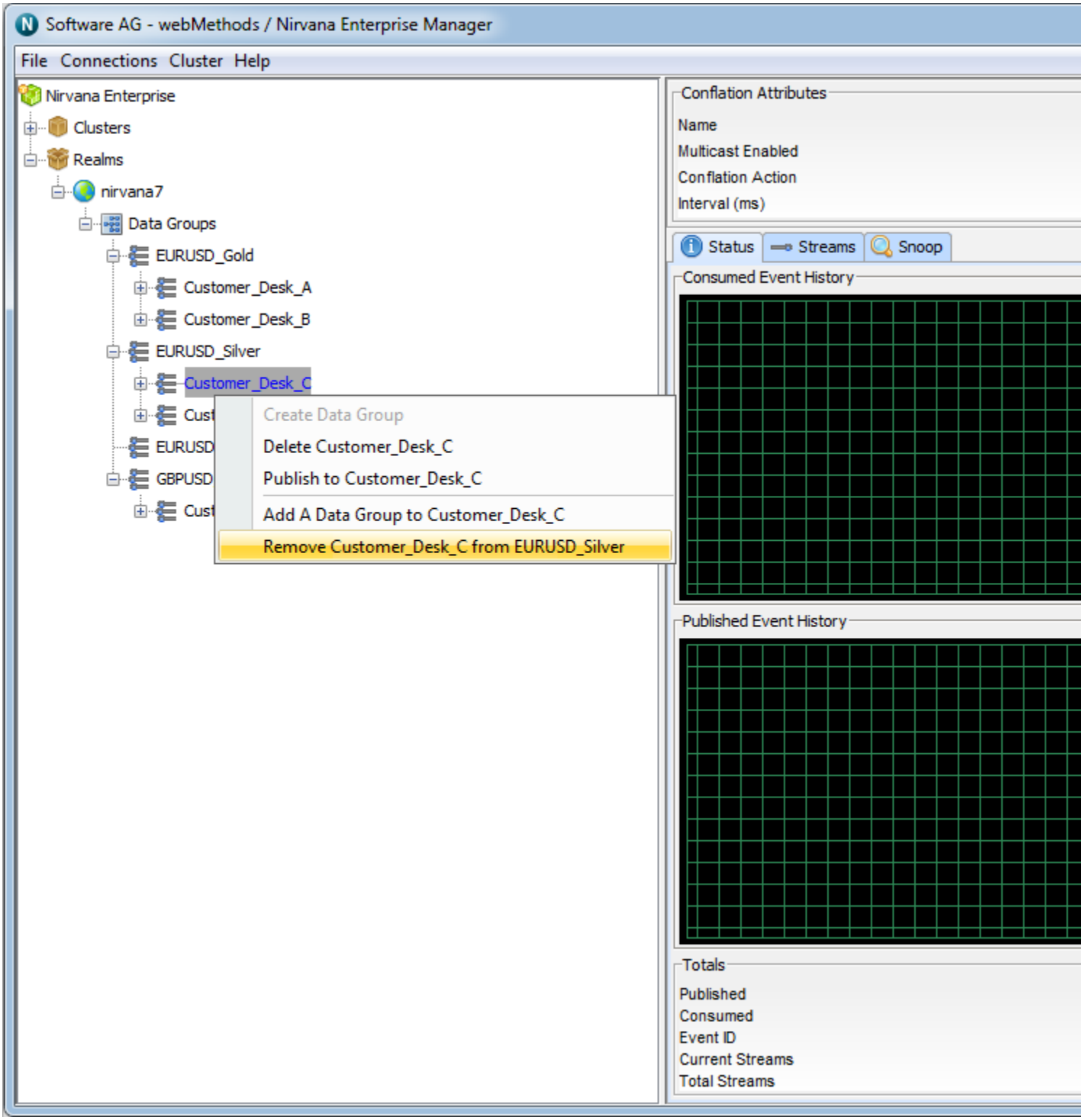
Removing Data Groups from Data Groups

In this section, we will remove an existing data group from a data group.

Assume that in the following example structure of data groups, we would like to remove the existing **Customer_Desk_C** data group from the **EURUSD_Silver** data group.

First, choose the data group that is going to be removed from its "parent" data group. In this case, it's the **Customer_Desk_C** data group we created earlier.

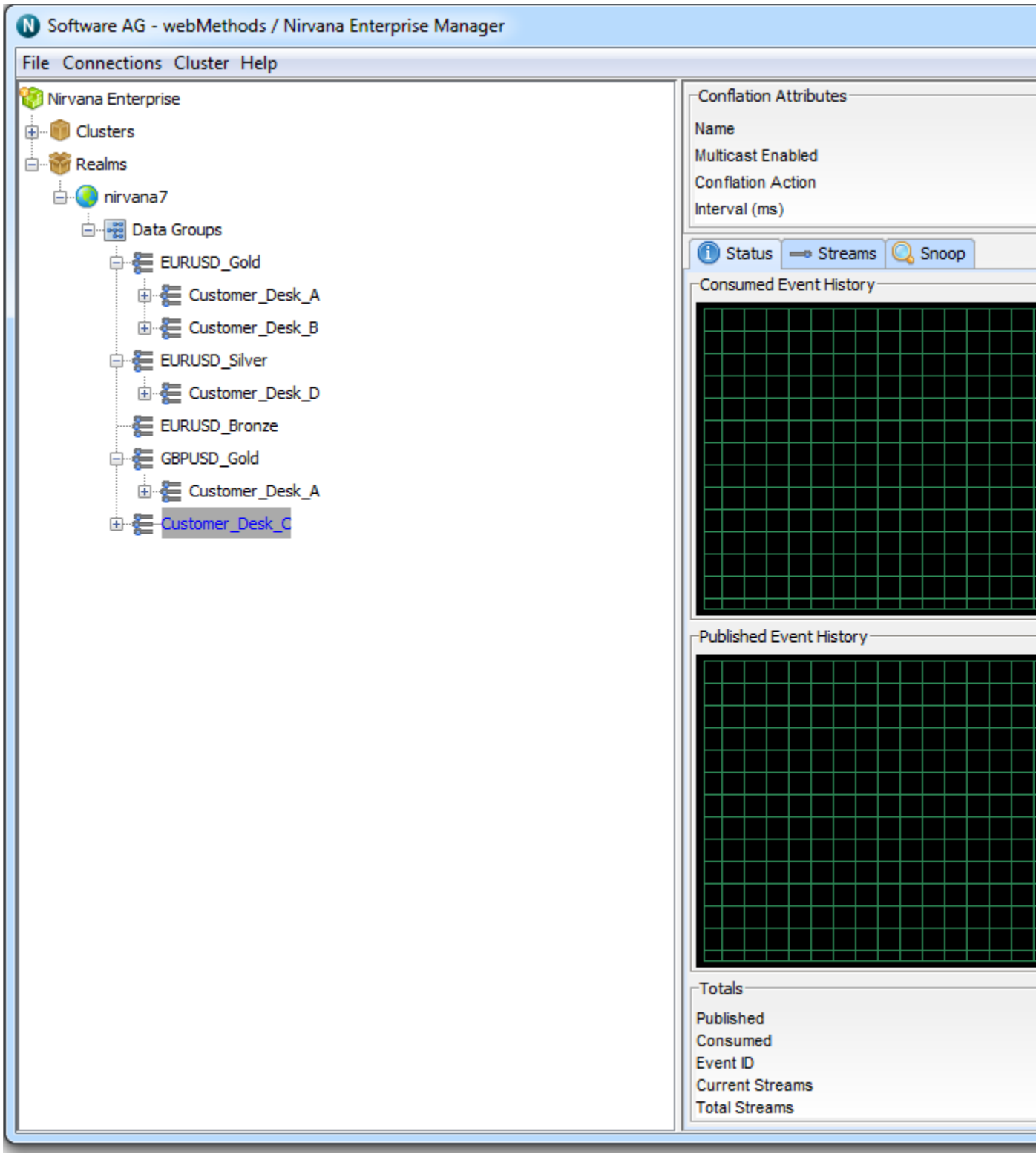
Right-click its icon, and the following context menu appears:



Click the "Remove Customer_Desk_C from EURUSD_Silver" context menu option, then click OK on the confirmation dialog.

The data group, having been removed, will either:

- Be moved to the top level **Data Groups** node if it has no other parent data groups, or
- Appear in other nodes in the tree if it has at least one other parent data group.



In this example, the **Customer_Desk_C** data group was not a member of any other data groups, so having been removed from the **EURUSD_Silver** data group, it now appears in the top level **Data Groups** node.

Deleting Data Groups

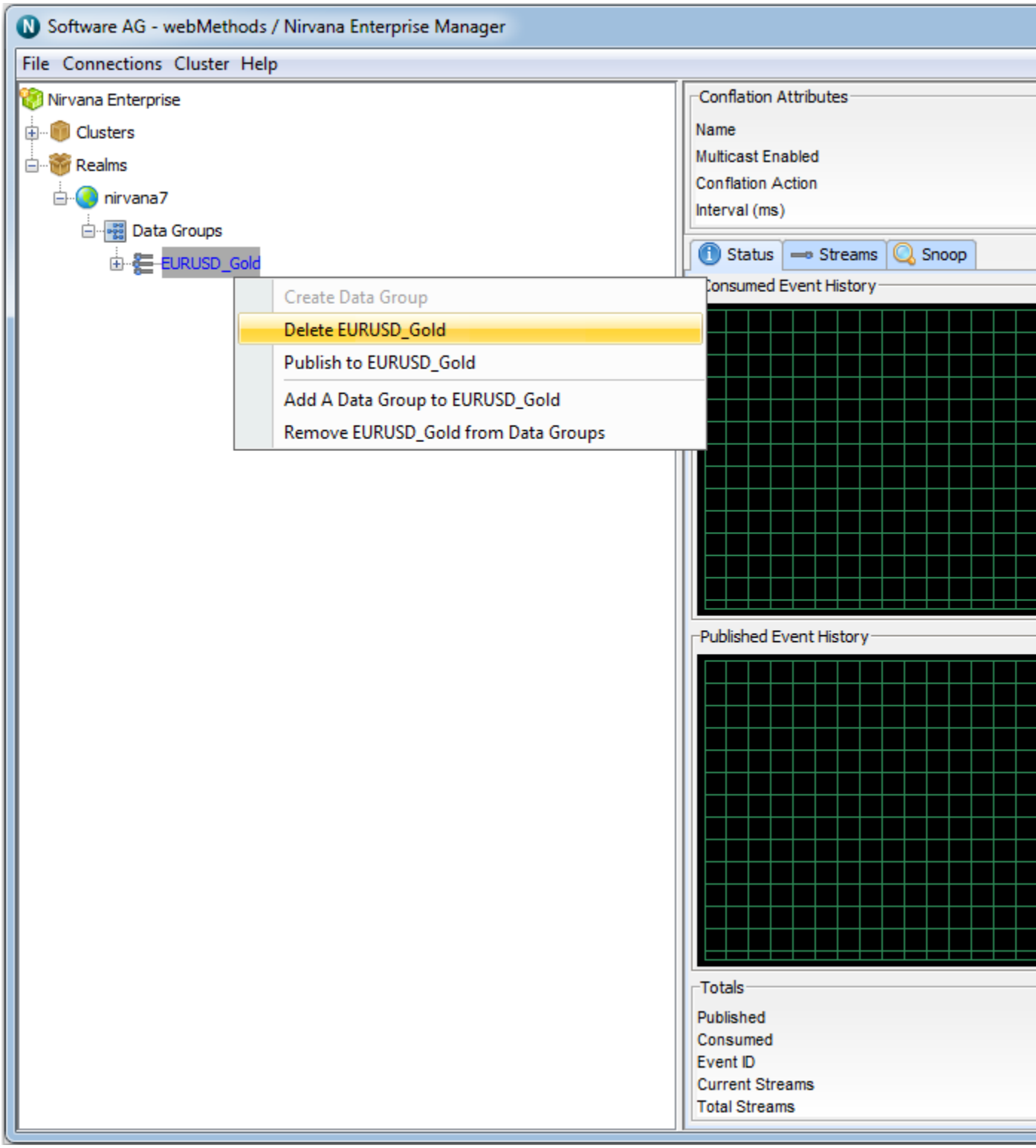
There are two ways of deleting a data group using the Enterprise Manager:

1. by navigating the **Data Groups** tree
2. by typing in its name

Note that if a deleted data group is a member of more than one parent data group, then it will be deleted from all of them, and will no longer be defined for use elsewhere.

Deleting by Navigating the Data Groups Tree

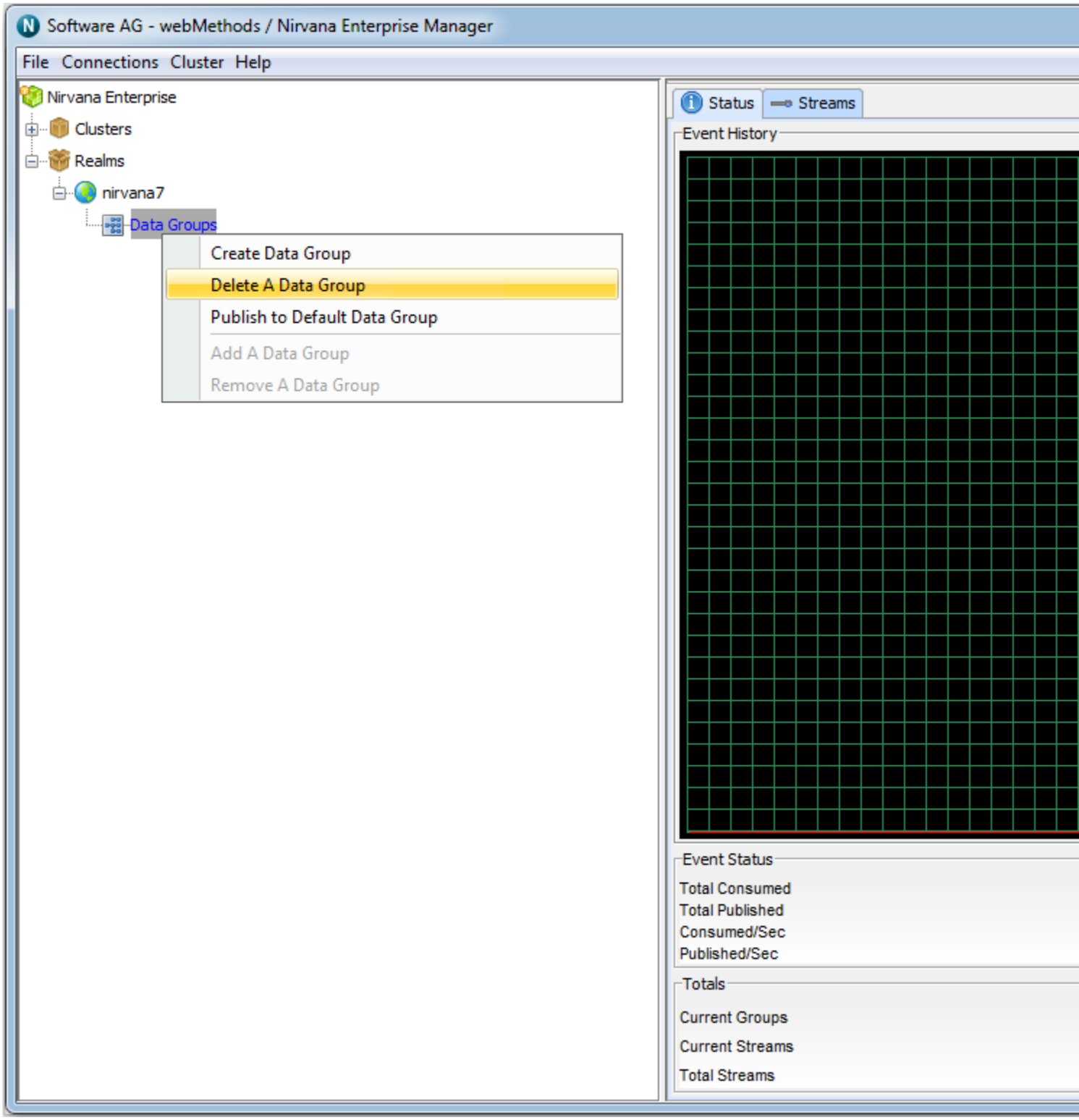
In Enterprise Manager, locate the Data Group Node (in this example, **EURUSD_Gold**), and right click on it to bring up a context menu:



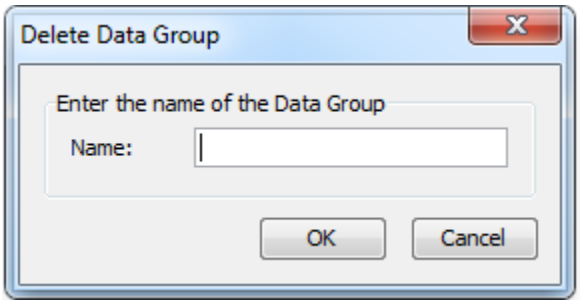
Click OK on the confirmation dialog, and the data group will be deleted.

Deleting by Typing in the Data Group Name

In Enterprise Manager, right-click the **Data Groups** node and select the **Delete A Data Group** context menu option:



In the resulting dialog box, type in the name of the data group to be deleted:



Click OK on the confirmation dialog, and the data group will be deleted.

Container Administration

Container Status

When you select a container (folder) from the namespace, one of the available panels to select is labeled 'Totals'.

The Totals panel for a container provides status information for resources and services contained within the selected container branch of the namespace tree.

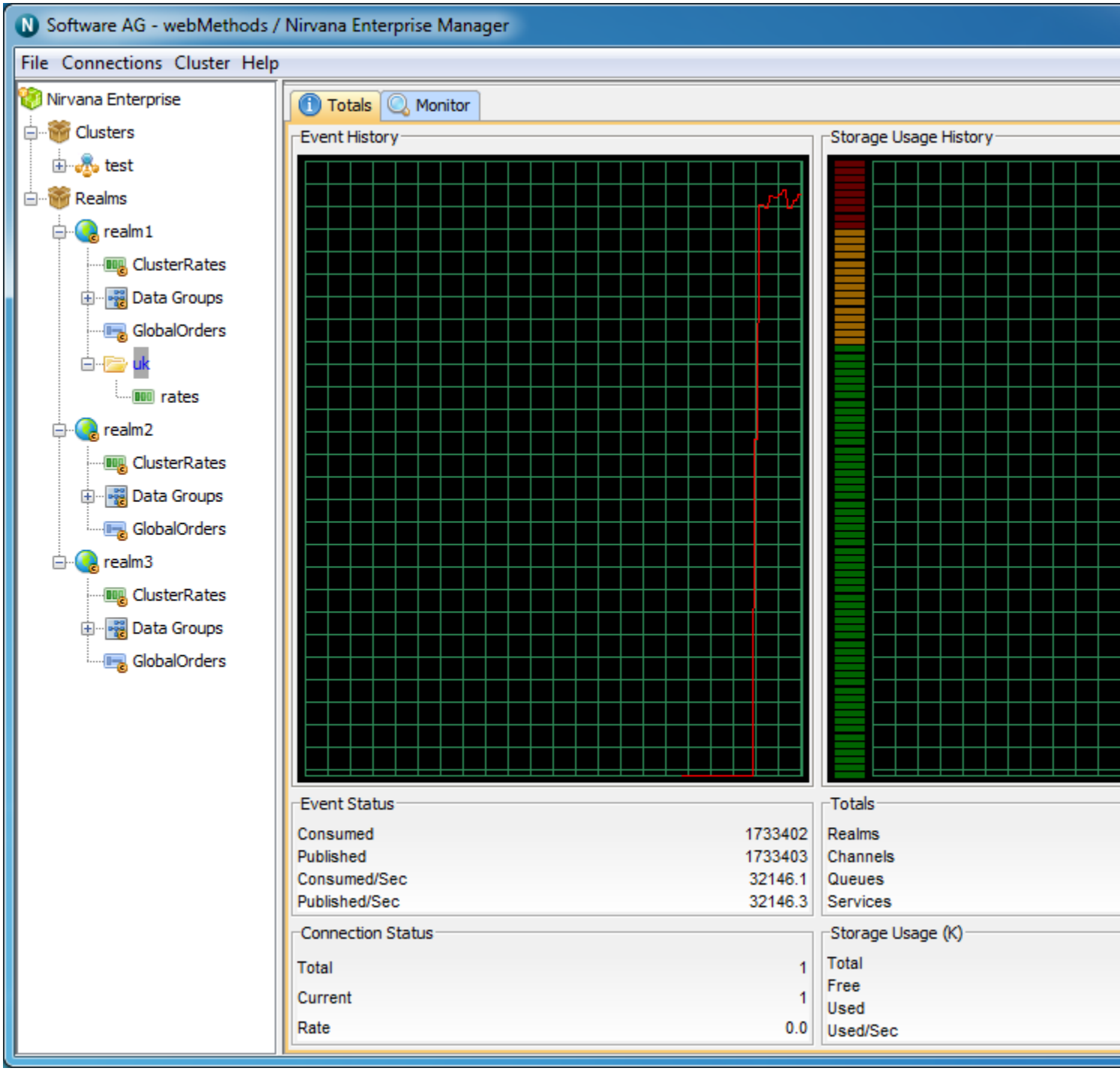
The status information shown within this panel is explained below.

The Totals panel is split into 2 main sections. The top most section of this panel shows 2 graphs, one demonstrates **Event History**, and the other **Storage Usage History**.

The event history graph shows the rates that events are published (red) and consumed (yellow) across all channels, queues and services found within the selected container.

The storage usage history graph shows the total amount of storage space used by each channel, queue and service found within the selected container.

Both graphs are updated every time a status event is received from the realm in which the container exists. The image below demonstrates the Container status graphs as described.



The bottom section of the panel displays 4 sections of information, **Event Status**, **Totals**, **Connection Status** and **Storage Usage**. These panels and the information displayed are described below.

Event Status

The Event Status section describes the following :

- **Published** - The total number of events published to all channels, queues and services within the container
- **Consumed** - The total number of events consumed from all channels, queues and services within the container
- **Published/Sec** - The number of events published to all channels, queues and services, per second within the container
- **Consumed/Sec** - The number of events consumed from all channels, queues and services, per second within the container

Totals

The Totals section describes the following :

- **Realms**- The number of realms mounted within this container
- **Channels**- The number of channels that exist within this container
- **Queues**- The number of queues that exist within this container
- **Services**- Total number of services that exist within this container

Connection Status

The Connection Status section describes the following :

- **Total** - The total number of connections made to channels, queues and services within this container
- **Current** - The current number of connections made to channels, queues and services within this container
- **Rate** - The number of connections being made per second to channels, queues and services within this container

Storage Usage (KB)

The Memory Usage section describes 4 values :

- **Total** - The total amount of KB used by channels, queues and services found within this container
- **Free** - The free memory available in the Realm JVM
- **Used** - The amount of memory available in the Realm JVM
- **Change** - The amount of change in Realm JVM memory between newest update and previous update

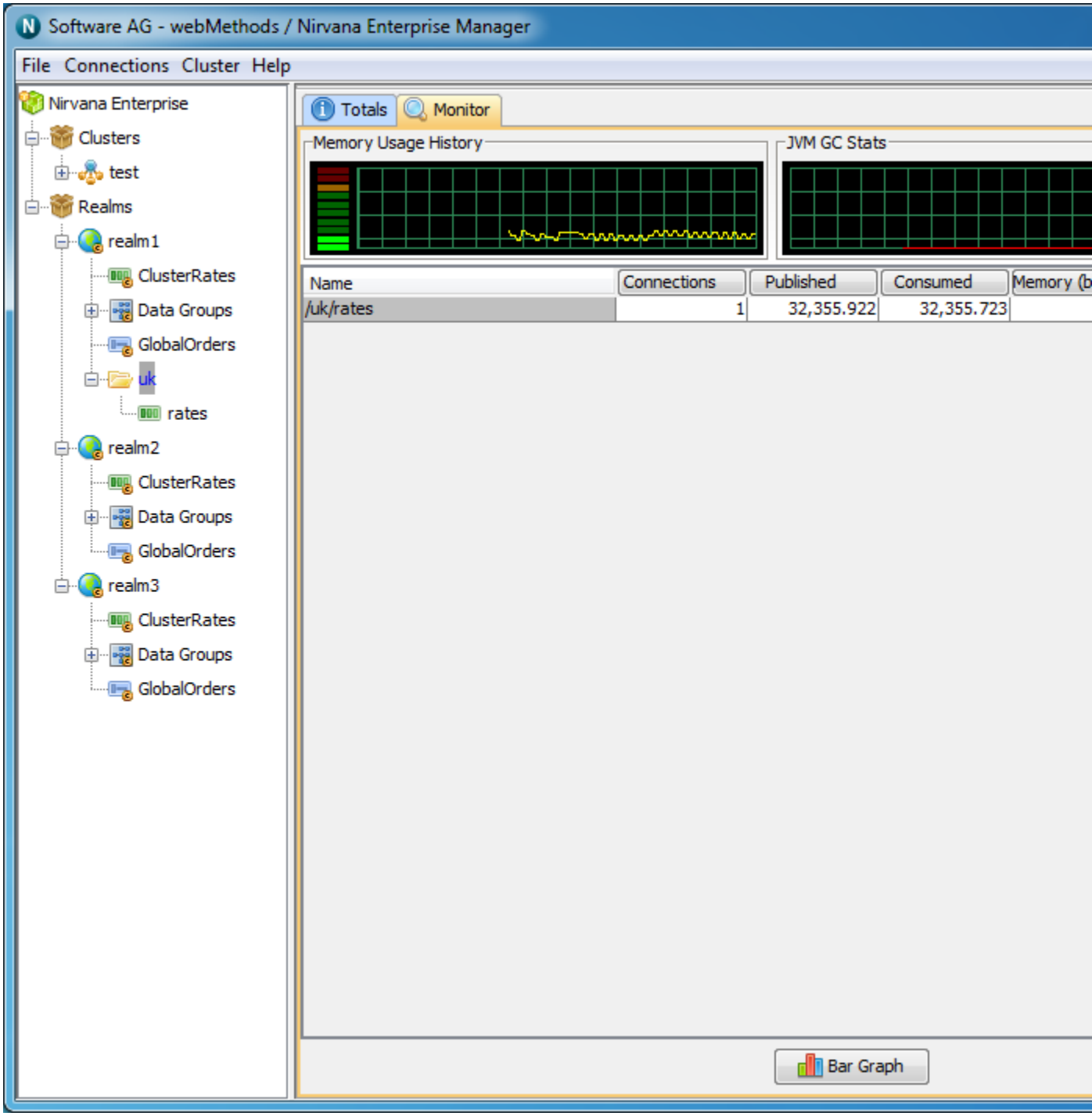
Container Monitor Panel

When you select a container (folder) from the namespace, one of the available panels to select is labeled 'Monitor'.

The Monitor panel provides a view not unlike 'top' for UNIX systems or task manager for Windows systems. Its main purpose is to present the user with a high level view of usage. The usage is based on channels found within the container node.

The Monitor panel comprises 2 sections. The top-most section contains a real time graph illustrating the realm memory usage in the same way the Realm Status panel (see "Viewing a Realm" on page 16) displays memory usage. This section also contains a summary showing the number of mounted realms, the number of channels, the number of queues and the number of services.

The image below demonstrates the Monitor panel for a container within a clustered realm.



Channel Usage

The next section of the Monitor panel displays a table showing multiple columns and rows. This table represents channel usage throughout the realm. Each row in the table represents a channel. Channel usage can be measured a number of ways. Each measurement corresponds to a column within the table. By clicking on one of the column headers, all known channels found within the container will be sorted according to their value for the selected column. For example, one of the

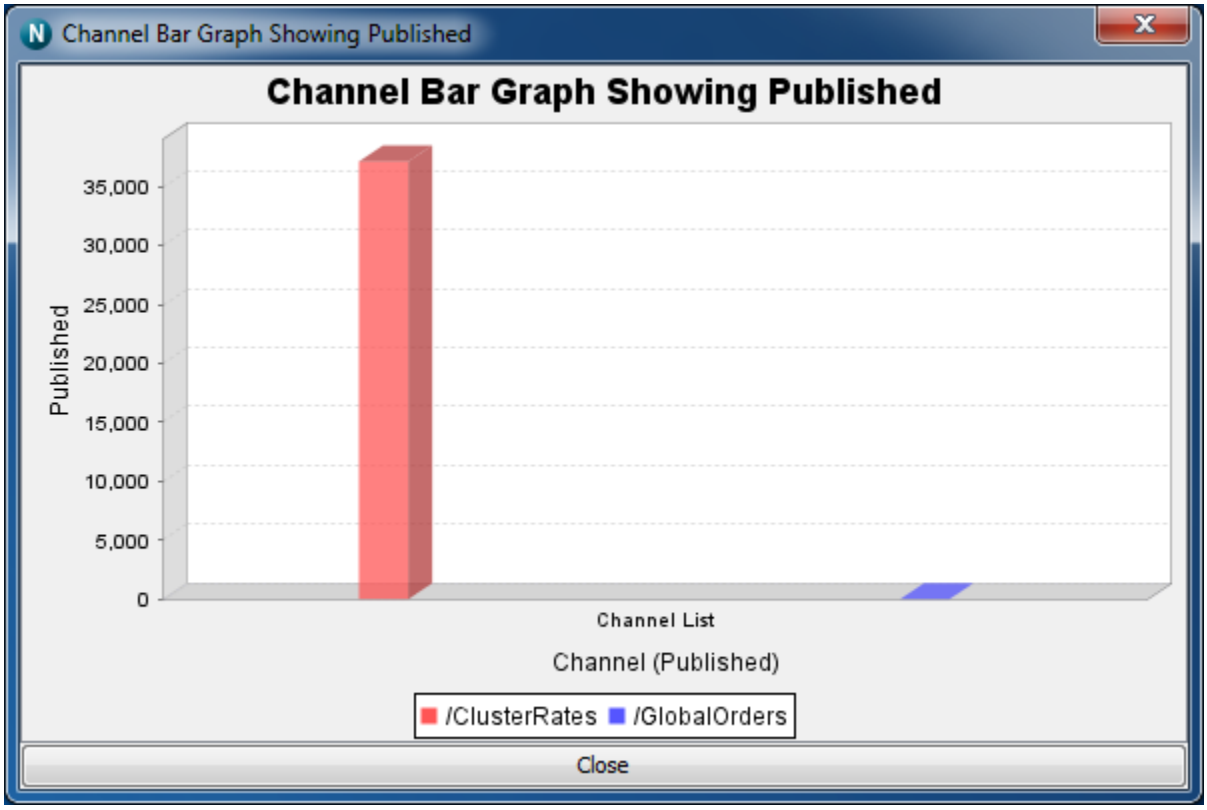
columns is 'Connections', i.e. the number of current consumers on the channel. By clicking on the column header labeled 'Connections', the table will be sorted according to the number of consumers each channel has. The channel with the most number of consumers will appear at the top of the table.

Each column used for channel usage measurements is described below:

- **Connections** - The number of consumers the channel has
- **Published** - The rate of events published per status interval
- **Consumed** - The rate of events consumed per status interval
- **Memory (bytes)** - The number of bytes the channel uses from the JVM memory
- **% Memory** - The percentage of overall JVM memory used by this channel
- **Disk** - The amount of disk space used by this channel, only relevant for persistent / mixed channels

Monitor Graphs

The monitor panel provides a method of graphing channel usage. It uses a 3D graph package from SourceForge (http://sourceforge.net/projects/jfreechart/) to display the items in each table as columns in a 3D vertical bar chart. The bar charts can be update live as the values in the tables are updated. Once a column is selected, simply click on the button labeled 'Bar Graph' under either the channel or connections table and a graph panel will appear, as shown in the image below showing a graph of the number of events published for channels within the container.



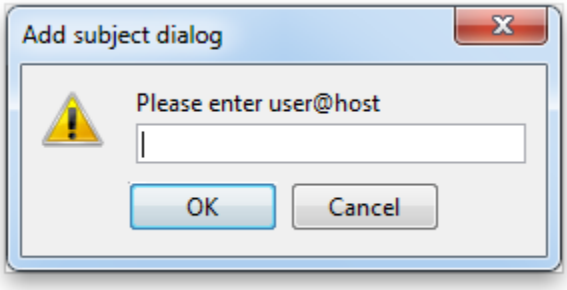
Right-clicking anywhere within the graph will show a pop-up menu of options. One of the options is labeled 'Start Live Update', which will ensure the graph consumes updates as and when they occur to the table. Once the live update is started, you can also stop the live update by once again right clicking on the graph and selecting 'Stop Live Update'.

You can also print the graph, and save the graph image as a '.png' file, as well as alter the properties of the graph and its axis.

Using ACLs for role-based Security

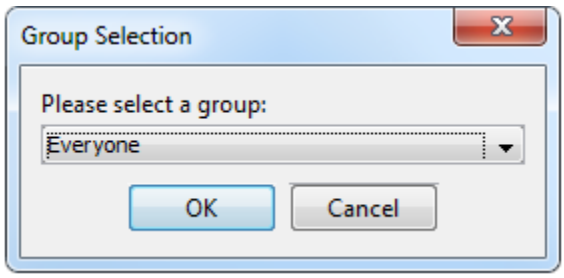
Adding ACLs

The Enterprise Manager allows Access Control Lists (ACLs) to be controlled via the ACL panel which is displayed for each object within the namespace. These panel allows users to add entries to the ACL, as well as remove the selected entry. The image below shows the dialog for adding an ACL entry.



Clicking on the 'OK' button will add the subject to the selected objects ACL list.

Similarly, once they have been defined, Security Groups (see "Nirvana Admin API - Nirvana Security Groups" on page 597) may be added into ACL Lists by clicking the "Add Group" button and selecting the desired group as shown:



When an entry is selected from the ACL panel, and the 'Delete' button is selected, you will be prompted to confirm the deletion.

After any changes made to the ACLs, only when the 'Apply' button is clicked will those changes be sent to the realm server for processing. Clicking the 'Cancel' button will discard any changes made and revert back to the state the Realm server has for the ACL.

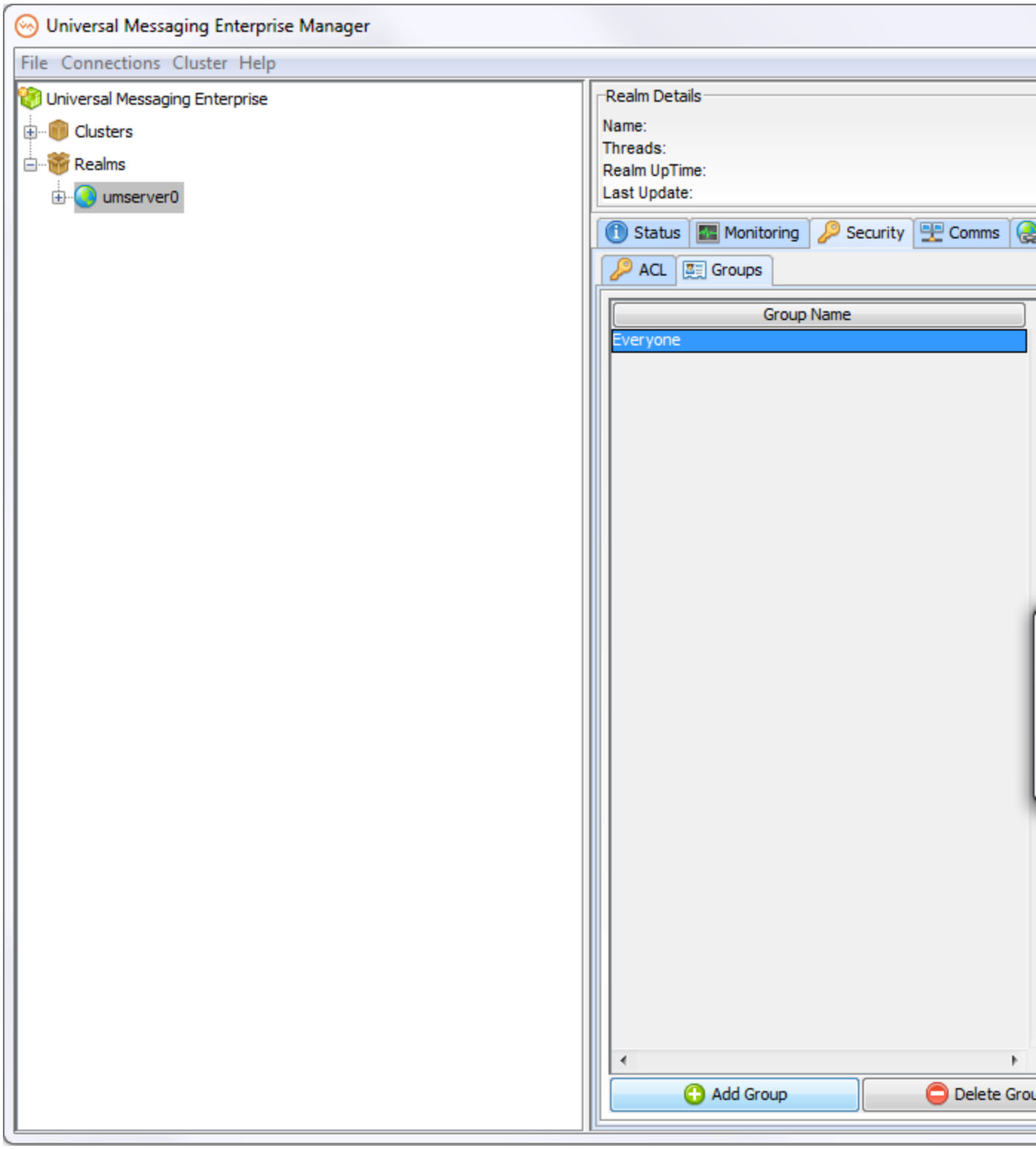
To read more about the entitlements for each object, follow the links below:

- "Security Groups" on page 597
- "Realm ACL" on page 214
- "Channel ACL" on page 216
- "Queue ACL" on page 219
- "Interface VIA ACL" on page 221

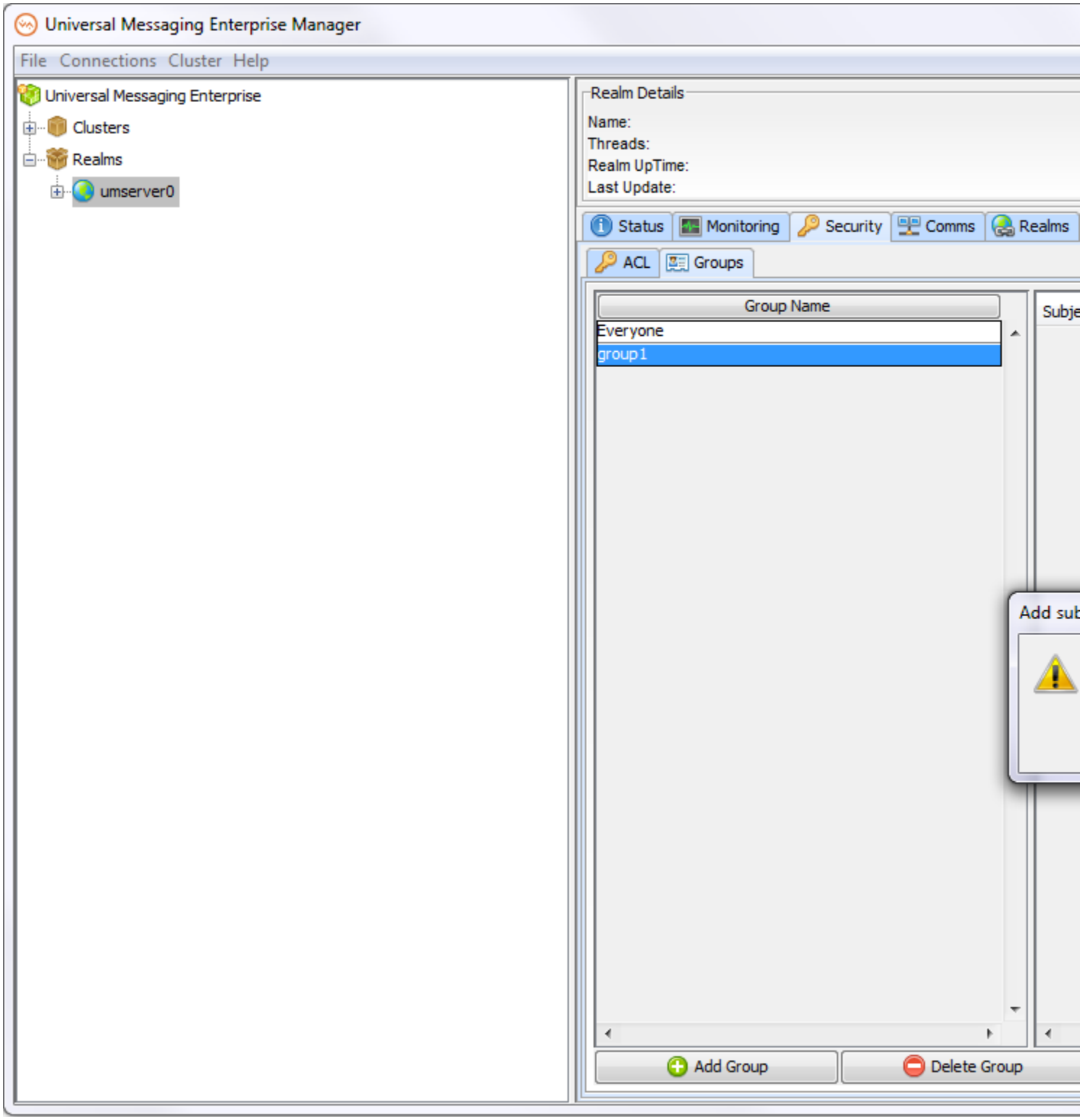
Security Groups

Security groups contain a list of subjects (username & host pairs) and, in addition, may contain other Security Groups. Once a Security group is defined, the group can be added to ACL lists like normal subject(user@host) entries are added and permissioned. This allows for "sets" of users to be defined and granted permissions through a single entry in an ACL list, rather than each user having an entry.

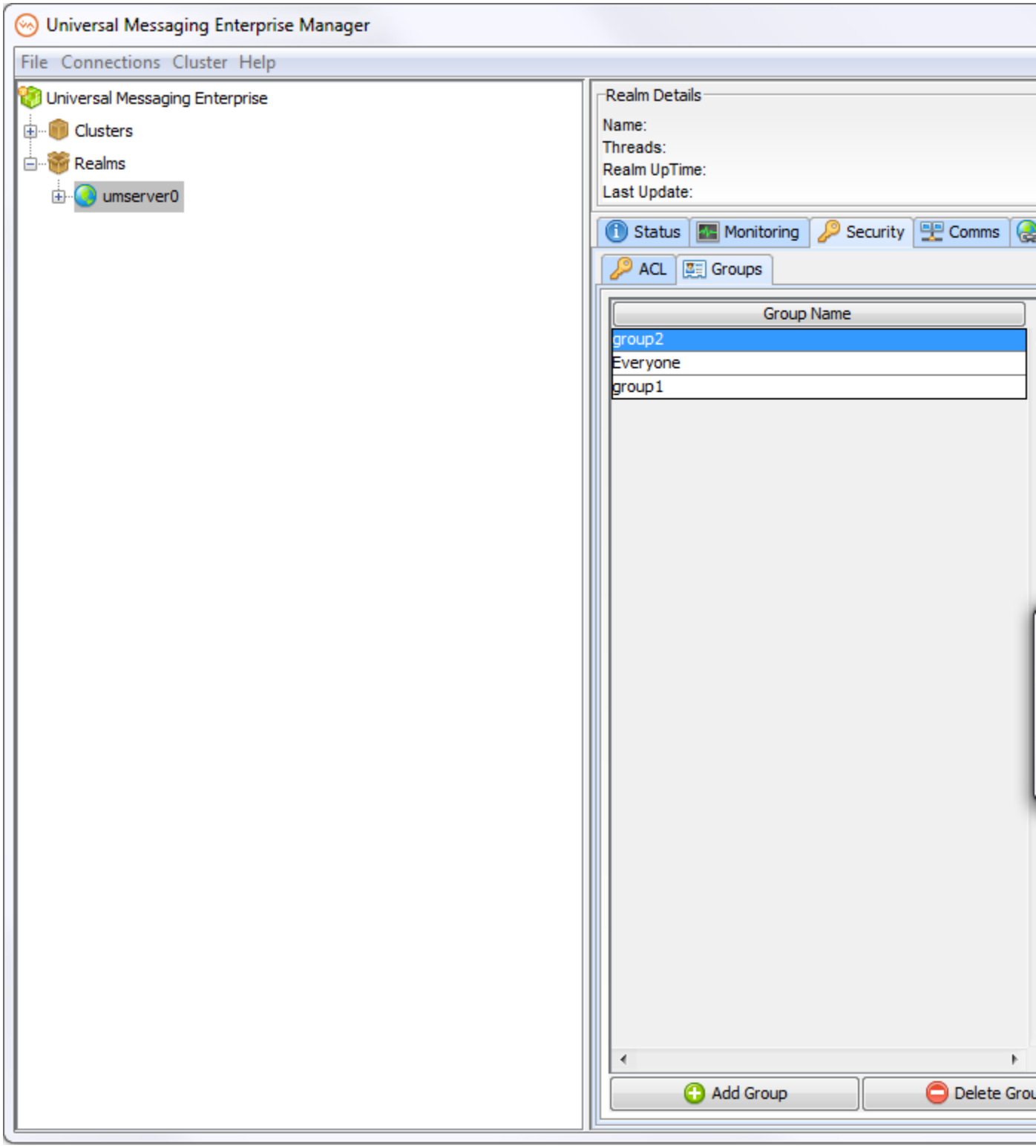
Before adding a Security Group to an ACL, it must first be created. This can be done programmatically or via the Enterprise Manager, as shown below.



Once the group has been created, user@host subjects can be added to the group using the "Add Member" button:



Alternatively, groups can be added as members of other groups by using the "Add Group" button. This will present you with a dropdown list of existing groups to choose from:



Membership of Security Groups can be altered dynamically, and the changes will be reflected in the permissions for all ACL lists where the security group is an entry in the ACL list.

As with all ACLs in Universal Messaging, privileges are cumulative. This means that, for example, if a user is in a group which has publish permissions on a channel, but not subscribe permissions,

the user will no be able to subscribe on the channel. Then, if an ACL entry is added on the channel for his specific username/host pair, with subscribe but no publish permissions, the user will then be able to both subscribe(from the non-group ACL permission), and publish (from the group ACL permission).

Realm Entitlements

Realm ACLs

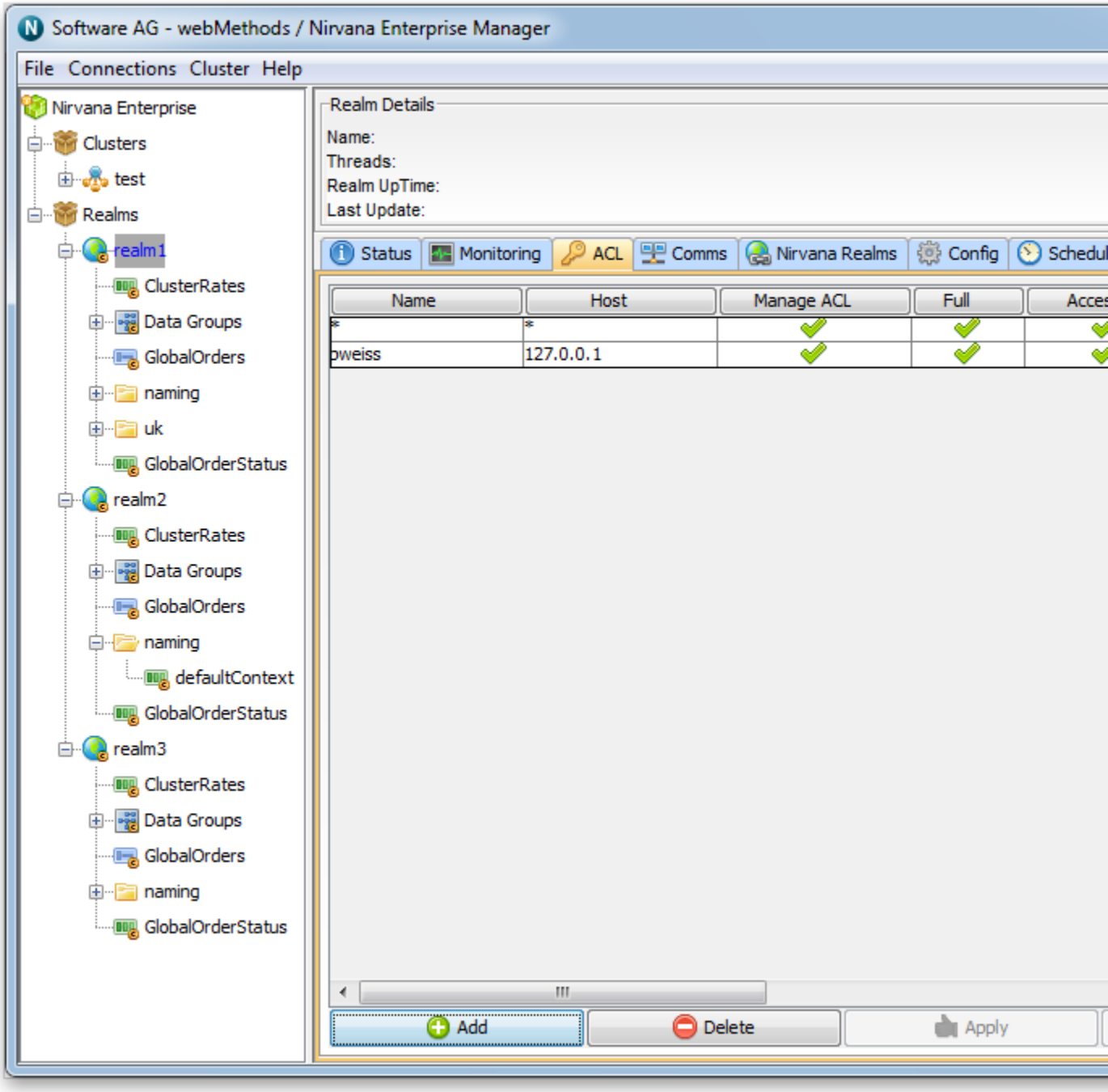
As mentioned in the security introduction (see "Using ACLs for role-based Security" on page 209), in order to perform operations within a Realm clients connecting to the realm must be given the correct entitlements.

In order for a client to connect to a Universal Messaging Realm server there must be a Realm ACL which allows them to do so. A Realm ACL contains a list of subjects and their entitlements (i.e. what operations they can perform within the realm).

Using the Enterprise manager, one can add to, remove or modify entries within a realm ACL.

ACLs can also be managed via the Universal Messaging administration API.

To view a Realm ACL, click on a realm node within the namespace of the Enterprise Manager, and select the 'ACL' tab. This will display the realm ACL and the list of subjects and their associated permissions for the realm. The following image displays and example of a realm ACL.



As you can see above, the realm ACL has a number of subject entries and operations that each subject is able to perform on the realm. The operations that can be performed on a realm are described below in the order in which they appear in the ACL panel above:

- Manage ACL - Allows the subject to get and manage the list of ACL entries.

  Note:

This permission is a combination of two permissions at the Administration API level. The boolean setModify() API function allows/denies permission to change an ACL value, and the boolean setList() API function allows/denies permission to access the current list of ACLs. If both of these functions return the value true, Manage ACL is allowed, otherwise Manage ACL is not allowed.

If the green check icon is displayed in the Manage ACL field, the corresponding two API functions for this field are set to true.

The value of this permission cannot be changed in the Enterprise Manager.

- Full - Has complete access to the secured object
- Access - Can actually connect to this realm
- Configure - Can set run time parameters on the realm
- Channels - Can add/delete channels on this realm
- Realm - Can add / remove realms from this realm
- Admin API - Can use the nAdminAPI package
- Manage DataGroups - Can add / remove data groups from this realm
- Pub DataGroups - Can publish to data groups (including default) on this realm
- Own DataGroups - Can add / delete publish to data groups even when they were not created by the user

The green check icon shows that a subject is permitted to perform the operation. For example, the subject *@* is shown as having no permissions for this realm. The minimum requirement for a client to use a realm is the 'Access' privilege. Without this privilege for the *@* subject, any Universal Messaging client attempting to connect, whose subject does not appear in the ACL list, will not be able to establish a session with the Realm Server.

In order to modify the permissions for a subject, you simply need to click on the cell in the ACL table for the subject and the operation you wish to modify permissions for. For example, if you want to grant the *@* user the 'Access' realm privilege, you would simply click on the *@* row at the column labelled 'access'. This would turn the cell from blank to a green check icon.

After making any changes, you then need to click on the 'Apply' button which will notify the Realm Server of the ACL change.

Any ACL changes that are made by other Enterprise Manager users, or from any programs using the Universal Messaging Admin API to modify ACLs will be received by all other Enterprise Managers. This is because ACL changes are automatically sent to all Universal Messaging Admin API clients, the Enterprise Manager being one of those clients.

Any changes made to a realm ACL where the realm is part of a cluster will be replicated to all other cluster realms.

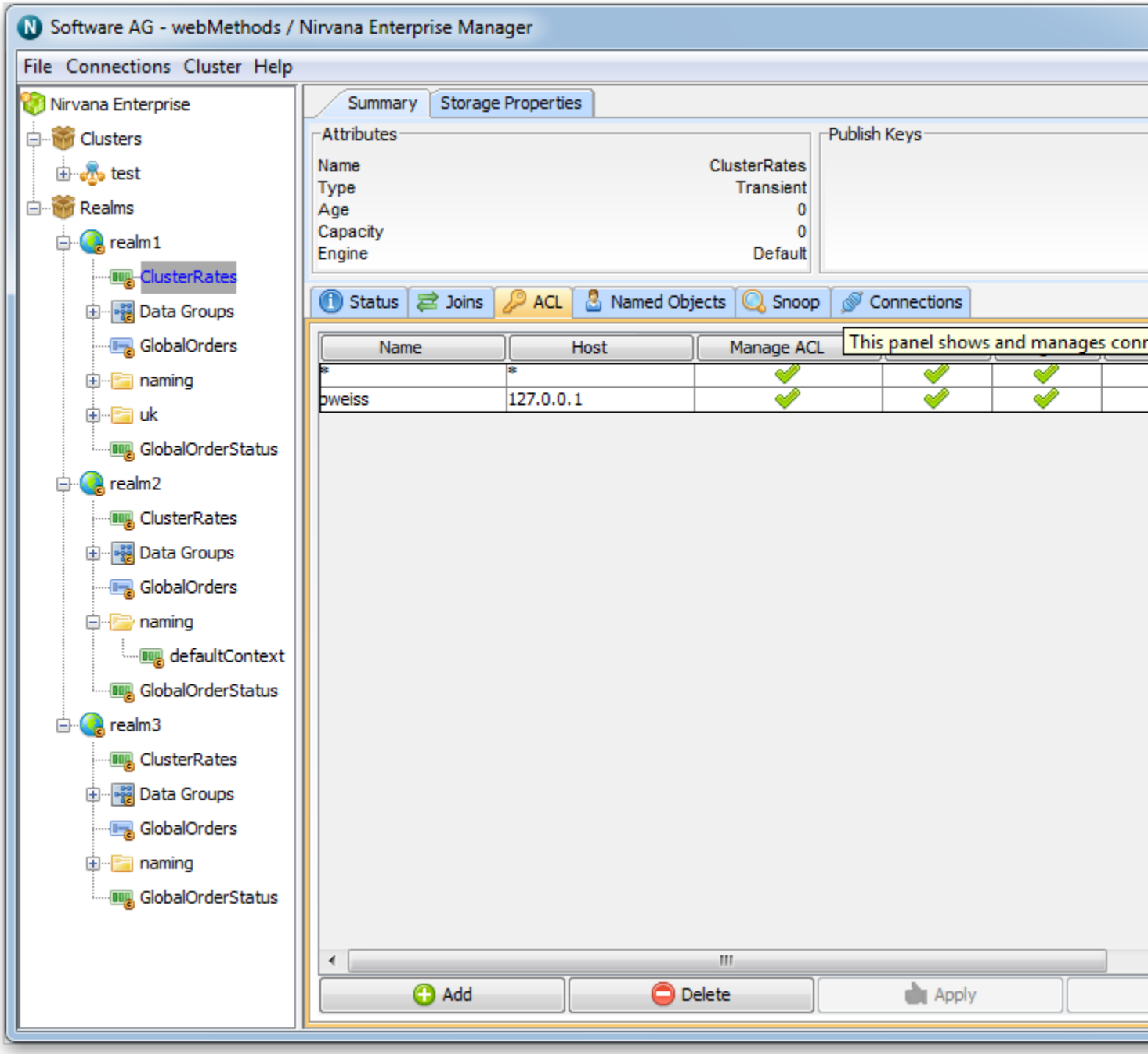
Channel Entitlements

Channel ACLs

Once clients have established a session with a Universal Messaging Realm server, and they have successfully been authenticated and the subject has the correct user entitlements, in order to perform operations on channel objects, the correct entitlements must be granted to the subject on the required channels. Each channel has an associated ACL that contains a list of subjects and a set of privileges the subject is given for operations on the channel.

Using the Enterprise Manager, one can add to, remove or modify entries within the channel ACL.

To view a channel ACL, click on a channel node within the namespace of the Enterprise Manager, and select the 'ACL' tab. This will display the channel ACL and the list of subjects and their associated permissions for the channel. The following image displays and example of a channel ACL.



As you can see above, the channel ACL has a number of subject entries and operations that each subject is able to perform on the channel. The operations that can be performed on a channel are described below in the order in which they appear in the ACL panel above:

- Manage ACL - Allows the subject to get and manage the list of ACL entries.

  Note:

This permission is a combination of two permissions at the Administration API level. The boolean setModify() API function allows/denies permission to change an ACL value, and the boolean setList() API function allows/denies permission to access the current list of ACLs. If both of these functions return the value true, Manage ACL is allowed, otherwise Manage ACL is not allowed.

If the green check icon is displayed in the Manage ACL field, the corresponding two API functions for this field are set to true. If you remove the green check icon, this sets the corresponding two API functions for this field to false.

- Full - Has complete access to the secured object
- Purge - Can delete events on this channel
- Subscribe - Can subscribe for events on this channel
- Publish - Can publish events to this channel
- Named - Can the user connect using a named (durable) subscriber

The green check icon shows that a subject is permitted to perform the operation. For example, if there is a subject *@* with only subscribe permissions for this channel, this means that any client who has successfully established a session and has obtained a reference to this channel within their application code can only subscribe to the channel and read events.

In order to modify the permissions for a subject, you simply need to click on the cell in the ACL table for the subject and the operation you wish to modify permissions for. For example, if you want to remove the subscribe permission for the *@* subject you would simply click on the *@* row at the column labelled 'subscribe'. This would turn the cell from blank to a green check icon. This would also ensure that only those subjects listed in the ACL and with sufficient privileges, would be able to perform any operations on the channel.

After making any changes, you then need to click on the 'Apply' button which will notify the Realm Server of the ACL change for that channel.

Any ACL changes that are made by other Enterprise Manager users, or from any programs using the Universal Messaging Admin API to modify ACLs will be received by all other Enterprise Managers. This is because ACL changes are automatically sent to all Universal Messaging Admin API clients, the Enterprise Manager being one of those clients.

Any changes made to a channel ACL where the channel is a cluster channel will be replicated to all other instances of the cluster channel in all other cluster realms.

Queue Entitlements

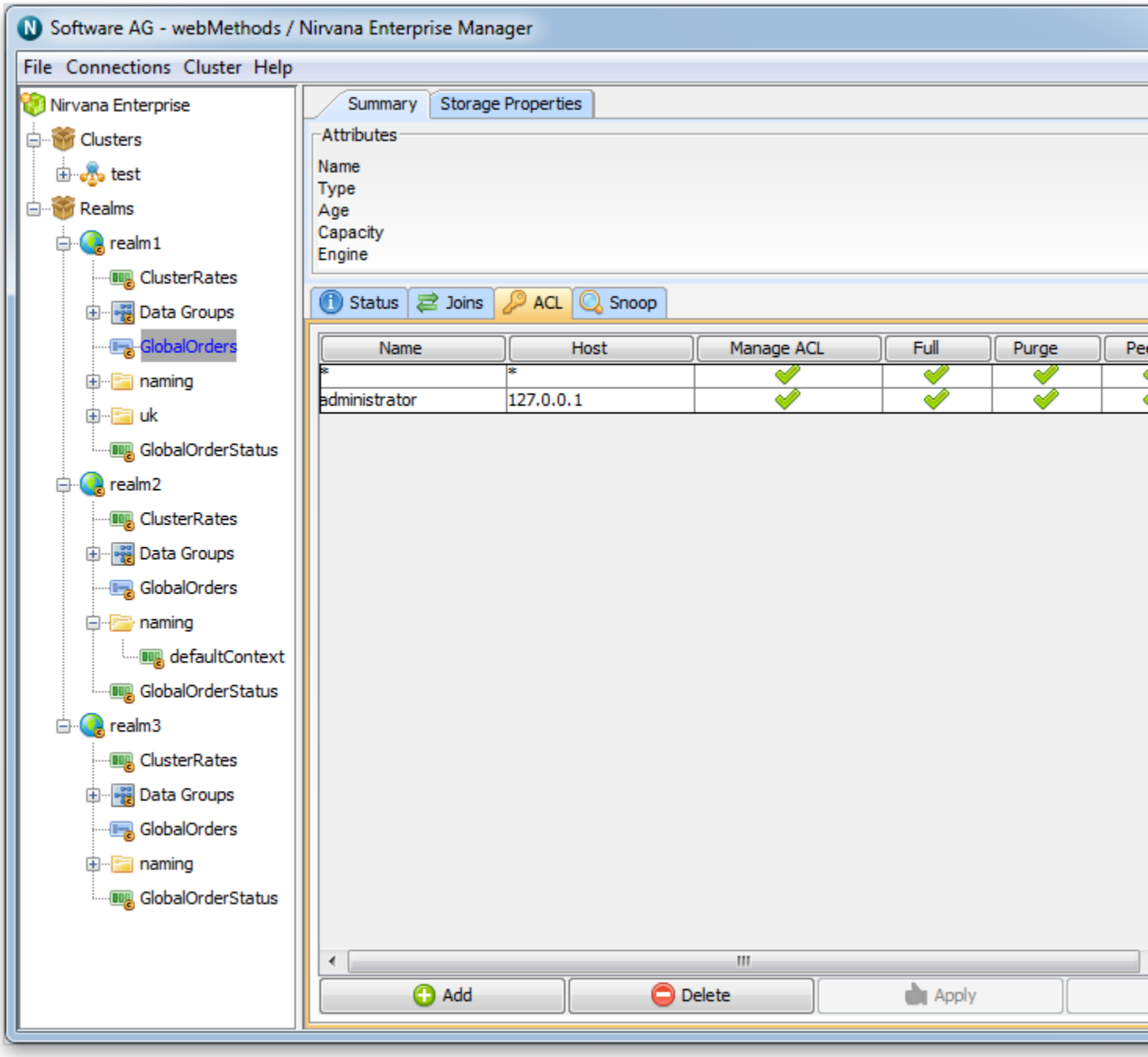
Queue ACLs

Once clients have established a session with a Universal Messaging Realm server, and they have successfully been authenticated and the subject has the correct user entitlements, in order to perform operations on queue objects, the correct entitlements must be granted to the subject on

the required queue. Each queue has an associated ACL that contains a list of subjects and a set of privileges the subject is given for operations on the queue.

Using the Enterprise Manager, one can add to, remove or modify entries within the queue ACL.

To view a queue ACL, click on a queue node within the namespace of the Enterprise Manager, and select the 'ACL' tab. This will display the queue ACL and the list of subjects and their associated permissions for the queue. The following image displays and example of a queue ACL.



As you can see above, the queue ACL has a number of subject entries and operations that each subject is able to perform on the queue. The operations that can be performed on a queue are described below in the order in which they appear in the acl panel above:

- Manage ACL - Allows the subject to get and manage the list of ACL entries.

  Note:

  This permission is a combination of two permissions at the Administration API level. The boolean setModify() API function allows/denies permission to change an ACL value, and the boolean setList() API function allows/denies permission to access the current list of ACLs. If both of these functions return the value true, Manage ACL is allowed, otherwise Manage ACL is not allowed.

  If the green check icon is displayed in the Manage ACL field, the corresponding two API functions for this field are set to true. If you remove the green check icon, this sets the corresponding two API functions for this field to false.

- Full - Has complete access to the secured object
- Purge - Can delete events on this channel
- Peak - Can snoop this queue (non destructive read)
- Push - Can publish events to this queue
- Pop - Can Consume events on this queue (destructive read)

The green check icon shows that a subject is permitted to perform the operation. For example, the subject *@* is shown as having only peek permissions for this queue. This means that any client who has successfully established a session and has obtained a reference to this queue within their application code can only subscribe to the queue and read events.

In order to modify the permissions for a subject, you simply need to click on the cell in the ACL table for the subject and the operation you wish to modify permissions for. For example, if I wanted remove the peek permission for the *@* subject I would simply click on the *@* row at the column labelled 'peek'. This would turn the cell from blank to a green check icon. This would also ensure that only those subjects listed in the ACL and with sufficient privileges, would be able to perform any operations on the queue.

After making any changes, you then need to click on the 'Apply' button which will notify the Realm Server of the ACL change for that queue.

Any ACL changes that are made by other Enterprise Manager users, or from any programs using the Universal Messaging Admin API to modify ACLs will be received by all other Enterprise Managers. This is because ACL changes are automatically sent to all Universal Messaging Admin API clients, the Enterprise Manager being one of those clients.

Any changes made to a channel ACL where the queue is a cluster queue will be replicated to all other instances of the cluster queue in all other cluster realms.

Interface VIA Rules

Each interface defined within a Universal Messaging Realm server can have an associated ACL list, known as a VIA list.

The VIA list enables list of users to be defined who are entitled to connect to the Universal Messaging realm using a specific protocol 'via' a specific interface.

If for example, a realm has an HTTP (nhp) interface running on port 10000, and we also want a sockets (nsp) interface running on port 15000, and you want all external clients to connect using the nhp interface, and all internal clients to connect using the nsp interface, this can be achieved by providing the nhp and nsp interfaces with a list of subjects that are able to connect via the different interfaces.

This ensures that any user that tries to connect via the nsp interface who is not part of the nsp interface VIA list but exists in the nhp via list will be rejected and will not be able to establish a connection via nsp. The same will apply for the nhp interface. Alternatively, by simply adding a list of via entries to the nhp interface (and leaving the nsp via list empty), any user trying to connect via nsp interface who is found in any other interface via list will be rejected. This allows you to tie specific users to specific interfaces.

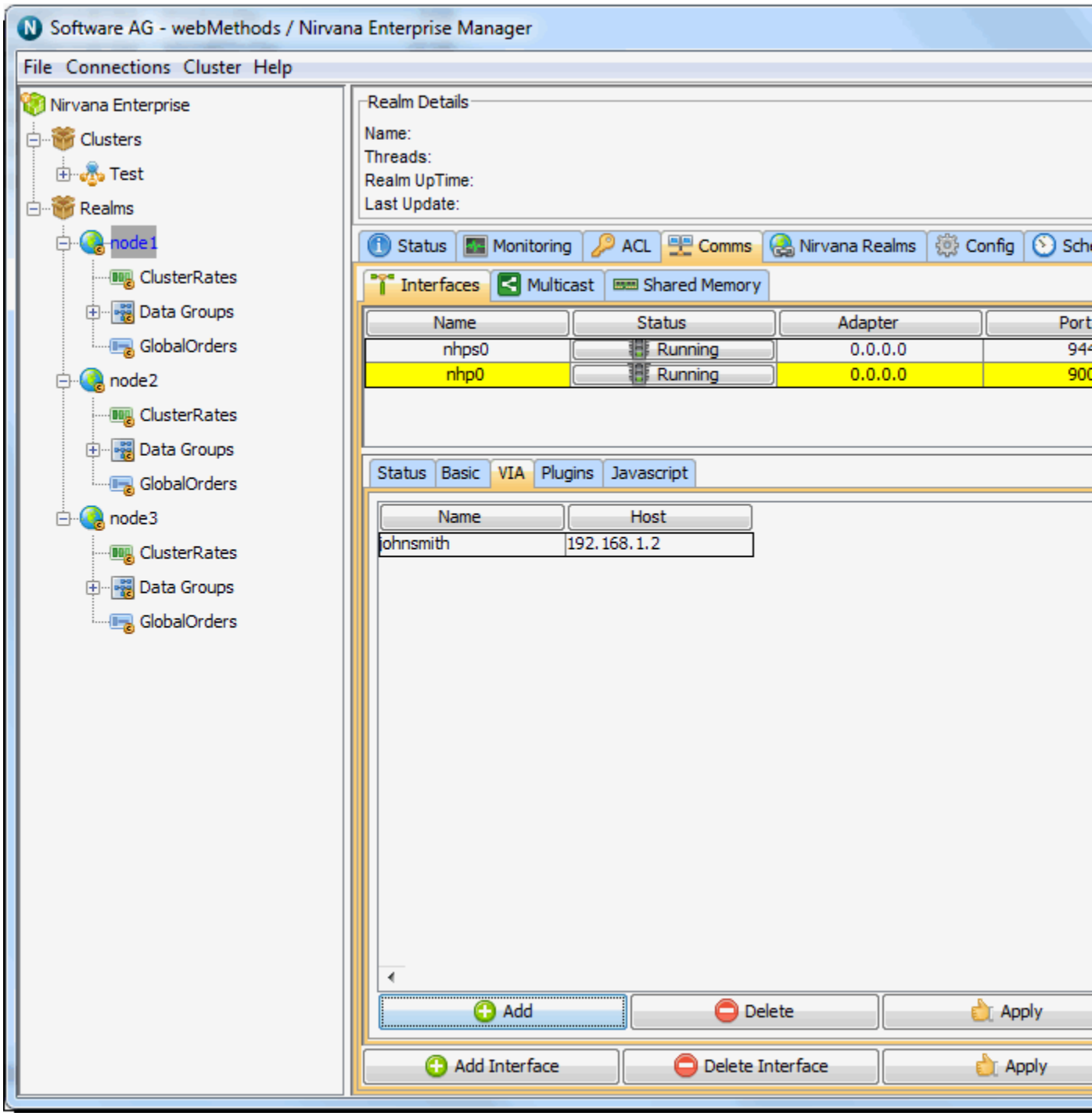
The default behaviour for all interfaces is that when no VIA lists exist on any defined interfaces, all users can connect on any interface (Realm ACLs permitting, see "Realm Entitlements" on page 214). When a user subject exists on an interface, that subject cannot use any other interface other than the one they are listed in.

This is an extra level of security that allows administrators of Realm Servers to define a strict approach to who can connect to the realm via specific protocols. This is particularly useful if for example you run many services on a single Universal Messaging realm server and wish to ensure that specific clients / groups of clients are using completely separate interfaces.

Interface ACL (VIA List)

In order to view the VIA list for an interface, select the realm where the interface is running, and then select the 'Interfaces' tab in the Enterprise Manager. From the interface list for the realm, select the interface from the table of interfaces, and choose the tab labelled 'VIA' from the bottom of the interface panel. The image below shows the result of an acl entry being added to the default socket interface running on port 9000. By adding this entry, the user johnsmith@192.168.1.2 can only use the nsp0 interface which is using the sockets protocol on port 9000.

As with all Universal Messaging ACLs wildcards are fully supported so that for example, *@192.168.1.2 or johnsmith@* are both relevant enforceable VIA rules.



Interface VIA entries can be added to by clicking on the 'Add' button from the VIA panel and entering the subject. Entries can be removed by selecting the entry and clicking the 'Delete' button.

Any changes to the interface VIA list will not take effect at the server until the 'Apply' button has been clicked on the VIA panel. Changes can also be disregarded without updating the server by clicking on the 'Cancel' button on the VIA list panel.

Scheduling

Overview

Universal Messaging provides a sophisticated scheduling engine that enables tasks to be executed as server-side scripts on a Realm Server at specific times or when certain conditions occur within the realm. This enables realm servers to automate important tasks, enabling them to self-manage without the need for intervention by administrators or externally scheduled tasks. The scripts consist of initial tasks, triggered tasks and / or calendar tasks.

Administrators of Universal Messaging Realm servers can provide scripts that outline the conditions and tasks to be performed which are then interpreted by the server. The server converts the scripts into the actual tasks to be completed, and executes them under the correct conditions.

This section guides you through the basic tasks that can be executed by a Realm Server, time based scheduling and conditional triggers, as well as how to write, modify and deploy scheduling scripts.

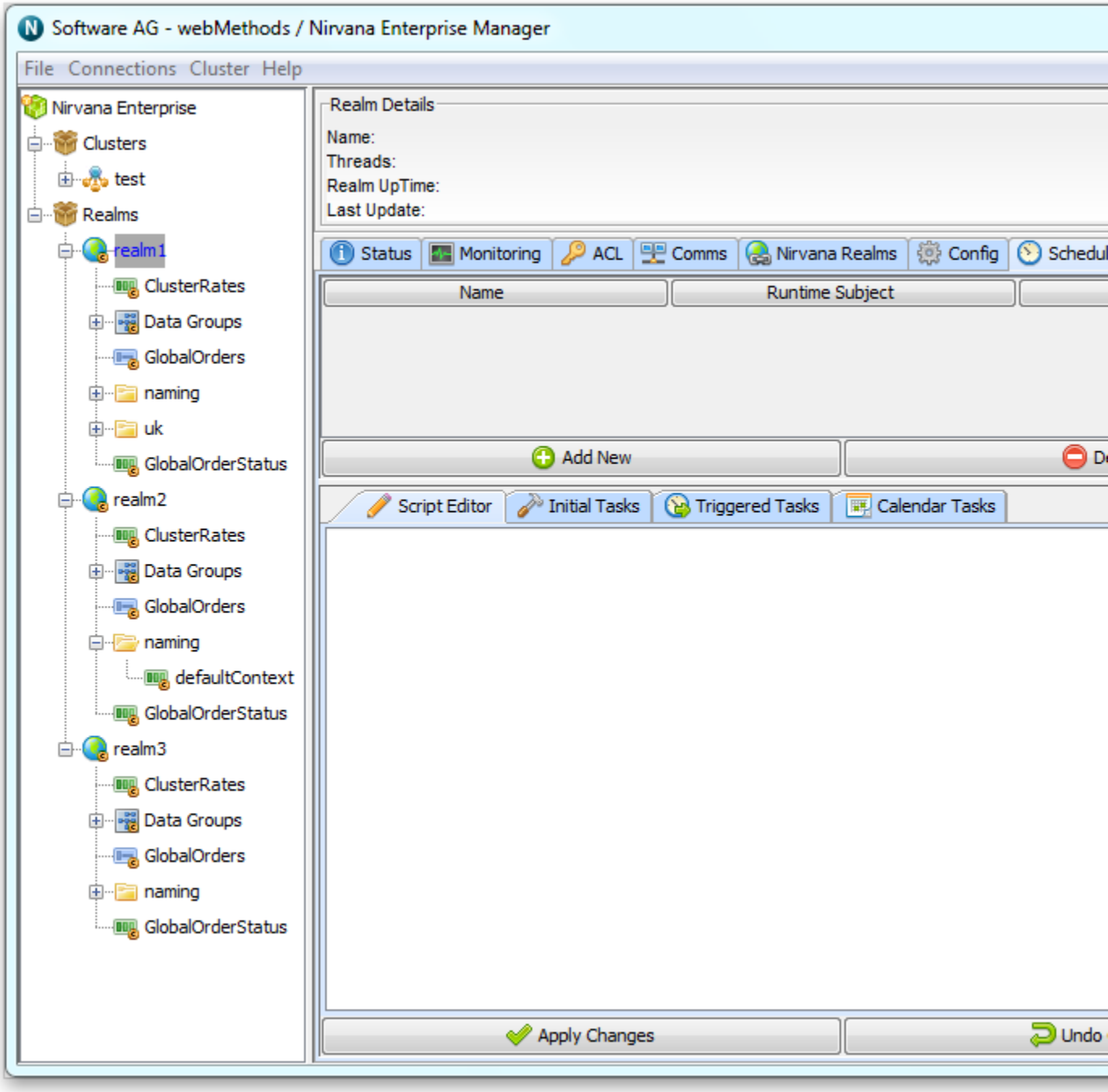
Note:
The Scheduler feature of the Enterprise Manager is deprecated in Universal Messaging version 10.2 and will be removed in a subsequent release.

Creating and Editing Scheduler Scripts

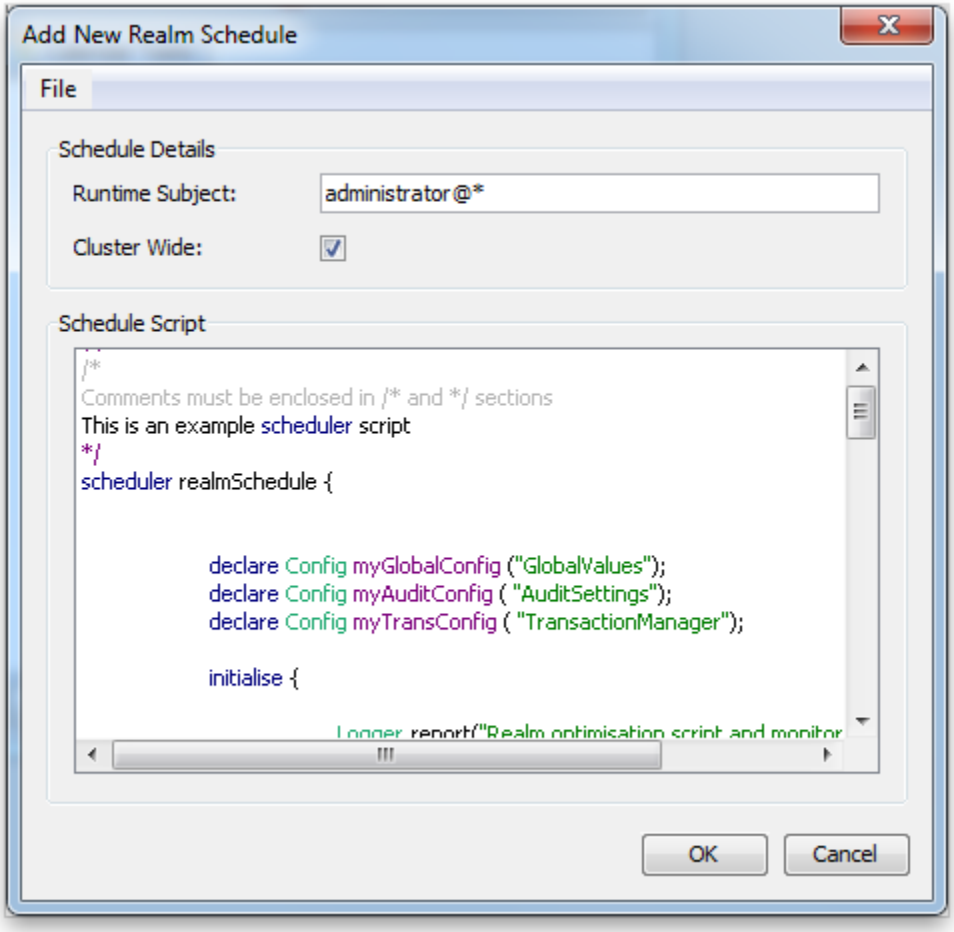
The Universal Messaging Enterprise Manager provides a scheduler panel that enables the user to view, add, delete and edit scheduler scripts. To view the scheduler panel, select the realm from the namespace and click on the 'Scheduler' tab.

The scheduler panel displays all scripts that have been deployed to the server within a table. This table shows the name of the schedule, as defined within the script, the user name of the person the script will be executed using (i.e. the user name of the Enterprise Manager user who deployed the script, as well as whether the script is to be deployed cluster wide (i.e. to all realms within the cluster node).

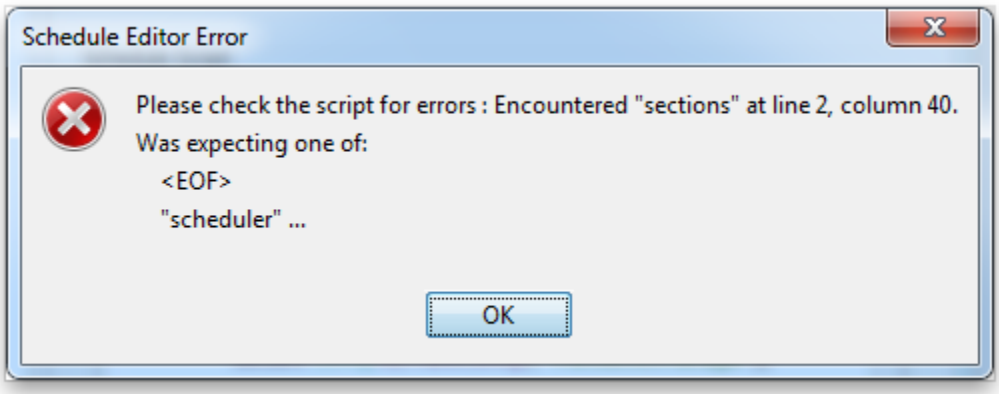
The image below shows the scheduler panel with no scheduling entries.



The button labelled 'Add New' when clicked will display a dialog containing a script editing panel that has been designed to assist with the creation of scheduling scripts. The scheduling grammar is discussed in more detail in the writing scripts section (see "Scheduling Script Language Summary" on page 232), as well as the calendar, triggers and tasks sections (see "Calendar Schedules (Time-based Triggers)" on page 237, "Conditional Triggers for Executing Tasks" on page 239 and "Scheduling Tasks" on page 250). The image below shows the script editor panel.



Once your script is complete, in order to deploy the schedule to the server, you need to click on the 'OK' button. Once clicked, if there are any errors or problems with the script, you will be presented with a dialog similar to the image below.

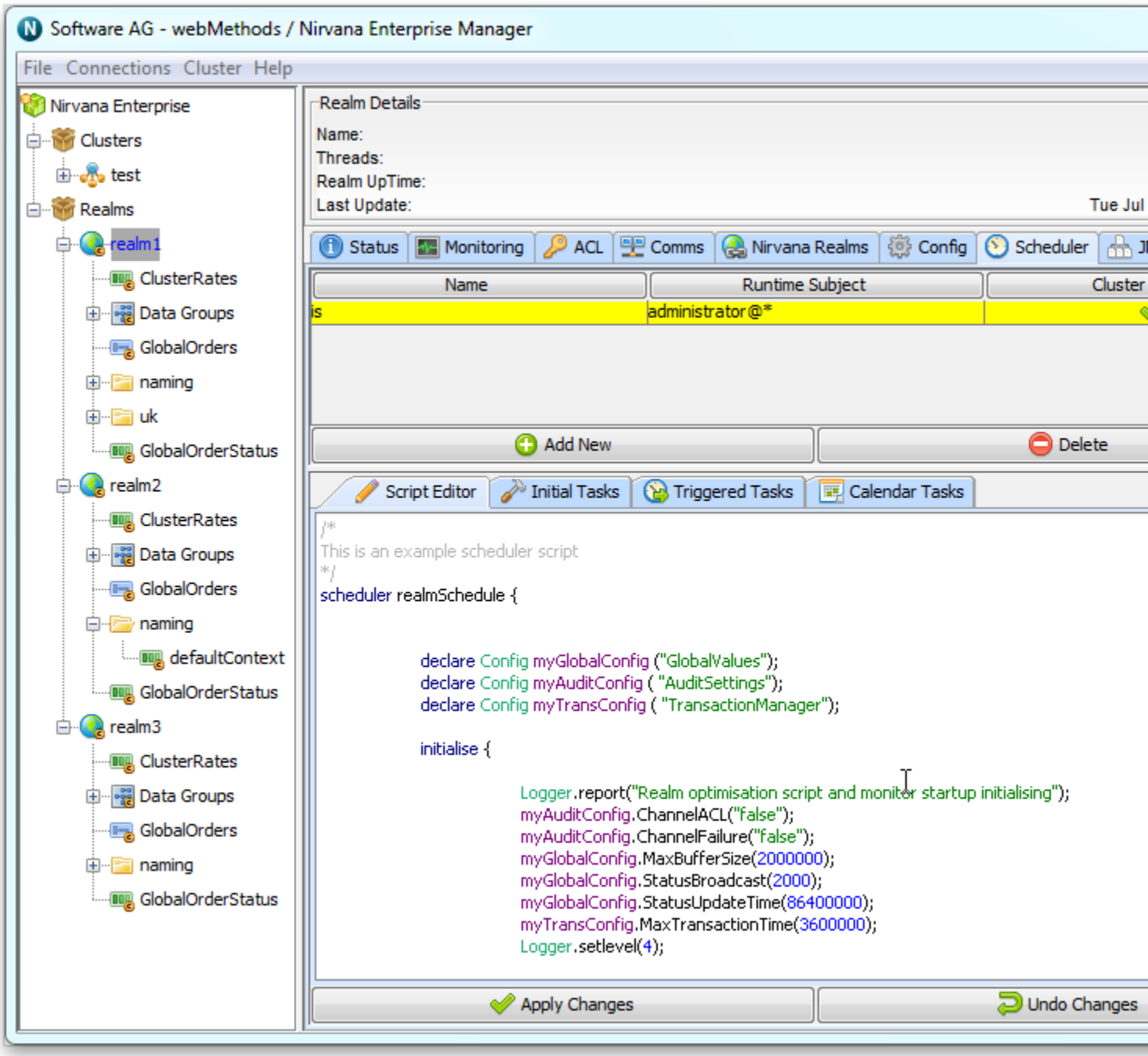


If the script does not contain any errors, the script editor panel will close and the new scheduler script will then appear within the scheduler table. Clicking on the newly created scheduler within the table will enable you to delete, vew and edit the schedule. The image below shows the newly created scheduler script once selected. There are 4 scheduler panels available for each scheduler

selected from the table. Each panel is represented by a tab on the bottom half of the main scheduler panel. Each of these panels is discussed below.

Script Editor Panel

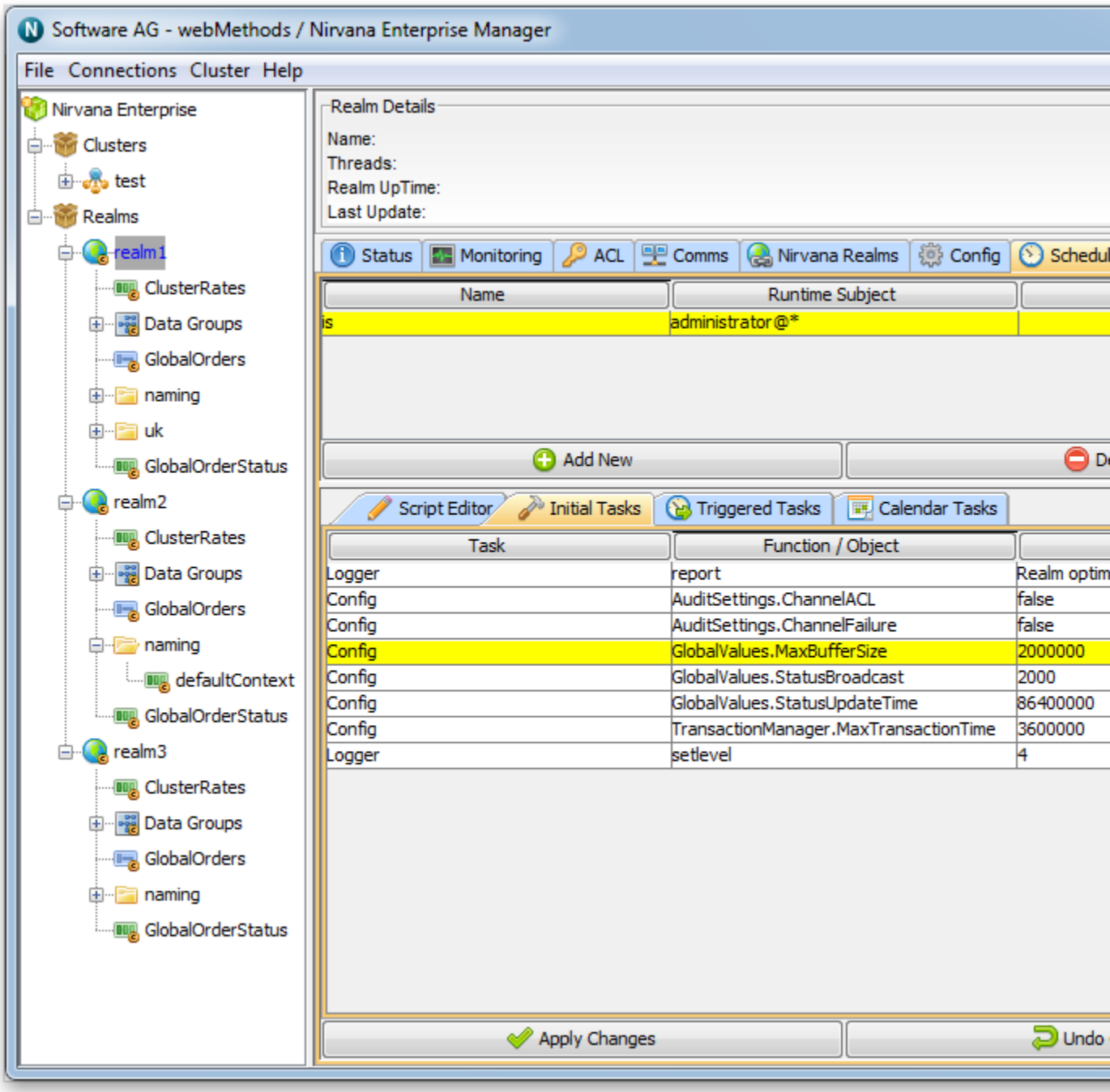
The script editor panel is denoted by the tab labelled 'Script Editor' and provides the same editor view found when you first add a new script. This panel is a simple editor pane that enables you to modify the scheduler triggers and tasks. The image below shows this panel selected from the available tabs.



Initial Tasks Panel

The initial tasks panel is denoted by the tab labelled 'Initial Tasks' and represents those tasks defined within the initialisation section of the scheduler script. Each initial task is represented as a row in a table with 3 columns. Column 1 labelled 'Task' is the task object (see "Scheduling Tasks" on page 250). Column 2 labelled 'Function / Object' represents the details of the task, fo example, if the task was to purge a channel, column 2 would show 'purge'. Column 3 labelled

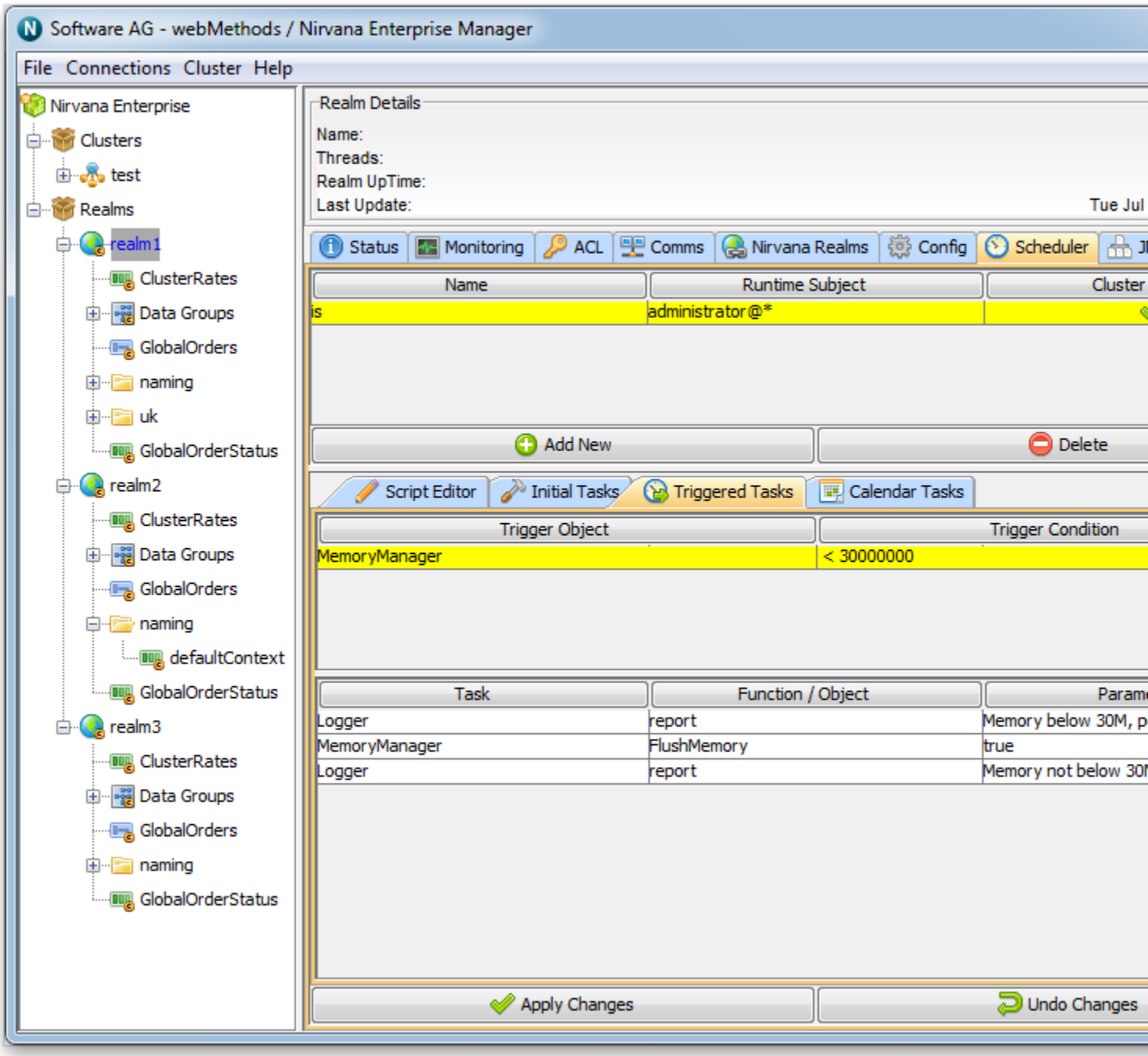
'Parameter' shows any parameters listed in the scheduler script for the given task. The image below shows an example of the Initial Tasks tab being selected.



Triggered Tasks

The triggered tasks panel is denoted by the 'Triggered Tasks' tab. This panel displays those tasks that are triggered based on some conditional triggers. Each conditional trigger is shown as a row in the table within this panel. Selecting a trigger from this table will then display the tasks to be

executed when this trigger is fired. Each task is shown in a table similar to that found in the Initial Tasks panel. The image below shows the triggered tasks panel being selected.



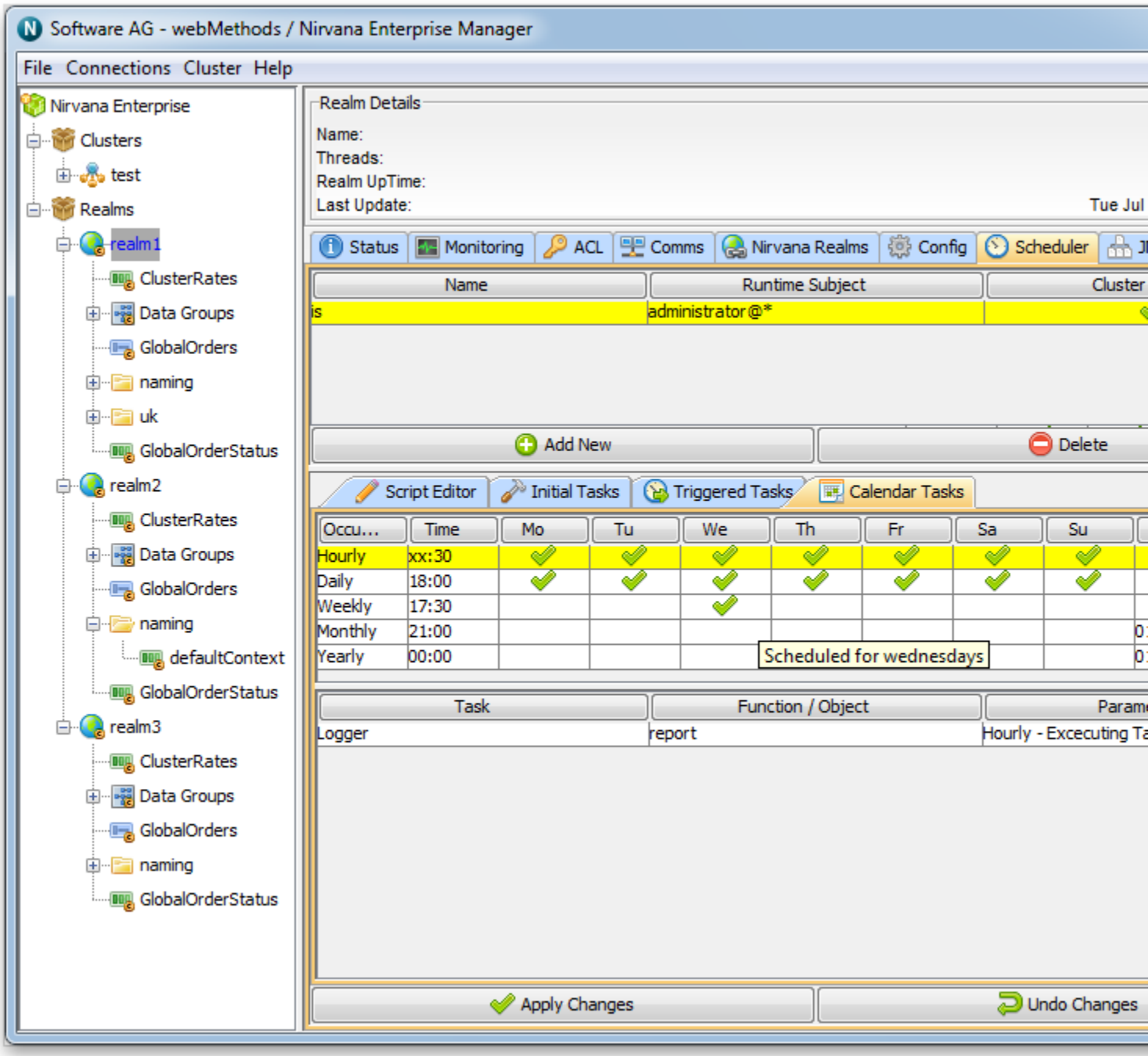
Calendar Tasks

The final panel is the calendar tasks panel and is denoted by the 'Calendar Tasks' tab. This panel shows the tasks that are scheduled to run at specific times. Each calendar task is shown as a row within a table. This table has a total of 11 columns. The first 2 columns show the frequency and

time. The frequency is either 'Hourly', 'Daily', 'Weekly', 'Monthly' or 'Yearly' and the time is specified as HH:MM. For hourly schedules, the HH (hours) will be displayed as XX which denotes every hour.

Columns 3 to 9 represent which days of the week the task will run, starting from Monday ('Mo'). A green circle means the task will run on that day. The last 2 columns represent the Day and Month the task will run.

Selecting one of the rows in the table will display the actual tasks that will be executed in a similar table to that found in the triggered tasks panel. The image below shows the calendar tasks panel with a task selected.



Any changes made to the schedule within the script editor panel can either be deployed to the realm server by clicking on the 'Apply Changes' button or discarded by clicking on the 'Undo Changes' button.

Schedule entries can be deleted from the server by selecting them from the main scheduler table and clicking on the 'Delete' button.

Scheduling Script Language Summary

Universal Messaging scheduling works by interpreting scripts written using a simple grammar. Administrators of realms can deploy as many scheduling scripts as they wish to each Realm Server.

This section will cover the basic structure of a Universal Messaging scheduling script, and then show how to write a script and deploy it to the Realm Server.

Follow the links below to view the guide for each of these:

- "Scheduling Grammar" on page 233
- "Declarations" on page 235
- "Initial Tasks" on page 235
- "Every Clause" on page 236
- "When Clause" on page 236
- "Else Clause" on page 237

Scheduling Grammar

The grammar for scheduling scripts is extremely simple to understand. The script must conform to a predefined structure and include elements that map to the grammar expected by the Realm Server Scheduler Engine.

In its simplest form the Universal Messaging scheduler syntax starts with the command *'scheduler'*. This tells the parser that a new scheduler task is being defined. This is followed by the name of the scheduler being defined, this is a user defined name. For example:

```
scheduler myScheduler {
}
```

Within this structure, triggers and tasks are defined. A task is the actual operation the server will perform, and it can be executed at a certain time or frequency, or when a condition occurs. Within the scheduler context the following verbs can be used to define tasks to be executed.

- **declare** : Used to define the name of a trigger for later user
- **initialise** : Is the first thing run when a scheduler is started (also run when the realm server starts up)
- **every** : Used to define a time/calendar based event
- **when** : Used to define a conditional trigger and the list of tasks to execute when it fires
- **else** : Used after a conditional trigger that will fire if the condition evaluates to false

The following shows the basic grammar and structure of a scheduling script.

```
/*
Comment block
*/
scheduler <User defined Name> {
declare <TRIGGER_DECLARATION>+
initialise {
```

```
<TASK_DECLARATION>+
}
/*
Time based tasks
*/
every <TIME_EXPRESSION> {
<TASK_DECLARATION>+
}
when ( <TRIGGER_EXPRESSION> ) {
<TASK_DECLARATION>+
} else {
<TASK_DECLARATION>+
}
```

where :

- TRIGGER_DECLARATION ::= <TRIGGER> <NAME> (<TRIGGER_ARGUMENT_LIST>)
- TRIGGER ::= Valid trigger. Learn more about triggers at "Conditional Triggers for Executing Tasks" on page 239.
- TRIGGER_ARGUMENT_LIST ::= Valid comma separated list of arguments for the trigger
- TASK_DECLARATION ::= Valid task. Learn more about tasks at "Scheduling Tasks" on page 250.
- TRIGGER_EXPRESSION ::=

  <TRIGGER_EXPRESSION><LOGICAL_OPERATOR><TRIGGER_EXPRESSION> | <TRIGGER> | <NAME> <COMPARISON_OPERATOR> <VALUE>
- TIME_EXPRESSION ::=

  <HOURLY_EXPRESSION> | <DAILY_EXPRESSION> | <WEEKLY_EXPRESSION> | <MONTHLY_EXPRESSION> | <YEARLY_EXPRESSION>
- HOURLY_EXPRESSION ::= <MINUTES>
- DAILY_EXPRESSION ::= <HOUR> <COLON> <MINUTES>
- WEEKLY_EXPRESSION ::= <DAYS_OF_WEEK> <SPACE> <HOUR> <COLON> <MINUTES>
- MONTHLY_EXPRESSION ::= <DAY_OF_MONTH> <SPACE> <HOUR> <COLON> <MINUTES>
- YEARLY_EXPRESSION ::= <DAY_OF_MONTH> <HYPHEN> <MONTH> <SPACE> <HOUR> <COLON> <MINUTES>
- MINUTES ::= Minutes past the hour, i.e. a value between 00 and 59
- HOUR ::= Hour of the day, i.e. a value between 00 and 23
- DAYS_OF_WEEK ::=

  <DAY_OF_WEEK> | <DAY_OF_WEEK> <SPACE> <DAY_OF_WEEK>
- DAY_OF_WEEK ::= Mo | Tu | We | Th | Fr | Sa | Su

- DAY_OF_MONTH ::= Specific day of the month to perform a task, i.e. a value between 01 and 28
- MONTH ::= The month of the year, JAN, FEB, MAR etc.
- NAME ::= The variable name for a trigger
- COMPARISON_OPERATOR ::= > | => | < | <= | == | !=
- LOGICAL_OPERATOR ::= AND | OR
- COLON ::= The ":" character
- SPACE ::= The space character
- HYPHEN ::= The "-" character
- + ::= indicates that this can occur multiple times
- VALUE ::= Any valid string or numeric value.

Declarations

The declarations section of the script defines any triggers and assigns them to local variable names. The grammar notation defined above specifies that the declaration section of a schedule script can contain multiple declarations of triggers. For example, the following declarations section would be valid based on the defined grammar:

```
declare Config myGlobalConfig ("GlobalValues");
declare Config myAuditConfig ( "AuditSettings");
declare Config myTransConfig ( "TransactionManager");
```

The above declarations define 3 variables that refer to the the Config trigger. The declared objects can be used in a time based trigger declaration, conditional triggers and to perform tasks on.

Initialise

The initialise section of the schedule script defines what tasks are executed straight away by the server when the script is deployed. These initial tasks are also executed every time the Realm Server is started. An example of a valid initialise section of a schedule script is shown below:

```
initialise {
Logger.report("Realm optimisation script and monitor startup initialising");
myAuditConfig.ChannelACL("false");
myAuditConfig.ChannelFailure("false");
myGlobalConfig.MaxBufferSize(2000000);
myGlobalConfig.StatusBroadcast(2000);

myTransConfig.MaxTransactionTime(3600000);
Logger.setlevel(4);
}
```

The example above ensures that each time a server starts, the tasks declared are executed. Using the variables defined in the declarations section, as well as the Logger task, the server will always ensure that the correct configuration values are set on the server whenever it starts.

Every Clause

The every clause defines those tasks that are executed at specific times and frequencies as defined in the grammar above. Tasks can be executed every hour at a specific time pas the hour, every day at a certain time, every week on one or more days at specific times or day, every month on a specific day of the month and a specific day of the year.

The grammar above defines how to declare an every clause. Based on this grammar the following examples demonstrate how to declare when to perform tasks :

```
Hourly Example (Every half past the hour, log a message to the realm server log)
every 30 {
Logger.report("Hourly - Executing Tasks");
}
Daily Example (Every day at 18:00, perform maintenance on the customerSales channel
)
every 18:00 {
Logger.report("Daily - performing maintenance");
Store.maintain("/customer/sales");
}
Weekly Example (Every week, on sunday at 17:30, purge the customer sales channel)
every Su 17:30 {
Logger.report("Weekly - Performing Purge");
Store.purge("/customer/sales");
}
Monthly Example (Every 1st of the month, at 21:00, stop all interfaces and start them
 again)
every 01 21:00 {
Logger.report("Monthly - Stopping interfaces and restarting");
Interface.stopAll();
Interface.startAll();
}
Yearly Example (Every 1st of the January, at 00:00, stop all interfaces and start them
 again)
every 01-Jan 00:00 {
Logger.report("Yearly - Stopping interfaces and restarting");
Interface.stopAll();
Interface.startAll();
}
```

When Clause

The when clause defines a trigger that evaluates a specific value and executes a task if the evaluation result is 'true'. The grammar for the when clause is defined above. The following example shows a valid when clause :

```
when (MemoryManager.FreeMemory < 30000000) {
Logger.report("Memory below 30M, performing some clean up");
FlushMemory(true);
}
```

The above example will trigger the Realm Server JVM to call garbage collection when the amount of free memory drops to below 30MB.

Else Clause

The else clause defines an alternative action to the when clause if the when clause evaluates to 'false'. The grammar for the else clause is defined above. The following example shows a valid when clause :

```
when (MemoryManager.FreeMemory < 30000000) {
Logger.report("Memory below 30M, performing some clean up");
FlushMemory(true);
} else {
Logger.report("Memory not below 30M, no clean up required");
}
```

The above example will trigger the Realm Server JVM to call garbage collection when the amount of free memory drops to below 30MB.

To view a sample scheduling script, see the section "Scheduler Examples" on page 261.

Calendar Schedules (Time-based Triggers)

Calendar schedules are triggered at specific times, either hourly, daily, weekly, monthly or yearly. Each calendar trigger is declared using the 'every' keyword. For basic information on the grammar for calendar schedules, please read the section on time based triggers in the writing scripts help file (see "Scheduling Script Language Summary" on page 232). The calendar, or time based triggers are signified by using the 'every' keyword. The values entered after the keyword represent hourly, daily, weekly, monthly or yearly frequency that the defined tasks will be executed. See "Hourly Triggers" on page 237, "Daily Triggers" on page 238, "Weekly Triggers" on page 238, "Monthly Triggers" on page 238, "Yearly Triggers" on page 239.

This section will describe in more detail the variations of the calendar trigger grammar.

Hourly Triggers

Hourly triggers have the simplest grammar. The value after the 'every' keyword represents the minutes past the hour that the tasks will be executed. For example, specifying '00' means that the tasks are executed on the hour, every hour. If you specify '30' the tasks will be executed at half past the hour every hour:

```
/*
 Execute every hour on the hour
*/
every 00 {
}
```

```
/*
 Execute every hour at half past the hour
*/
every 30 {
}
```

Daily Triggers

Daily triggers are executed every day at a specific time. The time of day is written as 'HH:MM', in a 24 hour clock format and represents the exact time of day that the tasks are executed. For example, specifying '18:00' means the tasks are executed every day at 6pm. If you specify '08:30' the tasks will be executed at 8.30am every morning.

```
/*
 Execute day at 6pm
*/
every 18:00 {
}
```

```
/*
 Execute day at 8.30am
*/
every 08:30 {
}
```

Weekly Triggers

Weekly triggers are executed on specific days of the week at a specific time, in the format 'DD HH:MM' . The days are represented as a 2 character string being one of : Su; Mo; Tu; We; Th; Fr; Sa, and you can specify more than one day. The time of day is written as 'HH:MM', in a 24 hour clock format and represents the exact time on each given day that the tasks are executed. For example, specifying 'Fr 18:00' means the tasks are executed every friday at 6pm. If you specify 'Mo Tu We Th Fr 18:30' the tasks will be executed every week day at 6.30pm.

```
/*
 Execute every friday at 6pm
*/
every Fr 18:00 {
}
```

```
/*
 Execute every week day at 6.30pm
*/
every Mo Tu We Th Fr 18:30 {
}
```

Monthly Triggers

Monthly triggers are executed on a specific day of the month at a specific time, in the format 'DD HH:MM' . The day is represented as a 2 digit number between 1 and 28. The time of day is written as 'HH:MM', in a 24 hour clock format and represents the exact time on the given day of the month that the tasks are executed. For example, specifying '01 18:00' means the tasks are executed on the 1st of every month at 6pm.

```
/*
 Execute on the first of every month at 6pm
*/
every 01 18:00 {
}
```

Yearly Triggers

Yearly triggers are executed on a specific day and month of the year at a specific time, in the format 'DD-MMM HH:MM'. The day of the month is represented as a 2 digit number between 1 and 31, and the month is represented as a 3 character string being one of : Jan; Feb; Mar; Apr; May; Jun; Jul; Aug; Sep; Oct; Nov; Dec. The time of day is written as 'HH:MM', in a 24 hour clock format and represents the exact time on the given day and month of the year that the tasks are executed. For example, specifying '01-Jan 18:00' means the tasks are executed on the 1st of January every year at 6pm.

```
/*
 Execute on the first of january every year at 6pm
 */
every 01-Jan 18:00 {
}
```

Conditional Triggers for Executing Tasks

Conditional triggers execute tasks when specific conditions occur. Each defined trigger has a number of attributes that can be used as part of the trigger expression and evaluated to determine whether the tasks are executed. For basic information on the grammar for conditional triggers, please read the section on conditional triggers in the writing scripts help file (see "Scheduling Script Language Summary" on page 232). The conditional triggers are signified by using the 'when' keyword. The expression entered after the keyword represent the trigger object(s) and the values to be checked against.

This section describes in detail the triggers that are available and how to use them within a trigger expression :

- "Trigger Expressions" on page 240
- "Store Triggers" on page 240
- "Interface Triggers" on page 241
- "Memory Triggers" on page 242
- "Realm Triggers" on page 242
- "Cluster Triggers" on page 243
- "Counter Triggers" on page 243
- "Timer Triggers" on page 244
- "Config Triggers" on page 244

To view examples of scheduling scripts, see "Scheduler Examples" on page 261.

Trigger Expressions

A trigger expression is constructed from the definition of the trigger object(s) to be evaluated and the values that will be used in the comparison. The trigger used in the expression can be either the actual trigger object, or the declared name of the trigger from the declarations section of the script (see "Scheduling Script Language Summary" on page 232). Multiple triggers can be used in the expression using conditional operators (AND | OR).

For example, the following expression can be used to evaluate when a Realm's Interface accept threads are exhausted 5 times. When this happens, the accept threads will be increased by 10. This schedule will continually monitor the state of the interface and self-manage the accept threads so the realm server is always able to accept connections from clients.

```
scheduler realmInterfaceSchedule {
declare Interface myNHP ("nhp0");
declare Counter myCounter("myExhaustedThreads");
when (myNHP.idleThreads == 0) {
Logger.report("NHP0 Interface has no idles Threads");
myCounter.inc();
}
when (myCounter >= 5) {
Logger.report("Increasing the accept thread count on NHP0");
myNHP.threads("+10");
myCounter.reset();
}
}
```

The above schedule will monitor the number of times the accept threads are exhausted and when the counter trigger hits 5 times, the number of threads will be increased by 10.

The next section will describe the available trigger objects and the available triggers on those objects that can be used within

Store Triggers - Channel / Queue based triggers

Store triggers are declared using the following syntax as an example:

```
declare Store myChannel("/customer/sales");
```

The table below lists those triggers that can be evaluated on a Store object, such that the trigger expression will look like :

```
when (myChannel.connections > 100) {
}
```

Trigger Object	Parameters	Description
connections	None	Trigger on the number of connections for the channel or queue
freeSpace	None	Trigger on the amount of free space available in the store

Trigger Object	Parameters	Description
		(used space - size of all purged events)
usedSpace		Trigger on the amount of used space available in the store (size of all event on disk or memory)
numOfEvents	None	Trigger on the number of events on the channel / queue
filter	Valid filter String	Trigger when an event that matches the filter is published to the channel / queue

Interface Triggers - Universal Messaging Interface based triggers

Interface triggers are declared using the following syntax as an example:

```
declare Interface myNHP("nhp0");
```

The table below lists those triggers that can be evaluated on an Interface object, such that the trigger expression will look like :

```
when (myNHP.connections > 100) {
}
```

Trigger Object	Parameters	Description
connections	None	Trigger on the number of connections for the interface
authentication	None	Trigger on the average authentication time for clients on an interface
failedConnections	None	Trigger on the number of failed authentication attempts
exhaustedTime	None	Trigger on the average amount of time the interface accept thread pool has been exhausted
idleThreads	None	Trigger on the number of idle interface accept pool threads
exhaustedCount	None	Trigger on the number of times an interface accept thread pool is exhausted (i.e. idle == 0)

Trigger Object	Parameters	Description
state	None	Trigger when an interface is in a certain state

MemoryManager Triggers - Universal Messaging JVM Memory Management based triggers

MemoryManager triggers are declared using the following syntax as an example:

```
declare MemoryManager mem;
```

The table below lists those triggers that can be evaluated on the memory management object, such that the trigger expression will look like :

```
when (mem.freeMemory < 1000000) {
}
```

Trigger Object	Parameters	Description
freeMemory	None	Trigger when the realm server's JVM has a certain amount of free memory
totalMemory	None	Trigger when the realm server's JVM has a certain amount of total memory
outOfMemory	None	Trigger when the realm server JVM runs out of memory

Realm Triggers - Universal Messaging Realm based triggers

Realm triggers are declared using the following syntax as an example:

```
declare Realm myRealm("productionmaster");
```

The table below lists those triggers that can be evaluated on the realm object, such that the trigger expression will look like :

```
when (realm.connections > 1000) {
}
```

Trigger Object	Parameters	Description
connections	None	Trigger when the realm server current connections reaches a certain number

Trigger Object	Parameters	Description
eventsSentPerSecond	None	Trigger when the realm server's events per second sent rate reaches a certain value
eventsReceivedPerSecond	None	Trigger when the realm server's events per second sent received reaches a certain value

Cluster Triggers - Universal Messaging Cluster based triggers

Cluster triggers are declared using the following syntax as an example, assuming a cluster is made up of 4 realms:

```
declare Cluster myNode1("realm1");
declare Cluster myNode2("realm2");
declare Cluster myNode3("realm3");
declare Cluster myNode4("realm4");
```

The table below lists those triggers that can be evaluated on the cluster object, such that the trigger expression will look like :

```
when ( Cluster.nodeOnline("realm1") == true ){
}
```

Trigger Object	Parameters	Description
hasQuorum	None	Trigger when cluster has quorum == true or false
isMaster	None	Trigger when a cluster realm is voted master
nodeOnline	None	Trigger when a cluster realm is online or offline

Counter Triggers - Counter value based triggers

Counter triggers allow you to keep a local count of events occurring with the Universal Messaging scheduler engine. The values of the Counters can be incremented / decremented and reset within the tasks section of a trigger declaration. Counter triggers are declared using the following syntax as an example:

```
declare Counter counter1 ("myCounter");
```

The counter trigger can be evaluated by referencing the Counter object itself, such that the trigger expression will look like :

```
when ( counter1 > 5) {
}
```

Timer Triggers - Timer based triggers

Timer triggers allow you to start a timer that will keep track of how long (in seconds) it has been running and then evaluate the running time within a trigger expression. Time triggers are declared using the following syntax as an example:

```
declare Timer reportTimer ("myTimer");
```

The timer trigger can be evaluated by referencing the timer object itself, such that the trigger expression will look like :

```
when ( reportTimer == 60 ) {
}
```

Config Triggers - Universal Messaging configuration triggers

Config triggers refer to any of the configuration values available in the Config panel for a realm. Any configuration value can be used as part of a trigger expression. Config triggers are declared using the following syntax as an example (the example refers to the 'Global Values' configuration group):

```
declare Config myGlobal ("Global Values");
```

The table below lists the triggers that can be evaluated on a Config object, such that the task expression will look like :

```
when (myGlobal.MaxNoOfConnections == -1) {
}
```

Trigger Object	Parameters	Description
Global Values		
SchedulerPoolSize	None	The number of threads assigned to the scheduler
MaxNoOfConnections	None	Sets the maximum concurrent connections to the server, -1 indicates no restriction
StatusBroadcast	None	The number of ms between status events being published
NHPTimeout	None	The number of milliseconds the server will wait for client authentication
NHPScanTime	None	The number of milliseconds that the server will wait before scanning for client timeouts

Trigger Object	Parameters	Description
StampDictionary	None	Place Universal Messaging details into the dictionary (true/false)
ExtendedMessageSelector	None	If true, allows the server to use the extended message selector syntax (true/false)
ConnectionDelay	None	When the server has exceeded the connection count, how long to hold on to the connection before disconnecting
SupportVersion2Clients	None	Allow the server to support older clients (true/false)
SendRealmSummaryStats	None	If true sends the realms status summary updates (true/false)
Audit Settings		
RealmMaintenance	None	Log to the audit file any realm maintenance activity
InterfaceManagement	None	Log to the audit file any interface management activity
ChannelMaintenance	None	Log to the audit file any channel maintenance activity
QueueMaintenance	None	Log to the audit file any queue maintenance activity
ServiceMaintenance	None	Log to the audit file any service maintenance activity
JoinMaintenance	None	Log to the audit file any join maintenance activity
RealmSuccess	None	Log to the audit file any successful realm interactions
ChannelSuccess	None	Log to the audit file any successful channel interactions
QueueSuccess	None	Log to the audit file any successful queue interactions

Trigger Object	Parameters	Description
ServiceSuccess	None	Log to the audit file any successful realm interactions
JoinSuccess	None	Log to the audit file any successful join interactions
RealmFailure	None	Log to the audit file any unsuccessful realm interactions
ChannelFailure	None	Log to the audit file any unsuccessful channel interactions
QueueFailure	None	Log to the audit file any unsuccessful queue interactions
ServiceFailure	None	Log to the audit file any unsuccessful service interactions
JoinFailure	None	Log to the audit file any unsuccessful join interactions
RealmACL	None	Log to the audit file any unsuccessful realm acl interactions
ChannelACL	None	Log to the audit file any unsuccessful channel acl interactions
QueueACL	None	Log to the audit file any unsuccessful queue acl interactions
ServiceACL	None	Log to the audit file any unsuccessful service acl interactions
Client Timeout Values		
EventTimeout	None	The amount of ms the client will wait for a response from the server
DisconnectWait	None	The maximum amount of time to wait when performing an operation when disconnected before throwing session not connected exception

Trigger Object	Parameters	Description
TransactionLifeTime	None	The default amount of time a transaction is valid before being removed from the tx store
KaWait	None	The amount of time the client will wait for keep alive interactions between server before acknowledging disconnected state
LowWaterMark	None	The low water mark for the connection internal queue. When this value is reached the outbound internal queue will again be ready to push event to the server
HighWaterMark	None	The high water mark for the connection internal queue. When this value is reached the internal queue is temporarily suspended and unable to send events to the server. This provides flow control between publisher and server.
QueueBlockLimit	None	The maximum number of milliseconds a queue will have reached HWM before notifying listeners
QueueAccessWaitLimit	None	The maximum number of milliseconds it should take to gain access to a queue to push events before notifying listeners
QueuePushWaitLimit	None	The maximum number of milliseconds it should take to gain access to a queue and to push events before notifying listeners
Cluster Config		
HeartBeatInterval	None	Heart Beat interval in milliseconds

Trigger Object	Parameters	Description
EventsOutStanding	None	Number of events outstanding
Event Storage		
CacheAge	None	The time in ms that cached events will be kept in memory for
ThreadPoolSize	None	The number of threads allocated to perform the management task on the channels
ActiveDelay	None	The time in milliseconds that an active channel will delay between scans
IdleDelay	None	The time in milliseconds that an idle channel will delay between scans
Fanout Values		
ConcurrentUser	None	The number of client threads allowed to execute concurrently in the server
KeepAlive	None	The number of milliseconds between the server will wait before sending a heartbeat
QueueHighWaterMark	None	The number of events in a client output queue before the server stops sending events
QueueLowWaterMark	None	The number of events in the clients queue before the server resumes sending events
MaxBufferSize	None	The maximum buffer size that the server will accept
PublishDelay	None	How long to delay the publisher when subscribers queue start to fill, in milliseconds
PublishExpiredEvents	None	Publish expired events at server startup (true/false)

Trigger Object	Parameters	Description
JVM Management		
MemoryMonitoring	None	Number of milliseconds between monitoring memory usage on the realm
WarningThreshold	None	The memory threshold when the server starts to scan for objects to release
EmergencyThreshold	None	The memory threshold when the server starts to aggressively scan for objects to release
ExitOnDiskIOError	None	If true, the server will exit if it gets a I/O Exception
Join Config		
MaxEventsPerSchedule	None	Number of events that will be sent to the remote server in one run
MaxQueueSizeToUse	None	The maximum events that will be queued on behalf of the remote server
ActiveThreadPoolSize	None	The number of threads to be assigned for the join recovery
IdleThreadPoolSize	None	The number of threads to manage the idle and reconnection to remote servers
Logging Config		
fLoggerLevel	None	The server logging level
RecoveryDaemon		
ThreadPool	None	Number of threads to use for client recovery
EventsPerBlock	None	The number of events to send in one block

Trigger Object	Parameters	Description
TransactionManager		
MaxTransactionTime	None	Time in milliseconds that a transaction will be kept active
MaxEventsPerTransaction	None	The maximum number of events per transaction, a 0 indicates no limit
TTLThreshold	None	The minimum time in milliseconds, below which the server will not store the Transaction ID

Scheduling Tasks

Tasks are executed by either time based (calendar, see "Calendar Schedules (Time-based Triggers)" on page 237) or conditional triggers (see "Conditional Triggers for Executing Tasks" on page 239). There are a number of tasks that can be executed by the Universal Messaging Scheduling engine. Each task corresponds to a unit of work that performs an operation on the desired object within a Universal Messaging realm.

This section will discuss the available tasks that can be declared within a Universal Messaging scheduling script :

- "Task Expressions" on page 250
- "Store Tasks" on page 251
- "Interface Tasks" on page 252
- "Memory Tasks" on page 253
- "Counter Tasks" on page 253
- "Timer Tasks" on page 254
- "Config Tasks" on page 254

To view examples of scheduling scripts, see "Scheduler Examples" on page 261.

Task Expressions

Task expressions are comprised of the object on which you wish to perform the operation, and the required parameters. For more information on the grammar for task expressions, please see the section (see "Scheduling Script Language Summary" on page 232). The following sections will describe the task objects and the parameters required to perform them. The example below demonstrates both Interface, Logger and Counter tasks.

```
scheduler realmInterfaceSchedule {
declare Interface myNHP ("nhp0");
declare Counter myCounter("myExhaustedThreads");
when (myNHP.idleThreads == 0) {
Logger.report("NHP0 Interface has no idles Threads");
myCounter.inc();
}
when (myCounter >= 5) {
Logger.report("Increasing the accept thread count on NHP0");
myNHP.threads("+10");
myCounter.reset();
}
}
```

Store Tasks - Channel / Queue operations

Store tasks can be used by first of all declaring the desired object as in the following syntax:

```
declare Store myChannel("/customer/sales");
```

The table below lists those tasks available on a Store object, such that the task expression will look like :

```
when (myChannel.numOfEvents < 100) {
myChannel.maintain();
}
```

Task Object	Syntax	Description

maintain	`Store.maintain("*");`
	`Store.maintain("/customer/sales");`
	`myChannel.maintain();`
		Perform maintenance on a channel so that any purged events are removed from the channel or queue event store.

publish	`myChannel.publish("Byte array data",` `"tag", "key1=value1:key2:value2" );`	Publish an event to the channel / queue, using the given byte array, event tag and event dictionary values.

purge	`myChannel.purge();`
	`myChannel.purge(0, 100000);`
	`myChannel.purge(0, 10000,`
	`"key1 = 'value1'");`
		Purge all events on a channel, or events between a start and end eid, or using a purge filter.

createChannel	`myChannel.createChannel(0, 0, "P");`	Create the channel using the name it was declared as, and the ttl, capacity and type specified in the parameters
createQueue	`myChannel.createQueue(0, 0, "P");`	Create the queue using the name it was declared as, and

Task Object	Syntax	Description
		the ttl, capacity and type specified in the parameters

Interface Tasks - Universal Messaging Interface operations

Interface tasks are operations that can be performed on all interfaces or individually declared interfaces. To declare an interface use the following syntax as an example:

```
declare Interface myNHP("nhp0");
```

The table below lists those tasks that can be executed on an Interface object, such that the task expression will look like :

```
when (myNHP.connections > 1000) {
myNHP.threads("+10");
}
```

Task Object	Syntax	Description
stop	`myNHP.stop();` `Interface.stop("nhp0");`	Stop the interface
start	`myNHP.start();` `Interface.start("nhp0");`	Start The interface
stopAll	`Interface.stopAll();`	Stop all interfaces on the realm
startAll	`Interface.startAll();`	Start all interfaces on the realm
authTime	`myNHP.authTime(20000);` `myNHP.authTime("+10000");`	Set the interface authentication time to a value, or increase / decrease it by a value.
backlog	`myNHP.backlog(200);` `myNHP.backlog("+100");`	Set the interface backlog time to a value, or increase / decrease it by a value.
autoStart	`myNHP.autoStart("true");` `myNHP.autoStart("false");`	Set whether an interface is automatically started when the realm is started.
advertise	`myNHP.advertise("true");` `myNHP.advertise("false");`	Set whether an interface is available to

Task Object	Syntax	Description
		clients using the admin API.
certificateValidation	`myNHP.certificateValidation("true");` `myNHP.certificateValidation("false");`	Set whether an interface (SSL) requires clients to provide a certificate to authenticate.
threads	`myNHP.threads(10);` `myNHP.threads("+10");`	Set the interface accept threads to a value or increase / decrease it by a value.

MemoryManager Triggers - Universal Messaging JVM Memory Management operations

MemoryManager triggers are declared using the following syntax as an example:

```
declare MemoryManager mem;
```

The table below lists those triggers that can be evaluated on the memory management object, such that the task expression will look like :

```
when (mem.freeMemory < 1000000) {
}
```

Task Object	Syntax	Description
flush	`mem.flush(true);` `mem.flush(false);`	Cause the JVM to call garbage collection, and optionally release used memory

Counter Tasks - Counter tasks

Counter tasks allow you to increment, decrement, set and reset a local counter within the Universal Messaging scheduling engine. Counter tasks are declared using the following syntax as an example:

```
declare Counter counter1 ("myCounter");
```

The counter task can be executing by referencing the Counter object itself, and calling one of a number of available tasks. The basic counter task expression will look like :

```
when ( counter1 > 5) {
counter1.reset();
}
```

The table below shows the tasks that can be executed on the Counter task.

Task Object	Syntax	Description
dec	counter1.dec()	Decrement the counter by 1
inc	counter1.inc()	Increment the counter by 1
set	counter1.set(5)	Set the counter to a value
reset	counter1.reset()	Reset the counter to 0

Timer Tasks - Timer operations

Timer tasks allow you to start, stop and reset the timer. Time tasks are declared using the following syntax as an example:

```
declare Timer reportTimer ("myTimer");
```

The timer task can be executed by referencing the timer object itself, such that the task expression will look like :

```
when ( reportTimer == 60 ) {
reportTimer.reset();
}
```

The table below shows the tasks that can be executed on the Counter task.

Task Object	Syntax	Description
start	reportTimer.start()	Start the timer
inc	reportTimer.stop()	Stop the timer
reset	reportTimer.reset()	Reset the timer

Config Tasks - Channel / Queue based triggers

Config tasks can be used to set any configuration value available in the Config panel for a realm. Any configuration value can be used as part of a trigger task expression. Config tasks are declared using the following syntax as an example (below example refers to the 'Global Values' configuration group):

```
declare Config myGlobal ("Global Values");
declare Config myAudit ("Audit Settings");
declare Config myClientTimeout ("Client Timeout Values");
declare Config myCluster ("Cluster Config");
declare Config myEventStorage ("Event Storage");
declare Config myFanout ("Fanout Values");
declare Config myJVM ("JVM Management");
declare Config myJoinConfig ("Join Config");
declare Config myLoggingConfig ("Logging Config");
declare Config myRecovery ("RecoveryDaemon");
declare Config myTXMgr ("TransactionManager");
```

The table below lists those tasks that can be evaluated on a config object, such that the task expression will look like :

```
when (myGlobal.MaxNoOfConnections == -1) {
myGlobal.MaxNoOfConnections(1000);
}
```

Trigger Object	Syntax	Description
Global Values		
SchedulerPoolSize	myGlobal.SchedulerPoolSize(10);	The number of threads assigned to the scheduler
MaxNoOfConnections	myGlobal.MaxNoOfConnections(-1);	Sets the maximum concurrent connections to the server, -1 indicates no restriction
StatusBroadcast	myGlobal.StatusBroadcast(2000);	The number of ms between status events being published
NHPTimeout	myGlobal.NHPTimeout(2000);	The number of milliseconds the server will wait for client authentication
NHPScanTime	myGlobal.NHPScanTime(10000);	The number of milliseconds that the server will wait before scanning for client timeouts
StampDictionary	myGlobal.StampDictionary(true);	Place Universal Messaging details into the dictionary (true/false)
ExtendedMessageSelector	myGlobal.ExtendedMessageSelector (true);	If true, allows the server to use the extended message selector syntax (true/false)
ConnectionDelay	myGlobal.ConnectionDelay(2000);	When the server has exceeded the connection count, how long to hold on to the connection before disconnecting

Trigger Object	Syntax	Description
SupportVersion2Clients	myGlobal.SupportVersion2Clients (true);	Allow the server to support older clients (true/false)
SendRealmSummaryStats	myGlobal.SendRealmSummaryStats (true);	If true sends the realms status summary updates (true/false)
Audit Settings		
RealmMaintenance	myAudit.RealmMaintenance (false);	Log to the audit file any realm maintenance activity
InterfaceManagement	myAudit.InterfaceManagement (false);	Log to the audit file any interface management activity
ChannelMaintenance	myAudit.ChannelMaintenance (false);	Log to the audit file any channel maintenance activity
QueueMaintenance	myAudit.QueueMaintenance (false);	Log to the audit file any queue maintenance activity
ServiceMaintenance	myAudit.ServiceMaintenance (false);	Log to the audit file any service maintenance activity
JoinMaintenance	myAudit.JoinMaintenance(false);	Log to the audit file any join maintenance activity
RealmSuccess	myAudit.RealmSuccess(false);	Log to the audit file any successful realm interactions
ChannelSuccess	myAudit.ChannelSuccess(false);	Log to the audit file any successful channel interactions
QueueSuccess	myAudit.QueueSuccess(false);	Log to the audit file any successful queue interactions

Trigger Object	Syntax	Description
ServiceSuccess	myAudit.ServiceSuccess(false);	Log to the audit file any successful realm interactions
JoinSuccess	myAudit.JoinSuccess(false);	Log to the audit file any successful join interactions
RealmFailure	myAudit.RealmFailure(false);	Log to the audit file any unsuccessful realm interactions
ChannelFailure	myAudit.ChannelFailure(false);	Log to the audit file any unsuccessful channel interactions
QueueFailure	myAudit.QueueFailure(false);	Log to the audit file any unsuccessful queue interactions
ServiceFailure	myAudit.ServiceFailure(false);	Log to the audit file any unsuccessful service interactions
JoinFailure	myAudit.JoinFailure(false);	Log to the audit file any unsuccessful join interactions
RealmACL	myAudit.RealmACL(false);	Log to the audit file any unsuccessful realm acl interactions
ChannelACL	myAudit.ChannelACL(false);	Log to the audit file any unsuccessful channel acl interactions
QueueACL	myAudit.QueueACL(false);	Log to the audit file any unsuccessful queue acl interactions
ServiceACL	myAudit.ServiceACL(false);	Log to the audit file any unsuccessful service acl interactions
Client Timeout Values		
EventTimeout	myClientTimeout.EventTimeout (10000);	The amount of ms the client will wait for a

Trigger Object	Syntax	Description
		response from the server
DisconnectWait	myClientTimeout.DisconnectWait (30000);	The maximum amount of time to wait when performing an operation when disconnected before throwing session not connected exception
TransactionLifeTime	myClientTimeout.TransactionLifeTime (10000);	The default amount of time a transaction is valid before being removed from the tx store
KaWait	myClientTimeout.KaWait(10000);	The amount of time the client will wait for keep alive interactions between server before acknowledging disconnected state
LowWaterMark	myClientTimeout.LowWaterMark (200);	The low water mark for the connection internal queue. When this value is reached the outbound internal queue will again be ready to push event to the server
HighWaterMark	myClientTimeout.HighWaterMark (500);	The high water mark for the connection internal queue. When this value is reached the internal queue is temporarily suspended and unable to send events to the server. This provides flow control between publisher and server.
QueueBlockLimit	myClientTimeout.QueueBlockLimit (5000);	The maximum number of milliseconds a queue will have reached HWM before notifying listeners

Trigger Object	Syntax	Description
QueueAccessWaitLimit	myClientTimeout.QueueAccessWaitLimit (10000);	The maximum number of milliseconds it should take to gain access to a queue to push events before notifying listeners
QueuePushWaitLimit	myClientTimeout.QueuePushWaitLimit (12000);	The maximum number of milliseconds it should take to gain access to a queue and to push events before notifying listeners
Cluster Config		
HeartBeatInterval	myCluster.HeartBeatInterval (60000);	Heart Beat interval in milliseconds
EventsOutStanding	myCluster.EventsOutStanding (10);	Number of events outstanding
Event Storage		
CacheAge	myEventStorage.CacheAge(360000);	The time in ms that cached events will be kept in memory for
ThreadPoolSize	myEventStorage.ThreadPoolSize(2);	The number of threads allocated to perform the management task on the channels
ActiveDelay	myEventStorage.ActiveDelay(1000);	The time in milliseconds that an active channel will delay between scans
IdleDelay	myEventStorage.IdleDelay(60000);	The time in milliseconds that an idle channel will delay between scans
Fanout Values		
ConcurrentUser	myFanout.ConcurrentUser(5);	The number of client threads allowed to

Trigger Object	Syntax	Description
		execute concurrently in the server
KeepAlive	myFanout.KeepAlive(60000);	The number of milliseconds between the server will wait before sending a heartbeat
QueueHighWaterMark	myFanout.QueueHighWaterMark(500);	The number of events in a client output queue before the server stops sending events
QueueLowWaterMark	myFanout.QueueLowWaterMark(200);	The number of events in the clients queue before the server resumes sending events
MaxBufferSize	myFanout.MaxBufferSize(1024000);	The maximum buffer size that the server will accept
PublishDelay	myFanout.PublishDelay(100);	How long to delay the publisher when subscribers queue start to fill, in milliseconds
PublishExpiredEvents	myFanout.PublishExpiredEvents(true);	Publish expired events at server startup (true/false)
Join Config		
MaxEventsPerSchedule	myJoinConfig.MaxEventsPerSchedule (200);	Number of events that will be sent to the remote server in one run
MaxQueueSizeToUse	myJoinConfig.MaxQueueSizeToUse (50);	The maximum events that will be queued on behalf of the remote server
ActiveThreadPoolSize	myJoinConfig.ActiveThreadPoolSize (4);	The number of threads to be assigned for the join recovery

Trigger Object	Syntax	Description
IdleThreadPoolSize	myJoinConfig.IdleThreadPoolSize (2);	The number of threads to manage the idle and reconnection to remote servers
Logging Config		
fLoggerLevel	myLoggingConfig.fLoggerLevel(4);	The server logging level
RecoveryDaemon		
ThreadPool	myRecovery.ThreadPool(5);	Number of threads to use for client recovery
EventsPerBlock	myRecovery.EventsPerBlock(300);	The number of events to send in one block
TransactionManager		
MaxTransactionTime	myTXMgr.MaxTransactionTime (1000);	Time in milliseconds that a transaction will be kept active
MaxEventsPerTransaction	myTXMgr.MaxEventsPerTransaction (1000);	The maximum number of events per transaction, a 0 indicates no limit
TTLThreshold	myTXMgr.TTLThreshold(1000);	The minimum time in milliseconds, below which the server will not store the Transaction ID

Scheduler Examples

Below is a list of example scheduling scripts that can help you become accustomed to writing Universal Messaging Scheduling scripts.

- "Generic Example" on page 262
- "Store Triggers" on page 263
- "Interface Triggers" on page 264
- "Memory Triggers" on page 264

- "Realm Triggers" on page 265
- "Cluster Triggers" on page 265
- "Counter Triggers" on page 266
- "Timer Triggers" on page 266
- "Config Triggers" on page 267

Universal Messaging Scheduling : Example Realm Script

```
/*
Comments must be enclosed in /* and */ sections
This is an example scheduler script
*/
scheduler realmSchedule {
 declare Config myGlobalConfig ("GlobalValues");
 declare Config myAuditConfig ( "AuditSettings");
 declare Config myTransConfig ( "TransactionManager");
 initialise {
 Logger.report("Realm optimisation script and monitor startup initialising");
 myAuditConfig.ChannelACL("false");
 myAuditConfig.ChannelFailure("false");
 myGlobalConfig.MaxBufferSize(2000000);
 myGlobalConfig.StatusBroadcast(2000);

 myTransConfig.MaxTransactionTime(3600000);
 Logger.setlevel(4);
 }
 every 30 {
 Logger.report("Hourly - Executing Tasks");
 }
 every 18:00 {
 Logger.report("Daily - performing maintenance");
 Store.maintain("/customer/sales");
 }
 every We 17:30 {
 Logger.report("Weekly - Performing Purge");
 Store.purge("/customer/sales");
 }
 every 01 21:00 {
 Logger.report("Monthly - Stopping interfaces and restarting");
 Interface.stopAll();
 Interface.startAll();
 }
 every 01-Jan 00:00 {
 Logger.report("Yearly - Stopping interfaces and restarting");
 Interface.stopAll();
 Interface.startAll();
 }
 when (MemoryManager.FreeMemory <30000000) {
 Logger.report("Memory below 30M, performing some clean up");
 MemoryManager.FlushMemory("true");
 } else {
 Logger.report("Memory not below 30M, no clean up required");
 }
}
```

Universal Messaging Scheduling : Store Triggers Example

```
scheduler myStore {
 declare Store myPubChannel("myChannel");
 declare Store myPubQueue("myQueue");
/*
Create the channels if they do not exist on the server
*/
 initialise{
 myPubChannel.createChannel( 0, 0, "P");
 myPubQueue.createQueue( 0, 0, "M");
 myPubChannel.publish("Data to store in the byte array", "tag info",
     "key1=value1:key2=value2" );
 myPubQueue.publish("Data to store in the byte array", "tag info",
     "key1=value1:key2=value2" );
 }
/*
At 4:30 each morning perform maintenance on the stores to release unused space
*/
 every 04:30 {
 myPubQueue.maintain();
 myPubChannel.maintain();
 }
/*
Every hour publish an event to the Channel
*/
 every 0 {
 myPubChannel.publish("Data to store in the byte array", "tag info",
     "key1=value1:key2:value2" );
 myPubQueue.publish("Data to store in the byte array", "tag info",
     "key1=value1:key2:value2" );
 }
/*
Every 1/2 hour purge the channels/queue
The purge takes 3 optional parameters
 StartEID
 EndEID
 Filter string
So it could be
 myPubChannel.purge(0, 100000);
or
 myPubChannel.purge(0, 10000, "key1 = 'value1'");
*/
 every 0 {
 myPubChannel.purge();
 myPubQueue.purge();
 }
/*
 When the number of events == 10 we purge the channel
*/
 when(myPubChannel.numOfEvents == 10){
 myPubChannel.purge();
 }
/*
 When the free space is greater then 60% perform maintenance
*/
 when(myPubChannel.freeSpace> 60){
 myPubChannel.maintain();
```

```
 }
/*
 When the number of connections on a channel reach 20 log an entry
*/
 when(myPubChannel.connections == 20){
 Logger.report("Reached 20 connections on the channel");
 }
/*
 Maintain all channels and queues at midnight every night
*/
 every 00:00 {
 Store.maintain("*");
 }
}
```

Universal Messaging Scheduling : Interface Triggers Example

```
scheduler realmInterfaceSchedule {
 declare Interface myNHP ("nhp0");
 declare Counter myCounter("myExhaustedThreads");
 when (myNHP.idleThreads == 0) {
 Logger.report("NHP0 Interface has no idles Threads");
 myCounter.inc();
 }
 when (myCounter>= 5) {
 Logger.report("Increasing the accept thread count on NHP0");
 myNHP.threads("+10");
 myCounter.reset();
 }
}
}
```

Universal Messaging Scheduling : Memory Triggers Example

```
scheduler myMemory {
/*
 Declare the MemoryManager task/trigger. Not really required to do
*/
declare MemoryManager mem;
/*
 Just using the MemoryManager task / trigger and not the declared mem as an example.
*/
 when (MemoryManager.freeMemory <10000000){
 MemoryManager.flush(false);
 }
/*
 Now when the Free Memory on the realm drops below 1000000 bytes force the
 realm to release ALL available memory
*/
 when ( mem.freeMemory <1000000){
 mem.flush(true);
 }
/*
 This is the same as the one above, except not using the declared name.
*/
 when ( MemoryManager.freeMemory <1000000){
 MemoryManager.flush(true);
 }
```

```
/*
 totalMemory available on the realm
*/
 when ( MemoryManager.totalMemory <20000000 ){
 Logger.report("Declared Memory too small for realm");
 }
/*
 Out Of Memory counter, increments whenever the realm handles an out of memory exception
*/
 when ( MemoryManager.outOfMemory> 2){
 Logger.report("Realm has run out of memory more then the threshold allowed");
 }
}
```

Universal Messaging Scheduling : Realm Triggers Example

```
scheduler realmSchedule {
 declare Realm myRealm ("productionmaster");
 declare Config myGlobalConfig ( "GlobalValues");
 when (Realm.connections> 1000) {
 Logger.report("Reached 1000 connections, setting max connections");
 myGlobalConfig.MaxNoOfConnections(1000);
 }
 when (Realm.eventsSentPerSecond> 10000) {
 Logger.report("Reached 10000 events per second, reducing max connection count by
100");
 myGlobalConfig.MaxNoOfConnections("-100");
 }
}
```

Universal Messaging Scheduling : Cluster Triggers Example

```
/*
This script tests the cluster triggers. It is assumed the cluster is created with 4
realms
named realm1, realm2, realm3, realm4
*/
scheduler myCluster{
 declare Cluster myNode1("realm1");
 declare Cluster myNode2("realm2");
 declare Cluster myNode3("realm3");
 declare Cluster myNode4("realm4");
/*
 This will trigger when realm1 is online to the cluster
*/
 when ( myNode1.nodeOnline == true ){
 Logger.report("Realm1 online");
 }
/*
 This can also be written as
*/
 when ( Cluster.nodeOnline("realm1") == true ){
 Logger.report("Realm1 online");
 }
 when ( myNode2.nodeOnline == true ){
 Logger.report("Realm2 online");
 }
 when ( myNode3.nodeOnline ==true ){
```

```
 Logger.report("Realm3 online");
 }
 when ( myNode4.nodeOnline == true ){
 Logger.report("Realm4 online");
 }
 when ( Cluster.hasQuorum == true ){
 Logger.report("Cluster now has quorum and is running" );
 }
 when ( Cluster.isMaster("realm1") == true){
 Logger.report("This local realm is the master realm of the cluster");
 }
}
```

Universal Messaging Scheduling : Counter Trigger Example

```
scheduler myCounter{
/*
 Define some new counters
*/
 declare Counter counter1 ("myCounter");
 declare Counter counter2 ("myAdditional");
/*
 When the counter reaches 5 reset it to 0;
*/
 when(counter2> 5 ){
 counter2.reset();
 }
/*
 If counter1 is less then 3 then increment the value
*/
 when(counter1 <3){
 counter1.inc();
 counter2.dec();
 }
/*
 if Counter2 equals 0 then set counter1 to 5
*/
 when(counter2 == 0 ){
 counter1.set(5);
 }
}
```

Universal Messaging Scheduling : Time Triggers Example

```
scheduler myTimers{
/*
 Define some new timers
*/
 declare Timer reportTimer ("myTimer");
 declare Timer testTimer ("myDelay");
initialise{
 testTimer.stop();
}
/*
In 60 seconds log a report and start the second timer
*/
 when(timer == 60){
 Logger.report("Timer has fired!");
```

```
 testTimer.start();
 }
/*
When the second timer hits 30 seconds, log it and reset all timers to do it again
*/
 when(testTimer == 30){
 Logger.report("Test dela fired, resetting timers");
 testTimer.reset();
 testTimer.stop();
 timer.reset();
 }
}
```

Universal Messaging Scheduling : Configuration Example

```
scheduler myConfig {
/*
 Declare local names for the Connection Config and the Logging Config configuration
groups.
 Can be used for both triggers and tasks
*/
 declare Config myConnectionConfig ("Connection Config");
 declare Config myLoggingConfig ("Logging Config");
/*
 When this scheduler task is initialised, set the Realms log level to 2
*/
 initialise{
 myLoggingConfig.fLoggerLevel(2);
 }
/*
 Then if the log level is ever set to 0, automatically reset it to 2.
*/
 when(myLoggingConfig.fLoggerLevel == 0){
 myLoggingConfig.fLoggerLevel(2);
 }
/*
 If the maximum number of connections on the realm is less than 0,
 implying no limit, then set it to 100.
*/
 when(myConnectionConfig.MaxNoOfConnections <0){
 myConnectionConfig.MaxNoOfConnections(100);
 }
}
```

Integration with JNDI

Universal Messaging supports integration with JNDI through its own provider for JNDI. Universal Messaging's provider for JNDI enables clients using *Universal Messaging Provider for JMS* to locate references to JMS administered objects.

As with all Java APIs that interface with host systems, JNDI is independent of the system's underlying implementation. In the case of the Universal Messaging product, the JNDI provider stores object references in the Universal Messaging channel `/naming/defaultContext`, which is the channel representing the Universal Messaging Initial Context for JNDI, and locates the references to the objects using a channel iterator. Note that if a realm is part of a cluster, the channel will be created on all cluster realm servers. This ensures that any object references bound into the context

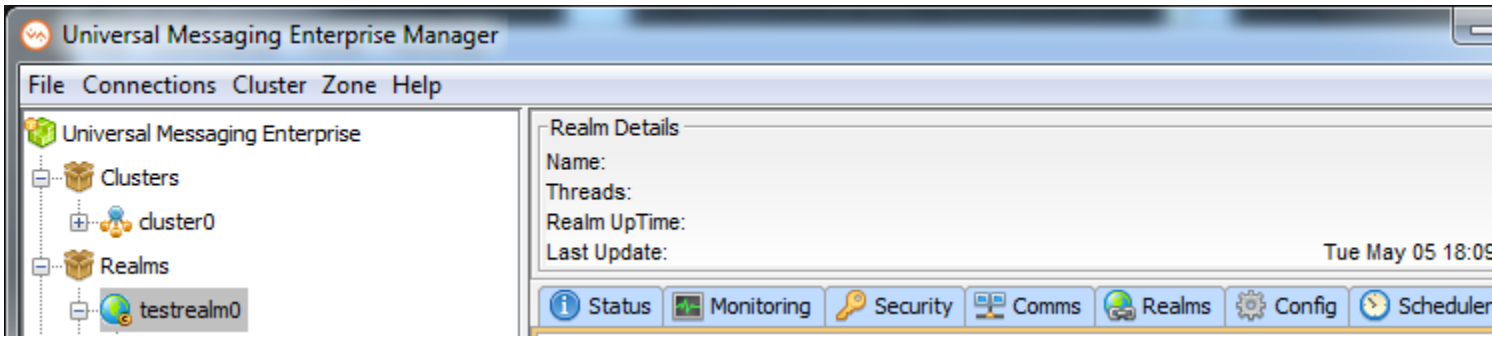
are available on each realm server in the cluster. See the section “Creating The Initial Context” for information about how and when the channel for the Initial Context is created.

Setting Up the Context and Connection Factories for JNDI

The provider for JNDI can be managed using the Enterprise Manager tool, by selecting any realm node from the namespace tree, and then clicking on the **JNDI** tab in the right hand panel. This opens the JNDI panel for the selected realm. The JNDI panel enables the creation of the provider and Initial Context for JNDI, and of TopicConnectionFactory and QueueConnectionFactory references for JMS, as well as references to Topics and Queues.

Creating The Initial Context

When you select a realm node from the namespace tree, one of the tabs on the right hand side of the Enterprise Manager will be labelled **JNDI**.



Selecting this tab will display the default JNDI panel for a realm.

If the Universal Messaging channel `/naming/defaultContext` has not been created yet, you will be prompted to create it.

The Initial Context uses the (potentially clustered) Universal Messaging context channel to store all JNDI references. This channel is called `/naming/defaultContext`. When the channel is initially created, full permissions are assigned to the first client who creates it and to all other users and clients who wish to use the channel.

Removing/destroying the Initial Context is as simple as deleting the `/naming/defaultContext` channel. This will of course result in the loss of all existing JNDI references (so make sure you don't accidentally delete this channel).

If the Initial Context no longer exists, then clicking the **Create JNDI context** button at the bottom of the screen will recreate it (though it will not contain any of its previous JNDI entries/references).

Viewing the JNDI Namespace

Whenever you open the JNDI Panel, Enterprise Manager will enable display of the JNDI Namespace. The JNDI Namespace is displayed as a tree structure within the **Namespace** section of the panel. The root of this tree will be the JNDI Provider URL.

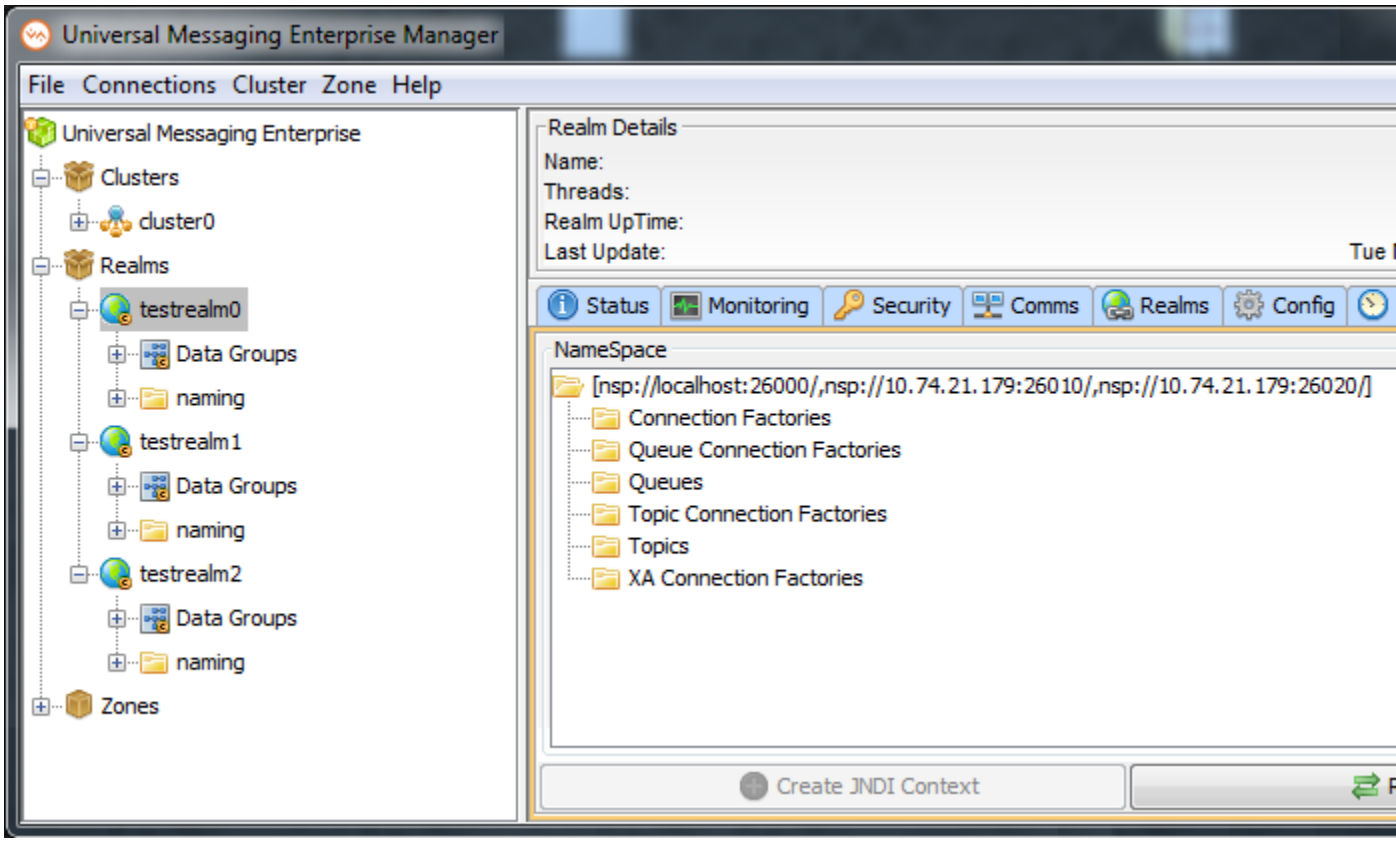
In the case of a cluster, the display will show a comma-separated list of RNAME values for each server in the cluster.

If you are using a *horizontal scalability* connection factory, the URL syntax allows you to specify multiple connection URLs, where each connection URL can specify either a standalone realm or a cluster.

The JNDI namespace tree renders 6 "folders":

- Connection Factories
- Queue Connection Factories
- Queues
- Topic Connection Factories
- Topics
- XA Connection Factories

The image below shows this view after the JNDI panel has been loaded with the JNDI namespace tree expanded:



You can also use the **Refresh** button to update the contents of the JNDI namespace tree with any changes done outside the current instance of the Enterprise Manager.

Creating Connection Factories

By using a *connection factory* you can connect to both topics and queues. With a *topic connection factory* (see description below) you connect only to topics, and with a *queue connection factory* (also described below) you connect to queues. Normally connection factories can be used as a more generic replacement for both topic connection factories and queue connection factories.

How to create a connection factory

To create a connection factory, proceed as follows:

1. Select the node **Connection Factories** and open the context menu of this node.
2. In the dialog, supply a name to be displayed for the new connection factory, for example "connectionFactory2". Also provide a connection URL, for example "nsp://localhost:9000".

   You can specify a cluster of realms by specifying a comma-separated list of connection URLs, for example "nsp://localhost:9000,nsp://localhost:9010".

   I f you want to use a *horizontal scalability* connection factory, you can specify several connection URLs, where each connection URL can point to a standalone realm or a cluster. In this case, each connection URL is bounded by a set of round brackets - "(" and ")".

   Examples:

   - (UM1)(UM2)(UM3)(UM4) - Indicates 4 standalone realms, namely UM1, UM2, UM3 and UM4, so 4 connections will be constructed here.
   - (UM1,UM2)(UM3,UM4) - Indicates 2 clusters, one consisting of UM1 and UM2 and the other consisting of UM3 and UM4, so only 2 connections will be constructed here.
   - (UM1)(UM2,UM3)(UM4) - Indicates one cluster consisting of UM2 and UM3, and two standalone realms, namely UM1 and UM4. A total of 3 connections will be constructed here

   For the round-robin URL syntax, the following rules apply:

   - Each set of brackets must contain at least one valid connection URL.
   - There is no limit on the number of sets of brackets in the URL.
   - Each set of bracket indicates a unique connection, and the realm names within each sets of brackets will be supplied unchanged to the underlying implementation.

   For more information on *horizontal scalability* connection factories, see the section "Provider for JMS" in the Developer Guide.
3. Use the dropdown list to select the appropriate durable type for durable consumers for topics that can be created using this connection factory.

   The following durable types are supported:

   - Named: There can be only one active consumer at a time.

- Shared: Multiple durable consumers can connect to the same durable subscription and can consume messages in a round-robin manner.
- Shared-Queued: Multiple durable consumers can connect to the same durable subscription and can consume messages in a round-robin manner using a queue-based durable.
- Serial: Multiple durable consumers can connect to the same durable subscription and can consume messages in a serial manner.
- Priority: Multiple consumers can connect to the same durable subscription but there can be only one active consumer at any one time.

Durable subscribers can be defined only for topics; they are not available for queues.

4. Click **OK** to save your changes and close the dialog.

Editing Connection Factories

To edit an existing connection factory, proceed as follows:

1. Expand the node **Connection Factories** to display the available connection factories. Then double-click the required connection factory to open the edit dialog.
2. Make your changes in the dialog.
3. Click **OK** to save your changes and close the dialog.

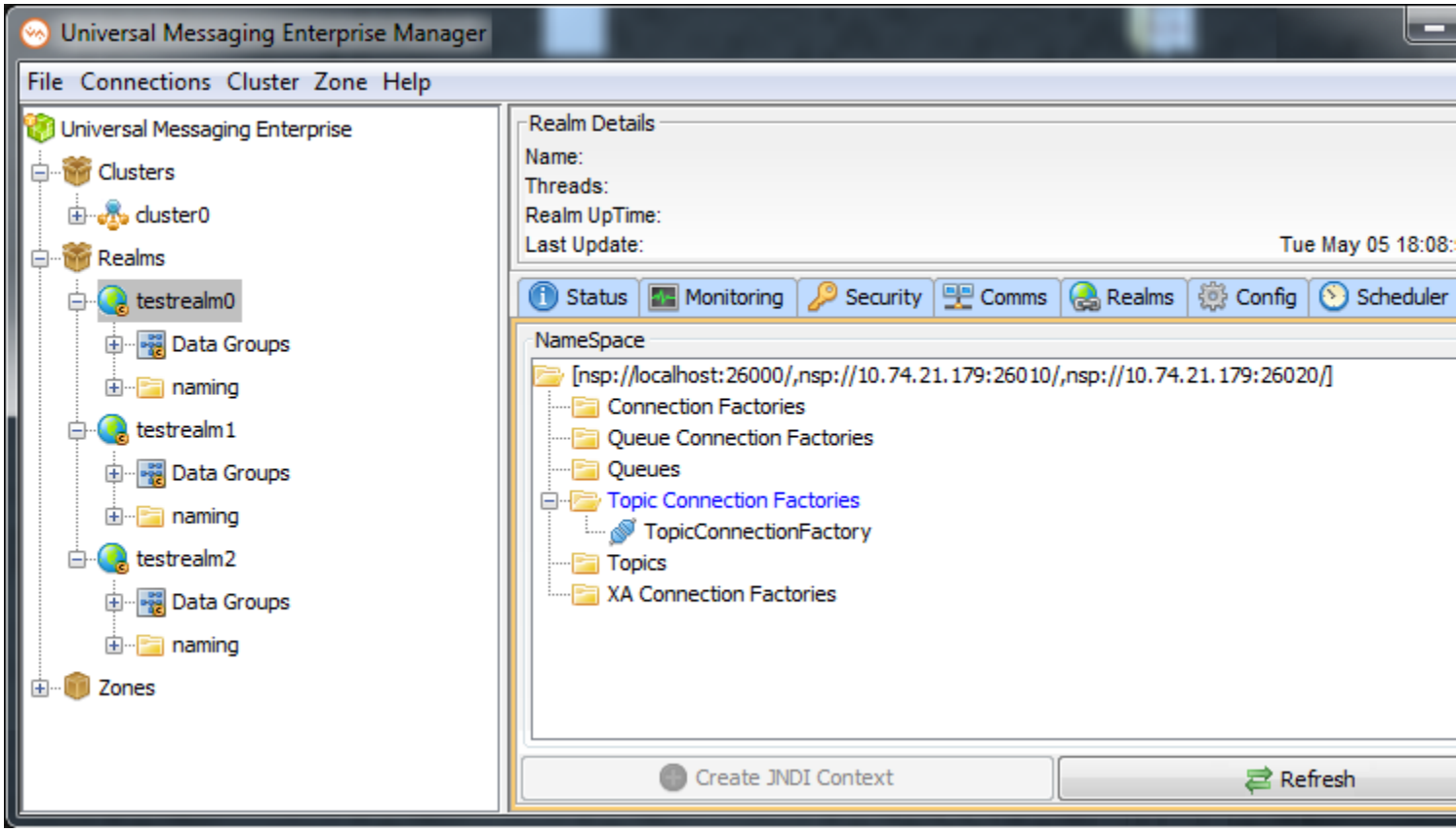
Creating Topic and Queue Connection Factories

In order to allow JMS clients to use the Universal Messaging Context Factory to reference objects via JNDI, we first of all need to create Topic and Queue connection factories.

To create a topic connection factory, proceed as follows:

1. Select the tree node labelled **Topic Connection Factories** and select the menu option **New Topic Connection Factory**.
2. In the dialog, supply a name for the connection factory. Enter any name (in this example, we will use the name **TopicConnectionFactory**). Also provide a connection URL, for example "nsp://localhost:9000".
3. Select the desired durable type from the dropdown list.
4. Click **OK** to save your changes and close the dialog.

You will see that a new node has been created under the **Topic Connection Factories** folder with the name that you entered. The image below shows the JNDI namespace with a newly created topic connection factory:



The Topic Connection Factory object you just created is actually stored as an event, published onto the `/naming/defaultContext` channel. This event is what will be referenced by JMS clients when they attempt to find the details for the connection factory.

To create a queue connection factory, proceed as follows:

1. Select the tree node labelled **Queue Connection Factories** and select the menu option **New Queue Connection Factory**.
2. In the dialog, supply a name for the connection factory. Also provide a connection URL, for example "nsp://localhost:9000".
3. Click **OK** to save your changes and close the dialog.

You will see that a new node has been created under the **Queue Connection Factories** folder with the name that you entered.

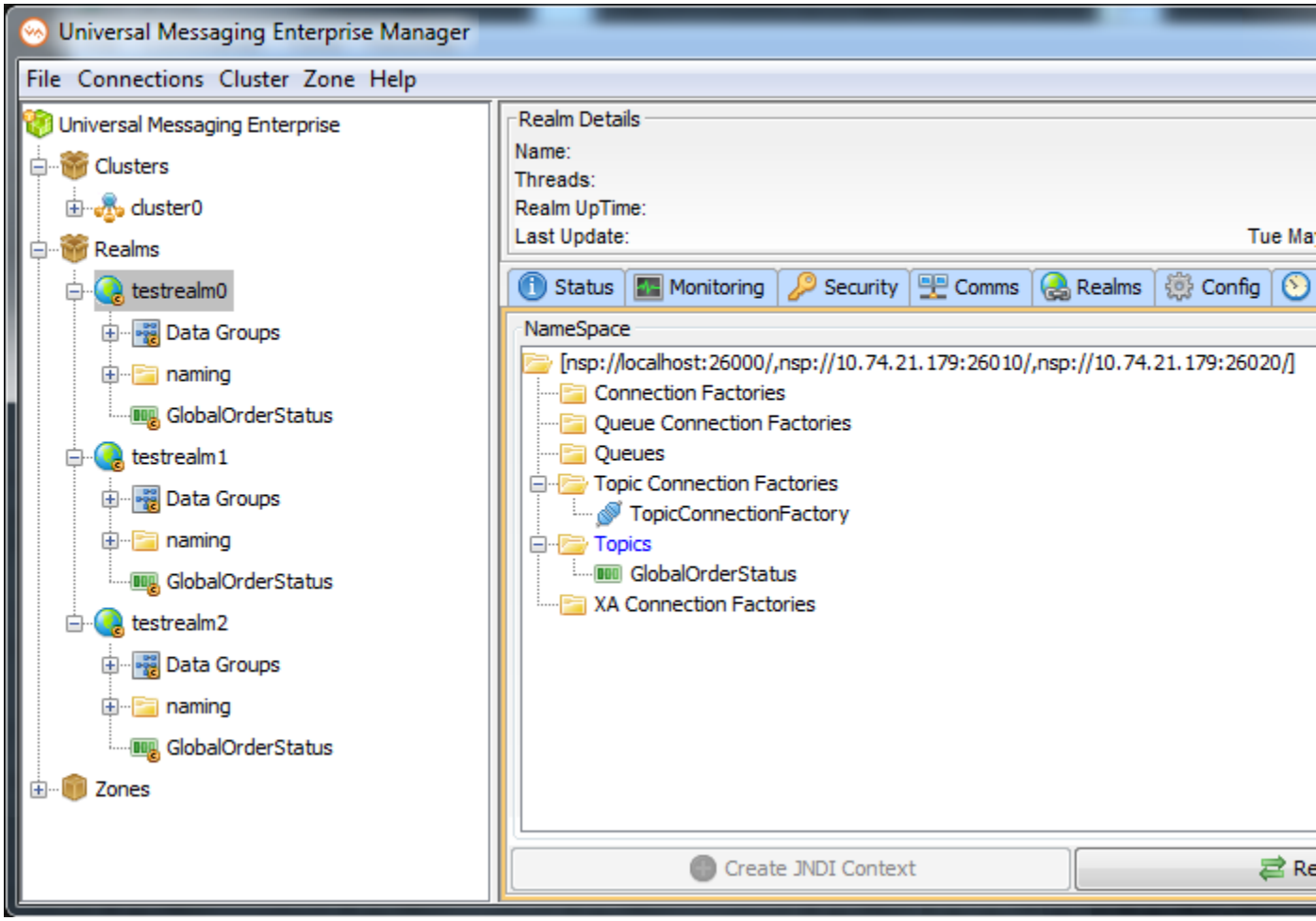
Note:
Durable types are only applicable to topics or normal connection factories.

Creating References to Topics and Queues

When JMS clients use the Universal Messaging Initial Context for JNDI, they also reference the topics and queues from the same Initial Context. In order for these clients to access these objects we need to create references to each topic and queue. Creating such references will also create the

underlying channel or queue if it does not already exist; note that channels or queues created in this way will have the same default permissions as channels or queues created manually.

In this example, we will add a new topic into the JNDI namespace that corresponds to a Universal Messaging channel that already exists as a cluster channel. To do this, first, right-click on the folder called **Topics** within the JNDI namespace, and select the menu option **New Topic**. If we enter the name **GlobalOrderStatus**, then a new object will be created under the **Topics** folder called **GlobalOrderStatus**. This is because, under the covers, a corresponding event was published to the `/naming/defaultContext` channel. JMS clients can thus look up the reference to this topic (channel) and begin using it within their application. The following image shows the newly created Topic within the JNDI namespace for the existing topic **GlobalOrderStatus**:



Viewing / Editing JNDI Settings

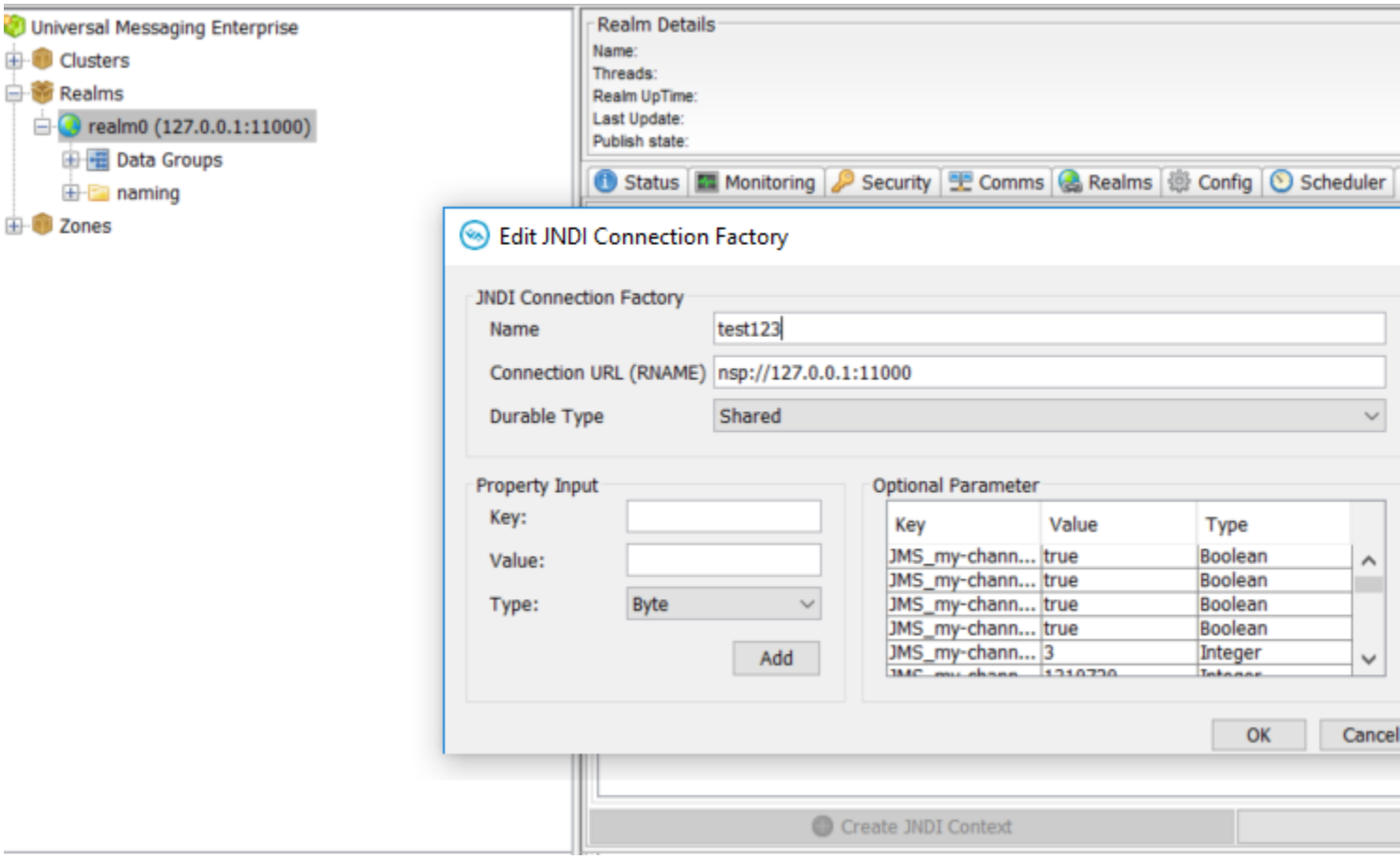
You can view and edit JNDI settings using the **JNDI** tab. With this tab you can also add new properties to existing JNDI entries.

Viewing JNDI Settings of a connection factory

To view the JNDI settings of an existing connection factory:

- Select a realm node from the namespace tree, and select the **JNDI** tab.
- Expand the node **Topic Connection Factories** to display the available connection factories.
- Double-click the required connection factory to open the dialog.

The dialog displays all of the mandatory and optional JNDI parameters in tabular format. The mandatory parameters are shown at the start of the dialog, and must always be present and contain a value. The optional parameters are parameters that are not required by JNDI, but which you can define for your own purposes. The optional parameters are listed under the heading **Optional Parameter**. The image below shows an example of JNDI settings for a connection factory.



Editing JNDI Settings of a connection factory

The dialog described above in the section *Viewing JNDI Settings of a connection factory* allows you to edit the values displayed. To edit a value, move the cursor to the appropriate field and enter a new value. The following general rules apply:

- For mandatory parameters, you can edit the value but not the name.
- For optional parameters, you can change the parameter's key, value and data type.

When you have finished making your edits, confirm the changes and close the dialog.

The edits you have just made cause an event containing the new JNDI settings to be published on the `/naming/defaultContext` channel. Also, the old event on the `/naming/defaultContext` channel containing the previous JNDI settings will be purged automatically.

Adding a New JNDI Parameter for a connection factory

In the view/edit dialog described above, there is a panel **Property Input**. This panel allows you to enter new optional JNDI settings for the connection factory, consisting of a key, value and data type.

To add a new optional setting, enter values for each of the key, value and data type fields, then click **Add**. A new row containing the definition will be added in the **Optional Parameter** table.

When you have finished adding new optional settings, confirm the changes and close the dialog.

The edits you have just made cause an event containing the new JNDI settings to be published on the `/naming/defaultContext` channel. Also, the old event on the `/naming/defaultContext` channel containing the previous JNDI settings will be purged automatically.

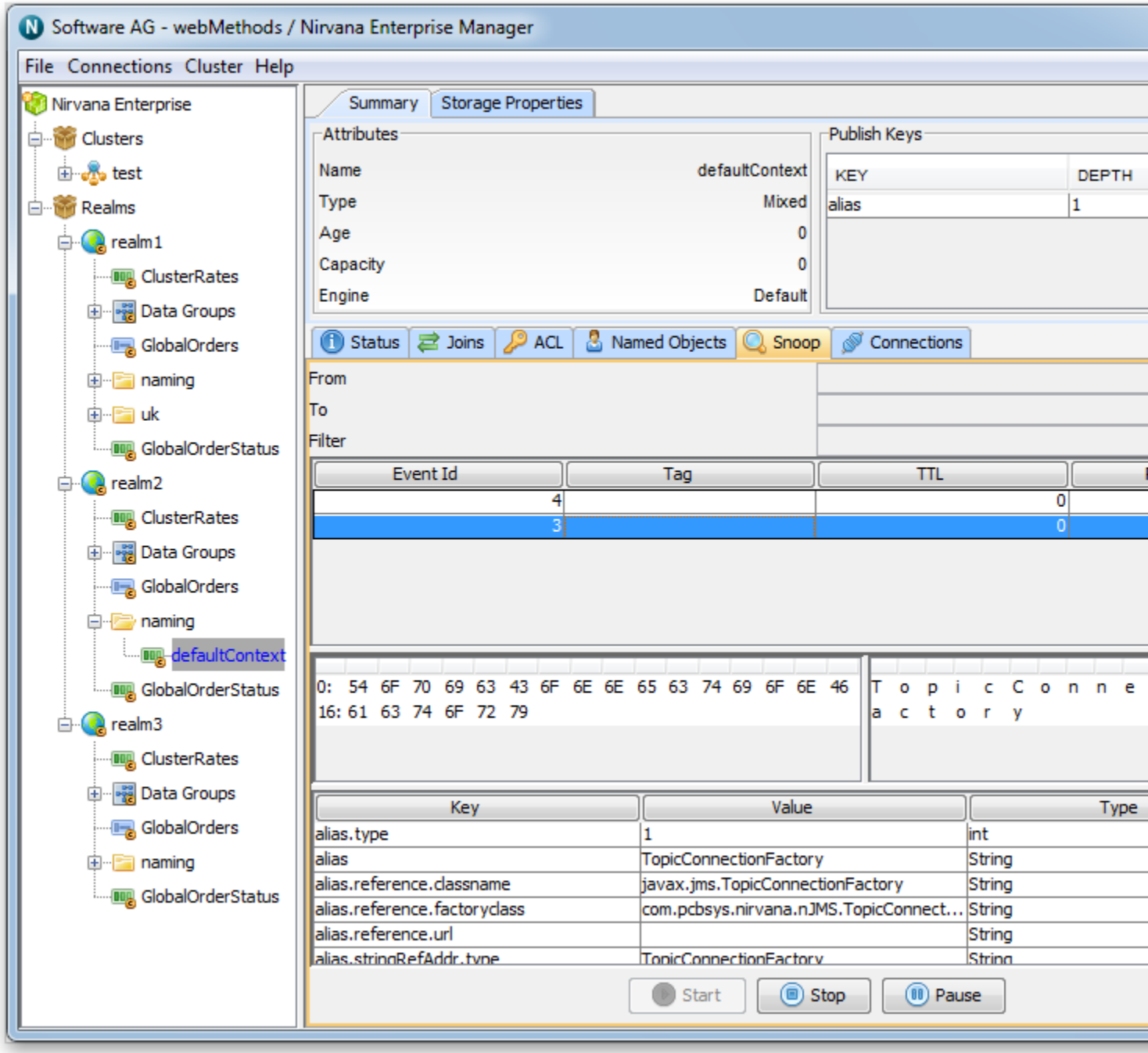
Viewing JNDI settings using Channel Snoop

As described above, JNDI settings are stored as events on the `/naming/defaultContext` channel. You can use the *Channel Snoop* feature (see the section "Channel Snoop" on page 156) to view the individual events on the `/naming/defaultContext` channel.

To view the individual events, first select the `/naming/defaultContext` channel within the Enterprise Manager namespace, then select the **Snoop** tab. In the Snoop display, click **Start**. Now you will see the events representing the JNDI entries that have been created. By selecting any event you will see the content of each event on the channel and the corresponding JNDI context information given to the JMS applications that will require it.

Note:
In most cases, Channel Snoop allows you not only to *view* events on any channel but also *edit* them (using the "edit and republish" feature). However, in the particular case of the `/naming/defaultContext` channel, the "edit and republish" feature for editing JNDI settings has been disabled. If you wish to edit JNDI settings in the `/naming/defaultContext` channel, you can ONLY do this via the **JNDI** tab, as described above in the section "Viewing / Editing JNDI Settings" on page 273.

The image below shows an example of the Topic Connection Factory created earlier using the JNDI panel:



Note:
If an old event on the `/naming/defaultContext` channel has been purged (as a result of the JNDI editing mechanism as described above) while you are viewing the Snoop display, you need to select **Stop** and then **Start** again to see the changes in the Snoop display.

TCP Interfaces, IP Multicast and Shared Memory

Overview of Working with Interfaces

Using Enterprise Manager, you can configure communication mechanisms including TCP Interfaces, IP Multicast and Shared Memory (SHM):

TCP Interfaces

Interfaces within a Universal Messaging Realm Server define a protocol, a network interface and a port number. When a Universal Messaging client connects to a realm using an RNAME, they are actually connecting to an Interface that has been created on the Universal Messaging Realm.

If a machine that is running a Universal Messaging Realm has multiple physical network interfaces, with different IP addresses, it is possible to bind specific protocols to specific ports. This way you are able to segment incoming network traffic to specific clients.

For example, if a realm is running on a machine that has an external internet facing network interface, as well as an internal interface, you can create a Universal Messaging interface that uses nhp or nhps on port 80 or 443 respectively using the external facing interface.

If on the other hand when there are multiple network interfaces, and you do not wish to segment network traffic for specific protocols, you can specify to bind to all known network interfaces to the specified protocol and port.

The default realm setting when you first install Universal Messaging creates a Universal Messaging Socket Protocol Interface that binds to port 9000, on all known network interfaces.

Once this basic understanding of Universal Messaging interfaces is understood, you can then set about performing a number of operations using the Universal Messaging Enterprise Manager:

- "Creating Interfaces" on page 278
- "Deleting Interfaces" on page 286
- "Creating SSL Interfaces" on page 286
- "Stopping Interfaces" on page 285
- "Starting Interfaces" on page 285
- "Interface Configuration" on page 282
- "JavaScript Interface Panel" on page 286
- "Modify Interfaces" on page 285
- "Interface Plugins" on page 289
- "Interface VIA rules" on page 221

IP Multicast

- "IP Multicast Configuration" on page 595

Shared Memory (SHM)

- "Shared Memory Configuration" on page 294

Creating Interfaces

This section describes how to create interfaces for a realm. For general information about interfaces, see "TCP Interfaces, IP Multicast and Shared Memory" on page 277.

To reach the dialog for defining and managing interfaces for a realm, proceed as follows:

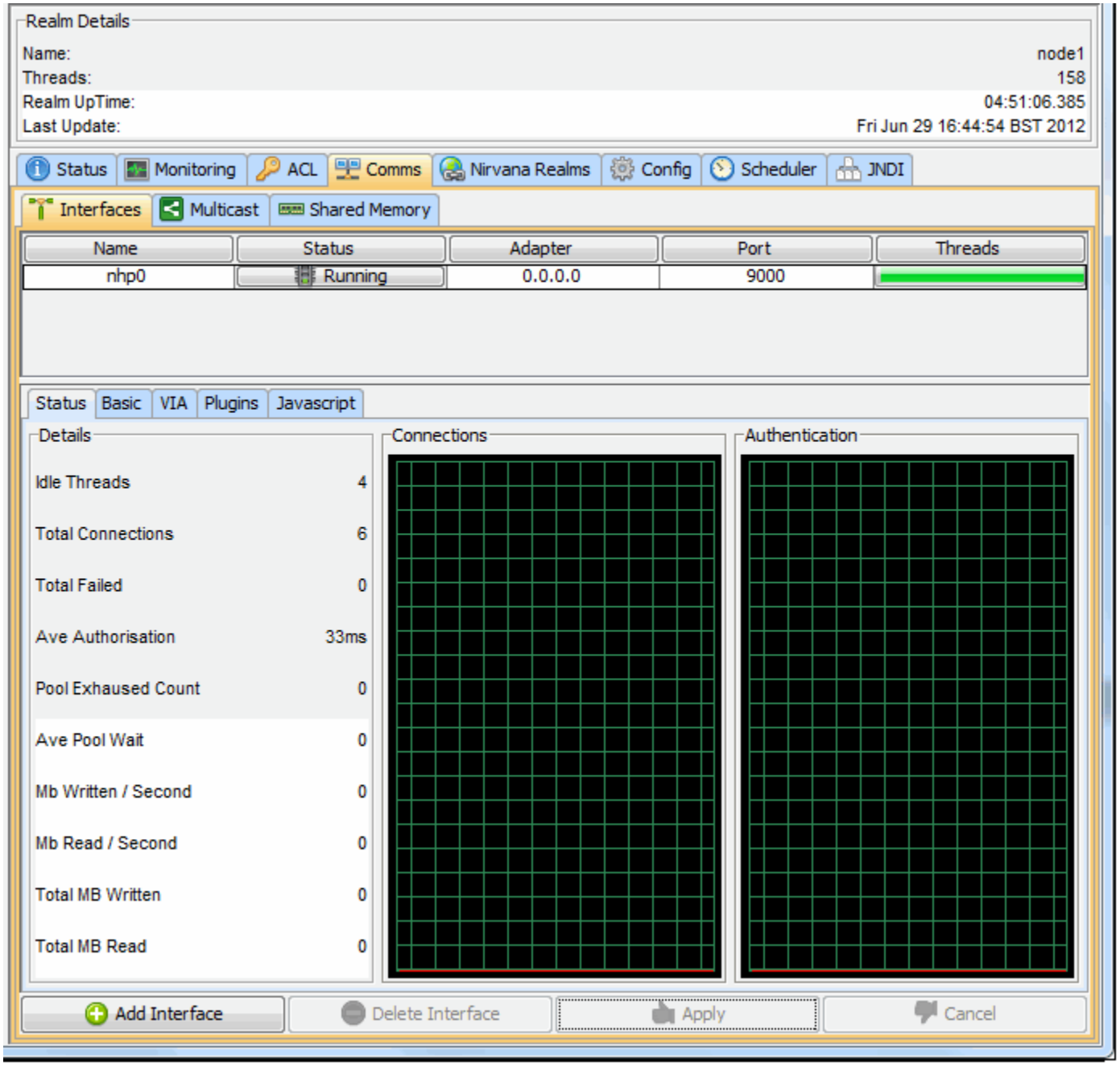
1. In the namespace tree in the navigation frame, select the realm for which you want to create an interface.
2. Select the **Comms** tab in the **Realm Details** frame.
3. Select the **Interfaces** tab.

You now see a table containing all of the available interfaces on a the selected realm.

The default interface is `nsp` (Universal Messaging Socket Protocol) and it binds to 0.0.0.0 (i.e. all known interfaces) on port 9000. Refer to the section "Usage of 0.0.0.0 When Defining Interfaces" on page 281 below for related information.

Please note that adding an SSL enabled interface (see "Creating an SSL network interface to a Universal Messaging Realm Server" on page 297) for either SSL enabled sockets or HTTPS requires some additional steps.

The image below shows the **Interfaces** tab containing the default realm interface.

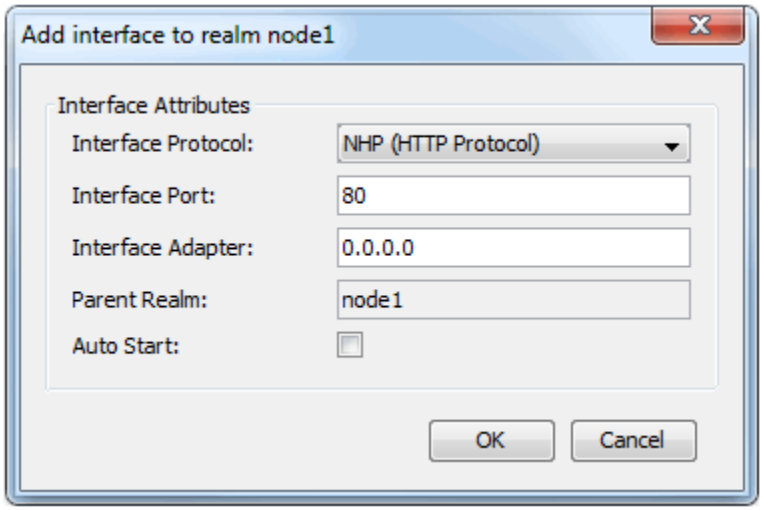


The interfaces table consists of 5 columns:

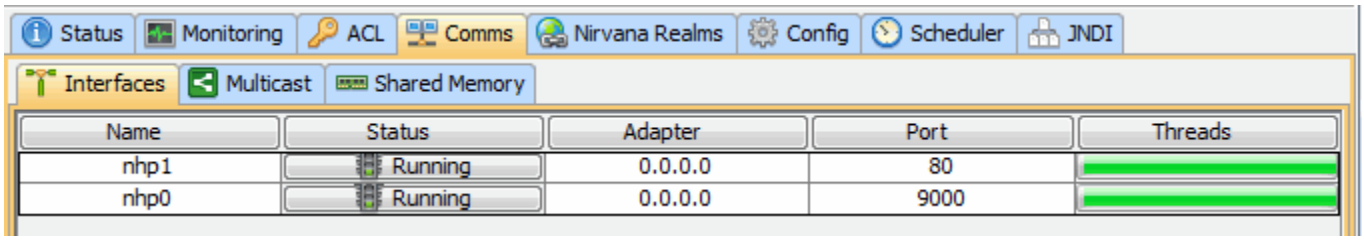
- **Name** : Defined as protocol + n, where n is a unique sequence number for the interfaces for that protocol.
- **Status** : Shows whether the interface is 'Running', 'Stopped' or 'Error'. The error status is shown if the interface has not been started due to an error.
- **Adapter** : The physical network interface to bind to, expressed either as an IP address or a hostname. The IP address 0.0.0.0 defines all known interfaces. You can use the hostname if you want the interface to be independent of the underlying IP address. Refer to the section "Usage of 0.0.0.0 When Defining Interfaces" on page 281 below for related information.

- **Port** : The port to bind to.
- **Threads** : An indicator for the number of accept threads the interface has free to accept connections. A full green bar denotes all are free.

To add a new interface, click on the **Add Interface** button, which will show a dialog that allows you to choose the protocol, the adapter, the port as well as whether the interface should be started automatically when it is created and also when the server restarts. This dialog is shown below:



In the example above, we have chosen to add a Universal Messaging HTTP Interface (nhp) that will be bound to all known network interfaces (0.0.0.0) on port 80. If you select the Auto Start option (by ticking the **Auto Start** checkbox), then click the **OK** button, this means that when the interface is created in the realm server, it will automatically be started. **Auto Start** will also cause that interface to be started whenever the realm is restarted. Once the interface has been created it will appear in the interfaces table as shown in the image below.

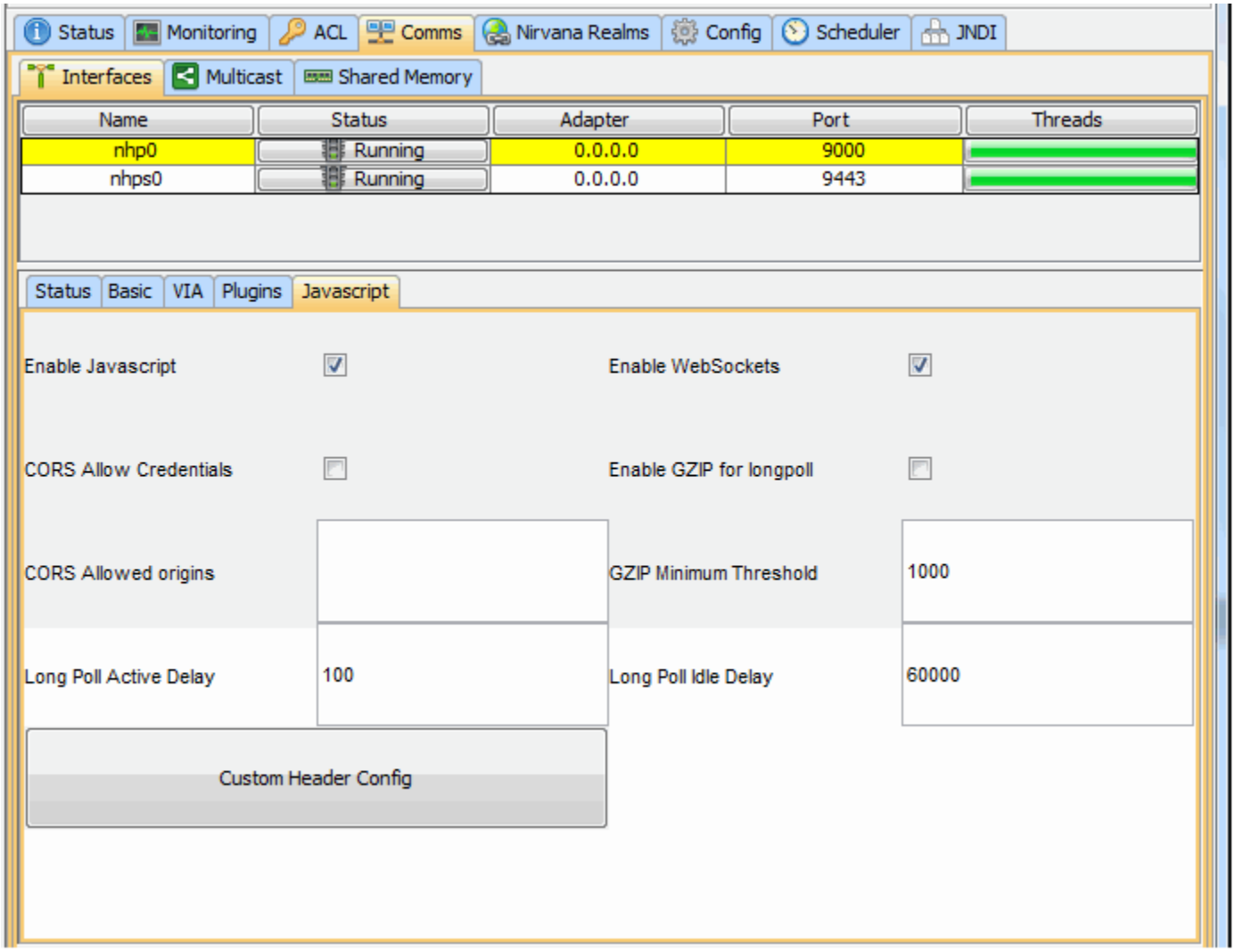


Further instructions on configuring Universal Messaging interfaces are also available in the section "Interface Configuration" on page 282.

In addition a VIA rule (see "Interface VIA Rules" on page 221) can be added to interfaces as a security enhancement.

HTTP / HTTPS Interface

The Javascript tab allows configuration of Comet delivery and is available for HTTP / HTTPS (nhp / nhps) interfaces.



Usage of 0.0.0.0 When Defining Interfaces

When a client connects to a server, the server will deliver all interfaces that are marked as "advertised" (this is set using the checkbox **Advertise Interface** under the panel **Comms > Interfaces > Basic**). If you have interfaces bound to `0.0.0.0` (i.e. all known interfaces), then this will include both `127.0.0.1` (localhost) and any IP address that the server node has. This means that the client will receive at least 2 interfaces that it will use to reconnect to the realm.

If the client connection needs to be restarted for any reason, the client will attempt to iterate through this list of interfaces until it is successful on reconnection. However, the order of iterating through this list is not deterministic.

On a successful connection, the Universal Messaging realm server will construct a client principal name in the format `<userName>@<IP-Address>` used to check permissions on realm resources (e.g. channels or queues), where `<IP-Address>` is the IP address of the machine where the client is running. The IP address of the client in turn depends on the network interface the client used to connect to the server. In the aforementioned example with a server adapter bound on all network interfaces (`0.0.0.0`), a local client (i.e. on the same machine as the server) may connect over the

loopback interface to the server (localhost) so the connection will come from `127.0.0.1`, but that same client may also connect over the real network interface, in which case the IP address will be the address of this network interface. Thus one and the same client may end up with different principal names when reconnecting to the realm server. This may lead to permission issues if a resource's default ACL has been established using one principal name, and is subsequently accessed with a different principal name after a reconnection.

To avoid this, you should either create the interface for an external IP address (not "localhost") or ensure that required ACLs are configured.

Interface Configuration

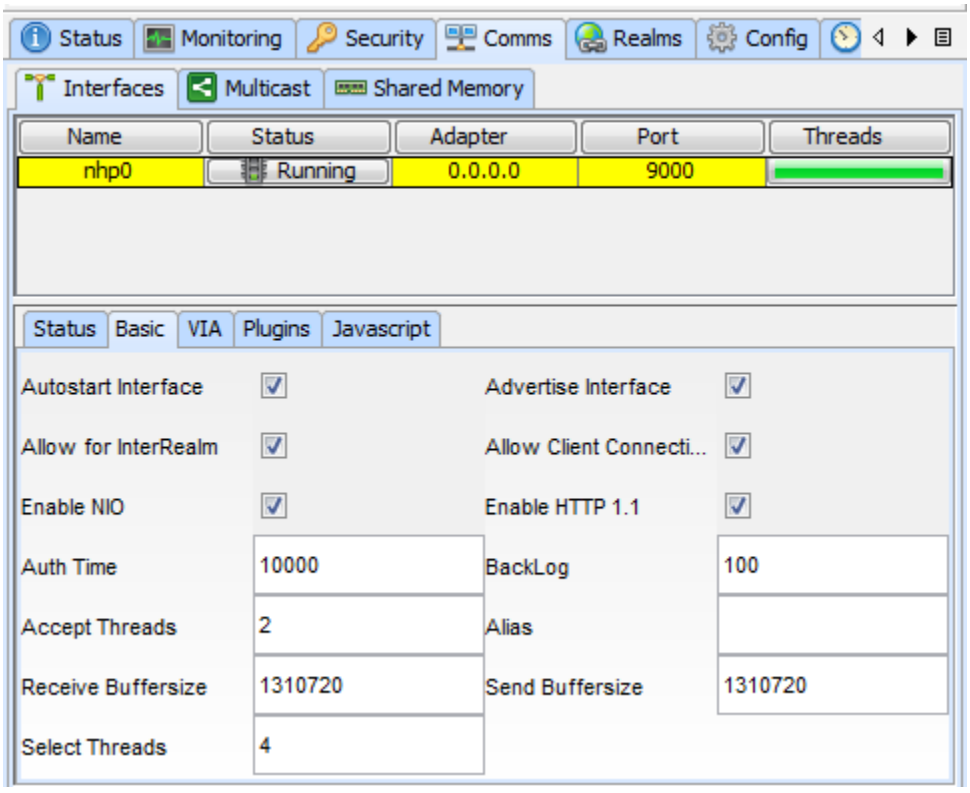
Each interface on a Universal Messaging Realm has a number of configurable attributes that determine the interface behavior. Some of these attributes are standard across all types of interface protocols, and some are specific to a particular protocol.

This section will describe the attributes that are common to interfaces of all types.

For additional information on specific interfaces types, see "TCP Interfaces, IP Multicast and Shared Memory" on page 277.

Basic Interface Attributes

When an interface is selected from the table of interfaces on the **Interfaces** tab, there are a number of attributes that are configurable for the interface. Below the interfaces table, there is a set of tabs, one of which is labelled **Basic**, as shown in the image below.



The basic interface configuration panel shows configurable attributes. These are explained in the following section:

Accept Threads

Each Universal Messaging realm interface contains a server socket. The **Accept Threads** attribute corresponds to the number of threads that are able to perform the `accept()` for a client connection. The `accept()` operation on a Universal Messaging interface performs the handshake and authentication for the client connection. For more heavily utilised interfaces, the accept threads will need to be increased. For example, on an nhp (http) or nhps (https) interface, each client request corresponds to a socket `accept()` on the interface, and so the more requests being made, the busier the interface will be, so the accept threads needs to be much higher than that of say an nsp (socket) interface. Socket interfaces maintain a permanent socket connection, and so the `accept()` is only performed once when the connection is first authenticated.

Advertise Interface

All interfaces that are advertised by a realm server are available to users (with the correct permissions) of the Universal Messaging Admin API. This property specifies whether the interface is indeed advertised to such users.

Alias

Each interface on a Universal Messaging Realm Server can have an associated alias in the form of host:port. This alias can be specified here.

For information on interface plugins please see "Interface plugins" on page 289.

For information on adding VIA rules for an interface please see "Interface VIA Rules" on page 221.

When you change any of these attributes, the changes need to be applied by clicking the **Apply** button. For more information, refer to the modifying interfaces documentation (see "Modifying Interfaces" on page 285).

Allow Client Connections

If this attribute is activated, clients are allowed to connect to the realm over this interface.

If this attribute is deactivated, clients are not allowed to connect to the realm over this interface. Note that Administration API connections, e.g. Enterprise Manager, count as client connections, so at least one of the available interfaces should allow such Administration API connections. If a realm has been defined with only one interface and you deactivate the **Allow Client Connections** attribute on the interface, this setting will be ignored. This is because essential administration tools like the Enterprise Manager would not otherwise be able to access the realm.

Allow for InterRealm

If this attribute is activated, the interface can be used for any of the following kinds of internal communication (i.e. Universal Messaging's own message passing) between realms:

- Inter-realm communication: between realms in the same cluster.
- Inter-zone communication: between realms in a zone.
- Inter-cluster communication: between realms in connected clusters.

Important:
If you do not activate this attribute for the interface, the interface cannot be used for any of the above scenarios.

If you activate **Allow for InterRealm** and deactivate **Allow Client Connections** for the same interface, the interface can **only** be used for internal communication between realms, so no communication with an external client is possible using such an interface. There are situations in which this configuration can be useful. For more information, see the section "Setting up Inter-Realm Communication" on page 119.

Autostart Interface

The **Autostart** attribute specifies whether the interface is started automatically when the Universal Messaging realm server is started. When this option is not selected, the interface must be started manually in order for it to be used by connecting clients. Please note that if **Autostart** is not set it must be started either manually or using the Universal Messaging Administration API whenever after the realm is started.

If **Autostart** is selected then the interface will be started once the **Apply** button is pressed.

Auth Time

The **Auth Time** attribute corresponds to the amount of time a client connection using this interface can take to perform the correct handshake with the realm server. For example, the default is 10000 milliseconds, but for some clients connecting on slow modems, and who are using the nhps (https) protocol, this default **Auth Time** may need to be increased. If any client connection fails to perform the handshake in the correct timeframe, the connection is closed by the realm server.

Backlog

The **Backlog** attribute specifies the maximum size of the incoming IP socket request queue. The operating system that the realm server is running on may specify a maximum value for this property. When the maximum queue size is reached the operating system will refuse incoming connections until the request queue reduces in size and more requests can be serviced. For more information on this value, please see the system administration documentation for your Operating System.

Enable HTTP 1.1

Enable the usage of HTTP 1.1 protocol on this interface.

Enable NIO

Specify whether NIO should be used for this interface.

Receive Buffersize

This specifies the size of the receive buffer on the socket

Select Threads

The **Select Threads** option specifies the number of threads allocated to monitor socket reads/writes on the interface if NIO is enabled. When a socket needs to be read, these threads will fire and pass on the request to the read thread pool. If the socket is blocked during a write, then when the socket is available to be written to, these threads will fire and the request will be passed on to the write thread pool. The number of select threads should not typically exceed the number of cores available.

Send Buffersize

This specifies the size of the send buffer on the socket.

Starting Interfaces

Interfaces can be started by selecting the realm node where the interface you want to start is running, and selecting the **Interfaces** tab. From the table of configured interfaces, select the interface you want to start and double-click on the row. Alternatively, you can click on the **Status** column for the interface. Both will present you with a dialog for starting the selected interface.

Clicking on the **Start** button will start the interface on the realm server.

Modifying Interfaces

Each interface within a Universal Messaging realm has a number of configuration attributes (see "Interface Configuration" on page 282) that can be modified using the Enterprise Manager. Once modified, these can be applied to the interface on the fly. Modifying an interface will cause it to restart, closing all connections to the interface. However, since Universal Messaging clients will automatically reconnect to the realm server, the service disruption should be minimal.

When you have modified the configuration attributes for the selected interface, the Interfaces panel contains a button labelled **Apply**. Clicking on this button will send the modified attributes to the realm server and apply them to the interface, causing it to restart. If there are any clients connected to the interface they will automatically reconnect after restart.

Stopping Interfaces

Interfaces can be stopped by selecting the realm node where the interface you want to stopped is running, and selecting the **Interfaces** tab. From the table of configured interfaces, select the interface you want to stop and double-click on the row. Alternatively, you can click on the **Status** column

for the interface. Both will have the same effect, and will present you with a dialog for stopping the selected interface.

Clicking on the **Stop** button will stop the interface on the realm server.

Deleting Interfaces

Interfaces can be deleted by simply selecting the realm node where the interface you want to delete is running, and selecting the **Interfaces** tab. From the table of configured interfaces, you can simply select the interface you want to delete and click the **Delete Interface** button.

When you delete an interface, Universal Messaging first stops the interface, then closes all clients connected to the interface, then removes the interface from the realm.

SSL Interfaces

Universal Messaging supports SSL encryption by providing two SSL-enabled protocols. These protocols enable clients to connect to a Universal Messaging realm server running a specific protocol on a port using all or specific physical network interfaces.

Defining an SSL-enabled interface ensures that clients wishing to connect to a realm server can do so only after presenting the correct SSL credentials and authenticating with the server.

SSL authentication occurs within the Universal Messaging handshake which uses the JVM's JSSE provider. This ensures that any unauthorized connections are SSL authenticated before any Universal Messaging specific operations can be performed.

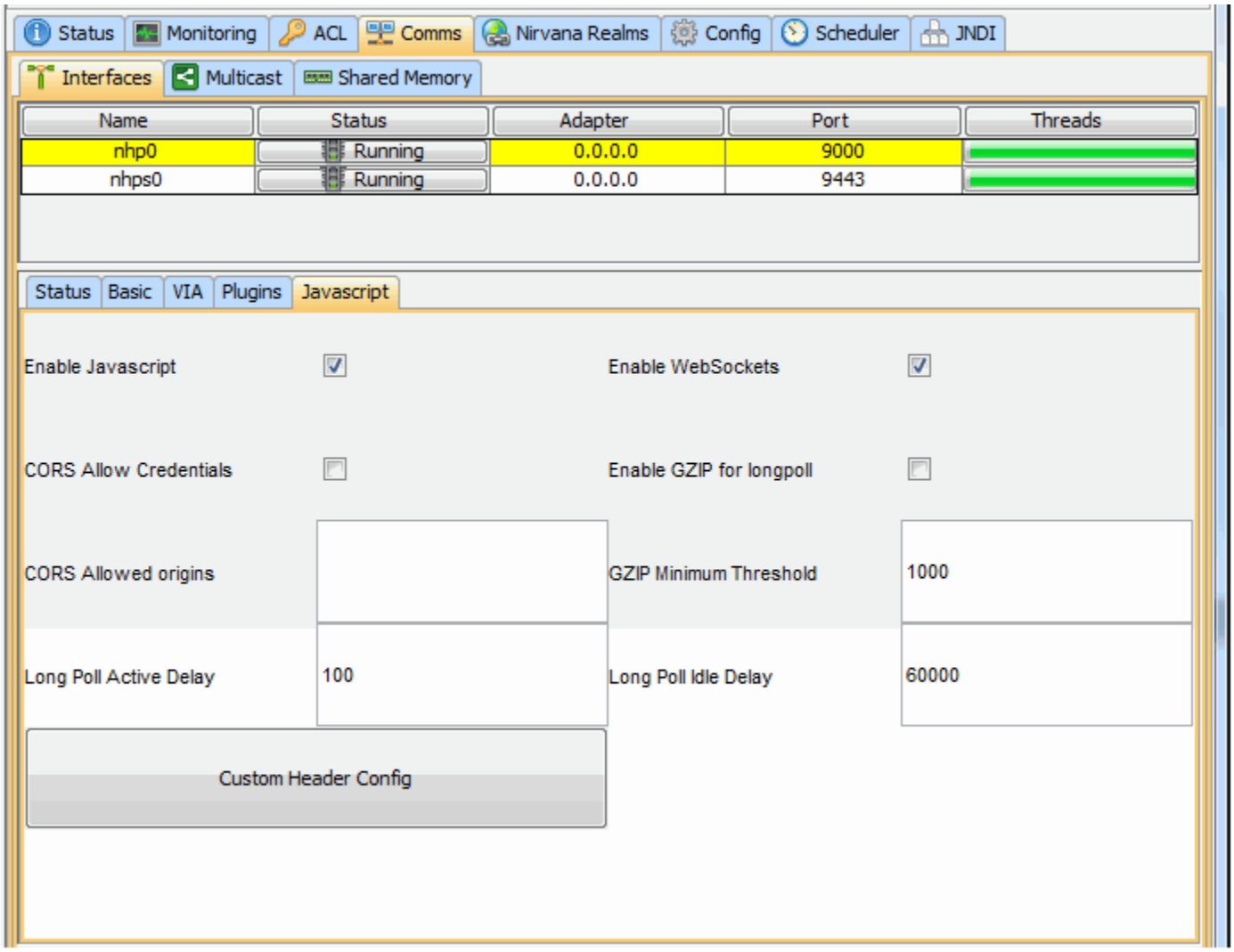
Creating an SSL enabled interface is the same as creating a non-SSL interface (see "Creating Interfaces" on page 278) except where there are a number of SSL-related attributes in addition to the basic attributes (see "Interface Configuration" on page 282).

For information on how to create an SSL interface using the Universal Messaging Enterprise Manager, see "Creating an SSL network interface to a Universal Messaging Realm Server" on page 297.

JavaScript Interface Panel

Universal Messaging HTTP and HTTPS (nhp and nhps) interfaces have configuration options specific to their communication with web clients using JavaScript. These options are accessible through the **JavaScript** panel when viewing an nhp or nhps interface.

The Interface Panel



JavaScript Interface Properties

Option Name	Description
Enable JavaScript	Recommended Setting: Enabled Allows JavaScript clients to connect on this interface.
Enable WebSockets	Recommended Setting: Enabled Toggles the ability for clients to communicate with the server using the HTML WebSocket Protocol on this interface.
CORS Allow Credentials	Recommended Setting: Enabled

Option Name	Description
	Toggles the server sending an "Access-Control-Allow-Credentials: true" header in response to XHR-with-CORS requests from the client. This is required if the application, or website hosting the application, or intermediate infrastructure such as reverse proxy servers or load balancers, uses cookies. Leave this enabled unless recommended otherwise by support. Disabling this will in most environments prevent all CORS-based drivers from working correctly.
CORS Allowed Origins	Recommended Setting: * A comma separated list of the host names (and IP addresses, if they appear in URLs) of the server/s which host your JavaScript application's HTML. Use an * (asterisk) as a wildcard value if you do not wish to limit the hosts that can serve applications to clients. This server will accept and respond with the required Access-Control-Allow-Origin header when requests originate from a hostname in this list. This header allows CORS enabled transport mechanism to bypass cross site security restrictions in modern browsers. It is important that this is set appropriately, or approximately half of the communication drivers available to JavaScript clients will fail.
Enable GZIP for LongPoll	Recommended Setting: Enabled This will allow the server to gzip responses sent to LongPoll clients. This can reduce network utilization on servers with many LongPoll clients. It increases CPU resource utilization.
GZIP Minimum Threshold	Recommended Setting: 1000 The minimum message size is bytes required for the server to begin compressing data sent to LongPoll clients.
Long Poll Active Delay	Recommended Setting: 100 The time between clients sending long poll requests to the server in milliseconds. Reducing this may reduce latency up to a certain threshold but will increase both client and server memory, CPU and network usage.
Long Poll Idle Delay	Recommended Setting: 25000 The time between clients sending long poll when the client is in idle mode. A client is put in idle mode when no communication takes place between client and server for a period of time. Reducing this may be necessary if a client is timing out owing to local TCP/IP settings, proxy settings, or other infrastructure settings, but will result in higher memory, CPU and network usage on both the client and the server. It is however vital that this value is lower than the timeouts used in any intermediate proxy server, reverse

Option Name	Description
	proxy server, load balancer or firewall. Since many such infrastructure components have default timeouts of as little as 30 seconds, a value of less than 30000 would be prudent. If long polling client sessions continually disconnect and reconnect, then lower this value.
Custom Header Config	Header Key/Value pairs which are sent in the HTTP packets to the client.

Interface plugins

Universal Messaging supports the concept of plugins that actively process client requests made to nhp and nhps interfaces. There available plugins are:

- File (Provides behavior similar to a web server)
- XML (Browse resources and events in XML)
- Proxy Passthrough (Enable http/s requests for specific URLs to be forwarded to another host:port)

The plugins are discussed in more detail in the section "Plugins" on page 305.

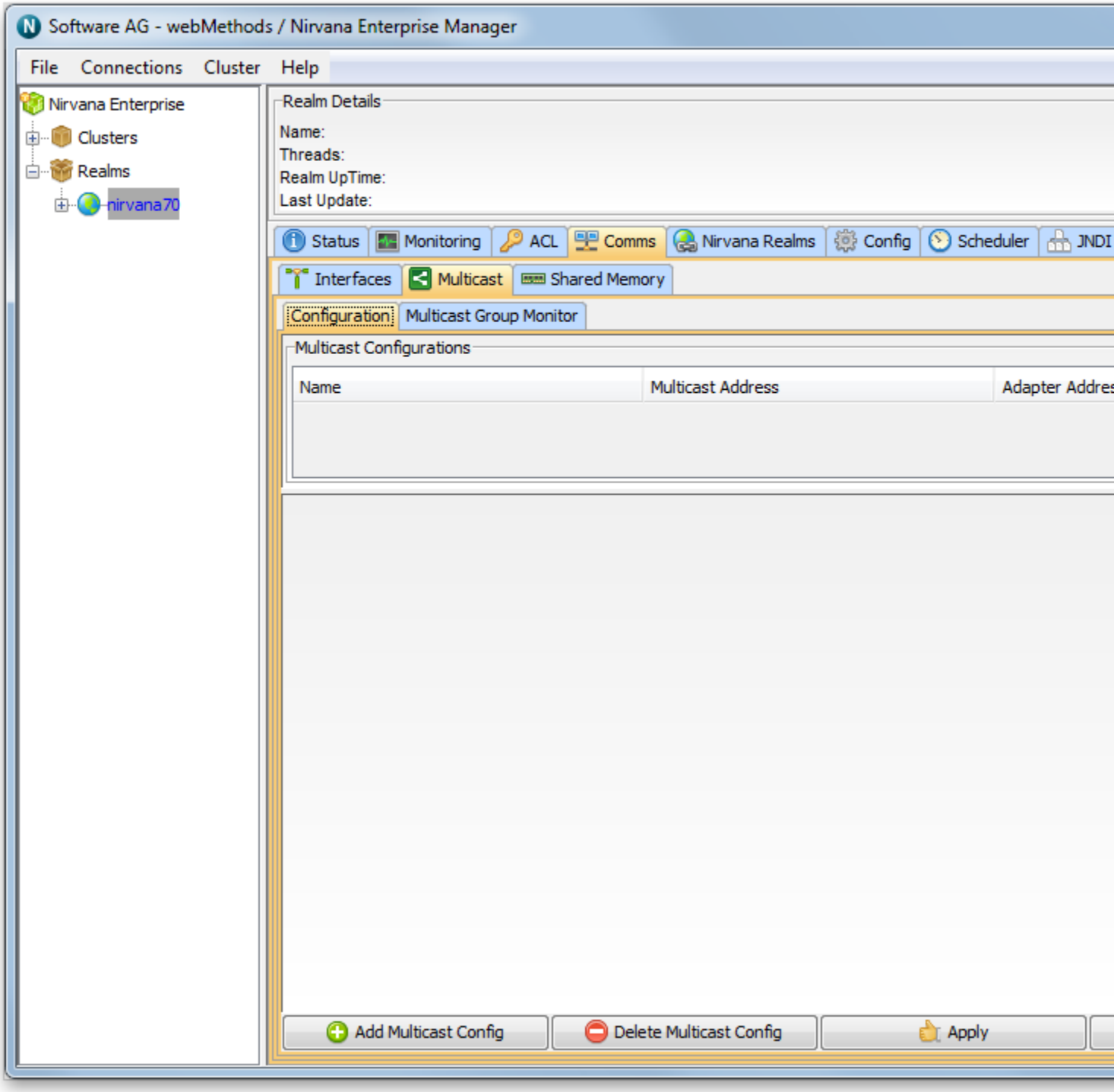
Multicast Configuration

Universal Messaging delivers 'ultra-low latency' to a large number of connected clients by including IP Multicast functionality for both the delivery of events to Data Group consumers as well as between inter-connected realms within a Universal Messaging cluster.

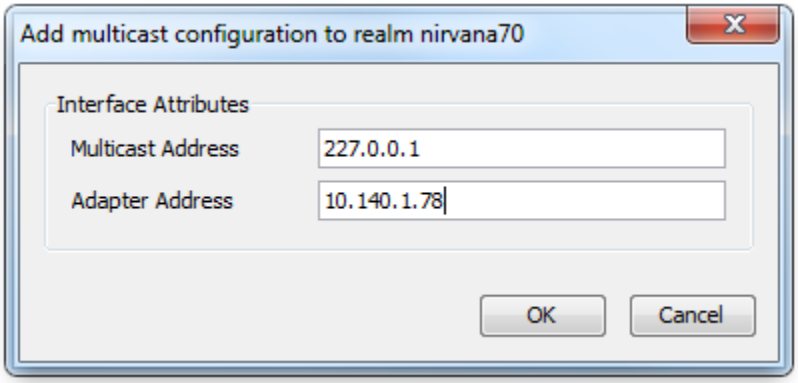
For a description of Multicast messaging, refer to the section *Multicast: An Overview* in the Concepts guide.

Setting Up Multicast for Data Group Delivery

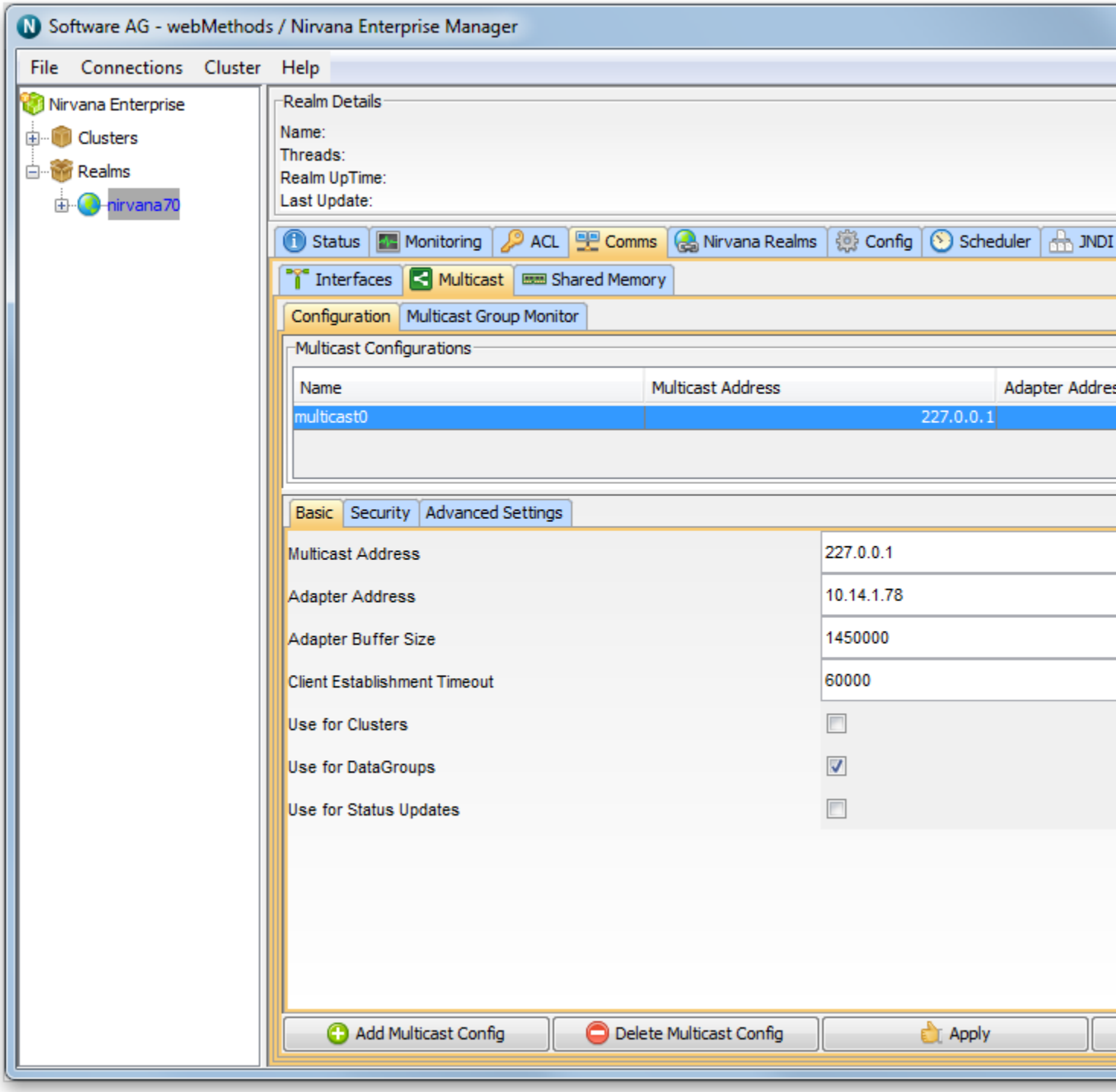
The first step in configuring a Universal Messaging Realm for Multicast Data Group delivery is to create the Multicast adapter configuration. Once you have the information described in the previous section, and your Universal Messaging realm is running, start the Enterprise Manager and connect to your realm. Once connected, select the realm node from the tree, and choose the Multicast tab in the right hand panel, as shown below.



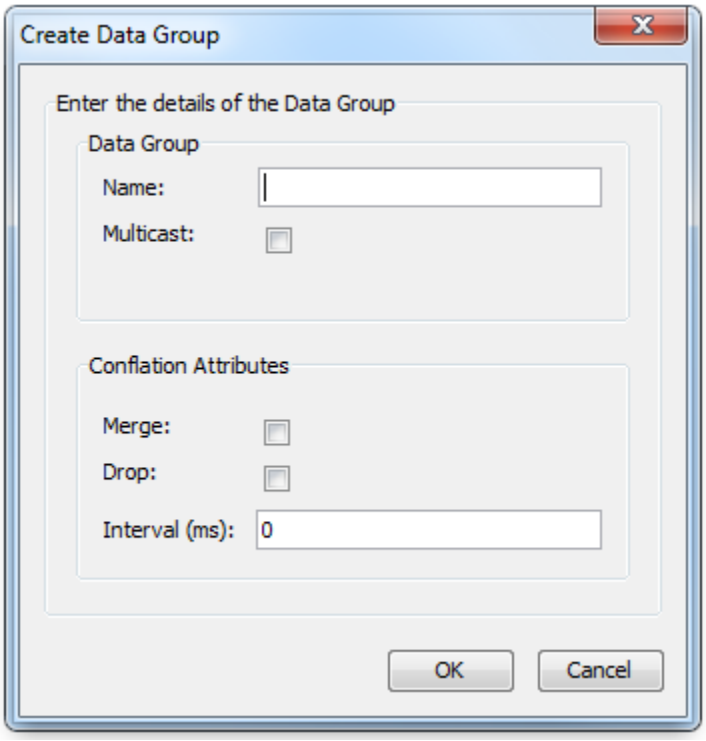
Clicking on the "Add Multicast Config" button opens a dialog that enables you to enter the Multicast IP Address, as well as the Network Adapter Address of your multicast configuration, as shown below.



When you click on ok in the dialog, the new Multicast configuration will appear in the table. You then need to select that the multicast configuration is to be used for Data Groups by clicking on the "Use for DataGroups" check box. Then click the "Apply" button and the configuration will be sent to the server. The completed multicast Configuration is shown in the table as seen in the image below.



Now you have created the Multicast configuration, you need to create your Multicast enabled Data Groups. To do this, simply click on the Data Groups node in the tree, and right click "Create Data Group". This will open up the standard create Data Group dialog but with an additional check box for enabling Multicast. This is shown in the image below.



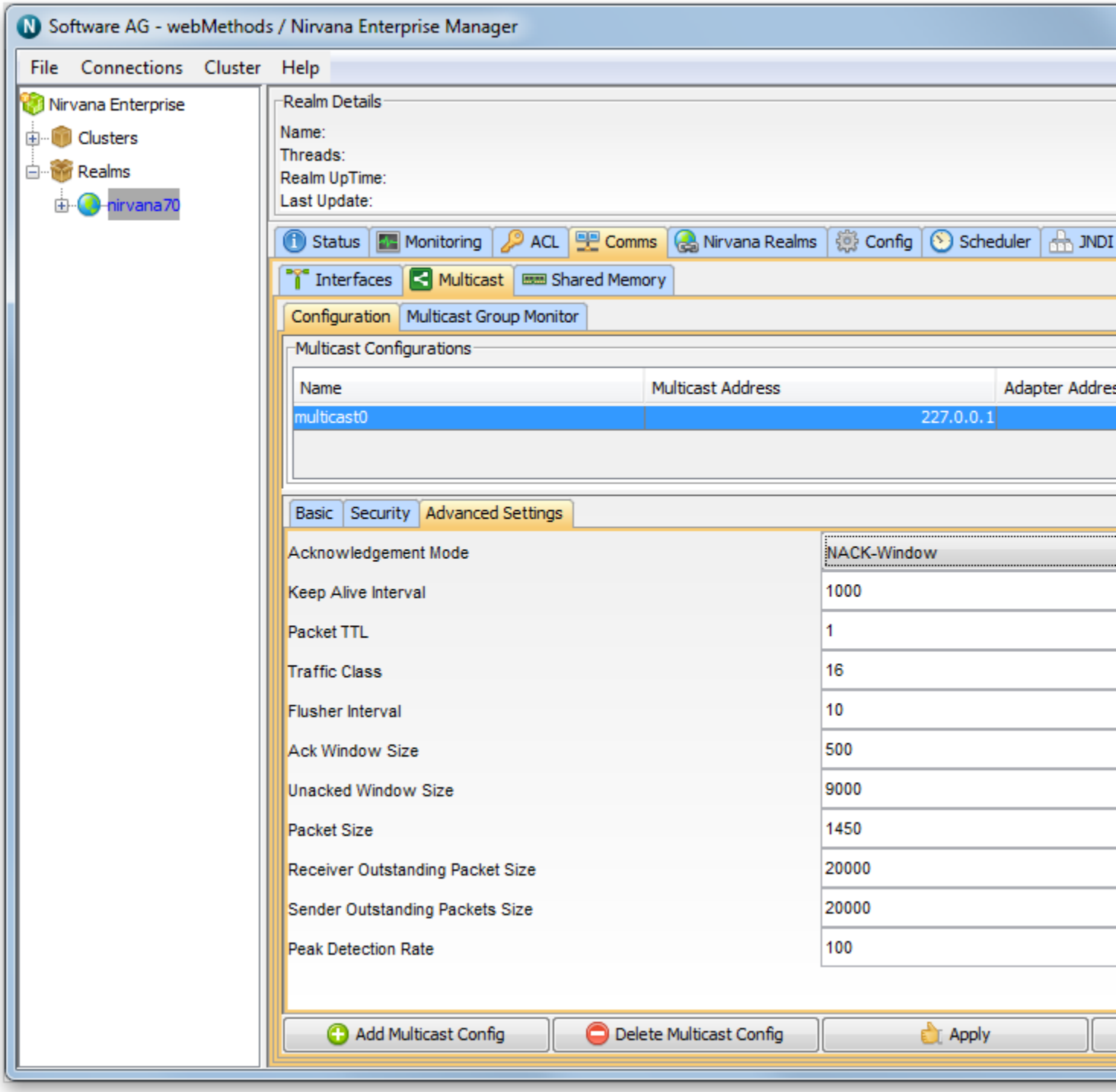
Now your data group is ready to be used for Multicast Delivery. If you are familiar with Universal Messaging Data Groups, you will be familiar with our example Data Group programs which you can use to test this out, or you may have your own Data Group setup that you can use. If your Data Groups are created programmatically, then the key thing to remember is that when you call the nSession.createDataGroup, you now need to also pass in an additional boolean that marks the Data Group as Multicast enabled.

Setting Up Multicast for Cluster Inter Realm Communication

If you have a clustered setup, and you wish to setup Multicast between your realms for the inter realm communication, the setup is the same, however on each realm that you create a Multicast Configuration, the configuration itself needs to set the "Use for Clusters" checkbox. The Multicast address can be the same for all realms, or you can choose a different Multicast Address per realm. With this feature enabled, each realm will know the Multicast address for each of the other realms in the cluster and will listen on these addresses for inter realm cluster communication.

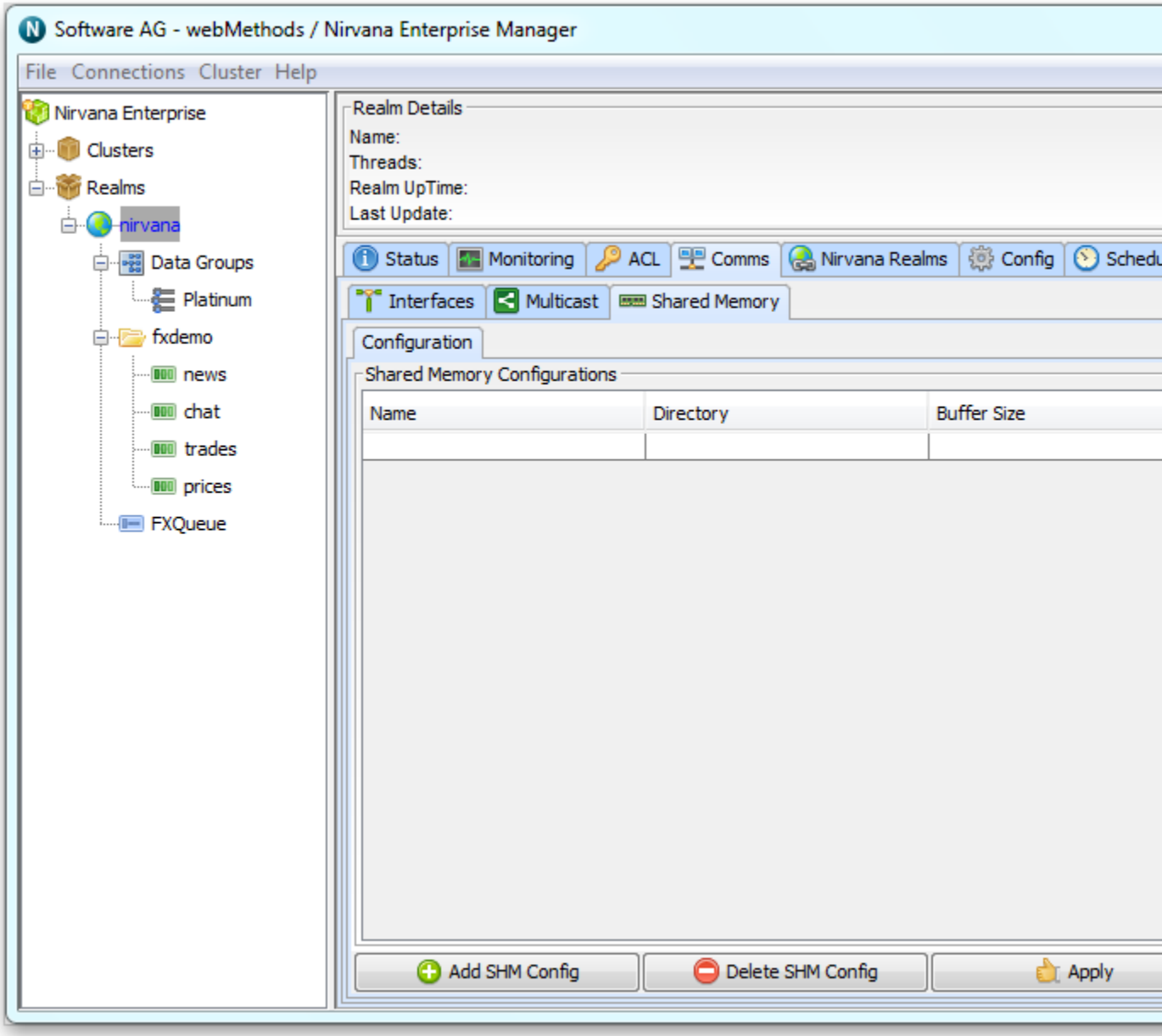
Advanced Multicast Settings

The default settings for the Multicast configurations you create are aimed at providing the lowest possible latency. With this in mind, the configuration is such that the multicast client will ack every 1 second, and the server will maintain a list of un-acked events (default 9000). Should the publish rate exceed 9000 per second, you may notice that the delivery rates might be quite irregular. This is down to the fact that the client will only acknowledge every 1 second, and so the server will automatically back off the delivery until it receives an acknowledgement from the client and can therefore clear its unacknowledged queue. If this happens, you can change both the Unacked Window Size to be > 9000 and the Keep Alive Interval (ack interval) to be less than 1 second (see image of the Advanced Settings tab below).

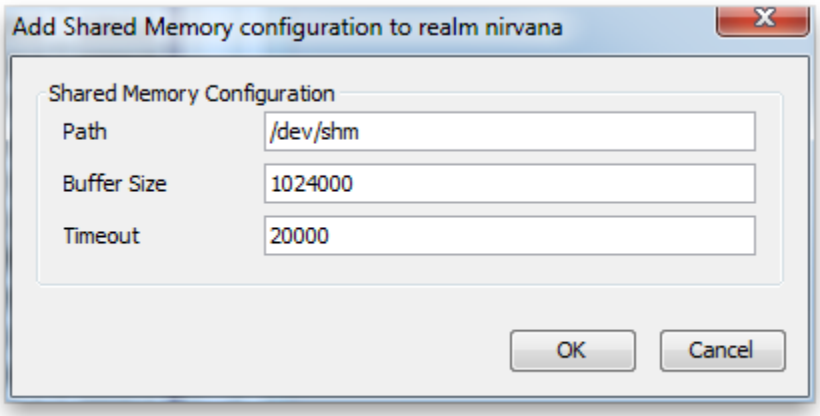


Shared Memory Configuration

In order to create a Shared Memory (SHM) interface you will need to select the realm node from the namespace tree to which you wish to add the interface, then in the right hand tabbed area there will be a tab labeled "Comms". Select it, and you will be presented with the "Shared Memory" tab:

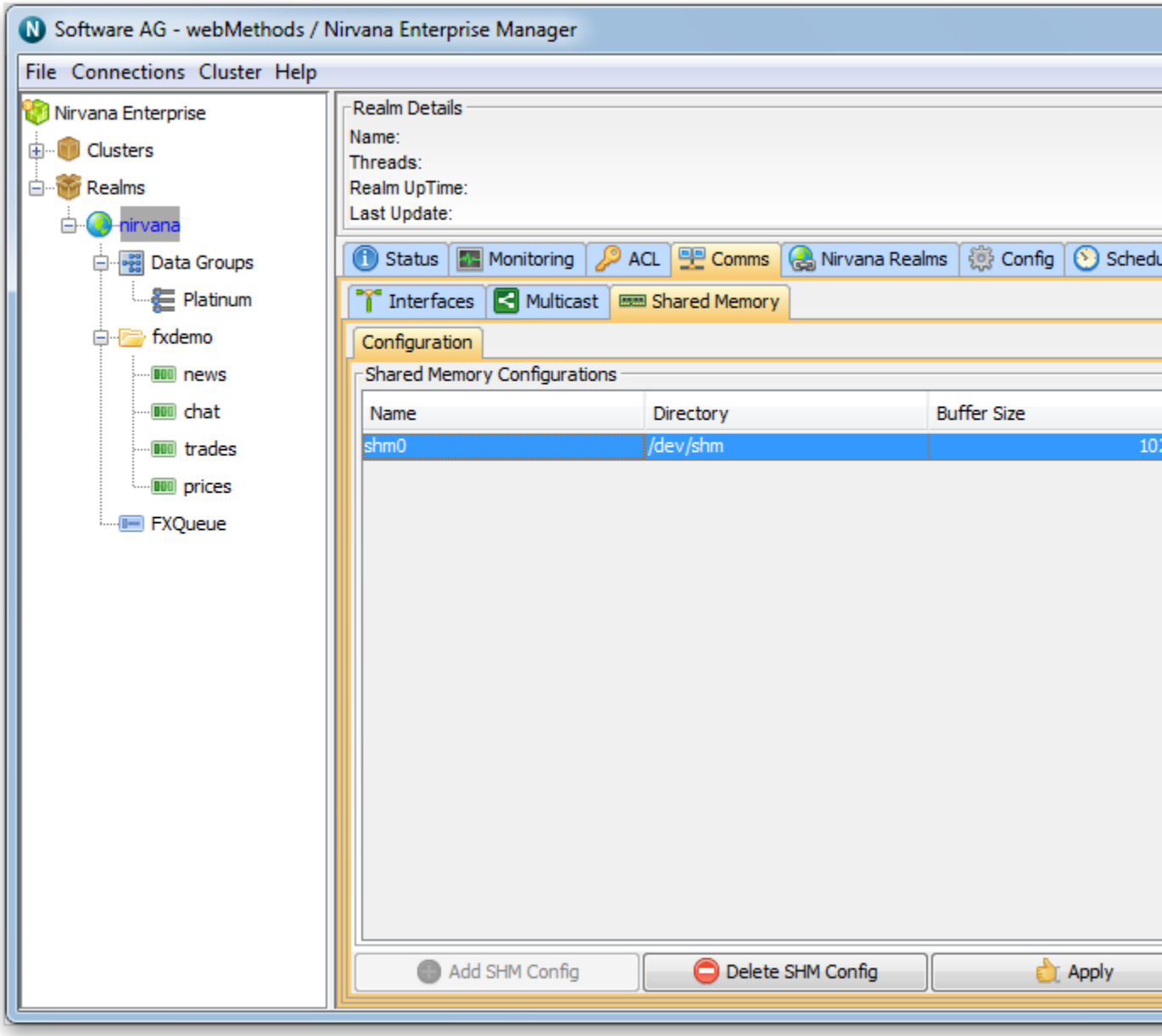


Now select "Add SHM Config" and you be presented with the below configuration box, it contains three parameters



1. Path: This is the directory within which the files needed for SHM communication will be created. (Please note that when choosing a path, ensure that the local user id of the server can access this directory, for example, /dev/shm will require root / super user access, or shm communication will not work)
2. Buffer Size: This is the size of the allocated memory in bytes a connection will use, it will also create a file of the same size which is used for mapping.
3. Timeout: This is the idle timeout for a connection, if no activity is detected on the connection it is closed.

Once you press okay the driver is created and ready to go. If you wish to edit any of those values you can edit them by double clicking any on the field you wish to change and then applying them with the apply button or resetting them by pressing cancel.

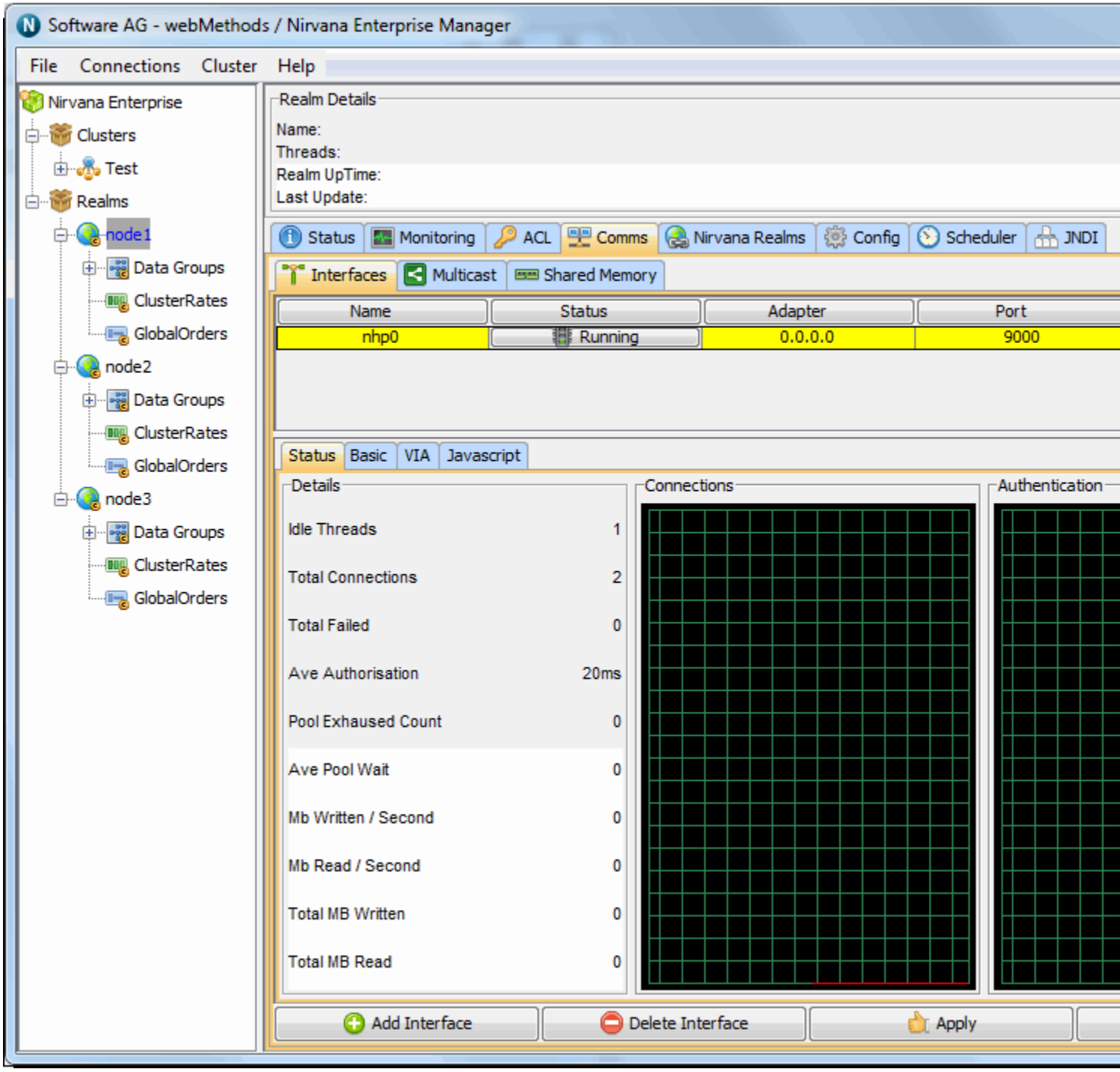


Creating an SSL network interface to a Universal Messaging Realm Server

Network Interfaces can be added to a Universal Messaging realm using the Universal Messaging Administration API or by using the Universal Messaging Enterprise Manager.

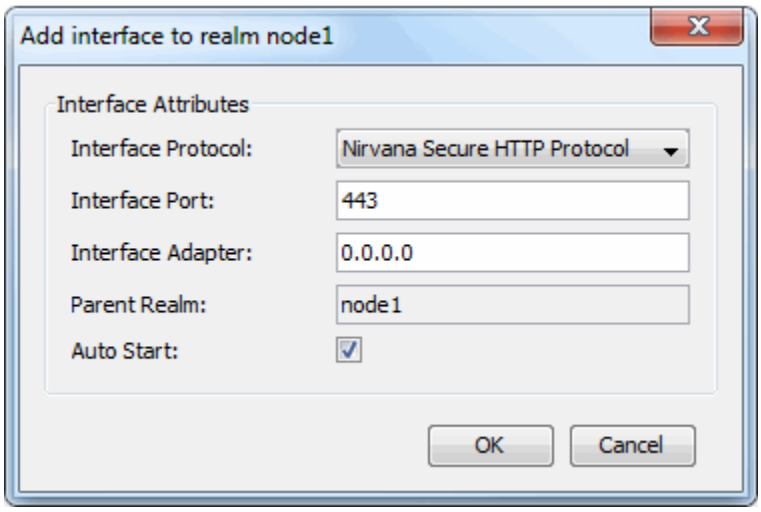
To add an SSL interface using the Enterprise Manager GUI, follow the steps below:

Step 1: Click on the interfaces panel for a realm. In the example below an interface is being added to the realm "node1". An interface could also be added however to any other realm shown in the Enterprise Manager. This ability makes centralized remote administration very easy using Universal Messaging.



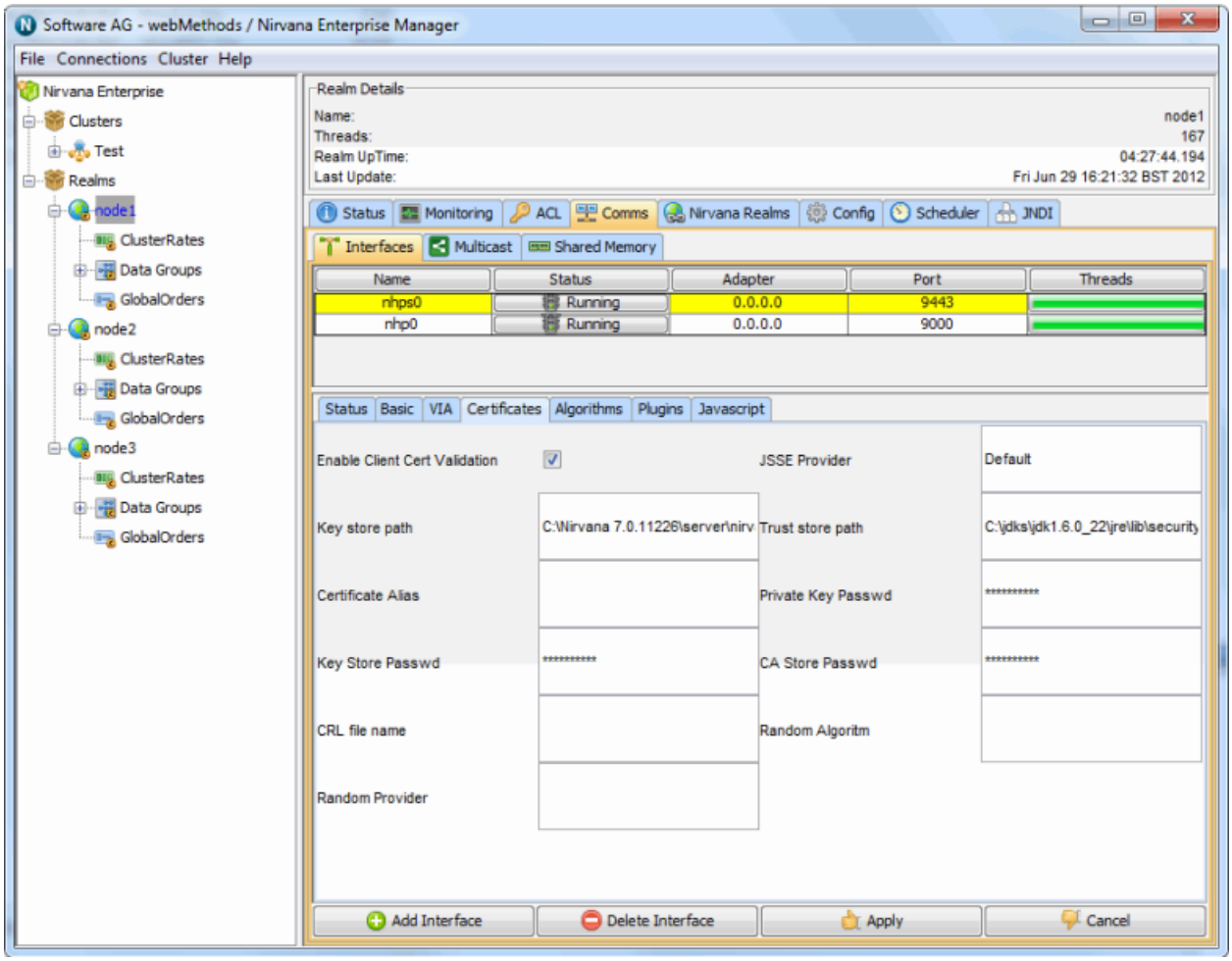
Step 2: Click on the Add Interface button in order to bring up the Add Interface dialog box. In the dialog choose the network protocol you would like to use for this interface. The choices are Sockets, Secure Sockets, HTTP and HTTPS. Choose either Secure Sockets or Secure HTTPS to add an SSL interface.

In this example HTTPS is chosen as the protocol and the interface is added to the network adapter 192.168.1.5. This will run the network interface on that IP Address. Alternatively, you could add a hostname that will resolve to the IP address of the chosen interface, or you can also specify 127.0.0.1 for localhost or 0.0.0.0 for all network interfaces on this machine.



Step 3: When a new interface is added, if the Auto Start option is not selected the realm interface will not start automatically when a realm starts, and it will have to be started manually.

After the interface has been added you should see the following in your interfaces panel:



In this example you can see that this realm now has 2 network interfaces and that the one just added ("nhps0") has been started.

If you did not choose to start the interface automatically, then in order to start the interface you need to click on the line containing the stopped traffic light. This will populate the tabs at the bottom with details for this interface.

Click on the Certificates tab. You will see that the first 2 text boxes have been automatically filled in. In the Universal Messaging download, we provide a utility called Certificate Generator (see "How to generate certificates for use" on page 301) that can generate sample .jks files containing certificates bound to localhost, for the server, the client and the truststore used by jsse. In this example we are going to use the sample jks files in order to demonstrate creating an SSL interface.

If you would like instructions on generating your own certificates (see "How to generate certificates for use" on page 301) for use with Universal Messaging please see our FAQ.

The text field titled 'Key store path' should contain something similar to:

```
c:\Universal Messaging\server\umserver\bin\server.jks
```

which should be the path to the sample Java keystore for the server, bound to localhost. The text field 'Trust store path' should contain something similar to the following:

```
c:\Universal Messaging\server\umserver\bin\nirvanacacerts.jks
```

Next, fill in the entries for the 'Key Store Passwd' and 'CA Store Passwd' with 'password'. This is the password for both the server keystore and the CA (truststore) keystore.

Next select the 'Basic' tab and click on the autostart interface checkbox. Clicking on this box means that the interface will be started automatically when the Universal Messaging realm server is started.

Note:
If you intend to use an SSL interface for inter-realm communication, you need to ensure that the option "Allow for InterRealm" is selected and the option "Allow Client Connections" is deselected. Similarly, If you intend to use an SSL interface for communication between clients and the realm, you need to ensure that the option "Allow for InterRealm" is deselected and the option "Allow Client Connections" is selected. Related information is available in the section "Setting up Inter-Realm Communication" on page 119.

Then click on apply and the Interface will be started.

Alternatively if you do not wish to autostart then double click on the line with the stopped traffic light. This will bring up a dialog which allows you to start that network interface.

If the network interface fails to start then please inspect the Universal Messaging log file via the messages tab. Please contact your software supplier if any other issues arise.

Similarly, if you wish to stop an interface, simply double-click on the interface you want to stop from the interface table, and click on the 'stop' button.

There is no limit to the number of network interfaces that can be added to a realm and each can have its own configuration such SSL chains etc applied. This allows you to isolate customers from each other while still using only one Universal Messaging realm server.

In this example we have used our own sample Java keystores which will only work when using the loopback interface of your realm server host. If you wish to provide SSL capabilities for remote connections, you must ensure you have your own keystores and valid certificate chains.

Connecting to an NHPS Interface

To connect to an nhps interface on a Universal Messaging server in the Enterprise Manager, you configure the following truststore and client keystore properties in the *Software AG_directory* \UniversalMessaging\java*instance_name*\bin\Admin_Tools_Common.conf file of the server:

- `set.default.CAKEYSTORE=<path_to_truststore>` - Required.
- `set.default.CAKEYSTOREPASSWD=<truststore_password>` - Required.
- `set.default.CKEYSTORE=<path_to_client_keystore>` - Required only when client authentication is enabled.
- `set.default.CKEYSTOREPASSWORD=<keystore_password>` - Required only when client authentication is enabled.

The certificates must be in .jks (java keystore) format.

Important:
If you have these properties configured both in the *Software AG_directory* \UniversalMessaging\java*instance_name*\bin\nenterprisemgr.conf file and Admin_Tools_Common.conf file, the values in nenterprisemgr.conf override the values in Admin_Tools_Common.conf. Software AG recommends that you configure the properties in the Admin_Tools_Common.conf file.

In addition, optionally, you can configure an nhps url to which clients connect by default. You specify the url as a value of the -DRNAME property in the nenterprisemgr.conf file of the server, for example:

```
wrapper.java.additional.3=-DRNAME=nhps://umserver:8000
```

Enabling Client Authentication

You use the **Enable Client Cert Validation** check box on the **Interfaces > Certificates** tab to enable or disable client authentication for an nhps interface on a Universal Messaging server. If you enable client authentication, you must specify the client keystore certificate and kesystore password as properties in the Admin_Tools_Common.conf file of the server instance.

How to generate certificates for use

Generating Demo / Development certificates

In order to generate a demo SSL certificate you can use the Java keytool utility or the Universal Messaging Certificate Generator utility.

Note:

The Certificate Generator utility is deprecated in Universal Messaging v10.2 and will be removed in a future version of the product.

The third-party Java keytool utility can be used to create and handle certificates. Keytool stores all keys and certificates in a keystore.

The Universal Messaging Certificate Generator utility can be used to generate a self signed server certificate, a self signed client certificate and a trust store for the above two.

You can run the Certificate Generator from the Start Menu on Windows by selecting the server/<realm name>/Create Demo SSL Certificates

Alternatively you can open a server Command Prompt and run the utility as required for your platform:

- Windows 32-bit systems:

  CertificateGenerator.exe

- UNIX-based systems:

  ./CertificateGenerator

- OS X:

  ./CertificateGenerator.command

This will generate 3 files:

- client.jks : Self signed certificate you could use if you have client certificate authentication enabled.
- server.jks : Self signed certificate with a CN=localhost . Please note: You can only connect to interfaces using this by specifying a localhost RNAME due to the HTTPS protocol restrictions.
- nirvanacacerts.jks: Keystore that contains the public certificate part of the 2 key pairs above. This should be used as a trust store by servers and clients.

It is also possible to customize some elements of these certificates stores such as the password, the host bound to the server CN attribute and they key size.This can be done by passing the following optional command line arguments to the Certificate Generator:

- Windows 32-bit systems:

  CertificateGenerator.exe <password> <host> <key size>

- UNIX-based systems:

  ./CertificateGenerator <password> <host> <key size>

- OS X:

  ./CertificateGenerator.command <password> <host> <key size>

Generating Production certificates

In order to obtain a real SSL Certificate, you must first generate a CSR (Certificate Signing Request). A CSR is a body of text that contains information specific to your company and domain name. This is a public key for your server.

The Java keytool utility can be used to create and handle certificates. Keytool stores all keys and certificates in a keystore. For a detailed description of keytool please see its documentation.

Step 1: Create a keystore

Use the keytool to create a keystore with a private/public keypair.

```
keytool -genkey -keyalg "RSA" -keystore keystore -storepass password -validity 360
```

You will be prompted for information about your organisation. Please note that when it asks for "User first and last name", please specify the hostname that Universal Messaging will be running on (e.g. www.yoursite.com).

Step 2: Create a certificate request

Use the keytool to create a certificate request.

```
keytool -certreq -keyalg "RSA" -file your.host.com.csr -keystore keystore
```

This will generate a file containing a certificate request in text format. The request itself will look someting like this :

```
-----BEGIN NEW CERTIFICATE REQUEST-----
 MIIBtTCCAR4CAQAwdTELMAkGA1UEBhMCVVMxDzANBgNVBAgTBmxvbmRvbjEPMA0GA1UEBxMGbG9u
 ZG9uMRQwEgYDVQQKEwtteS1jaGFubmVsczEMMAoGA1UECxMDYml6MSAwHgYDVQQDExdub2RlMjQ5
 Lm15LWNoYW5uZWxzLmNvbTCBnzANBeddiegkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAycg0MJ7PXkQM9sLj
 1vWa8+7Ce0FDU4tpcMXlL647dwok3uUGXuaz72DmFtb8Oninjawingsjxr MBDK9fXG9hqfDvxWGyU0DEgbn+Bg

O3XqmUbyI6eMzGdf0vTyBFSeQIinigomontoyaU9Ahq1T7C6zlryJ9n6XZTW79E5UcbSGjoNApBOgVOCPKBs7/CR
 hZECAwEAAaAAMA0GCSqGSIb3DQEBBAUAA4GBAB7TkFzQr+KvsZCV/pP5IT0c9tM58vMXkds2J7TY
 Op3AueMVixRo14ruLq1obbTudhc385pPgHLzO7QHEKI9gJnM5pR9yLL72zpVKPQ9XOImShvO05Tw
 0os69BjZeW8LTV60v4w3md47IeGE9typGGxBWscVbXzB4sgVlv0JtE7b
 -----END NEW CERTIFICATE REQUEST-----
```

Step 3: Submit your certificate request to a certificate supplier

Certificate vendors will typically ask you to paste the certificate request into a weborder form. This will be used as a public key to generate you private key. Please include the (BEGIN and END) tags when you paste the certificate request.

Please note that a cert of PKCS #7 format is required so that it can be imported back into keytool. (step 4)

The certificate vendor will then provide you with a certificate which that will look something like this:

Please paste this certificate into a file called your.host.com.cer [Note. please include the (BEGIN and END) tags]

```
-----BEGIN PKCS #7 SIGNED DATA-----
```

```
MIIFpAYJKoZIhvcNAQcCoIIFlTCCBZECAQExADALBgkqhkiG9w0BBwGgggV5MIIC
2DCCAkGgAwIBAgICErYwDQYJKoZIhvcNAQEEBQAwgYcxCzAJBgNVBAYTAlpBMSIw
IAYDVQQIExlGT1IgVEVTVElORyBQVVJQT1NFUyBPTkxZMR0wGwYDVQQKExRUaGF3
dGUgQ2VydGlmaWNhdGlvbjEXMBUGA1UECxMOVEVTVCBURVNUIFRFU1QxHDAaBgNV
BAMTE1RoYXd0ZSBUZXN0IENBIFJvb3QwHhcNMDQwOTA2MTYwOTIwWhcNMDQwOTI3
MTYwOTIwWjB1MQswCQYDVQQGEwJVUzEPMA0GA1UECBMGbG9uZG9uMQ8wDQYDVQQH
EwZsb25kb24xFDASBgNVBAoTC215LWNoYW5uZWxzMQwwCgYDVQQLEwNiaXoxIDAe
BgNVBAMTF25vZGUyNDkubXktY2hhbm5lbHMuY29tMIGfMA0GCSqGSIb3DQEBAQUA
A4GNADCBiQKBgQDJyDQwns9eRAz2wuPW9Zrz7sJ7QUNTi2lwxeUvrjt3CiTe5QZe
5rPvYOYW1vw6PGswEMr19cb2Gp8O/FYbJTQMSBuf4GA7deqZRvIjp4zMZ1/S9PIE
VJ5AhT0CGrVPsLrOWvIn2fpdlNbv0TlRxtIaOg0CkE6BU4I8oGzv8JGFkQIDAQAB
o2QwYjAMBgNVHRMBAf8EAjAAMDMGA1UdHwQsMCowKKAmoCSGImh0dHA6Ly93d3cu
dGhhd3RlLmNvbS90ZXN0Y2VydC5jcmwwHQYDVR0lBBYwFAYIKwYBBQUHAwEGCCsG
AQUFBwMCMA0GCSqGSIb3DQEBBAUAA4GBAHGPR6jxU/h1U4yZGt1BQoydQSaWW48e
r7slod/2ff66LwC4d/fymiOTZpWvbiYFH1ZG98XjAvoF/V9iNpF5ALfIkeyJjNj4
ZryYjxGnbBa77GFiS4wvUk1sngnoKpaxkQh24t3QwQJ8BRHWnwR3JraNMwDWHM1H
GaUbDBI7WyWqMIICmTCCAgKgAwIBAgIBADANBgkqhkiG9w0BAQQFADCBhzELMAkG
A1UEBhMCWkExIjAgBgNVBAgTGUZPUiBURVNUSU5HIFBVUlBPU0VTIE9OTFkxHTAb
BgNVBAoTFFRoYXd0ZSBDZXJ0aWZpY2F0aW9uMRcwFQYDVQQLEw5URVNUIFRFU1Qg
VEVTVDEcMBoGA1UEAxMTVGhhd3RlIFRlc3QgQ0EgUm9vdDAeFw05NjA4MDEwMDAw
MDBaFw0yMDEyMzEyMTU5NTlaMIGHMQswCQYDVQQGEwJaQTEiMCAGA1UECBMZRk9S
IFRFU1RJTkcgUFVSUE9TRVMgT05MWTEdMBsGA1UEChMUVGhhd3RlIENlcnRpZmlj
YXRpb24xFzAVBgNVBAsTDlRFU1QgVEVTVCBURVNUMRwwGgYDVQQDExNUaGF3dGUg
VGVzdCBDQSBSb290MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC1fZBvjrOs
fwzoZvrSlEH81TFhoRPebBZhLZDDE19mYuJ+ougb86EXieZ487dSxXKruBFJPSYt
tHoCin5qkc5kBSz+/tZ4knXyRFBO3CmONEKCPfdu9D06y4yXmjHApfgGJfpA/kS+
QbbiilNz7q2HLArK3umk74zHKqUyThnkjwIDAQABoxMwETAPBgNVHRMBAf8EBTAD
AQH/MA0GCSqGSIb3DQEBBAUAA4GBAIKM4+wZA/TvLItldL/hGf7exH8/ywvMupg+
yAVM4h8uf+d8phgBi7coVx71/lCBOlFmx66NyKlZK5mObgvd2dlnsAP+nnStyhVH
FIpKy3nsDO4JqrIgEhCsdpikSpbtdo18jUubV6z1kQ71CrRQtbi/WtdqxQEEtgZC
JO2lPoIWMQA=
-----END PKCS #7 SIGNED DATA-----
```

Step 4: Store the certificate in your keystore

Use the keytool to store the generated certificate :

```
keytool -keystore keystore -keyalg "RSA" -import -trustcacerts -file your.host.com.cer
```

Once step 4 is completed you now have a Universal Messaging server keystore and can add an SSL interface (see "Creating an SSL network interface to a Universal Messaging Realm Server" on page 297).

Please note that if you completed steps 1 to 4 for test certificates then you will also need to create a store for the CA root certificate as Universal Messaging will not be able to start the interface until it validates where it came from. Certificate vendors typically provide test root certificates which are not recognized by browsers etc. In this case you will need to add that cert to another store and use that as your cacert. When specifying certificates for a Universal Messaging SSL interface this would be specified as the Trust Store Path in the certificates tab.

If you are using anonymous SSL then you will have to provide this cacert to clients also as this will not be able to validate the Universal Messaging certificate without it. Please see the Security section of our Concepts guide for more information on configuring Universal Messaging clients to use certificates.

Plugins

The Universal Messaging Realm Server supports the concept of Plugins within the context of the NHP or NHPS protocols. The plugins are initiated when the underlying Universal Messaging driver receives an HTTP/S packet which is not part of the standard Universal Messaging protocol. At this point it passes the request over to the Plugin Manager to see if there is any registered plugin interested in the packet's URL. If there is, then the request is forwarded to this plugin for processing. There are several plugins supported by Universal Messaging. Please see below for available documentation:

- "File Plugin" on page 306
- "XML Plugin" on page 310
- "Proxy passthrough Plugin" on page 314
- "REST Plugin" on page 316
- "Servlet Plugin" on page 335

Note:
The following server plugins are deprecated in Universal Messaging v10.2 and will be removed in a future version of the product: *Graphics, XML, Proxy passthrough, Servlet*.

Configuration

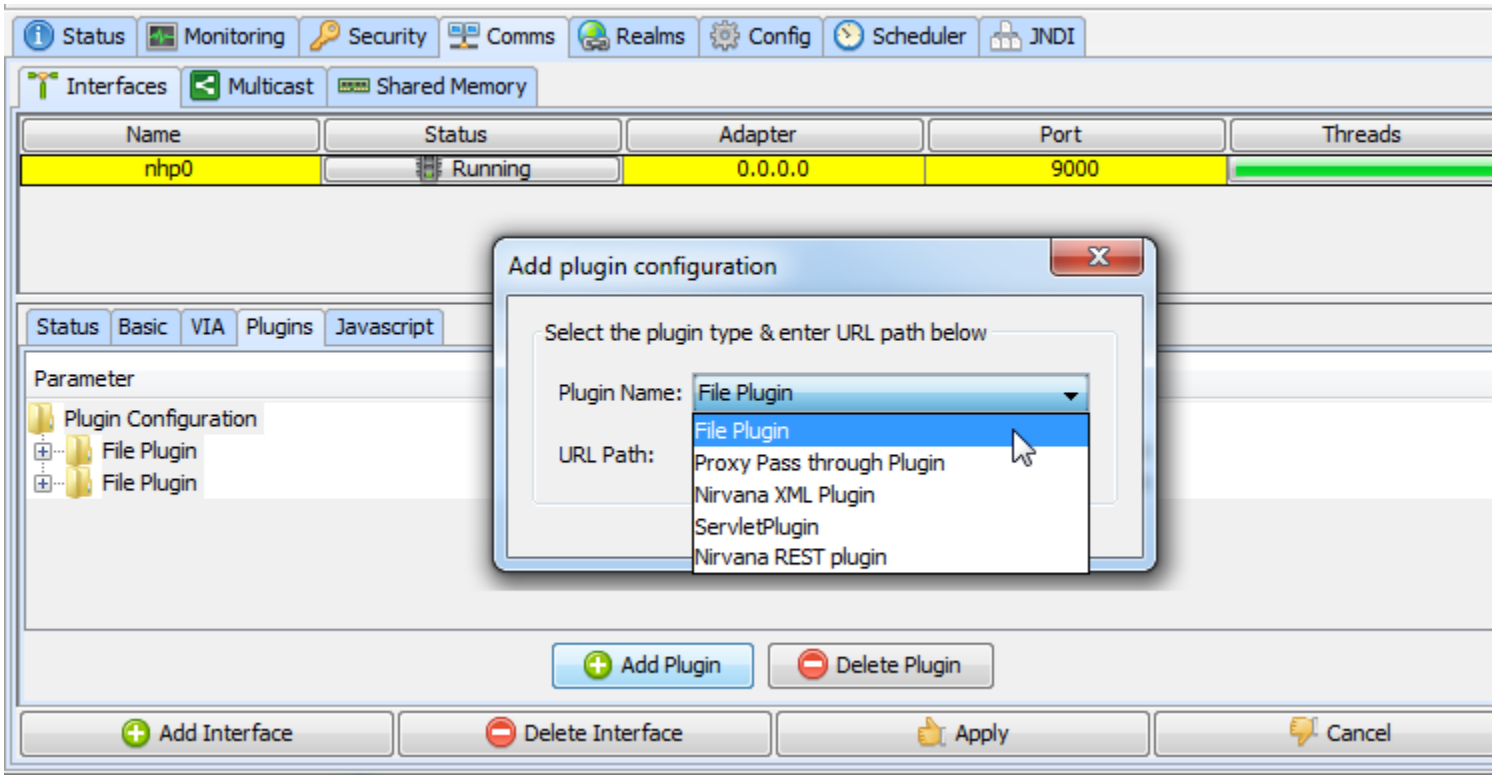
Configuration of a plugin can be done programmatically with the Administration API supplied with Universal Messaging or it can be done with the Enterprise Manager application. For the rest of this document the Enterprise Manager will be used to describe how to set up and use the plugins.

In order to add a plugin, first of all you need to have created the nhp or nhps interface (see "Creating Interfaces" on page 278) that will use the plugin within the realm where you wish to run the plugin.

Once the interface is created, proceed as follows to access the plugin configuration dialog:

1. In the navigation frame, select the realm where you want to add the plugin.
2. In the **Realm Details** frame, navigate to the list of defined interfaces for the realm, using **Comms > Interfaces**.
3. Select the interface from the table of configured interfaces.
4. Select the tab **Plugins** from the interface configuration panel.
5. Click **Add Plugin**. This displays the plugin configuration dialog, which enables you to choose which plugin you wish to add.

The diagram below shows a new file plugin about to be added to the known interface `nhp0`.



URL Path

When you configure a plugin, you are required to add a URL Path. The URL Path is what the realm server uses to determine if the request if destined for a plugin. If the server name and path within the URL supplied in the plugin configuration dialog match the server name and path within the request to a configured plugin, then this request is passed to the correct configured plugin for processing.

For example:

If a request with a URL of `http://realmServer/pluginpath/index.html` is made to the server, the file path will be extracted, i.e. `pluginpath/index.html`, and the configured plugins will be scanned for a match. If there is a file plugin configured with a URL Path `pluginpath`, then this plugin will get a request for `index.html`.

Similarly:

If a request with a URL of `http://realmServer/pluginpath/pictures/pic1.jpg` is received, then the same file plugin would get a request for `pictures/pic1.jpg`.

File Plugin

The file plugin enables the Universal Messaging realm server to serve static web pages. This can be used for example to have the realm server serve applets and supported files without the need for a dedicated web server. For example, if you are running a file plugin on your realm server host called `webhost`, on an nhp interface running on port 80, you could type in a URL within a web

browser `http://webhost:80/index.html` which will return the index page defined within the file plugin's base path directory.

This enables the realm server to act as a web server and can even be used to serve applets to client browsers that may directly communicate with the realm server and publish and consume events from channels.

Important:
The root file directory that the file plugin points to using the `BasePath` parameter can be any disk location. All files under that location are potentially visible to any HTTP client that can connect to the Universal Messaging realm server. We would recommend that you do not point the file plugin to a directory that contains any sensitive data, without also configuring suitable access controls. These could be at the network level (restricting network access to the server), in the file plugin configuration (it supports HTTP basic authentication with a username/password file) or by using file permissions at the Operating System level (so that sensitive data cannot be read by the realm server process). Or of course a combination of these.

Configuration

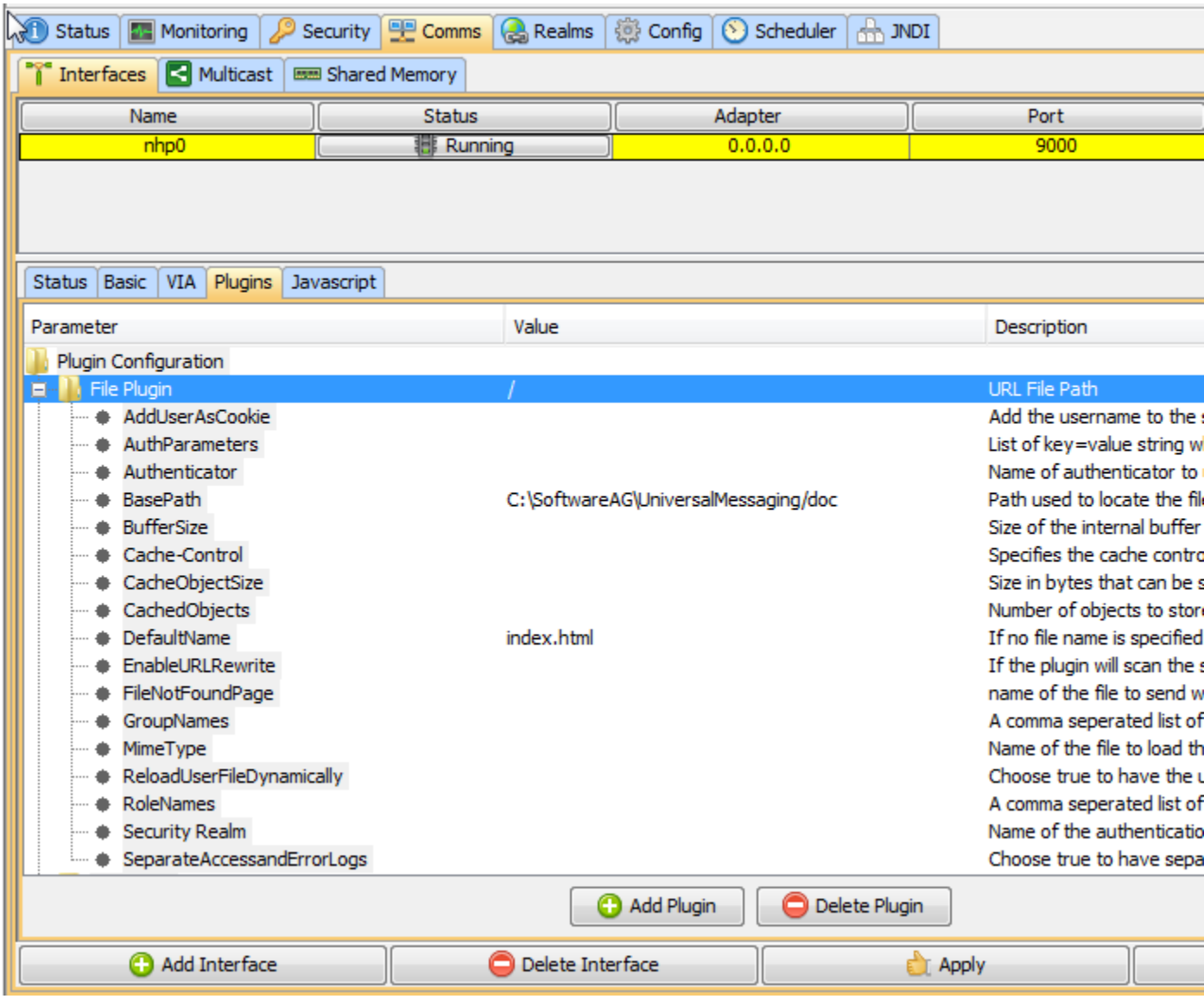
Once you have created the file plugin on the interface you require it on, you can then select it from the **Plugins** panel for the selected interface and enter values as you wish for the configuration parameters.

The file plugin requires configuration information defining its behavior as well as the location of the files it is required to serve to the clients. Below is a table that shows each configuration parameter and describes what it is used for.

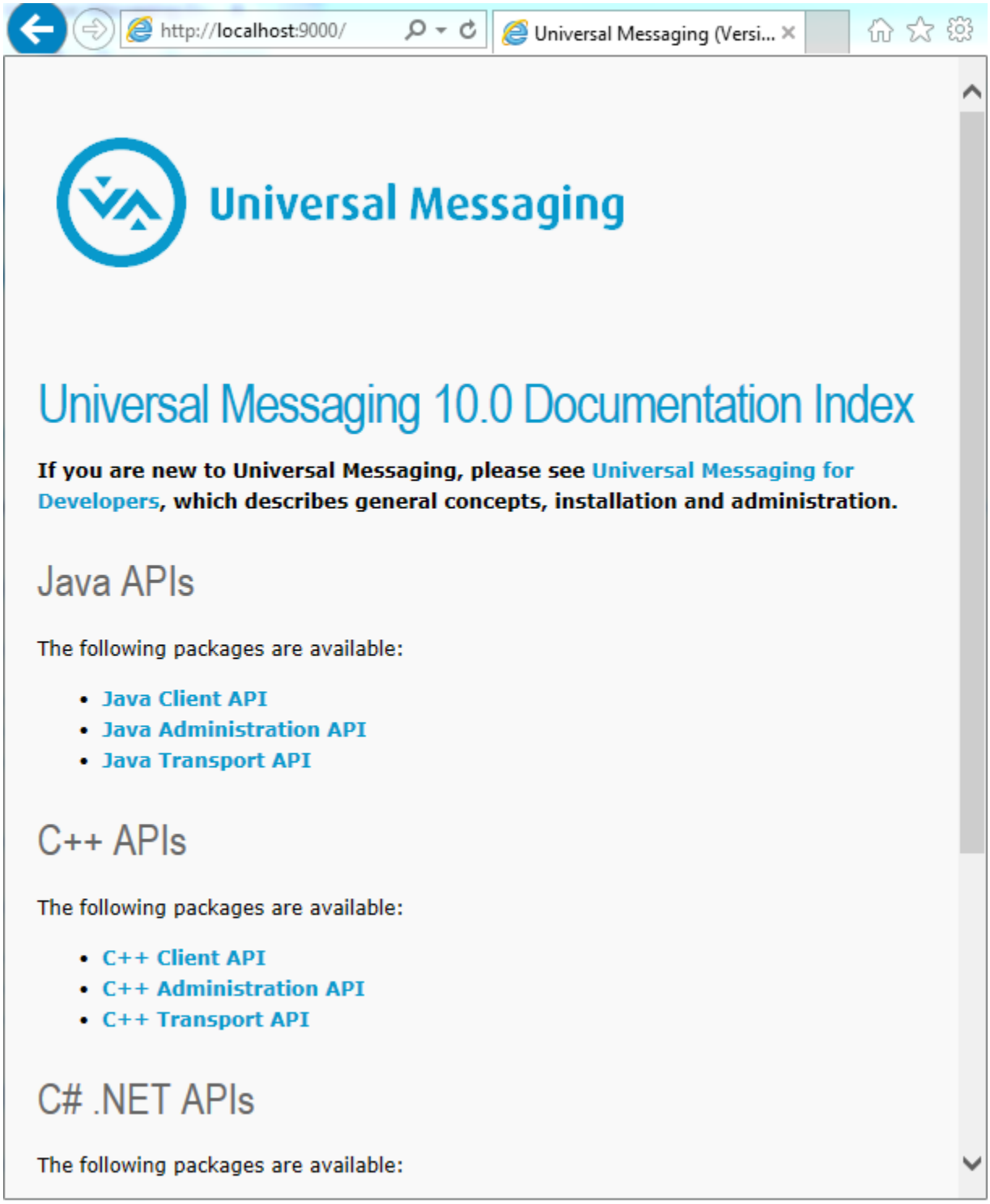
Parameter Name	Description	Default Value
BufferSize	Size of the internal buffer to use to send the data.	1024
BasePath	Path used to locate the files.	The `UniversalMessaging/doc` directory under the product installation directory.
DefaultName	If no file name is specified which file should be returned.	index.html
FileNotFoundPage	Name of the file to send when file cannot be located	None.
UserFile	Name of the file containing the usernames and passwords.	None.
Security Realm	Name of the authentication realm	None.
MimeType	Name of the file to load the mime type information from. The format of this file is : <mimetype> <fileExtension>	Built in types used.

Parameter Name	Description	Default Value
CachedObjects	Number of objects to store in the cache	100
CacheObjectSize	Size in bytes that can be stored in the cache	20K
SeparateAccessandErrorLogs	Choose true to have separate log files for the access and error logs.	FALSE

The image below shows the Enterprise Manager **Interfaces** panel with an nhp interface running on port 9000. This interface has a file plugin configured with the default settings and its URL path is `/`. The default BasePath setting is the `UniversalMessaging/doc` directory in the file hierarchy for your local product installation, which is where the default product installation places the Universal Messaging API docs. Once the plugin is created, you can click the **Apply** button which will restart the interface and enable the new file plugin.



From a browser, it is now possible to enter the URL `http://localhost:9000/` which will then render the default `index.html` page from the `UniversalMessaging/doc` directory for the API docs. The image below demonstrates the browser view from a realm that has a file plugin on an nhp interface on port 9000, and displaying the default API docs found in the `UniversalMessaging/doc` directory.



XML Plugin

The XML Plugin can be used to query the realm server, its queues and channels. It returns the data in XML format. This plugin also supports style sheets, so the XML can be transformed into HTML or any format required. For example, a client can publish XML data onto a Universal

Messaging realm's channel, then using a standard web browser, get the server to transform the XML into HTML via a stylesheet, thereby enabling the web browser to view events on the realm.

This functionality enables realm data to be viewed from a channel without any requirement for a Java client. All that is required is for the client to have a browser.

Important:
Never include XSL code from untrusted sources into the plugin's XSL code, as this can lead to a security risk for the client browser (or other client application) accessing the plugin. The Universal Messaging realm server itself is not at risk, since it does not execute the plugin's XSL code.

Configuration

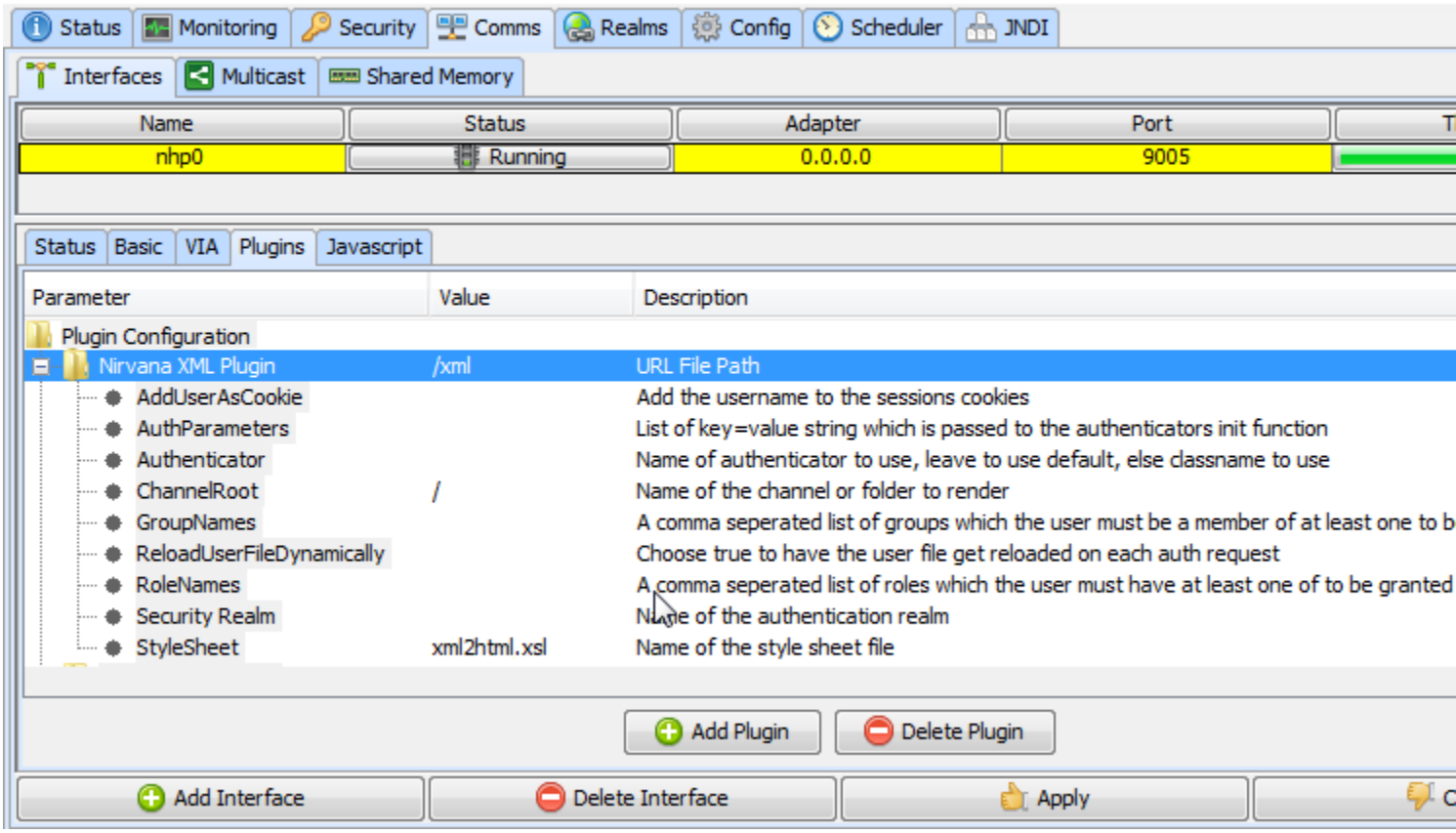
Once you have created the XML plugin on the interface you require it on, you can then select it from the **Plugins** panel for the selected interface and enter values as you wish for the configuration parameters.

The XML plugin requires configuration information relating to its behavior as well as the entry point in the namespace for the channels you wish to make available to serve to the clients. Below is a table that shows each configuration parameter and describes what it is used for.

Parameter Name	Description	Default Value
ChannelRoot	Name of the channel or folder to render.	/
Security Realm	Name of the authentication realm	None.
StyleSheet	Name of the style sheet file to use to process the resulting XML.	None. If you specify a filename without a path, the default path is `UniversalMessaging/server/<InstanceName>/bin` under the product installation root location.
UserFile	Name of the file containing the usernames and passwords	None.

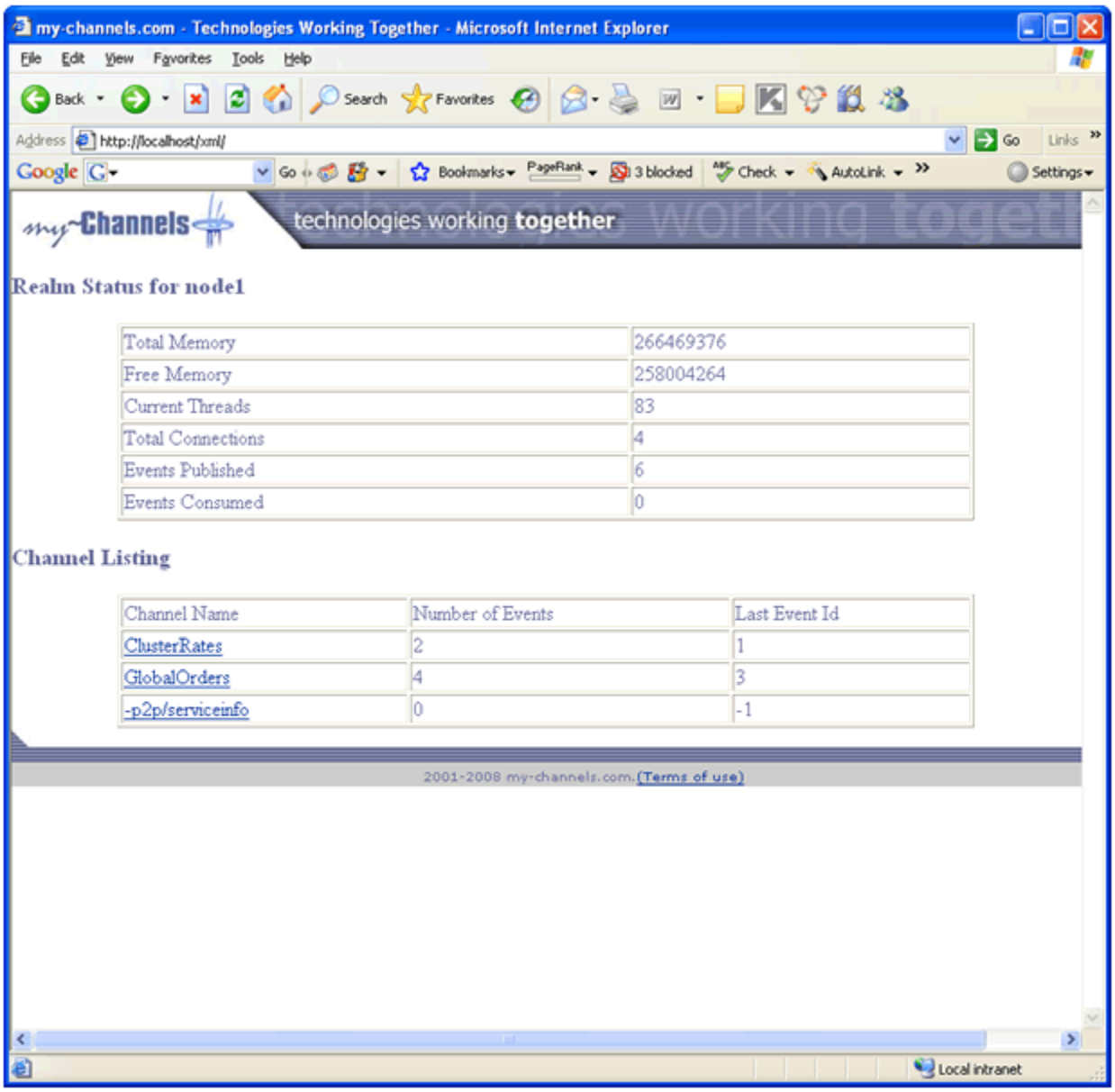
Note:
As a starting point for creating your own stylesheet, you can use the stylesheet `xml2html.xsl` that is supplied in the `UniversalMessaging/doc/style` directory in the file hierarchy for your local product installation.

The image below shows the Enterprise Manager **Interface** panel with an nhp interface running on port 9005. This interface has an XML plugin configured to use the `xml2html.xsl` stylesheet and its URL path as `/xml`. The default ChannelRoot setting is `/`, which is the root of the namespace, i.e. all channels. Once the plugin is created, you can click the **Apply** button which will restart the interface and enable the new XML plugin.

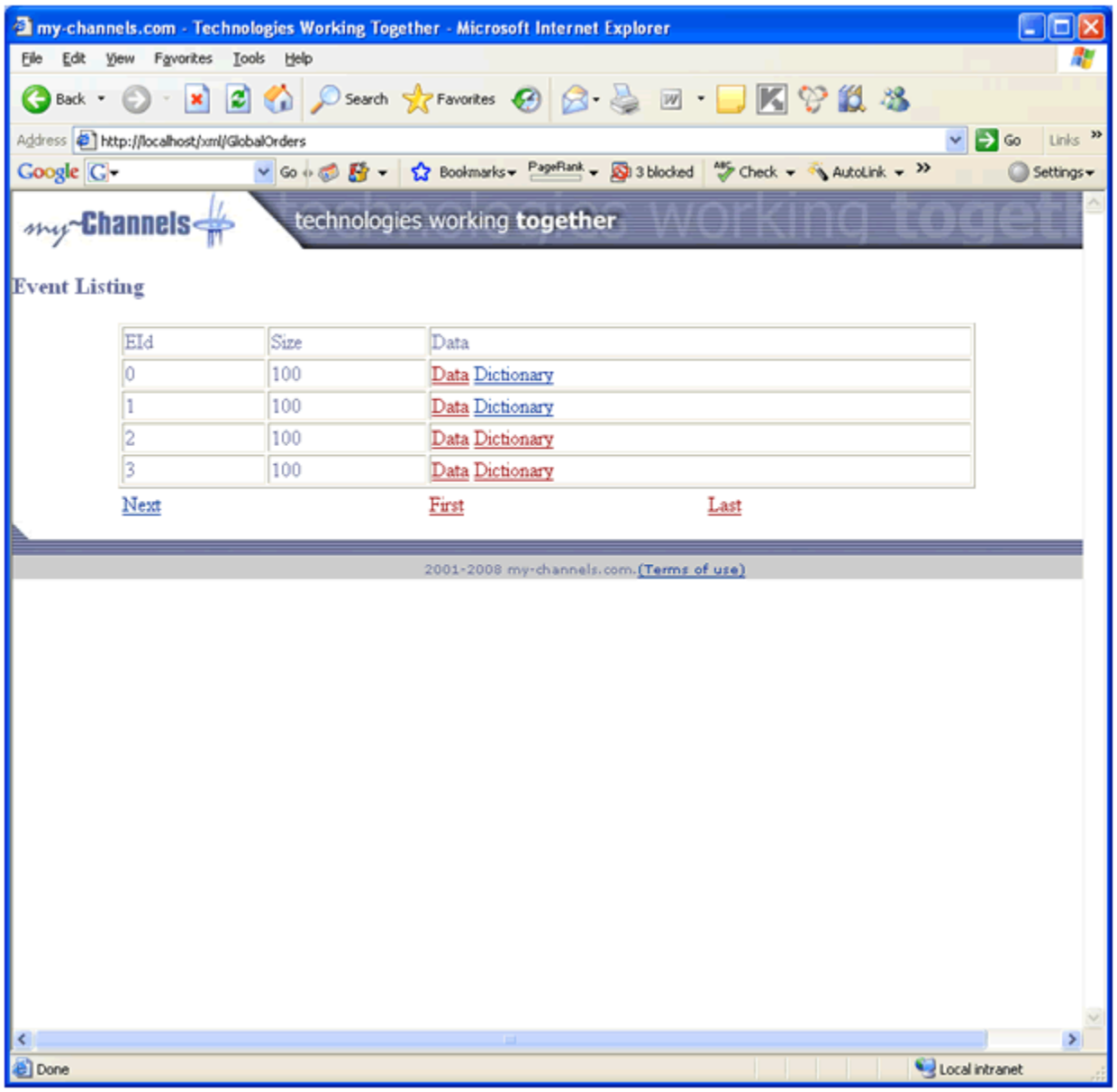


From a browser, you can now enter the URL `http://localhost:9005/xml/` which will render the realm information page using the stylesheet. The image below demonstrates the browser view from a realm that has an XML plugin on an nhp interface on port 9005.

If you use the stylesheet `xml2html.xsl` delivered with the product, the result will be similar to the display below. The realm information is displayed at the top of the page, and the information on resources is shown beneath.



As you can see above, each resource is shown as a link within a table showing the information obtainable from the XML plugin. Clicking on a channel link will then take you to another page that has been rendered by the XML plugin which will show you the list of events on a channel. The image below shows the event list for the JNDI naming channel.



The XML plugin will determine whether the events on the channel contain byte data, dictionaries or XML documents and return the relevant elements within the XML document. The stylesheet applied to the XML document then examines each element to find out how to render it within the browser. Each event on the channel or queue is shown in the table with event ID, its size in bytes and links to either the byte data, the dictionary or the XML data. These links are generated by the stylesheet. Clicking on the data or dictionary links will again return an XML document from the XML plugin that will be rendered to show either the base64 encoded event data or the event dictionary.

If any events contain XML documents, these will be returned directly from the XML plugin. The stylesheet provided will not render event XML documents, since the structure of these is unknown. You will need to provide your own stylesheet to render your own XML event documents.

Proxy Passthrough Plugin

The Proxy Passthrough Plugin can be used to forward http(s) requests from specific URLs to another host. For example, if you want to forward requests from one realm to another realm, or to another web server, you can use the proxy passthrough plugin.

This functionality enables realms to act as a proxy to forward URL requests to any host that accepts http(s) connections.

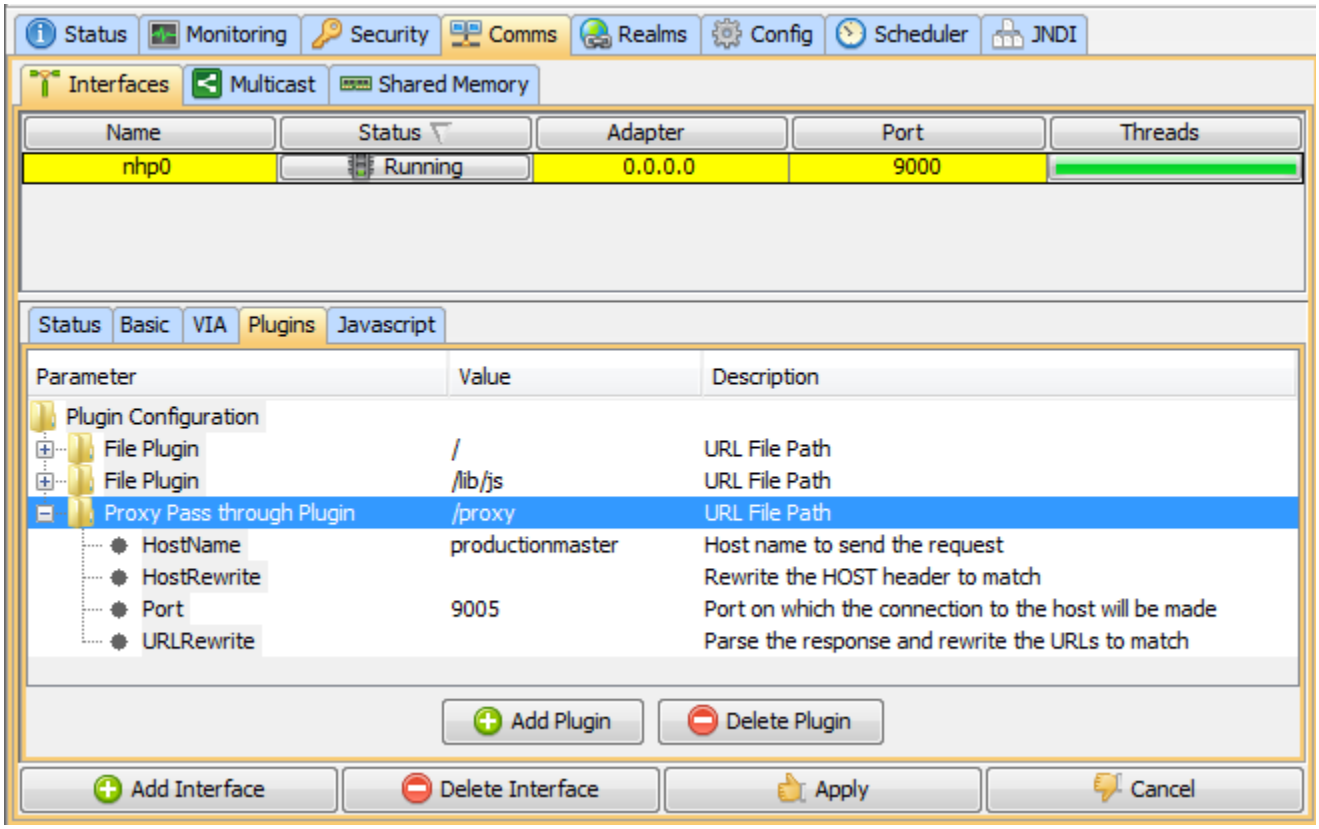
Configuration

Once you have created the proxy passthrough plugin on the interface you require it on, you can then select it from the plugins panel for the selected interface and enter values as you wish for the configuration parameters.

The proxy passthrough plugin requires configuration information relating to the host and port that requests will be forwarded to. Below is a table that shows each configuration parameter and describes what it is used for.

Parameter Name	Description	Default Value
HostName	Host name of the process that requests for the URL will be forwarded to	
Port	Port on which the requests will be sent to the host	80

The image below shows the Enterprise Manager **Interface** panel with an nhp interface running on port 9000. This interface has a proxy passthrough plugin configured to redirect requests from this interface using the URL path of `/proxy` and will forward these requests to any File Plugins and XML Plugins located on the `productionmaster` realm's nhp interface running on port 9005.



From a browser, it is now possible to enter the URL `http://localhost:9000/proxy/` which will redirect this request to the interface on the `productionmaster` realm interface running on port 9005. This will display the details of the `productionmaster` realm as if you had specified the URL `http://productionmaster:9005/` in your browser.

REST Plugin

The REST plugin allows access to the Universal Messaging REST API, and can be enabled on any HTTP or HTTPS (NHP or NHPS) interface. The Universal Messaging REST API is designed for publishing, purging and representing events published on channels and queues in 2 initial representations: JSON and XML.

The Universal Messaging REST API supports three different HTTP commands. GET is used for representations of events, POST for publishing and PUT for purging. Both XML and JSON support byte arrays, XML and Dictionary events for publishing, which map to native Universal Messaging event types. There are two MIME types available: text and application.

Configuration

Once you have created the REST plugin on the interface you require it on, you can then select it from the plugins panel for that interface and enter values as desired for the configuration parameters.

Parameter Name	Description	Default Value
AddUserAsCookie	Add the username to the session's cookies.	Blank
AuthParameters	A list of key=value strings, which are passed to the Authenticator's init() function.	Blank
Authenticator	Classname of Authenticator to use. If blank, no authentication is used.	Blank
EnableStatus	Enables Realm status details. Default is disabled, for security reasons.	False
GroupNames	A comma separated list of groups. The user must be a member of at least one in order to be granted access.	Blank
IncludeTypeInfo	Includes type information for event dictionaries.	False
NamespaceRoot	Name of the namespace folder to be used as root.	Blank
ReloadUserFileDynamically	If set to true and authentication is enabled, fAuthenticator.reload() is called on each request.	True
RoleNames	A comma-separated list of names. The user must have at least one to gain access.	Blank
Security Realm	Name of the authentication realm.	Blank
SessionTimeout	Time in seconds to time-out inactive http sessions.	300

The REST plugin supports WADL documentation which is accessible through the HTTP OPTIONS command. Once you have completed setting up your REST plugin, you can verify it works by opening a browser to the NHP interface in the mount URL path, and appending the query string `?method=options`. For example, for an NHP interface running on port 9000 on localhost, and having the plugin mounted on "/rest", open a browser to `http://localhost:9000/rest/API?method=options`.

Following this will display an HTML version of the full Universal Messaging REST API documentation which is generated by applying an XSL processor to the WADL XML document. The XML document itself can be obtained by accessing the plugin URL without the `?method=options` query string. For example, the curl command line tool can be used as follows:

```
curl -XOPTIONS http://localhost:9000/rest/API
```

What follows is a summary of the three HTTP commands for both XML and JSON, and what functionality each provides, as well as detailed examples of requests and responses for each command.

XML: GET

Provides XML representations of channels/queues or events in a channel or queue as specified by the ChannelOrQueue parameter. The parameter is represented by the URI Path following the REST Plugin mount.

If the value supplied corresponds to a Universal Messaging namespace container, the representation returned is a list of channels and queues present in the container. If the value supplied corresponds to a channel or queue then the representation returned is a list of events. Finally if the value supplied does not correspond to either a container or a channel / queue a 404 response will be returned with no body.

Available response representations:

- "text/xml" on page 319
- "application/xml" on page 319

XML: POST

Allows publishing of an event to a channel or queue specified by the ChannelOrQueue parameter, which is represented by the URI Path following the REST Plugin mount. For example `http://localhost:9000/rest/API/xml/testchannel` expects an XML byte, XML or dictionary event to be published to channel **testchannel**.

Acceptable request representations:

- "text/xml" on page 324
- "application/xml" on page 324

Available response representations:

- "text/xml" on page 328
- "application/xml" on page 328

XML: PUT

Allows purging of 1 or more events already published on a channel or queue specified by the ChannelOrQueue parameter, which is represented by the URI Path following the REST Plugin mount. For example `http://localhost:9000/rest/API/xml/testchannel` expects a request to purge events to be published to channel **testchannel**. Purging can be specified by EID and selector.

Acceptable request representations:

- "text/xml" on page 328
- "application/xml" on page 328

Available response representations:

- "text/xml" on page 328
- "application/xml" on page 328

JSON: GET

Provides JSON representations of channels/queues or events in a channel or queue as specified by the ChannelOrQueue parameter. The parameter is represented by the URI Path following the REST Plugin mount.

If the value supplied corresponds to a Universal Messaging namespace container, the representation returned is a list of channels and queues present in the container. If the value supplied corresponds to a channel or queue then the representation returned is a list of events. Finally if the value supplied does not correspond to either a container or a channel / queue a 404 response will be returned with no body.

Available response representations:

- "application/json" on page 329

JSON: POST

Allows publishing of an event to a channel or queue specified by the ChannelOrQueue parameter, which is represented by the URI Path following the REST Plugin mount. For example `http://localhost:9000/rest/API/json/testchannel` expects a JSON byte, XML or dictionary event to be published to channel **testchannel**.

Acceptable request representations:

- "application/json" on page 332

Available response representations:

- "application/json" on page 333

JSON: PUT

Allows purging of 1 or more events already published on a channel or queue specified by the ChannelOrQueue parameter, which is represented by the URI Path following the REST Plugin mount. For example `http://localhost:9000/rest/API/json/testchannel` expects a request to purge events to be published to channel **testchannel**. Purging can be specified by EID and selector.

Acceptable request representations:

- "application/json" on page 334

Available response representations:

- "application/json" on page 334

Representation: XML

XML REPRESENTATION : An XML representation of channels/queues or events in a channel or queue as specified by the ChannelOrQueue parameter.

Should the parameter point to an existing container, the response code is 200 and the response looks like this:

```
<Nirvana-RealmServer-ChannelList NumberOfChannels="2">
    <!--Constructed by my-channels Nirvana REST-Plugin :
        Wed Mar 02 16:07:28 EET 2011-->
    <Channel EventsConsumed="0" EventsPublished="0" LastEventID="-1"
        Name="testqueue" NumberEvents="0"
        fqn="http://localhost:8080/rest/API/xml/testqueue"/>
    <Channel EventsConsumed="0" EventsPublished="2" LastEventID="223"
        Name="testchannel" NumberEvents="2"
        fqn="http://shogun:8080/rest/API/xml/testchannel"/>
</Nirvana-RealmServer-ChannelList>
```

If the REST plugin is configured to include realm status, some additional information about the realm is presented:

```
<Nirvana-RealmServer-ChannelList NumberOfChannels="2">
    <!--Constructed by my-channels Nirvana REST-Plugin :
        Wed Mar 02 16:07:28 EET 2011-->
    <Channel EventsConsumed="0" EventsPublished="0" LastEventID="-1"
        Name="testqueue" NumberEvents="0"
        fqn="http://localhost:8080/rest/API/xml/testqueue"/>
    <Channel EventsConsumed="0" EventsPublished="2" LastEventID="223"
        Name="testchannel" NumberEvents="2"
        fqn="http://shogun:8080/rest/API/xml/testchannel"/>
    <RealmStatus FreeMemory="498101048" RealmName="nirvana6" Threads="87"
        TotalConnections="0" TotalConsumed="0"
        TotalMemory="530186240" TotalPublished="2"/>
</Nirvana-RealmServer-ChannelList>
```

Should the parameter point to an existing channel or queue, the response code is 200 and the response looks like this:

```
<Nirvana-RealmServer-EventList>
    <!--Constructed by my-channels Nirvana REST-Plugin :
        Wed Mar 02 16:10:57 EET 2011-->
    <Details ChannelName="http://localhost:8080/rest/API/xml/testsrc"
        FirstEvent=
          "http://localhost:8080/rest/API/xml/testsrc?Data=Dictionary&amp;EID=first"

        LastEID="223"
        LastEvent=
           "http://localhost:8080/rest/API/xml/testsrc?Data=Dictionary&amp;EID=last"
       NextLink="http://localhost:8080/rest/API/xml/testsrc?EID=224" StartEID="222"/>
   <Event ByteLink="http://localhost:8080/rest/API/xml/testsrc?Data=Byte&amp;EID=222"

        DataSize="9" EID="222" Tag="Test Tag" hasByte="true"/>
    <Event
        DictionaryLink=
           "http://localhost:8080/rest/API/xml/testsrc?Data=Dictionary&amp;EID=223"
        EID="223" hasDictionary="true"/>
</Nirvana-RealmServer-EventList>
```

You can follow the provided links to view individual events. If you choose to look at an individual byte event, the response code is 200 and the response looks like this:

```
<Nirvana-RealmServer-RawData>
    <!--Constructed by my-channels Nirvana REST-Plugin :
```

```
        Wed Mar 02 16:13:17 EET 2011-->
    <EventData ChannelName="http://localhost:8080/rest/API/xml/testsrc" EID="222">
        <Data>
            <![CDATA[VGVzdCBCb2R5]]>
        </Data>
        <Tag>
            <![CDATA[Test Tag]]>
        </Tag>
    </EventData>
</Nirvana-RealmServer-RawData>
```

If you choose to look at an individual XML event, the response code is 200 and the response looks like this:

```
<Nirvana-RealmServer-XMLData>
    <!--Constructed by my-channels Nirvana REST-Plugin :
        Wed Mar 02 16:13:17 EET 2011-->
    <EventData ChannelName="http://localhost:8080/rest/API/xml/testsrc" EID="222"
        isDOM="true">
        <Data>
            <myUserDataTag>
                Some User Data
            </myUserDataTag>
        </Data>
        <Tag>
            <![CDATA[Test Tag]]>
        </Tag>
    </EventData>
</Nirvana-RealmServer-XMLData>
```

If you choose to look at an individual Dictionary event, the response code is 200 and the response looks like this:

```
<DictionaryData isPersistent="true" TTL="0">
    <Data Key="testdouble">
        <![CDATA[1.0]]>
    </Data>
    <Data Key="testinteger">
        <![CDATA[1]]>
    </Data>
    <Data Key="teststring">
        <![CDATA[teststringvalue]]>
    </Data>
    <Dictionary Key="testdictionary">
        <Data Key="testdouble">
            <![CDATA[1.0]]>
        </Data>
        <Data Key="testinteger">
            <![CDATA[1]]>
        </Data>
        <Data Key="teststring">
            <![CDATA[teststringvalue]]>
        </Data>
        <Data Key="testlong">
            <![CDATA[1]]>
        </Data>
        <Data Key="testfloat">
            <![CDATA[1.0]]>
        </Data>
```

```
        <Data Key="testcharacter">
            <![CDATA[a]]>
        </Data>
        <Data Key="testboolean">
            <![CDATA[true]]>
        </Data>
    </Dictionary>
    <Data Key="testlong">
        <![CDATA[1]]>
    </Data>
    <Data Key="testfloat">
        <![CDATA[1.0]]>
    </Data>
    <Data Key="testcharacter">
        <![CDATA[a]]>
    </Data>
    <Data Key="testboolean">
        <![CDATA[true]]>
    </Data>
    <DataArray Key="teststringarray">
        <ArrayItem Index="0">
            <![CDATA[one]]>
        </ArrayItem>
        <ArrayItem Index="1">
            <![CDATA[two]]>
        </ArrayItem>
        <ArrayItem Index="2">
            <![CDATA[three]]>
        </ArrayItem>
    </DataArray>
    <DataArray Key="testbytearray">
        <ArrayItem Index="0">
            <![CDATA[YSBib2R5]]>
        </ArrayItem>
    </DataArray>
    <DataArray Key="testdictionaryarray">
        <ArrayItem Index="0">
            <Data Key="testdouble">
                <![CDATA[1.0]]>
            </Data>
            <Data Key="testinteger">
                <![CDATA[1]]>
            </Data>
            <Data Key="teststring">
                <![CDATA[teststringvalue]]>
            </Data>
        </ArrayItem>
        <ArrayItem Index="1">
            <Data Key="testdouble">
                <![CDATA[1.0]]>
            </Data>
            <Data Key="testinteger">
                <![CDATA[1]]>
            </Data>
            <Data Key="teststring">
                <![CDATA[teststringvalue]]>
            </Data>
        </ArrayItem>
    </DataArray>
</DictionaryData>
```

If the rest plugin is configured to include type information in representations, dictionary event representations will include them. In this case, responses looks like this:

```
<DictionaryData isPersistent="true" TTL="0">
    <Data Key="testdouble" Type="2">
        <![CDATA[1.0]]>
    </Data>
    <Data Key="testinteger" Type="4">
        <![CDATA[1]]>
    </Data>
    <Data Key="teststring" Type="0">
        <![CDATA[teststringvalue]]>
    </Data>
    <Dictionary Key="testdictionary">
        <Data Key="testdouble" Type="2">
            <![CDATA[1.0]]>
        </Data>
        <Data Key="testinteger" Type="4">
            <![CDATA[1]]>
        </Data>
        <Data Key="teststring" Type="0">
            <![CDATA[teststringvalue]]>
        </Data>
        <Data Key="testlong" Type="1">
            <![CDATA[1]]>
        </Data>
        <Data Key="testfloat" Type="5">
            <![CDATA[1.0]]>
        </Data>
        <Data Key="testcharacter" Type="6">
            <![CDATA[a]]>
        </Data>
        <Data Key="testboolean" Type="3">
            <![CDATA[true]]>
        </Data>
    </Dictionary>
    <Data Key="testlong" Type="1">
        <![CDATA[1]]>
    </Data>
    <Data Key="testfloat" Type="5">
        <![CDATA[1.0]]>
    </Data>
    <Data Key="testcharacter" Type="6">
        <![CDATA[a]]>
    </Data>
    <Data Key="testboolean" Type="3">
        <![CDATA[true]]>
    </Data>
    <DataArray ArrayType="0" Key="teststringarray">
        <ArrayItem Index="0">
            <![CDATA[one]]>
        </ArrayItem>
        <ArrayItem Index="1">
            <![CDATA[two]]>
        </ArrayItem>
        <ArrayItem Index="2">
            <![CDATA[three]]>
        </ArrayItem>
    </DataArray>
```

```
    <DataArray ArrayType="7" Key="testbytearray">
        <ArrayItem Index="0">
            <![CDATA[YSBib2R5]]>
        </ArrayItem>
    </DataArray>
    <DataArray ArrayType="9" Key="testdictionaryarray">
        <ArrayItem Index="0">
            <Data Key="testdouble" Type="2">
                <![CDATA[1.0]]>
            </Data>
            <Data Key="testinteger" Type="4">
                <![CDATA[1]]>
            </Data>
            <Data Key="teststring" Type="0">
                <![CDATA[teststringvalue]]>
            </Data>
        </ArrayItem>
        <ArrayItem Index="1">
            <Data Key="testdouble" Type="2">
                <![CDATA[1.0]]>
            </Data>
            <Data Key="testinteger" Type="4">
                <![CDATA[1]]>
            </Data>
            <Data Key="teststring" Type="0">
                <![CDATA[teststringvalue]]>
            </Data>
        </ArrayItem>
    </DataArray>
</DictionaryData>
```

Finally, should the parameter point to a non existing container or channel / queue, the response code is 404 without a response body

XML PUBLISH REQUEST

XML Byte events can be represented as follows:

```
<EventData isDom="false" isPersistent="true" TTL="0">
    <Data>
        <![CDATA[YSBib2R5]]>
    </Data>
    <Tag>
        <![CDATA[YSB0YWc=]]>
    </Tag>
</EventData>
```

Important:
data and tag should always be submitted in base64 encoded form.

XML DOM events can be represented as follows:

```
<EventData isDom="true" isPersistent="true" TTL="0">
    <Data>
        <![CDATA[YSBib2R5]]>
    </Data>
    <Tag>
        <![CDATA[YSB0YWc=]]>
    </Tag>
```

```
</EventData>
```

Important:
data and tag should always be submitted in base64 encoded form.

XML Dictionary events can be represented as follows:

```
<DictionaryData isPersistent="true" TTL="0">
    <Data Key="testdouble">
        <![CDATA[1.0]]>
    </Data>
    <Data Key="testinteger">
        <![CDATA[1]]>
    </Data>
    <Data Key="teststring">
        <![CDATA[teststringvalue]]>
    </Data>
    <Dictionary Key="testdictionary">
        <Data Key="testdouble">
            <![CDATA[1.0]]>
        </Data>
        <Data Key="testinteger">
            <![CDATA[1]]>
        </Data>
        <Data Key="teststring">
            <![CDATA[teststringvalue]]>
        </Data>
        <Data Key="testlong">
            <![CDATA[1]]>
        </Data>
        <Data Key="testfloat">
            <![CDATA[1.0]]>
        </Data>
        <Data Key="testcharacter">
            <![CDATA[a]]>
        </Data>
        <Data Key="testboolean">
            <![CDATA[true]]>
        </Data>
    </Dictionary>
    <Data Key="testlong">
        <![CDATA[1]]>
    </Data>
    <Data Key="testfloat">
        <![CDATA[1.0]]>
    </Data>
    <Data Key="testcharacter">
        <![CDATA[a]]>
    </Data>
    <Data Key="testboolean">
        <![CDATA[true]]>
    </Data>
    <DataArray Key="teststringarray">
        <ArrayItem Index="0">
            <![CDATA[one]]>
        </ArrayItem>
        <ArrayItem Index="1">
            <![CDATA[two]]>
        </ArrayItem>
```

```
        <ArrayItem Index="2">
            <![CDATA[three]]>
        </ArrayItem>
    </DataArray>
    <DataArray Key="testbytearray">
        <ArrayItem Index="0">
            <![CDATA[YSBib2R5]]>
        </ArrayItem>
    </DataArray>
    <DataArray Key="testdictionaryarray">
        <ArrayItem Index="0">
            <Data Key="testdouble">
                <![CDATA[1.0]]>
            </Data>
            <Data Key="testinteger">
                <![CDATA[1]]>
            </Data>
            <Data Key="teststring">
                <![CDATA[teststringvalue]]>
            </Data>
        </ArrayItem>
        <ArrayItem Index="1">
            <Data Key="testdouble">
                <![CDATA[1.0]]>
            </Data>
            <Data Key="testinteger">
                <![CDATA[1]]>
            </Data>
            <Data Key="teststring">
                <![CDATA[teststringvalue]]>
            </Data>
        </ArrayItem>
    </DataArray>
</DictionaryData>
```

Optionally, dictionary events can include type information (see "Types" on page 334). This allows the Universal Messaging REST API to preserve these types when publishing the event. The types are defined as byte constants to keep typed dictionary events compact in size.

XML Dictionary events (with type information) can be represented as follows:

```
<DictionaryData isPersistent="true" TTL="0">
    <Data Key="testdouble" Type="2">
        <![CDATA[1.0]]>
    </Data>
    <Data Key="testinteger" Type="4">
        <![CDATA[1]]>
    </Data>
    <Data Key="teststring" Type="0">
        <![CDATA[teststringvalue]]>
    </Data>
    <Dictionary Key="testdictionary">
        <Data Key="testdouble" Type="2">
            <![CDATA[1.0]]>
        </Data>
        <Data Key="testinteger" Type="4">
            <![CDATA[1]]>
        </Data>
        <Data Key="teststring" Type="0">
```

```
            <![CDATA[teststringvalue]]>
        </Data>
        <Data Key="testlong" Type="1">
            <![CDATA[1]]>
        </Data>
        <Data Key="testfloat" Type="5">
            <![CDATA[1.0]]>
        </Data>
        <Data Key="testcharacter" Type="6">
            <![CDATA[a]]>
        </Data>
        <Data Key="testboolean" Type="3">
            <![CDATA[true]]>
        </Data>
    </Dictionary>
    <Data Key="testlong" Type="1">
        <![CDATA[1]]>
    </Data>
    <Data Key="testfloat" Type="5">
        <![CDATA[1.0]]>
    </Data>
    <Data Key="testcharacter" Type="6">
        <![CDATA[a]]>
    </Data>
    <Data Key="testboolean" Type="3">
        <![CDATA[true]]>
    </Data>
    <DataArray ArrayType="0" Key="teststringarray">
        <ArrayItem Index="0">
            <![CDATA[one]]>
        </ArrayItem>
        <ArrayItem Index="1">
            <![CDATA[two]]>
        </ArrayItem>
        <ArrayItem Index="2">
            <![CDATA[three]]>
        </ArrayItem>
    </DataArray>
    <DataArray ArrayType="7" Key="testbytearray">
        <ArrayItem Index="0">
            <![CDATA[YSBib2R5]]>
        </ArrayItem>
    </DataArray>
    <DataArray ArrayType="9" Key="testdictionaryarray">
        <ArrayItem Index="0">
            <Data Key="testdouble" Type="2">
                <![CDATA[1.0]]>
            </Data>
            <Data Key="testinteger" Type="4">
                <![CDATA[1]]>
            </Data>
            <Data Key="teststring" Type="0">
                <![CDATA[teststringvalue]]>
            </Data>
        </ArrayItem>
        <ArrayItem Index="1">
            <Data Key="testdouble" Type="2">
                <![CDATA[1.0]]>
            </Data>
            <Data Key="testinteger" Type="4">
```

```
                <![CDATA[1]]>
            </Data>
            <Data Key="teststring" Type="0">
                <![CDATA[teststringvalue]]>
            </Data>
        </ArrayItem>
    </DataArray>
</DictionaryData>
```

Important:
byte[ ] types should always be submitted in base64 encoded form.

XML PUBLISH RESPONSE : A XML representation to indicate the status of attempting to publish an event to the channel or queue specified by the ChannelOrQueue parameter.

Should the publish call be successful, the response code is 201 and the response looks like this:

```
<Nirvana-RealmServer-PublishRequest>
    <response value="ok"/>
</Nirvana-RealmServer-PublishRequest>
```

Should the publish call fail for any reason, the response code is 400 and the response looks like this:

```
<Nirvana-RealmServer-Error>
    <response value="failInput"/>
    <errorcode value="ErrorCode"/>
    <errormessage value="Error Message"/>
</Nirvana-RealmServer-Error>
```

XML PURGE REQUEST : A XML representation of a Purge Request that indicates the event(s) to purge.

A XML purge request looks as follows:

```
<Nirvana-RealmServer-PurgeRequest startEID="10" endEID="20" purgeJoins="false">
    <selector>
        <![CDATA[]]>
    </selector>
</Nirvana-RealmServer-PurgeRequest>
```

XML PURGE RESPONSE : A XML representation to indicate the status of attempting to purge an event to the channel or queue specified by the ChannelOrQueue parameter.

Should the purge call be successful, the response code is 200 and the response looks like this:

```
<Nirvana-RealmServer-PurgeRequest>
    <response value="ok"/>
</Nirvana-RealmServer-PurgeRequest>
```

Should the purge call fail for any reason, the response code is 400 and the response looks like this:

```
<Nirvana-RealmServer-Error>
    <response value="failInput"/>
    <errorcode value="ErrorCode"/>
    <errormessage value="Error Message"/>
</Nirvana-RealmServer-Error>
```

Representation: JSON

JSON REPRESENTATION : A JSON representation of channels/queues or events in a channel or queue as specified by the ChannelOrQueue parameter.

Should the parameter point to an existing container, the response code is 200 and the response looks like this:

```
{
    "Channels":
    [ {
        "EventsConsumed": "0",
        "EventsPublished": "0",
        "LastEventID": "-1",
        "Name": "testqueue",
        "NumberEvents": "0",
        "fqn": "http://localhost:8080/rest/API/json/testqueue"
    }, {
        "EventsConsumed": "0",
        "EventsPublished": "0",
        "LastEventID": "212",
        "Name": "testchannel",
        "NumberEvents": "0",
        "fqn": "http://localhost:8080/rest/API/json/testchannel"
    } ],
    "Comment": "Constructed by my-channels Nirvana REST-Plugin :
                Wed Mar 02 11:38:30 EET 2011",
    "Name":
    "Nirvana-RealmServer-ChannelList",
    "NumberOfChannels": "2",
}
```

If the REST plugin is configured to include realm status, some additional information about the realm is presented:

```
{
    "Channels":
    [ {
        "EventsConsumed": "0",
        "EventsPublished": "0",
        "LastEventID": "-1",
        "Name": "testqueue",
        "NumberEvents": "0",
        "fqn": "http://localhost:8080/rest/API/json/testqueue"
    }, {
        "EventsConsumed": "0",
        "EventsPublished": "0",
        "LastEventID": "212",
        "Name": "testchannel",
        "NumberEvents": "0",
        "fqn": "http://localhost:8080/rest/API/json/testchannel"
    } ],
    "Comment": "Constructed by my-channels Nirvana REST-Plugin :
                Wed Mar 02 11:38:30 EET 2011",
    "Name": "Nirvana-RealmServer-ChannelList",
    "NumberOfChannels": "2",
    "Realm": {
```

```
        "FreeMemory": "503291048",
        "RealmName": "nirvana6",
        "Threads": "104",
        "TotalConnections": "1",
        "TotalConsumed": "0",
        "TotalMemory": "530186240",
        "TotalPublished": "0"
    }
}
```

Should the parameter point to an existing channel or queue, the response code is 200 and the response looks like this:

```
{
    "ChannelName": "http://localhost:8080/rest/API/json/testsrc",
    "Comment": "Constructed by my-channels Nirvana REST-Plugin : Wed
    Mar 02 12:19:22 EET 2011",
    "Events":
    [ {
        "ByteLink": "http://localhost:8080/rest/API/json/testsrc?Data=Byte&EID=213",
        "DataSize": "9",
        "EID": "213",
        "Tag": "Test Tag",
        "hasByte": "true"
    }, {
        "DictionaryLink":
             "http://localhost:8080/rest/API/json/testsrc?Data=Dictionary&EID=214",
        "EID": "214",
        "hasDictionary": "true"
    } ],
    "FirstEvent":
"http://localhost:8080/rest/API/json/testsrc?Data=Dictionary&EID=first",
    "LastEID": "214",
   "LastEvent": "http://localhost:8080/rest/API/json/testsrc?Data=Dictionary&EID=last",
    "Name": "Nirvana-RealmServer-EventList",
    "NextLink": "http://localhost:8080/rest/API/json/testsrc?EID=215",
    "StartEID": "213"
}
```

You can follow the provided links to view individual events. If you choose to look at an individual byte event, the response code is 200 and the response looks like this:

```
{
    "ChannelName": "http://localhost:8080/rest/API/json/testsrc",
    "Comment": "Constructed by my-channels Nirvana REST-Plugin :
                Wed Mar 02 12:21:46 EET 2011",
    "Data": "VGVzdCBCb2R5",
    "EID": "213",
    "Name": "Nirvana-RealmServer-RawData",
    "Tag": "Test Tag"
}
```

If you choose to look at an individual XML event, the response code is 200 and the response looks like this:

```
{
    "ChannelName": "http://localhost:8080/rest/API/json/testsrc",
    "Comment": "Constructed by my-channels Nirvana REST-Plugin :
                Wed Mar 02 12:21:46 EET 2011",
```

```
    "Data": "VGVzdCBCb2R5",
    "EID": "213",
    "Name": "Nirvana-RealmServer-XMLData",
    "Tag": "Test Tag"
}
```

If you choose to look at an individual Dictionary event, the response code is 200 and the response looks like this:

```
{
    "dictionary":
    {
        "testboolean": [true],
        "testcharacter": ["a"],
        "testdictionary": [
        {
            "testboolean": [true],
            "testcharacter": ["a"],
            "testdouble": [1],
            "testfloat": [1],
            "testinteger": [1],
            "testlong": [1],
            "teststring": ["teststringvalue"]
        }],
        "testdouble": [1],
        "testfloat": [1],
        "testinteger": [1],
        "testlong": [1],
        "teststring": ["teststringvalue"],
        "teststringarray": [[
            "one",
            "two",
            "three"
        ]]
    },
    "isPersistent": true
}
```

If the rest plugin is configured to include type information in representations, dictionary event representations will include them. In this case, responses looks like this:

```
{
    "dictionary":
    {
        "testboolean": [ true, 3 ],
        "testcharacter": [ "a", 6 ],
        "testdictionary":
        [ {
            "testboolean": [ true, 3 ],
            "testcharacter": [ "a", 6 ],
            "testdouble": [ 1, 2 ],
            "testfloat": [ 1, 5 ],
            "testinteger": [ 1, 4 ],
            "testlong": [ 1, 1 ],
            "teststring": [ "teststringvalue", 0 ]
        }, 9 ],
        "testdouble": [ 1, 2 ],
        "testfloat": [ 1, 5 ],
        "testinteger": [ 1, 4 ],
```

```
        "testlong": [ 1, 1 ],
        "teststring": [ "teststringvalue", 0 ],
        "teststringarray":
        [[
            "one",
            "two",
            "three"
        ], 100, 0 ]
    },
    "isPersistent": true
}
```

Finally, should the parameter point to a non existing container or channel / queue, the response code is 404 without a response body

JSON PUBLISH REQUEST

JSON Byte events can be represented as follows:

```
{
    "data": "VGVzdCBCb2R5",
    "isPersistent": true,
    "tag": "VGVzdCBUYWc="
}
```

Important:
data and tag should always be submitted in base64 encoded form.

JSON DOM events can be represented as follows:

```
{
    "data": "VGVzdCBCb2R5",
    "isDOM": true,
    "isPersistent": true,
    "tag": "VGVzdCBUYWc="
}
```

Important:
data and tag should always be submitted in base64 encoded form.

JSON Dictionary events can be represented as follows:

```
{
    "dictionary":
    {
        "testboolean": [true],
        "testcharacter": ["a"],
        "testdictionary":
        [{
            "testboolean": [true],
            "testcharacter": ["a"],
            "testdouble": [1],
            "testfloat": [1],
            "testinteger": [1],
            "testlong": [1],
            "teststring": ["teststringvalue"]
        }],
        "testdouble": [1],
```

```
        "testfloat": [1],
        "testinteger": [1],
        "testlong": [1],
        "teststring": ["teststringvalue"],
        "teststringarray":
        [[
            "one",
            "two",
            "three"
        ]]
    },
    "isPersistent": true
}
```

Optionally, dictionary events can include type information (see "Types" on page 334). This allows the Universal Messaging REST API to preserve these types when publishing the event. The types are defined as byte constants to keep typed dictionary events compact in size.

Dictionary events (with type information) can be represented as follows:

```
{
    "dictionary":
    {
        "testboolean": [ true, 3 ],
        "testcharacter": [ "a", 6 ],
        "testdictionary":
        [ {
            "testboolean": [ true, 3 ],
            "testcharacter": [ "a", 6 ],
            "testdouble": [ 1, 2 ],
            "testfloat": [ 1, 5 ],
            "testinteger": [ 1, 4 ],
            "testlong": [ 1, 1 ],
            "teststring": [ "teststringvalue", 0 ]
        }, 9 ],
        "testdouble": [ 1, 2 ],
        "testfloat": [ 1, 5 ],
        "testinteger": [ 1, 4 ],
        "testlong": [ 1, 1 ],
        "teststring": [ "teststringvalue", 0 ],
        "teststringarray":
        [ [
            "one",
            "two",
            "three"
        ], 100, 0 ]
    },
    "isPersistent": true
}
```

Important:
byte[] types should always be submitted in base64 encoded form.

JSON PUBLISH RESPONSE : A JSON representation to indicate the status of attempting to publish an event to the channel or queue specified by the ChannelOrQueue parameter.

Should the publish call be successful, the response code is 201 and the response looks like this:

```
{
    "Response": "OK"
}
```

Should the publish call fail for any reason, the response code is 400 and the response looks like this:

```
{
    "errorcode": "ErrorCode",
    "errormessage": "Error Message",
    "response": "failInput"
}
```

JSON PURGE REQUEST : A JSON representation of a Purge Request that indicates the event(s) to purge.

A JSON purge request looks as follows:

```
{
    "endEID": 20,
    "purgeJoins": false,
    "selector": "",
    "startEID": 10
}
```

JSON PURGE RESPONSE : A JSON representation to indicate the status of attempting to purge an event to the channel or queue specified by the ChannelOrQueue parameter

Should the purge call be successful, the response code is 200 and the response looks like this:

```
{
    "Response": "OK"
}
```

Should the purge call fail for any reason, the response code is 400 and the response looks like this:

```
{
    "errorcode": "ErrorCode",
    "errormessage": "Error Message",
    "response": "failInput"
}
```

Types

Type	ID
String	0
Long	1
Double	2
Boolean	3
Integer	4

Type	ID
Float	5
Character	6
Byte	7
Short	8
Dictionary	9
Array	100

Servlet Plugin

The servlet plugin enables the Universal Messaging realm server to serve Java servlets.

Configuration

Once you have created the servlet plugin on an interface, you can then select it from the **Plugins** panel for the interface and configure the plugin parameters.

The servlet plugin requires configuration information relating to its behavior, as well as the location of the servlets it is required to serve to the clients. Below is a table that shows each configuration parameter and describes what each is used for.

To ensure security, the `EnforceConfigFile` option can be set to true; this allows only those classes specified in the configuration file to be loaded. Alternatively, the `EnforceStrictClassLoader` option can be set; this prevents classes being loaded from different class loaders to that of the servlet, and thereby also prevents arbitrary classes from being loaded.

Parameter Name	Description	Default Value
AddUserAsPlugin	Add the username to the session cookies.	false
AuthParameters	List of key=value string which is passed to authenticators init function.	
AddUserAsPlugin	Classname of authenticator to use, leave blank for default	(default)
EnableClassReload	Automatically reload servlet class if it changes	true
EnforceConfigFile	If true, only servlets within the ServletConfigFile will be executed.	true
EnforceStrictClassLoader	If true, only servlets loaded by the initial class loader will be executed. Any classes loaded by parent loader will be ignored.	true

Parameter Name	Description	Default Value
GroupNames	A comma separated list of groups to which a user must be a member of to be granted access.	
MimeType	Name of the file to load the mime type information from. The format of the file is same as the apache mime types.	
Properties	File containing the servlet properties. The file should be a java properties file that contains one property per line prefixed with the full class name. For example for a servlet class com.example.Servlet defining a property called RNAME you should have a line as follows: com.example.Servlet.RNAME=nsp://localhost:9000	
ReloadUserFileDynamically	If true, the user file will get reloaded on each auth request.	true
RoleNames	A comma separated list of groups to which a user must have one to be granted access.	
Security Realm	Name of the authentication realm.	
Servlet Config File	File which contains all the valid servlets which will run. The file should be a text file containing one full servlet class name per line, indicating only these should be allowed to run. For example having a single line com.example.Servlet would mean that only that servlet will be allowed to run irrespective of how many exist in the server classpath.	
Servlet Path	Directory in which to locate servlet classes	
SessionTimeout	Time in seconds before timeout of servlet session not in use.	

Exporting and Importing Realm XML Configurations

You can export a realm configuration into an XML representation, and then import the XML representation into another realm, using the Enterprise Manager. The XML export and import functionality enables you to automatically configure multiple realms based on a standard structure, for example when you want to clone realms and their internal structure.

You can export specific elements of a realm or the entire realm structure. The exported XML can contain any or all of the following elements:

- Clusters

- Realm access control lists (ACLs)
- Channels
- Channel ACLs
- Queues
- Queue ACLs
- Configuration parameters
- JNDI assets
- Durables
- Interfaces
- Plugins
- Scheduling information

After you exported the realm configuration, you can import the XML into another realm. Importing the XML automatically creates and configures the objects selected for import from the XML file. The export and import marshal the realm objects from their Administration API representation into XML and back again.

Exporting a Realm Configuration into an XML File

Use the following procedure to export a realm or specific elements of a realm into an XML file.

To export a realm configuration into an XML file

1. In the Enterprise Manager, go to **Realms** and select the realm that you want to export.
2. Right-click the realm node and select **Export Realm to XML**.
3. In the **Export to** field, specify the path to the file to which you want to export the realm configuration.
4. Select the realm elements to export:
   - Select **Export all** to export the entire realm structure.
   - Select one or more options under **Realm Export**, **Channels**, **Cluster Export**, **Interfaces**, **Queues**, and **Data Groups** to export specific elements of the realm structure.
5. Click **OK**.

Importing a Realm Configuration from an XML File

Before importing an XML representation of a realm configuration to another realm, you must export the realm configuration as described in "Exporting a Realm Configuration into an XML File" on page 337.

Use the following procedure to import a realm or specific elements of a realm from an XML file into another realm.

To import a realm configuration from an XML file

1. In the Enterprise Manager, go to **Realms** and select the realm into which you want to import the XML configuration.
2. Right-click the realm node and select **Import Realm from XML**.
3. In the **Import from** field, specify the path to the file from which you want to import the realm configuration.
4. Select the realm elements to import:
   - Select **Import all** to import the entire realm structure.
   - Select one or more options under **Realm Export**, **Channels**, **Cluster Export**, **Interfaces**, **Queues**, and **Data Groups** to import specific elements of the realm structure.

   Important:
   Importing a configuration that contains clustered transient channels and queues will fail because transient channels and queues are not supported in a cluster. To resolve this situation, you can use the **Convert cluster-wide transient channels and queues to mixed** option to automatically convert all clustered transient channels and queues to mixed. However, be aware that converting clustered transient channels and queues to mixed might cause additional disk and memory consumption for events published to these channels and queues. If you are importing into a realm that is part of a cluster, the **Convert cluster-wide transient channels and queues to mixed** option is selected by default.
5. Click **OK**.

Using the clusterWide Attribute for Channels and Queues

When you export channels or queues to an XML file, each channel or queue in the XML file has a `clusterWide` attribute. If you export a clustered channel or queue, the attribute is set to `true`. If you export a non-clustered channel or queue, the attribute is set to `false`.

Before you import the XML file into a realm, you can manually edit the XML file and modify the `clusterWide` attribute of each channel or queue, depending on how you want to import the channel or queue. To import a channel or queue as clustered while doing an import on a clustered realm, set `clusterWide` to `true`. To import a channel or queue as non-clustered, set `clusterWide` to `false`.

Sample XML File for Import

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<NirvanaRealm name="realm0" exportDate="2021-01-12+02:00" comment="Realm configuration
 from realm0" version="BuildIdentifier" buildInfo="BuildIdentifier">
    <RealmConfiguration>
        <ConfigGroup name="Audit Settings">
            <ConfigItem name="ChannelACL" value="true"/>
            <ConfigItem name="ChannelFailure" value="true"/>
            <ConfigItem name="ChannelMaintenance" value="false"/>
            <ConfigItem name="ChannelSuccess" value="false"/>
            <ConfigItem name="DataGroup" value="false"/>
            <ConfigItem name="DataGroupFailure" value="false"/>
            <ConfigItem name="DataStream" value="false"/>
            <ConfigItem name="Group" value="true"/>
            <ConfigItem name="GroupMembers" value="true"/>
            <ConfigItem name="InterfaceManagement" value="true"/>
            <ConfigItem name="JoinFailure" value="true"/>
            <ConfigItem name="JoinMaintenance" value="true"/>
            <ConfigItem name="JoinSuccess" value="false"/>
            <ConfigItem name="QueueACL" value="true"/>
            <ConfigItem name="QueueFailure" value="true"/>
            <ConfigItem name="QueueMaintenance" value="false"/>
            <ConfigItem name="QueueSuccess" value="false"/>
            <ConfigItem name="RealmACL" value="true"/>
            <ConfigItem name="RealmFailure" value="true"/>
            <ConfigItem name="RealmMaintenance" value="true"/>
            <ConfigItem name="RealmSuccess" value="false"/>
            <ConfigItem name="SnoopStream" value="false"/>
        </ConfigGroup>
        <ConfigGroup name="Client Timeout Values">
            <ConfigItem name="EventTimeout" value="60000"/>
            <ConfigItem name="HighWaterMark" value="3000"/>
            <ConfigItem name="LowWaterMark" value="1000"/>
            <ConfigItem name="QueueAccessWaitLimit" value="200"/>
            <ConfigItem name="QueueBlockLimit" value="500"/>
            <ConfigItem name="QueuePushWaitLimit" value="200"/>
            <ConfigItem name="TransactionLifeTime" value="20000"/>
        </ConfigGroup>
        <ConfigGroup name="Cluster Config">
            <ConfigItem name="ClientQueueSize" value="1000"/>
            <ConfigItem name="ClientQueueWindow" value="100"/>
            <ConfigItem name="ClientStateDelay" value="5000"/>
            <ConfigItem name="ClusterMode" value="0"/>
            <ConfigItem name="DisableHTTPConnections" value="true"/>
            <ConfigItem name="DisconnectWait" value="30000"/>
            <ConfigItem name="DisconnectWhenNotReady" value="false"/>
            <ConfigItem name="EnableMulticast" value="true"/>
            <ConfigItem name="EnableStoreRecoveryRetry" value="true"/>
            <ConfigItem name="EnginePipelineSize" value="2"/>
            <ConfigItem name="FormationTimeout" value="120000"/>
            <ConfigItem name="HeartBeatInterval" value="120000"/>
            <ConfigItem name="InitialConnectionTimeout" value="30000"/>
            <ConfigItem name="IsCommittedDelay" value="5000"/>
            <ConfigItem name="MasterRequestTimeout" value="60000"/>
            <ConfigItem name="MasterVoteDelay" value="10000"/>
            <ConfigItem name="MasterWaitTimeout" value="10000"/>
            <ConfigItem name="PublishQueueEnabled" value="true"/>
            <ConfigItem name="QueueConsistencyCheck" value="false"/>
            <ConfigItem name="QueueSize" value="1000"/>
            <ConfigItem name="StateChangeScan" value="60000"/>
            <ConfigItem name="SyncPingSize" value="1000"/>
```

```
        </ConfigGroup>
        <ConfigGroup name="Comet Config">
            <ConfigItem name="BufferSize" value="5120"/>
            <ConfigItem name="EnableLogging" value="false"/>
            <ConfigItem name="Timeout" value="60000"/>
        </ConfigGroup>
        <ConfigGroup name="Connection Config">
            <ConfigItem name="AllowBufferReuse" value="true"/>
            <ConfigItem name="BufferManagerCount" value="16"/>
            <ConfigItem name="BufferPoolSize" value="100"/>
            <ConfigItem name="BufferSize" value="102400"/>
            <ConfigItem name="CometReadTimeout" value="20000"/>
            <ConfigItem name="ConnectionDelay" value="60000"/>
            <ConfigItem name="IdleDriverTimeout" value="300000"/>
            <ConfigItem name="IdleSessionTimeout" value="300000"/>
            <ConfigItem name="KeepAlive" value="60000"/>
            <ConfigItem name="MaxBufferSize" value="20971520"/>
            <ConfigItem name="MaxNoOfConnections" value="-1"/>
            <ConfigItem name="MaxWriteCount" value="30"/>
            <ConfigItem name="NetworkMonitorThreads" value="4"/>
            <ConfigItem name="PriorityQueueCount" value="10"/>
            <ConfigItem name="PriorityReadSpinLockMaxConnections" value="2"/>
            <ConfigItem name="PriorityReadSpinLockTime" value="500"/>
            <ConfigItem name="PriorityReadType" value="1"/>
            <ConfigItem name="QueueHighWaterMark" value="3000"/>
            <ConfigItem name="QueueLowWaterMark" value="1000"/>
            <ConfigItem name="ReadCount" value="10"/>
            <ConfigItem name="UseDirectBuffering" value="true"/>
            <ConfigItem name="WriteHandlerType" value="3"/>
            <ConfigItem name="whEventThresholdCount" value="350"/>
            <ConfigItem name="whEventThresholdTime" value="500"/>
            <ConfigItem name="whMaxEventsBeforeFlush" value="100"/>
            <ConfigItem name="whMaxEventsPerSecond" value="100000"/>
            <ConfigItem name="whMaxTimeBetweenFlush" value="2"/>
        </ConfigGroup>
        <ConfigGroup name="Data Stream Config">
            <ConfigItem name="MonitorTimer" value="10000"/>
            <ConfigItem name="OffloadMulticastWrite" value="false"/>
            <ConfigItem name="SendInitialMapping" value="true"/>
        </ConfigGroup>
        <ConfigGroup name="DurableConfig">
            <ConfigItem name="DurableNameFiltering" value="false"/>
            <ConfigItem name="QueuedExtendedException" value="false"/>
        </ConfigGroup>
        <ConfigGroup name="Environment Config">
            <ConfigItem name="AvailableProcessors" value="4"/>
            <ConfigItem name="Embedded" value="false"/>
            <ConfigItem name="InterRealmProtocolVersion" value="1"/>
            <ConfigItem name="JavaVendor" value="Oracle Corporation"/>
            <ConfigItem name="JavaVersion" value="1.8.0_271"/>
            <ConfigItem name="NanosecondSupport" value="true"/>
            <ConfigItem name="OSArchitecture" value="amd64"/>
            <ConfigItem name="OSName" value="Windows 10"/>
            <ConfigItem name="OSVersion" value="10.0"/>
            <ConfigItem name="ProcessId" value="33468"/>
            <ConfigItem name="ServerBuildDate" value="12-Feb-1964"/>
            <ConfigItem name="ServerBuildNumber" value="BuildNumberHere"/>
            <ConfigItem name="ServerReleaseDetails" value="BuildIdentifier"/>
            <ConfigItem name="ServerVersion" value="RealmServerVersion"/>
        </ConfigGroup>
```

```
        <ConfigGroup name="Event Storage">
            <ConfigItem name="ActiveDelay" value="1000"/>
            <ConfigItem name="AutoDeleteScan" value="5000"/>
            <ConfigItem name="AutoMaintenanceThreshold" value="50"/>
            <ConfigItem name="CacheAge" value="60000"/>
            <ConfigItem name="EnableStoreCaching" value="false"/>
            <ConfigItem name="IdleDelay" value="10000"/>
            <ConfigItem name="JMSEngineAutoPurgeTime" value="5000"/>
            <ConfigItem name="JMSEngineIndividualPurgeEnabled" value="false"/>
            <ConfigItem name="MaintenanceFileSizeThreshold" value="104857600"/>
            <ConfigItem name="MaintenanceMemoryThreshold" value="104857600"/>
            <ConfigItem name="PageSize" value="5000"/>
            <ConfigItem name="QueueDeliveryPersistencePolicy" value="1"/>
            <ConfigItem name="StoreReadBufferSize" value="32768"/>
            <ConfigItem name="SyncBatchSize" value="50"/>
            <ConfigItem name="SyncServerFiles" value="false"/>
            <ConfigItem name="SyncTimeLimit" value="20"/>
            <ConfigItem name="ThreadPoolSize" value="4"/>
        </ConfigGroup>
        <ConfigGroup name="Fanout Values">
            <ConfigItem name="ConnectionGrouping" value="true"/>
            <ConfigItem name="DelayPublishOnCapacity" value="true"/>
            <ConfigItem name="HonourSharedDurableCapacity" value="true"/>
            <ConfigItem name="IteratorWindowSize" value="100"/>
            <ConfigItem name="JMSQueueMaxMultiplier" value="10"/>
            <ConfigItem name="ParallelThreadPoolSize" value="2"/>
            <ConfigItem name="PeakPublishDelay" value="1"/>
            <ConfigItem name="PublishDelay" value="1"/>
            <ConfigItem name="PublishExpiredEvents" value="true"/>
            <ConfigItem name="SendEndOfChannelAlways" value="false"/>
            <ConfigItem name="SendPubEventsImmediately" value="true"/>
            <ConfigItem name="SyncQueueDelay" value="1000"/>
            <ConfigItem name="SyncQueuePublisher" value="false"/>
        </ConfigGroup>
        <ConfigGroup name="Global Values">
            <ConfigItem name="AllowRealmAdminFullAccess" value="true"/>
            <ConfigItem name="CacheJoinInfoKeys" value="true"/>
            <ConfigItem name="DisableExplicitGC" value="true"/>
            <ConfigItem name="EnableCaching" value="false"/>
            <ConfigItem name="EnableDNSLookups" value="true"/>
            <ConfigItem name="EnableWeakReferenceCleanup" value="true"/>
            <ConfigItem name="ExtendedMessageSelector" value="true"/>
            <ConfigItem name="HTTPCookieSize" value="14"/>
            <ConfigItem name="NHPScanTime" value="5000"/>
            <ConfigItem name="NHPTimeout" value="120000"/>
            <ConfigItem name="OverrideEveryoneUser" value="false"/>
            <ConfigItem name="PauseServerPublishing" value="false"/>
            <ConfigItem name="SendRealmSummaryStats" value="false"/>
            <ConfigItem name="StampDictionary" value="true"/>
            <ConfigItem name="StampHost" value="true"/>
            <ConfigItem name="StampTime" value="true"/>
            <ConfigItem name="StampTimeUseHPT" value="false"/>
            <ConfigItem name="StampTimeUseHPTScale" value="0"/>
            <ConfigItem name="StampUser" value="true"/>
            <ConfigItem name="StatusBroadcast" value="5000"/>
        </ConfigGroup>
        <ConfigGroup name="Inter-Realm Comms Config">
            <ConfigItem name="EstablishmentTime" value="30000"/>
            <ConfigItem name="KeepAliveInterval" value="10000"/>
            <ConfigItem name="KeepAliveResetTime" value="35000"/>
```

```
            <ConfigItem name="MaximumReconnectTime" value="20000"/>
            <ConfigItem name="MinimumReconnectTime" value="1000"/>
            <ConfigItem name="Timeout" value="120000"/>
            <ConfigItem name="WriteDelayTimeout" value="30000"/>
        </ConfigGroup>
        <ConfigGroup name="JVM Management">
            <ConfigItem name="EmergencyThreshold" value="94"/>
            <ConfigItem name="EnableJMX" value="false"/>
            <ConfigItem name="ExitOnDiskIOError" value="true"/>
            <ConfigItem name="ExitOnInterfaceFailure" value="false"/>
            <ConfigItem name="IORetryCount" value="10"/>
            <ConfigItem name="IOSleepTime" value="500"/>
            <ConfigItem name="JMXRMIServerURLString"
value="service:jmx:rmi:///jndi/rmi://localhost:9999/server"/>
            <ConfigItem name="MemoryMonitoring" value="2000"/>
            <ConfigItem name="ThrottleAllPublishersAtThreshold" value="true"/>
            <ConfigItem name="WarningThreshold" value="85"/>
        </ConfigGroup>
        <ConfigGroup name="Join Config">
            <ConfigItem name="ActiveThreadPoolSize" value="2"/>
            <ConfigItem name="IdleThreadPoolSize" value="1"/>
            <ConfigItem name="MaxEventsPerSchedule" value="1000"/>
            <ConfigItem name="MaxQueueSizeToUse" value="100"/>
            <ConfigItem name="RemoteJoinAckBatchSize" value="100"/>
            <ConfigItem name="RemoteJoinAckInterval" value="1000"/>
            <ConfigItem name="UseQueuedLocalJoinHandler" value="false"/>
        </ConfigGroup>
        <ConfigGroup name="Logging Config">
            <ConfigItem name="DefaultLogSize" value="10000000"/>
            <ConfigItem name="DiagnosticsEnabledStores" value="!"/>
            <ConfigItem name="DisplayCurrentThread" value="true"/>
            <ConfigItem name="EmbedTag" value="false"/>
            <ConfigItem name="EnableLog4J" value="false"/>
            <ConfigItem name="EnableStatusLog" value="true"/>
            <ConfigItem name="LogManager" value="1"/>
            <ConfigItem name="RolledLogFileDepth" value="10"/>
            <ConfigItem name="fLoggerLevel" value="0"/>
        </ConfigGroup>
        <ConfigGroup name="Metric Config">
            <ConfigItem name="EnableEventMemoryMonitoring" value="true"/>
            <ConfigItem name="EnableMetrics" value="true"/>
        </ConfigGroup>
        <ConfigGroup name="Plugin Config">
            <ConfigItem name="EnableAccessLog" value="true"/>
            <ConfigItem name="EnableErrorLog" value="true"/>
            <ConfigItem name="EnablePluginLog" value="true"/>
            <ConfigItem name="MaxNumberOfPluginThreads" value="200"/>
            <ConfigItem name="PluginTimeout" value="30000"/>
        </ConfigGroup>
        <ConfigGroup name="Protobuf Config">
            <ConfigItem name="CacheEventFilter" value="true"/>
        </ConfigGroup>
        <ConfigGroup name="Protocol AMQP Config">
            <ConfigItem name="AnonymousUser" value="anonymous_amqp"/>
            <ConfigItem name="BufferSize" value="10240"/>
            <ConfigItem name="DefaultNodeMode" value="0"/>
            <ConfigItem name="Enable" value="true"/>
            <ConfigItem name="EnableWriteThread" value="false"/>
            <ConfigItem name="EngineLoopCount" value="50"/>
            <ConfigItem name="MaxFrameSize" value="10000000"/>
```

```
            <ConfigItem name="MaxThreadPoolSize" value="10"/>
            <ConfigItem name="MinThreadPoolSize" value="1"/>
            <ConfigItem name="QueuePrefix" value="queue://"/>
            <ConfigItem name="SASL_Anonymous" value="true"/>
            <ConfigItem name="SASL_CRAM-MD5" value="false"/>
            <ConfigItem name="SASL_DIGEST-MD5" value="false"/>
            <ConfigItem name="SASL_Plain" value="false"/>
            <ConfigItem name="SubscriberCredit" value="1000"/>
            <ConfigItem name="Timeout" value="60000"/>
            <ConfigItem name="TopicPrefix" value="topic://"/>
            <ConfigItem name="TransformToUse" value="2"/>
        </ConfigGroup>
        <ConfigGroup name="Protocol MQTT Config">
            <ConfigItem name="DisconnectClientsOnPublishFailure" value="true"/>
            <ConfigItem name="Enable" value="true"/>
            <ConfigItem name="EnableAutoCreateTopics" value="true"/>
            <ConfigItem name="EnforceAlphaNumericClientID" value="false"/>
            <ConfigItem name="IgnoreClientIDLength" value="true"/>
            <ConfigItem name="MaxOutstanding" value="64000"/>
            <ConfigItem name="QoS0AsTransient" value="false"/>
            <ConfigItem name="SessionStateTTL" value="259200000"/>
            <ConfigItem name="Strict" value="true"/>
            <ConfigItem name="SupportZeroLength" value="true"/>
        </ConfigGroup>
        <ConfigGroup name="RecoveryDaemon">
            <ConfigItem name="EventsPerBlock" value="500"/>
            <ConfigItem name="ThreadPool" value="4"/>
        </ConfigGroup>
        <ConfigGroup name="Server Protection">
            <ConfigItem name="EnableFlowControl" value="false"/>
            <ConfigItem name="FlowControlWaitTimeOne" value="16000"/>
            <ConfigItem name="FlowControlWaitTimeThree" value="10000"/>
            <ConfigItem name="FlowControlWaitTimeTwo" value="13332"/>
        </ConfigGroup>
        <ConfigGroup name="Thread Pool Config">
            <ConfigItem name="CommonPoolThreadSize" value="5"/>
            <ConfigItem name="ConnectionThreadPoolMaxSize" value="10"/>
            <ConfigItem name="ConnectionThreadPoolMinSize" value="4"/>
            <ConfigItem name="ConnectionThreadWaitTime" value="120000"/>
            <ConfigItem name="EnableConnectionThreadPooling" value="true"/>
            <ConfigItem name="MaxUnauthorisedCount" value="1000"/>
            <ConfigItem name="PendingTaskWarningThreshold" value="1000"/>
            <ConfigItem name="ReadThreadPoolMaxSize" value="100"/>
            <ConfigItem name="ReadThreadPoolMinSize" value="4"/>
            <ConfigItem name="SchedulerPoolSize" value="10"/>
            <ConfigItem name="SlowTaskWarningTime" value="5000"/>
            <ConfigItem name="StalledTasksWarningTime" value="60000"/>
            <ConfigItem name="ThreadDumpInterval" value="60000"/>
            <ConfigItem name="ThreadDumpOnSlowTask" value="false"/>
            <ConfigItem name="ThreadIdleQueueSize" value="10"/>
            <ConfigItem name="WriteThreadPoolMaxSize" value="1000"/>
            <ConfigItem name="WriteThreadPoolMinSize" value="5"/>
        </ConfigGroup>
        <ConfigGroup name="TransactionManager">
            <ConfigItem name="MaxEventsPerTransaction" value="0"/>
            <ConfigItem name="MaxTransactionTime" value="300000"/>
            <ConfigItem name="TTLThreshold" value="1000"/>
        </ConfigGroup>
    </RealmConfiguration>
<RealmPermissionSet>
```

```
        <RealmACLEntry listACLEntries="false" modifyACLEntries="false"
fullControl="true"
connectToRealm="false" changeRealmConfig="false" addremoveChannels="false"
addremoveJoins="false"
addremoveRealms="false" overrideConnectionCount="false" useAdminAPI="false"
manageDatagroups="false"
publishDatagroups="false" ownDatagroups="false" host="0:0:0:0:0:0:0:1" name="rgav"/>
        <RealmACLEntry listACLEntries="false" modifyACLEntries="false"
fullControl="true"
connectToRealm="false" changeRealmConfig="false" addremoveChannels="false"
addremoveJoins="false"
addremoveRealms="false" overrideConnectionCount="false" useAdminAPI="false"
manageDatagroups="false"
publishDatagroups="false" ownDatagroups="false" host="127.0.0.1" name="rgav"/>
        <RealmGroupACLEntry listACLEntries="true" modifyACLEntries="true"
fullControl="true"
connectToRealm="true" changeRealmConfig="true" addremoveChannels="true"
addremoveJoins="true"
addremoveRealms="true" overrideConnectionCount="true" useAdminAPI="true"
manageDatagroups="true"
publishDatagroups="true" ownDatagroups="true" groupname="Everyone"/>
    </RealmPermissionSet>
    <ClusterSet>
        <ClusterEntry name="cluster_1">
            <ClusterMember name="realm0" rname="nsp://localhost:11000"
canBeMaster="true"/>
            <ClusterMember name="realm1" rname="nsp://localhost:11010"
canBeMaster="true"/>
            <ClusterMember name="realm2" rname="nsp://localhost:11020"
canBeMaster="true"/>
        </ClusterEntry>
    </ClusterSet>
    <RealmSet>
        <RealmEntry name="realm1" rname="nhp://10.248.27.186:11010"/>
        <RealmEntry name="realm2" rname="nhp://10.248.27.186:11020"/>
    </RealmSet>
    <ChannelSet>
        <ChannelEntry>
          <ChannelAttributesEntry name="/customer/sales/JMSTopic" TTL="0" capacity="0"
 EID="0"
clusterWide="true" jmsEngine="true" mergeEngine="false" type="MIXED_TYPE"/>
           <StorePropertiesEntry HonorCapacityWhenFull="true" SyncOnEachWrite="false"

SyncMaxBatchSize="0" SyncBatchTime="0" PerformAutomaticMaintenance="true"
EnableCaching="true"
CacheOnReload="true" EnableReadBuffering="true" ReadBufferSize="10240" Priority="4"
EnableMulticast="false" MultiFileEventsPerSpindle="50000" StampDictionary="0"/>
            <ChannelPermissionSet>
                <ChannelACLEntry listACLEntries="false" modifyACLEntries="false"
fullControl="false" getLastEID="true" purgeEvents="false" subscribe="true"
publish="true"
useNamedSubcription="false" host="*" name="user"/>
                <ChannelACLEntry listACLEntries="true" modifyACLEntries="true"
fullControl="true"
getLastEID="true" purgeEvents="true" subscribe="true" publish="true"
useNamedSubcription="true"
host="192.168.1.2" name="user"/>
               <ChannelGroupACLEntry listACLEntries="false" modifyACLEntries="false"
```

```
fullControl="false" getLastEID="true" purgeEvents="false" subscribe="true"
publish="false"
useNamedSubcription="true" groupname="Everyone"/>
            </ChannelPermissionSet>
        </ChannelEntry>
        <ChannelEntry>
           <ChannelAttributesEntry name="/naming/defaultContext" TTL="0" capacity="0"
 EID="2"
clusterWide="true" jmsEngine="false" mergeEngine="false" type="MIXED_TYPE"/>
          <StorePropertiesEntry HonorCapacityWhenFull="true" SyncOnEachWrite="false"

SyncMaxBatchSize="0" SyncBatchTime="0" PerformAutomaticMaintenance="true"
EnableCaching="true"
CacheOnReload="true" EnableReadBuffering="true" ReadBufferSize="10240" Priority="4"
EnableMulticast="false" MultiFileEventsPerSpindle="50000" StampDictionary="0"/>
            <ChannelPermissionSet>
                <ChannelACLEntry listACLEntries="true" modifyACLEntries="true"
fullControl="true"
getLastEID="true" purgeEvents="true" subscribe="true" publish="true"
useNamedSubcription="true"
host="10.248.27.186" name="rgav"/>
                <ChannelGroupACLEntry listACLEntries="true" modifyACLEntries="true"
fullControl="true" getLastEID="true" purgeEvents="true" subscribe="true" publish="true"

useNamedSubcription="true" groupname="Everyone"/>
            </ChannelPermissionSet>
            <ChannelKeySet>
                <ChannelKeyEntry keyName="alias" keyDepth="1"/>
            </ChannelKeySet>
            <EventsSet>
                <Event id="0">
                    <EventAttribSet>
                        <EventAttrib name="nrvpub.time" type="Long"
value="1610454161489"/>
                        <EventAttrib name="nrvpub.host" type="String"
value="10.248.27.186"/>
                        <EventAttrib name="nrvpub.name" type="String" value="rgav"/>
                        <EventAttrib name="JMSDeliveryMode" type="String"
value="PERSISTENT"/>
                        <EventAttrib name="JMSPriority" type="Byte" value="4"/>
                    </EventAttribSet>
                    <EventPropSet>
                      <EventProp name="JMS_my-channels_EnableMultiplexedConnections"

type="Boolean" value="true"/>
                       <EventProp name="JMS_my-channels_RandomRNames" type="Boolean"

value="false"/>
                        <EventProp name="JMS_my-channels_RetryCommit" type="Boolean"

value="false"/>
                       <EventProp name="JMS_my-channels_ConxExceptionOnRetryFailure"

type="Boolean" value="false"/>
                        <EventProp name="JMS_my-channels_MaxReconnectAttempts"
type="Integer"
value="-1"/>
                        <EventProp name="JMS_my-channels_EnableSharedDurable"
type="Boolean"
value="true"/>
```

```
                          <EventProp name="JMS_my-channels_EnableSerialDurable"
type="Boolean"
value="false"/>
                        <EventProp name="JMS_my-channels_EnableSingleSharedDurableAck"

type="Boolean" value="false"/>
                          <EventProp name="JMS_my-channels_EnableSingleQueueAck"
type="Boolean"
value="false"/>
                          <EventProp name="JMS_my-channels_SyncWritesToDisc"
type="Boolean"
value="false"/>
                          <EventProp name="JMS_my-channels_SyncSendPersistent"
type="Boolean"
value="true"/>
                         <EventProp name="JMS_my-channels_InitialConnectionRetryCount"

type="Integer" value="2"/>
                        <EventProp name="JMS_my-channels_SyncBatchSize" type="Integer"

value="50"/>
                          <EventProp name="JMS_my-channels_SyncTime" type="Integer"
value="20"/>
                          <EventProp name="JMS_my-channels_GlobalStoreCapacity"
type="Integer"
value="0"/>
                         <EventProp name="JMS_my-channels_AutoAckCount" type="Integer"
 value="50"/>
                          <EventProp name="JMS_my-channels_WindowSize" type="Integer"
value="100"/>
                          <EventProp name="JMS_my-channels_useInfiniteWindowSize"
type="Boolean"
value="false"/>
                          <EventProp name="JMS_my-channels_UnAckedSize" type="Integer"
 value="100"/>
                        <EventProp name="JMS_my-channels_RedliveredSize" type="Integer"
 value="100"/>
                        <EventProp name="JMS_my-channels_ThreadPoolSize" type="Integer"
 value="30"/>
                          <EventProp name="JMS_my-channels_AutoReconnectAfterACL"
type="Boolean"
value="false"/>
                          <EventProp name="JMS_my-channels_ImmediateReconnect"
type="Boolean"
value="true"/>
                        <EventProp name="JMS_my-channels_ReconnectInterval" type="Long"
 value="2000"/>
                          <EventProp name="JMS_my-channels_UseJMSEngine" type="Boolean"
 value="true"/>
                        <EventProp name="JMS_my-channels_DisconnectAfterClusterFailure"

type="Boolean" value="true"/>
                        <EventProp name="JMS_my-channels_ConnectionTimeout" type="Long"

value="10000"/>
                          <EventProp name="JMS_my-channels_PermittedKeepAlivesMissed"
type="Integer" value="2"/>
                          <EventProp name="JMS_my-channels_SyncNamedTopicAcks"
type="Boolean"
value="true"/>
```

```
                        <EventProp name="JMS_my-channels_AdapterBuffer" type="Integer"
value="1310720"/>
                         <EventProp name="JMS_my-channels_WriteHandler" type="Integer"
 value="3"/>
                        <EventProp name="JMS_my-channels_SyncQueueAcks" type="Boolean"
 value="true"/>
                        <EventProp name="JMS_my-channels_SyncTopicAcks" type="Boolean"
 value="true"/>
                         <EventProp name="JMS_my-channels_AutoCreateResource"
type="Boolean"
value="true"/>
                         <EventProp name="JMS_my-channels_FollowMaster" type="Boolean"
 value="false"/>
                         <EventProp name="alias.type" type="Integer" value="1"/>
                         <EventProp name="alias" type="String"
value="ExampleConnectionFactory"/>
                         <EventProp name="alias.reference.classname" type="String"
value="javax.jms.ConnectionFactory"/>
                         <EventProp name="alias.reference.factoryclass" type="String"
value="com.pcbsys.nirvana.nJMS.ConnectionFactoryFactory"/>
                         <EventProp name="alias.reference.url" type="String"
value="{&amp;quot;JMS_my-channels_EnableMultiplexedConnections&amp;quot;:[3,true],&amp;quot;
JMS_my-channels_RandomRNames&amp;quot;:[3,false],&amp;quot;JMS_my-channels_RetryCommit&amp;quot;
:[3,false],&amp;quot;JMS_my-channels_ConxExceptionOnRetryFailure&amp;quot;:[3,false],&amp;quot;
JMS_my-channels_MaxReconnectAttempts&amp;quot;:[4,-1],&amp;quot;JMS_my-channels_EnableSharedDurable&amp;quot;
:[3,true],&amp;quot;JMS_my-channels_EnableSerialDurable&amp;quot;:[3,false],&amp;quot;
JMS_my-channels_EnableSingleSharedDurableAck&amp;quot;:[3,false],&amp;quot;
JMS_my-channels_EnableSingleQueueAck&amp;quot;:[3,false],&amp;quot;JMS_my-channels_SyncWritesToDisc&amp;quot;
:[3,false],&amp;quot;JMS_my-channels_SyncSendPersistent&amp;quot;:[3,true],&amp;quot;
JMS_my-channels_InitialConnectionRetryCount&amp;quot;:[4,2],&amp;quot;JMS_my-channels_SyncBatchSize&amp;quot;
:[4,50],&amp;quot;JMS_my-channels_SyncTime&amp;quot;:[4,20],&amp;quot;JMS_my-channels_GlobalStoreCapacity&amp;quot;
:[4,0],&amp;quot;JMS_my-channels_AutoAckCount&amp;quot;:[4,50],&amp;quot;JMS_my-channels_WindowSize&amp;quot;
:[4,100],&amp;quot;JMS_my-channels_useInfiniteWindowSize&amp;quot;:[3,false],&amp;quot;
JMS_my-channels_UnAckedSize&amp;quot;:[4,100],&amp;quot;JMS_my-channels_RedliveredSize&amp;quot;
:[4,100],&amp;quot;JMS_my-channels_ThreadPoolSize&amp;quot;:[4,30],&amp;quot;
JMS_my-channels_AutoReconnectAfterACL&amp;quot;:[3,false],&amp;quot;
JMS_my-channels_ImmediateReconnect&amp;quot;:[3,true],&amp;quot;JMS_my-channels_ReconnectInterval&amp;quot;
:[1,2000],&amp;quot;JMS_my-channels_UseJMSEngine&amp;quot;:[3,true],&amp;quot;
JMS_my-channels_DisconnectAfterClusterFailure&amp;quot;:[3,true],&amp;quot;
JMS_my-channels_ConnectionTimeout&amp;quot;:[1,10000],&amp;quot;
JMS_my-channels_PermittedKeepAlivesMissed&amp;quot;:[4,2],&amp;quot;
JMS_my-channels_SyncNamedTopicAcks&amp;quot;:[3,true],&amp;quot;JMS_my-channels_AdapterBuffer&amp;quot;
:[4,1310720],&amp;quot;JMS_my-channels_WriteHandler&amp;quot;:[4,3],&amp;quot;
JMS_my-channels_SyncQueueAcks&amp;quot;:[3,true],&amp;quot;JMS_my-channels_SyncTopicAcks&amp;quot;
:[3,true],&amp;quot;JMS_my-channels_AutoCreateResource&amp;quot;:[3,true],&amp;quot;
JMS_my-channels_FollowMaster&amp;quot;:[3,false]}"/>
                         <EventProp name="alias.stringRefAddr.type" type="String"
value="ConnectionFactory"/>
                         <EventProp name="alias.stringRefAddr.addr" type="String"
value="nsp://localhost:11000,nsp://localhost:11010,nsp://localhost:11020"/>
                    </EventPropSet>
                    <EventData>RXhhbXBsZUNvbm5lY3Rpb25GYWN0b3J5</EventData>
                </Event>
                <Event id="1">
                    <EventAttribSet>
                        <EventAttrib name="nrvpub.time" type="Long"
value="1610454171673"/>
```

```
                        <EventAttrib name="nrvpub.host" type="String"
value="10.248.27.186"/>
                        <EventAttrib name="nrvpub.name" type="String" value="rgav"/>
                        <EventAttrib name="JMSDeliveryMode" type="String"
value="PERSISTENT"/>
                        <EventAttrib name="JMSPriority" type="Byte" value="4"/>
                    </EventAttribSet>
                    <EventPropSet>
                        <EventProp name="alias.type" type="Integer" value="1"/>
                        <EventProp name="alias" type="String"
value="/customer/sales/JMSTopic"/>
                        <EventProp name="alias.reference.classname" type="String"
value="javax.jms.Topic"/>
                        <EventProp name="alias.reference.factoryclass" type="String"

value="com.pcbsys.nirvana.nJMS.TopicFactory"/>
                      <EventProp name="alias.reference.url" type="String" value=""/>
                        <EventProp name="alias.stringRefAddr.type" type="String"
value="Topic"/>
                        <EventProp name="alias.stringRefAddr.addr" type="String"
value="/customer/sales/JMSTopic"/>
                    </EventPropSet>
                    <EventData>L2N1c3RvbWVyL3NhbGVzL0pNU1RvcGlj</EventData>
                </Event>
                <Event id="2">
                    <EventAttribSet>
                        <EventAttrib name="nrvpub.time" type="Long"
value="1610454193768"/>
                        <EventAttrib name="nrvpub.host" type="String"
value="10.248.27.186"/>
                        <EventAttrib name="nrvpub.name" type="String" value="rgav"/>
                        <EventAttrib name="JMSDeliveryMode" type="String"
value="PERSISTENT"/>
                        <EventAttrib name="JMSPriority" type="Byte" value="4"/>
                    </EventAttribSet>
                    <EventPropSet>
                        <EventProp name="alias.type" type="Integer" value="1"/>
                        <EventProp name="alias" type="String"
value="/customer/sales/JMSQueue"/>
                        <EventProp name="alias.reference.classname" type="String"
value="javax.jms.Queue"/>
                        <EventProp name="alias.reference.factoryclass" type="String"

value="com.pcbsys.nirvana.nJMS.QueueFactory"/>
                      <EventProp name="alias.reference.url" type="String" value=""/>
                        <EventProp name="alias.stringRefAddr.type" type="String"
value="Queue"/>
                        <EventProp name="alias.stringRefAddr.addr" type="String"
value="/customer/sales/JMSQueue"/>
                    </EventPropSet>
                    <EventData>L2N1c3RvbWVyL3NhbGVzL0pNU1F1ZXVl</EventData>
                </Event>
            </EventsSet>
        </ChannelEntry>
        <ChannelEntry>
           <ChannelAttributesEntry name="/partner/sales" TTL="0" capacity="0" EID="0"

clusterWide="true" jmsEngine="false" mergeEngine="false" type="MIXED_TYPE"/>
           <StorePropertiesEntry HonorCapacityWhenFull="true" SyncOnEachWrite="false"
```

```
SyncMaxBatchSize="0" SyncBatchTime="0" PerformAutomaticMaintenance="true"
EnableCaching="true"
CacheOnReload="true" EnableReadBuffering="true" ReadBufferSize="10240" Priority="4"
EnableMulticast="false" MultiFileEventsPerSpindle="50000" StampDictionary="0"/>
            <ChannelPermissionSet>
                <ChannelACLEntry listACLEntries="false" modifyACLEntries="false"
fullControl="false" getLastEID="true" purgeEvents="false" subscribe="true"
publish="true"
useNamedSubcription="false" host="*" name="user"/>
                <ChannelACLEntry listACLEntries="true" modifyACLEntries="true"
fullControl="true"
getLastEID="true" purgeEvents="true" subscribe="true" publish="true"
useNamedSubcription="true"
host="192.168.1.2" name="user"/>
               <ChannelGroupACLEntry listACLEntries="false" modifyACLEntries="false"

fullControl="false" getLastEID="true" purgeEvents="false" subscribe="true"
publish="false"
useNamedSubcription="true" groupname="Everyone"/>
            </ChannelPermissionSet>
            <DurableSet>
                <durableEntry name="serial_durable" EID="-1" outstandingEvents="0"
clusterWide="true" persistent="true" shared="false" serial="true"/>
                <durableEntry name="shared_durable" EID="-1" outstandingEvents="0"
clusterWide="true" persistent="true" shared="true" serial="false"/>
            </DurableSet>
        </ChannelEntry>
    </ChannelSet>
    <QueueSet>
        <QueueEntry>
          <ChannelAttributesEntry name="/customer/sales/JMSQueue" TTL="0" capacity="0"
 EID="0"
clusterWide="true" jmsEngine="false" mergeEngine="false" type="MIXED_TYPE"/>
           <StorePropertiesEntry HonorCapacityWhenFull="true" SyncOnEachWrite="false"

SyncMaxBatchSize="0" SyncBatchTime="0" PerformAutomaticMaintenance="true"
EnableCaching="true"
CacheOnReload="true" EnableReadBuffering="true" ReadBufferSize="10240" Priority="4"
EnableMulticast="false" MultiFileEventsPerSpindle="50000" StampDictionary="0"/>
            <QueuePermissionSet>
                <QueueACLEntry listACLEntries="false" modifyACLEntries="false"
fullControl="false"
purge="false" peek="true" push="true" pop="true" host="*" name="user"/>
                <QueueACLEntry listACLEntries="true" modifyACLEntries="true"
fullControl="true"
purge="true" peek="true" push="true" pop="true" host="192.168.1.2" name="user"/>
                <QueueGroupACLEntry listACLEntries="false" modifyACLEntries="false"
fullControl="false" purge="false" peek="true" push="false" pop="false"
groupname="Everyone"/>
            </QueuePermissionSet>
        </QueueEntry>
        <QueueEntry>
            <ChannelAttributesEntry name="/partner/queries" TTL="0" capacity="0"
EID="0"
clusterWide="true" jmsEngine="false" mergeEngine="false" type="MIXED_TYPE"/>
           <StorePropertiesEntry HonorCapacityWhenFull="true" SyncOnEachWrite="false"

SyncMaxBatchSize="0" SyncBatchTime="0" PerformAutomaticMaintenance="true"
EnableCaching="true"
CacheOnReload="true" EnableReadBuffering="true" ReadBufferSize="10240" Priority="4"
```

```
EnableMulticast="false" MultiFileEventsPerSpindle="50000" StampDictionary="0"/>
            <QueuePermissionSet>
                <QueueACLEntry listACLEntries="false" modifyACLEntries="false"
fullControl="false"
purge="false" peek="true" push="true" pop="true" host="*" name="user"/>
                <QueueACLEntry listACLEntries="true" modifyACLEntries="true"
fullControl="true"
purge="true" peek="true" push="true" pop="true" host="192.168.1.2" name="user"/>
                <QueueGroupACLEntry listACLEntries="false" modifyACLEntries="false"
fullControl="false" purge="false" peek="true" push="false" pop="false"
groupname="Everyone"/>
            </QueuePermissionSet>
        </QueueEntry>
    </QueueSet>
    <DataGroupSet/>
    <RealmInterfaces>
        <RealmNHPInterface>
          <RealmInterface name="nhp0" port="11000" adapter="0.0.0.0" autostart="true"

advertise="true" authtime="10000" backlog="100" acceptThreads="10" selectThreads="2"

sendbuffersize="1310720" receivebuffersize="1310720" allowforinterrealm="true"
allowclientconnections="true" EnableNIO="true"/>
            <RealmInterfacePlugin mountPoint="/" name="File Plugin">
                <NirvanaFilePlugin>
                    <PluginConfigEntry name="AddUserAsCookie" value=""
description="Add the username to the sessions cookies"/>
                    <PluginConfigEntry name="AuthParameters" value=""
description="List of key=value string which is passed to the authenticators init
function"/>
                    <PluginConfigEntry name="Authenticator" value=""
description="Name of authenticator to use, leave to use default, else classname to
use"/>
                    <PluginConfigEntry name="BasePath"
value="C:\Users\RGAV\dev\NUM-15094\ide\realmDirectories\realm0\plugins\htdocs"
description="Path used to locate the files"/>
                    <PluginConfigEntry name="BufferSize" value=""
description="Size of the internal buffer to use to send the data"/>
                    <PluginConfigEntry name="Cache-Control" value=""
description="Specifies the cache control for the plugin"/>
                    <PluginConfigEntry name="CacheObjectSize" value=""
description="Size in bytes that can be stored in the cache"/>
                    <PluginConfigEntry name="CachedObjects" value=""
description="Number of objects to store in the cache"/>
                    <PluginConfigEntry name="DefaultName" value="index.html"
description="If no file name is specified which file should be returned"/>
                    <PluginConfigEntry name="EnableURLRewrite" value=""
description="If the plugin will scan the source and rewrite the urls"/>
                    <PluginConfigEntry name="FileNotFoundPage" value=""
description="name of the file to send when file can not be located"/>
                    <PluginConfigEntry name="GroupNames" value=""
description="A comma seperated list of groups which the user must be a member of at
least one to
be granted access"/>
                    <PluginConfigEntry name="MimeType" value="" description="Name of
 the file to
load the mime type information from"/>
                    <PluginConfigEntry name="ReloadUserFileDynamically" value=""
description="Choose true to have the user file get reloaded on each auth request"/>
```

```
                    <PluginConfigEntry name="RoleNames" value="" description="A comma
 seperated
list of roles which the user must have at least one of to be granted access"/>
                  <PluginConfigEntry name="Security Realm" value="" description="Name
 of the
authentication realm"/>
                    <PluginConfigEntry name="SeparateAccessandErrorLogs" value=""
description="Choose true to have separate log files"/>
                </NirvanaFilePlugin>
            </RealmInterfacePlugin>
            <JavascriptConfigEntry CORSAllowedOrigins="*" EnableJavaScript="true"
EnableWebSockets="true" CORSAllowCredentials="true" EnableGZipLP="true"
MinimumBytesBeforeGZIP="1000" AjaxLPIdleDelay="60000" AjaxLPActiveDelay="100"/>
        </RealmNHPInterface>
    </RealmInterfaces>
    <RealmSecurityGroupSet/>
    <RealmSchedulerSet>
        <Scheduler
source="c2NoZWR1bGVyIHJlYWxtTG9nU2NoZWR1bGUgewoJaW5pdGlhbGlzZSB7CgogCQlMb2dnZXI
ucmVwb3J0KCJEZWZhdWx0IHJlYWxtIGxvZyBmaWxlIGF1dG8gcm9sbCBpbml0aWFsaXNlCIpOwoKCSB9CgoJZXZlcnkgMDow
MCB7CgoJCUxvZ2dlci5yb2xsKCk7CgkJTG9nZ2VyLnJlcG9ydCTG9nIGF1dG9tYXRpY2FsbHkgcm9sbGVkIGJ5IGRlZmF1b
HQgc2NoZWR1bGVkIHNjmlwdC4gSWYgdGhpcyBpcyBub3QgcmVxdWlyZWQgcGxlYXNlIHJlbW92ZSB0aGUgc2NyaXB0IGZyb2
0gdGhlIHNlcnZlciIpOwoKCX0KfQ==" subject="[rgav@0:0:0:0:0:0:0:1, rgav@127.0.0.1]"
clusterWide="false"/>
    </RealmSchedulerSet>
</NirvanaRealm>
```

Using the Enterprise Viewer

The Enterprise Viewer is a read-only version of the Enterprise Manager. Its purpose is to allow clients to view the Universal Messaging environment without the need for special administration rights.

Basically, it offers the same views as the Enterprise Manager, but you cannot use it to modify your Universal Messaging environment in any way. This means, for example, that you cannot create or delete channels and queues, and you cannot publish any events.

Starting the Enterprise Viewer

Windows platforms

Windows users can start the Enterprise Viewer by selecting the appropriate component from the Universal Messaging group in the Windows Start menu.

You can also type a command of the following form on the command line:

```
<InstallDir>\UniversalMessaging\java\<InstanceName>\bin\nenterpriseview.exe
```

where <*InstallDir*> is the installation root location and <*InstanceName*> is the name of the Universal Messaging server.

UNIX-based platforms

You can launch the Enterprise Viewer on UNIX-based platforms by starting the `nenterpriseview` executable, which you can find at the following location:

```
<InstallDir>/UniversalMessaging/java/umserver/bin/nenterpriseview
```

3 Using Command Central to Manage Universal Messaging

- Managing Universal Messaging Using Command Central 354
- Securing Communication Between Command Central and Universal Messaging 354
- Securing Access to Command Central 356
- Instance Management 358
- Authentication Configuration 360
- Universal Messaging Configuration Types 360
- Universal Messaging Administration Types 387
- Snooping on Channels 391
- Snooping on Queues 393
- Publishing Events 396
- Universal Messaging Cloud Transformation 398
- Universal Messaging Logs 399
- Universal Messaging Inventory 399
- Universal Messaging Lifecycle Actions 399
- Universal Messaging KPIs 400
- Universal Messaging Run-time Monitoring Statuses 401
- Universal Messaging and the Command Line Interface 401

Managing Universal Messaging Using Command Central

You can configure and administer Universal Messaging server instances using the Command Central web or command-line interface.

Command Central uses one of the Universal Messaging ports (interfaces) for configuration and administration. Command Central checks the interfaces of a Universal Messaging server instance in the following order and chooses the first available interface to connect to the server:

1. Interfaces that use the HTTP protocol (nhp).
2. Interfaces that use the socket protocol (nsp).
3. Interfaces that use the HTTPS protocol (nhps).
4. Interfaces that use the SSL protocol (nsps).

Note:
Command Central does not use interfaces that use the shared memory protocol (shm).

In case of a disconnection between Command Central and the Universal Messaging server, Command Central uses the same order to connect to a new Universal Messaging port.

Securing Communication Between Command Central and Universal Messaging

If you want to guarantee secure communication between Command Central and Universal Messaging, you must have only an nhps or nsps port (interface) configured on the Universal Messaging server. When the Universal Messaging server instance is configured with a single nhps or nsps interface, Command Central uses this interface to connect automatically to the Universal Messaging server. By default, Command Central uses the same truststore file and, in case of client-side authentication, the same keystore file that are configured in the nhps or nsps interface.

If you want to specify truststore and keystore files that are different from the ones configured in the nhps or nsps interface, you can use either the standard Java Secure Socket Extension (JSSE) system properties or the Universal Messaging client system properties for secure communication. For information about how to configure the properties, see " Configuring the JSSE System Properties" on page 355 and " Configuring the Universal Messaging Client Properties" on page 355.

Considerations When Using System Properties to Specify Truststore and Keystore Files

Consider the following information before you use system properties to specify custom truststore and keystore files for secure communication between Command Central and a Universal Messaging server instance:

- If you want to connect to a Universal Messaging server instance that is part of a cluster or a zone, or that you plan to add to a cluster or a zone, ensure that the custom truststore contains the certificates of all server instances that are part of the cluster or zone.

- Configuring the standard JSSE system properties might impact all product instances that use secure sockets layer (SSL) in the same Platform Manager installation.

Configuring the JSSE System Properties

Use the following procedure to configure the JSSE system properties for a custom truststore and keystore to secure communication between Command Central and Universal Messaging.

Important:
Setting the JSSE system properties might impact all run-time components that use SSL in the same Platform Manager installation.

To configure the JSSE system properties

1. In the Command Central web user interface, go to the Platform Manager instance that is in the same installation as the Universal Messaging server.
2. Click **Configuration > Java System Properties > Edit**.
3. Add the following properties:
   - `com.softwareag.um.plugin.use.ssl.system.properties=true`
   - `javax.net.ssl.keyStore=<path to the custom keystore>` - Required only if client authentication is enabled on the nsps or nhps interface.
   - `javax.net.ssl.keyStorePassword=<password of the custom keystore>` - Required only if client authentication is enabled on the nsps or nhps interface.
   - `javax.net.ssl.trustStore=<path to the custom truststore>`
   - `javax.net.ssl.trustStorePassword=<password of the custom truststore>`
4. Click **Apply**.
5. Restart Platform Manager.

Configuring the Universal Messaging Client Properties

Use the following procedure to configure the Universal Messaging client system properties for a custom truststore and keystore to secure communication between Command Central and Universal Messaging.

To configure the Universal Messaging client system properties

1. In the Command Central web user interface, go to the Platform Manager instance that is in the same installation as the Universal Messaging server.

2. Click **Configuration > Java System Properties > Edit**.

3. Add the following properties:

   - `com.softwareag.um.plugin.use.ssl.system.properties=true`
   - `com.softwareag.um.client.ssl.keystore_path=<path to the custom keystore>` - Required only if client authentication is enabled on the nsps or nhps interface.
   - `com.softwareag.um.client.ssl.keystore_password=<password of the custom keystore>` - Required only if client authentication is enabled on the nsps or nhps interface.
   - `com.softwareag.um.client.ssl.certificate_alias=<the alias of the certificate in the keystore that Command Central should use>` - Required only if client authentication is enabled on the nsps or nhps interface and the keystore contains more than one certificate.
   - `com.softwareag.um.client.ssl.truststore_path=<path to the custom truststore>`
   - `com.softwareag.um.client.ssl.truststore_password=<password of the custom truststore>`

4. Click **Apply**.

5. Restart Platform Manager.

Switching Off the System Properties Mode

To stop using custom truststore and keystore files for secure communication between Command Central and a Universal Messaging server instance, you must set `com.softwareag.um.plugin.use.ssl.system.properties` to `false`.

To stop using custom truststore and keystore files

1. In the Command Central web user interface, go to the Platform Manager instance that is in the same installation as the Universal Messaging server instance.

2. Click **Configuration > Java System Properties > Edit**.

3. Set `com.softwareag.um.plugin.use.ssl.system.properties` to `false`.

4. Click **Apply**.

5. Restart Platform Manager.

Securing Access to Command Central

Secure access to Command Central by performing one or more of the following tasks:

Changing the Authentication Mode

1. Click the Instances tab to view all the available Universal Messaging server instances.

2. Click an Universal Messaging instance name to access the Dashboard.

   The Dashboard contains information about the specific Universal Messaging server instance such as status, alerts, and KPIs.

3. In the Details section of the Dashboard, click in the **Authentication** field to change the authentication mode.

   The **Authentication Mode** dialog box appears.

4. Select one of the following authentication modes:

   - **System default**: Use default authentication method.
   - **None**: No authentication method is used.
   - **Trusted**: Password-less authentication for predefined administrative user account.
   - **Delegated**: SAML-based authentication and authorization for the currently logged in user.
   - **Fixed user**: Authenticate the specified administrative user credentials. Provide a user name and password for the user.

   Note:
   To use basic authentication, you must change the authentication mode for a run-time component to **Fixed User**. Command Central uses basic authentication with a fixed user to communicate with Platform Manager. With **Fixed User** authentication, the authentication credentials for the Platform Manager will be fixed.

Verifying the Outbound Authentication Settings

Use the following steps to verify that Command Central is configured with the correct outbound authentication settings.

To verify that Command Central is configured with the correct user credentials

1. In Command Central, on the Overview tab for the product component, click . Check that the product status is **Online** and the JVM KPIs are updated.

2. On the Logs tab, check the product log for authentication errors.

Using Unix Shell Scripts to Change Connection Credentials for Managed Products

You can use the following sample UNIX shell script to configure basic authentication credentials for product components managed by Command Central.

```
NODE_ALIAS=local
USERNAME=Administrator
PASSWORD=secret
RCID=integrationServer-default
# RCID=MwsProgramFiles-default
# RCID=Universal-Messaging-nirvana
# RCID=OSGI-CTP
# RCID=OSGI-InfraDC

sagcc get configuration data $NODE_ALIAS $RCID COMMON-LOCAL-USERS-Administrator

-o administrator.xml
sed "s,/>,><Password>${PASSWORD}</Password></User>,g" administrator.xml >

 administrator_new.xml
sagcc update configuration data $NODE_ALIAS $RCID COMMON-LOCAL-USERS

-Administrator -o administrator_new.xml

# verify connection
sagcc get monitoring runtimestatus $NODE_ALIAS $RCID -e ONLINE
```

Instance Management

You can create and delete Universal Messaging server instances using Command Central web or command-line interface.

Creating an Instance

To create an instance

1. In the **Environments** pane, select the environment in which you want to create the new product instance.
2. Click the **Installations** tab.
3. Select the installation by clicking the installation name in the table.
4. Click the **Instances** tab.
5. Click , and select **Universal Messaging Server**.

6. Specify the following instance properties:

Property	Description
Instance name	Required. Name of the new Universal Messaging server instance. Case-insensitive and can include upper and lower case alphabetic characters, digits (0-9), underscores (_), and non-leading hyphens (-).
NHP interface binding	Specific host or IP address to bind. The instance will bind to all available interfaces if this field is left blank.
NHP interface port	Port number for the Universal Messaging server instance.
Data directory	Absolute path to the data directory of the Universal Messaging server instance. If you do not specify a value, the default path "*install directory*/UniversalMessaging/server/*instance name*" is used.
License file	The registered Universal Messaging license file. Select one of the registered license files from the list.
Configuration profile	Initial configuration settings for the Universal Messaging server instance. Options are: ■ **webMethods suite use cases** ■ **Standalone use cases** ■ **Custom**

7. Click **Next**, and then click **Finish**.

Deleting an Instance

To delete an instance

1. In the **Environments** pane, select the environment in which you want to create the new product instance.

2. Click the **Instances** tab.

3. Select the Universal Messaging instance to be deleted and then click .

4. Click **Ok** to confirm deletion and then click **Finish**.

Note:

The Windows service associated with the instance is automatically deleted when the instance is deleted.

Authentication Configuration

Perform the following steps to enable basic authentication for Universal Messaging server instance users. For more information about authentication, the Authentication Overview section in the Universal Messaging Concepts guide.

Important:
JAAS Authentication with Software AG Security infrastructure component is required for completing basic authentication configuration, for more information see the Server JAAS Authentication with Software AG Security infrastructure component section in the Universal Messaging Concepts guide.

1. Use the **Internal Users** configuration type in Command Central if you want to add new internal users.
2. Use the **Realm ACL** configuration type to add the internal users in the format *username@host*, and configure the ACLs. For more information see "Realm Access Control Lists (ACLs)" on page 372.
3. Change the default Authentication Mode from **None** to **Fixed User** and provide a new user name and password. See "Changing the Authentication Mode" on page 357 for more information.

Universal Messaging Configuration Types

You can configure the following configuration types for a Universal Messaging server instance:

- Internal Users
- Licenses
- Ports
- Memory
- Realm ACL
- Groups
- Properties
- JNDI Connection Factories
- JNDI Destinations
- Channels
- Queues
- Zone

- Java System Properties
- JVM Options
- Clustering

Working with Universal Messaging Configuration Types

Perform the following steps to add, edit, or delete Universal Messaging configuration type items in the Command Central web user interface.

Note:
The Universal Messaging server instance must be running during configuration.

To add, edit, or delete an item for an Universal Messaging configuration type

1. Select the Universal Messaging environment from the **Environment** pane, then click the instance from the **Instances** tab.

2. Click the **Configuration** tab.

3. Select the configuration type from the drop-down list.

   Universal Messaging displays the available or default values for the selected Universal Messaging configuration type.

4. To add an item for the Universal Messaging configuration type, click . Enter the required values, and click **Save**.

5. To edit an item for a configuration type, click on the item that you want to update and then click **Edit**. Make the necessary changes and click one of the following:

   - **Test** to test the configuration type item.
   - **Save** to save your changes.
   - **Cancel** to cancel the edits to the configuration type item.

6. To delete an item for a configuration instance, click .

User Management

You can access the user management configuration options for a Universal Messaging server instance by selecting a Universal Messaging instance name and selecting **Internal Users** from the configurations list. You can add new users, list existing users, change the password of an user, or delete a user from the user repository.

Information to authenticate the users of a Universal Messaging server instance is stored in the user repository users.txt file. The default location of the users.txt file is <*InstallDir*>/common/conf/users.txt. The users.txt file is generated only after you create a new internal user.

The path to the users.txt file is added in the jaas.conf file present in <*InstallDir*>/UniversalMessaging/server/<*InstanceName*>/bin/jaas.conf. If you specify a relative path in the jaas.conf file, the users.txt file will be created in a directory relative to the bin directory of the Universal Messaging server instance. You can provide a custom name instead of the default users.txt, and a custom path in the jaas.conf file.

You can also use the command line interface commands or internaluserrepo.bat/sh script in <*InstallDir*>/common/bin to configure users of a Universal Messaging server instance. For more information, see *Software AG Command Central and Software AG Platform Manager Command Reference*.

License Management

For a Universal Messaging server, you can configure the license, view the details of the license that is configured, and retrieve the location of the license file.

Note:
You cannot change the location of a Universal Messaging license file.

Ports Configuration

In the Command Central web user interface, you can view, create, enable, disable, or edit the following Universal Messaging server ports (interfaces):

- NSP
- NHP
- NHPS
- NSPS
- SHM

On the Configuration > Ports page, you can view all the ports configured for a Universal Messaging server instance, as well as the following basic attributes for a port:

Column	Description
Enabled	Whether a port is enabled or disabled.
Alias	An alias that is used to recognize the port. Each port (interface) on a Universal Messaging server instance has an associated alias.
Address	■ The number of an NSP, NHP, NHPS, or NSPS port. The port number is unique.

Column	Description
	■ The path of an SHM port.
Protocol	The type of the port. The port uses one of the following protocols: ■ NSP (Socket protocol) ■ NHP (HTTP protocol) ■ NHPS (HTTPS protocol) ■ NSPS (SSL protocol) ■ SHM (shared memory protocol)

Configuring an NSP Port

To configure an NSP port (interface) for a Universal Messaging server instance

1. In Command Central, navigate to **Environments > Instances > All > Universal-Messaging-*instanceName* > Configuration > Ports**.

2. Click and select **NSP**.

3. Specify values for the following fields:

Field	Description
Enabled	Enable or disable this NSP port.
Adapter alias	Provide an alias for the port. Each Universal Messaging server instance can have an associated alias in the form of *host*:*port*. This alias is used to inform other servers how to contact this server, if this server is behind a NAT or a Proxy Server. This alias is not the same as the Universal Messaging assigned interface alias.
Number	Required. Provide a unique port number. The port number must be unique and cannot be used again in a Universal Messaging server instance.
Backlog	Provide the size of the Internet Protocol (IP) socket queue.
Bind address	The IP address to which to bind this port, if your machine has multiple IP addresses and you want the port to use this specific address. You cannot change this attribute after you create the port.

Field	Description
Autostart interface	Automatically start this port when starting the Universal Messaging server instance.
Advertise interface	Allow the Universal Messaging server instance to send information about this port to the client.
Allow for inter-realm	Allow port communication between two or more Universal Messaging server instances. For example, allow communication between clusters, or joins.
Allow client connections	Allow clients to connect to this port.
Enable NIO	Enable NIO (Network Input/Output) on this port.
Enable policy server	Allow the ports to respond to policy requests. You can run a policy file server on a socket interface that will automatically handle these requests. Once this is set up, you will also need to set up a client access policy in the clientaccesspolicy.xml file in the `/install/server/name/plugins/htdocs` directory of the server.
Auth time	Provide the time in milliseconds (ms) that the Universal Messaging server instance waits for the client to complete the authentication process. Default is 10000 milliseconds.
Accept threads	Provide the number of threads processing the accepted sockets.
Select threads	Provide the number of threads allocated for selection.
Send buffer size	Provide the size of the socket send buffer.
Receive buffer size	Provide the size of the socket receive buffer.

For more information about the basic port properties, see "Interface Configuration" on page 282.

4. Optionally, click **Test** to validate the configuration.

5. Click **Save**.

Configuring an NHP Port

To configure an NHP port (interface) for a Universal Messaging server instance

1. In Command Central, navigate to **Environments > Instances > All > Universal-Messaging-*instanceName* > Configuration > Ports**.

2. Click and select **NHP**.

3. Specify values for the following fields:

Connection Basics

Field	Description
Enabled	Enable or disable this NHP port.
Adapter alias	Specify an alias for the port. Each interface on a Universal Messaging server instance can have an associated alias in the format *host*:*port*. This alias is used to tell other servers how to contact this server, if this server is behind a NAT or a Proxy Server. This alias is not the same as the Universal Messaging assigned interface alias.
Number	Required. Set port number. The port number must be unique and cannot be used again in a Universal Messaging server instance.
Backlog	Provide the size of the Internet Protocol (IP) socket queue.
Bind address	The IP address to which to bind this port, if your machine has multiple IP addresses and you want the port to use this specific address. You cannot change this attribute after you create the port.
Autostart interface	Automatically start this port when starting the Universal Messaging server instance.
Advertise interface	Allow the Universal Messaging server instance to send information about this port to the client.
Allow for inter-realm	Allow port communication between two or more Universal Messaging server instances. For example, allow communication between clusters, or joins.
Allow client connections	Allow clients to connect to this port.
Enable NIO	Enable NIO (Network Input/Output) on this port.
Enable HTTP 1.1	Enable the usage of HTTP 1.1 protocol on this port.
Auth time	Provide the time in milliseconds (ms) that the Universal Messaging server instance waits for the client to complete the authentication process. Default is 10000 milliseconds.
Accept threads	Provide the number of threads processing the accepted sockets.
Select threads	Provide the number of threads allocated for selection.
Send buffer size	Provide the size of the socket send buffer.

Field	Description
Receive buffer size	Provide the size of the socket receive buffer.

For more information about the basic port properties, see "Interface Configuration" on page 282.

JavaScript Interface Properties

Set these properties to configure communication with web clients using JavaScript.

For more information and description about the JavaScript interface properties, see "JavaScript Interface Panel" on page 286.

Field	Description
Enable JavaScript	Allow JavaScript client connections using this port (interface).
Enable WebSockets	Toggle the ability for clients to communicate with the server using the HTML WebSocket Protocol on this interface.
Enable GZIP for long poll	Enable GZIP compression on HTTP long poll connections.
CORS allow credentials	Allow Cross-Origin Resource Sharing (CORS) credentials.
CORS allowed origins	A comma separated list of the host names (and IP addresses, if they appear in URLs) of the servers that host your JavaScript application's HTML files. **Important:** If this property is not set correctly, many communication drivers available to JavaScript clients may fail.
Long poll active delay	The time between clients sending long poll requests to the server in milliseconds. Reducing this may reduce latency but will increase both client and server memory, CPU, and network usage.
GZIP minimum threshold	Set the minimum message size is bytes required for the server to begin compressing data sent to long poll clients.
Long poll idle delay	The time between clients sending long poll when the client is in idle mode.

Custom Headers

Custom headers are paired with Header Key/Value pairs which are sent in the HTTP packets to the client.

4. Optionally, click **Test** to validate the configuration.

5. Click **Save**.

Configuring an NHPS Port

To configure an NHPS port (interface) for a Universal Messaging server instance

1. In Command Central, navigate to **Environments > Instances > All > Universal-Messaging-*instanceName* > Configuration > Ports**.

2. Click and select **NHPS**.

3. Specify values for the following fields:

Connection Basics

Field	Description
Enabled	Enable or disable this NHPS port.
Adapter alias	Provide an alias for the port. Each Universal Messaging server instance can have an associated alias in the form of *host*:*port*. This alias is used to tell other servers how to contact this server, if this server is behind a NAT or a Proxy Server. This alias is not the same as the Universal Messaging assigned interface alias.
Number	Required. Set port number. The port number must be unique and cannot be used again in a Universal Messaging server instance.
Backlog	Set the size of the Internet Protocol (IP) socket queue.
Bind address	The IP address to which to bind this port, if your machine has multiple IP addresses and you want the port to use this specific address. You cannot change this attribute after you create the port.
Autostart interface	Automatically start this port when starting the Universal Messaging server instance.
Advertise interface	Allow the Universal Messaging server instance to send information about this port to the client.
Allow for inter-realm	Allow port communication between two or more Universal Messaging server instances. For example, allow communication between clusters, or joins.
Allow client connections	Allow clients to connect to this port.
Enable NIO	Enable NIO (Network Input/Output) on this port.

Field	Description
Enable HTTP 1.1	Enable the usage of HTTP 1.1 protocol on this port.
Auth time	Provide the time in milliseconds (ms) that the Universal Messaging server instance waits for the client to complete the authentication process. Default is 10000 milliseconds.
Accept threads	Provide the number of threads processing the accepted sockets.
Select threads	Provide the number of threads allocated for selection.
Send buffer size	Provide the size of the socket send buffer.
Receive buffer size	Provide the size of the socket receive buffer.

For more information about the basic port properties, see "Interface Configuration" on page 282.

JavaScript Interface Properties

Set these properties to configure communication with web clients using JavaScript.

For more information and description about the JavaScript interface properties, see "JavaScript Interface Panel" on page 286.

Field	Description
Enable JavaScript	Allow JavaScript client connections using this port (interface).
Enable WebSockets	Toggle the ability for clients to communicate with the server using the HTML WebSocket Protocol on this interface.
Enable GZIP for long poll	Enable GZIP compression on HTTP long poll connections.
CORS allow credentials	Allow Cross-Origin Resource Sharing (CORS) credentials.
CORS allowed origins	A comma-separated list of the host names (and IP addresses, if they appear in URLs) of the servers that host your JavaScript application's HTML files. **Important:** If this property is not set correctly, many communication drivers available to JavaScript clients may fail.
Long poll active delay	The time between clients sending long poll requests to the server in milliseconds. Reducing this may reduce latency but will increase both client and server memory, CPU, and network usage.
GZIP minimum threshold	Set the minimum message size is bytes required for the server to begin compressing data sent to long poll clients.

Field	Description
Long poll idle delay	The time between clients sending long poll when the client is in idle mode.

Custom Headers

Custom headers are paired with Header Key/Value pairs that are sent in the HTTP packets to the client.

Security Configuration

Field	Description
Client authentication	Whether or not Universal Messaging requires client certificates for all requests. Select: ■ **None** if Universal Messaging does not require client certificates for all requests. ■ **REQUIRE_CERTIFICATE** if you want Universal Messaging to require client certificates for all requests.
Keystore type	File type of the keystore file. Universal Messaging supports only the JKS file type.
Keystore server location	Location of the keystore file.
Keystore password	Password required to access the SSL certificate in the keystore file.
Keystore key password	Password required to access a specific private key in the keystore file.
Truststore type	File type of the truststore file. Universal Messaging supports only the JKS file type.
Truststore server location	Location of the truststore file.
Truststore password	Password required to access the SSL certificate in the truststore file.

4. Optionally, click **Test** to validate the configuration.

5. Click **Save**.

Configuring an NSPS Port

To configure an NSPS port (interface) for a Universal Messaging server instance

1. In Command Central, navigate to **Environments > Instances > All > Universal-Messaging-*instanceName* > Configuration > Ports**.

2. Click and select **NSPS**.

3. Specify values for the following fields:

Connection Basics

Field	Description
Enabled	Enable or disable this NSPS port.
Adapter alias	Provide an alias for the port. Each Universal Messaging server instance can have an associated alias in the form of *host*:*port*. This alias is used to tell other servers how to contact this server if this server is behind a NAT or a Proxy Server. This alias is not the same as the Universal Messaging assigned interface alias.
Number	Required. Set port number. The port number must be unique and cannot be used again in a Universal Messaging server instance.
Backlog	Set the size of the Internet Protocol (IP) socket queue.
Bind address	The IP address to which to bind this port, if your machine has multiple IP addresses and you want the port to use this specific address. You cannot change this attribute after you create the port.
Autostart interface	Automatically start this port when starting the Universal Messaging server instance.
Advertise interface	Allow the Universal Messaging server instance to send information about this port to the client.
Allow for inter-realm	Allow port communication between two or more Universal Messaging server instances. For example, allow communication between clusters, or joins.
Allow client connections	Allow clients to connect to this port.
Enable NIO	Enable NIO (Network Input/Output) on this port.
Enable policy server	Allow the ports to respond to policy requests. You can run a policy file server on a socket interface that will automatically handle these requests. Once this is set up, you will also need to set up a client access policy in the clientaccesspolicy.xml

Field	Description
	file in the `/install/server/name/plugins/htdocs` directory of the server.
Auth time	Provide the time in milliseconds (ms) that the Universal Messaging server instance waits for the client to complete the authentication process. Default is 10000 milliseconds.
Accept threads	Provide the number of threads processing the accepted sockets.
Select threads	Provide the number of threads allocated for selection.
Send buffer size	Provide the size of the socket send buffer.
Receive buffer size	Provide the size of the socket receive buffer.

For more information about the basic port properties, see "Interface Configuration" on page 282.

Security Configuration

Field	Description
Client authentication	Whether or not Universal Messaging requires client certificates for all requests. Select: ■ **None** if Universal Messaging does not require client certificates for all requests. ■ **REQUIRE_CERTIFICATE** if you want Universal Messaging to require client certificates for all requests.
Keystore type	File type of the keystore file. Universal Messaging supports only the JKS file type.
Keystore server location	Location of the keystore file.
Keystore password	Password required to access the SSL certificate in the keystore file.
Keystore key password	Password required to access a specific private key in the keystore file.
Truststore type	File type of the truststore file. Universal Messaging supports only the JKS file type.
Truststore server location	Location of the truststore file.
Truststore password	Password required to access the SSL certificate in the truststore file.

4. Optionally, click **Test** to validate the configuration.

5. Click **Save**.

Configuring an SHM Port

To configure an SHM port (interface) for a Universal Messaging server instance

1. In Command Central, navigate to **Environments > Instances > All > Universal-Messaging-*instanceName* > Configuration > Ports**.

2. Click and select **SHM**.

3. Specify values for the following fields:

Field	Description
Enabled	Enable or disable the SHM port.
Path	The directory in which the files required for SHM (shared memory) communication are created. The default value is `/dev/shm`. **Note:** When choosing a path, ensure that the local ID of the server can access this directory.
Buffer size	The size in bytes of the allocated memory that a connection uses. The default value is `1024000`. A file of the same size is also created for mapping.
Timeout	The idle time in milliseconds before a connection is closed. The default value is `20000`.

4. Optionally, click **Test** to validate the configuration.

5. Click **Save**.

Memory Configuration

You can view and update the initial memory size and maximum memory size of a Universal Messaging server instance.

Realm Access Control Lists (ACLs)

You can configure the permissions for a Universal Messaging server instance by editing one or more of the following parameters:

Configure...	Specify...
Subject	User name in the format *user@host*, or name of an existing group in the format *groupname*.
Manage ACL	Allow ACL management.
Full	Grant full privileges to the user or group.
Access	Allow access to the realm.
Configure	Allow realm configuration.
Channels	Allow channel configuration and administration.
Realm	Allow realm configuration and management.
Admin API	Allow use of Universal Messaging Admin API.
Manage data groups	Allow management of data groups.
Own data groups	Allow ownership of data groups.
Publish to data groups	Allow publishing to global data groups.

Group Management

You can add, edit, or delete a new security group using the Command Central web user interface. You can define subjects for a specific group, the subjects can be an existing group or an individual user in the format *user@host*.

For more information about Universal Messaging security groups, see "Security Groups" on page 210.

General Properties

You can configure a Universal Messaging server instance by editing the configuration parameters. The large number of Universal Messaging configuration parameters are organized into groups. The parameters in each group are organized into two categories: Basic and Advanced. The properties in the Basic category are commonly used. The properties in the Advanced category is less frequently used, and are intended for special cases or expert users.

Note:
You can view and configure properties only when the Universal Messaging server instance is Online.

Configure this group...	To modify...
Audit Settings	Parameters to configure what gets logged in the audit file.

Configure this group...	To modify...
Client Timeout Values	Parameters to configure client-server communication timeout settings.
Cluster Config	Parameters to configure clustering.
Comet Config	Parameters to configure Comet communication protocol connection.
Connection Config	Parameters to configure client-server connection.
Data Stream Config	Parameters to configure data streams.
Environment Config	System environment configuration parameters. **Note:** You cannot modify the environment configuration parameters, the parameters are read-only.
Event Storage	Parameters to configure how events are stored, and retrieved from the server.
Fanout Values	Parameters to configure delivery of events to the clients.
Global Values	Parameters to configure various global Universal Messaging server instance properties. For example, `AllowRealmAdminFullAccess`.
Inter-Realm Comms Config	Parameters to configure communication across Universal Messaging server instances.
JVM Management	Parameters to configure the JVM used by the Universal Messaging server.
Join Config	Parameters to modify join properties.
Logging Config	Parameters to modify logging configuration.
Metric Config	Parameters to enable or disable system metrics such as memory usage.
MQTT Config	Parameters to configure MQTT.
Plugin Config	Parameters for Universal Messaging server plugin configuration.
Protobuf Config	Parameters to configure Google protocol buffers.
Protocol AMQP Config	Parameters to configure AMQP connections.
Protocol MQTT Config	Parameters to configure MQTT connections.

Configure this group...	To modify...
RecoveryDaemon	Parameters to configure clients that are in recovery and replaying large number of events.
Server Protection	Parameters to configure server protection such as flow control of producer connections.
Thread Pool Config	Parameters for server thread pools.
TransactionManager	Parameters for Universal Messaging transaction engine.

For information about configuration group parameters and their values, see "Realm Configuration" on page 49.

JNDI Management

You can configure connection factories and destinations in a JNDI namespace using the Command Central web user interface or the command line. You can perform the following operations on JNDI entries:

- Create
- Get
- Update
- Delete

JNDI Connection Factories

You can perform create, get, update, and delete operations on the following connection factory types:

- Connection Factory
- Topic Connection Factory
- Queue Connection Factory
- XA Connection Factory

The table describes the connection factory configuration parameters:

Configure...	Specify...
Name	Required and unique. Name of the new connection factory. Once created, you cannot edit the JNDI connection factory name. For example, connectionfactory1.
Type	Required. Type of connection factory selected when creating the connection factory. For example, XA Connection Factory.

Configure...	Specify...
	Note: This field cannot be edited.
Connection URL	Required. Universal Messaging server URL for binding the connection factory. For example, nsp://umhostname:9000. A cluster of server instances is specified using a comma-separated list of connection URLs, for example, *nsp://localhost:9000,nsp://localhost:9010*. You can use a *horizontal scalability* connection factory to specify several connection URLs, where each connection URL can point to a standalone realm or a cluster. *Horizontal scalability* connection factories allow clients to publish messages to a set of servers or consume messages from a set of servers in a round-robin manner: ■ For round-robin publishing, one message or transaction gets published to the first realm node or cluster, the next message to the next realm node or cluster, and so on. ■ For round-robin consuming, there is no guarantee about the order in which the events are delivered. For a *horizontal scalability* connection factory, you specify several connection URLs, using the *horizontal scalability URL syntax*. See the section *URL Syntax for Horizontal Scalability* in the *Concepts* guide for details of this syntax. Example: (UM1,UM2)(UM3,UM4) - Indicates 2 clusters, one consisting of UM1 and UM2 and the other consisting of UM3 and UM4, so only 2 connections will be constructed here. **Note:** Round-robin delivery is not supported for XA Connection Factory.
Durable type	Select the durable type. The durable type **Named** is selected by default. **Note:** You can configure the durable type property only for Connection Factory and Topic Connection Factory.

JNDI Destinations

You can perform create, get, and delete operations on the following destination types:

Note:
Update operation for JNDI destinations is not supported.

- Topics
- Queues

The table describes the destination configuration parameters:

Configure...	Specify...
Lookup Name	Required and unique. Name of the JNDI destination. Once created, you cannot edit the JNDI destination name. Name can include upper and lower case alphabetic characters, digits (0-9), double colon (::), slash (/), and periods (.), for example, destination1. Use the double colon (::) for specifying nested name space, for example, destination1::destination2.
Destination Type	Required. Type of destination. **Note:** This field cannot be edited.
Store Name	Required and unique. Name of the JMS channel or queue. Once created, you cannot edit the store name. Store name can include upper and lower case alphabetic characters, digits (0-9), double colon (::), and slash (/).
Auto-Create JMS Channel	Select to enable auto-creation of JMS channel.

Note:

- Creating a connection factory and destination with the same name is not allowed for a Universal Messaging server instance.
- Deleting a JNDI destination will not delete the channel or queue that exists in the Universal Messaging server instance.

Channel Configuration

You can view, create, update, and delete a channel using the Command Central web or command-line interface.

You can configure the following properties:

Channel Properties

Property...	Description..
Name	Required. Name of the channel to be created.

Property...	Description..
	Note: Once a channel is created, you cannot edit the channel name.
Type	Type of channel. Universal Messaging channel types: ■ Transient ■ Simple ■ Reliable ■ Persistent ■ Mixed (default) ■ Off-heap ■ Paged
TTL (ms)	Specifies how long (in milliseconds) each event is retained on the channel after being published. For example, if you specify a TTL of 10000, the events on the channel will be automatically removed by the server after 10000 milliseconds. Specify 0 for events to remain on the channel indefinitely.
Capacity	Event capacity of the channel. Specifies the maximum number of events that can be on a channel, once published. Specify 0 to store unlimited events. The maximum channel capacity is 2147483646.
Dead event store	Channel or queue to be used to store events that are purged before being consumed.
Engine	Type of engine to be used for the channel. By default, Universal Messaging retains all events on the channel for a specified TTL, modify the retention behavior by selecting **JMS** engine or **Merge** engine.
Cluster-wide	Selected automatically if the Universal Messaging server instance is part of a cluster.
Protobuf descriptor	Absolute path to the Protocol Buffer (protobuf) descriptor file that is stored on the machine where Software AG Platform Manager is installed.

Storage Properties

Property...	Description..
Auto-maintenance	Select to retain events till they reach their TTL. Cancel the selection to purge events from the channel storage file.
Honor capacity	Select to prevent publishing of data when the channel is full. Cancel the selection to purge the oldest published event.
Enable caching	Select for the events to be stored in the cache memory and reused. Cancel the selection to read and stored in the file store.
Cache on reload	Select to enable caching during reload.
Enable read buffering	Select to enable read buffering for the store on the Universal Messaging server.
Enable multicast	Select to enable multicast client to receive events over multitask connections.
Read buffer size	Read buffer size in bytes.
Sync each write	Select to sync each write to the file system.
Sync batch size	Configurable only when **Sync each write** is selected. Number of events that is to be synced with the file system at once.
Sync batch time	Configurable only when **Sync each write** is selected. Time in milliseconds (ms) between syncs with the file system.
Fanout archive target	Target number of events that are written to an archive after fanout.
Priority	Priority range. 0 (lowest) to 9 (highest).
Events per spindle	Maximum number of events allowed per file.
Stamp dictionary	Select to stamp events on the channel by the server.

Channel Keys

Column...	Description...
Key name	Name of the channel publish key.
Depth	Depth of the channel publish key. Depth is the maximum number of events that can exist on a channel for a specific key name.

Channel ACL

Column...	Description...
Subject	User name in the format `user@host` or the name of an existing group.
Manage ACL	Select to allow the user or group to manage ACLs.
Full	Select to grant full privileges to the user or group.
Purge	Select to allow the user or group to delete events.
Subscribe	Select to allow the user or group to subscribe to events.
Publish	Select to allow the user or group to publish events.
Named (durable)	Select to allow the user or group to connect to this channel using durable subscriptions.

Joins

You can add and delete joins, but cannot edit channel join properties.

Column...	Description...
Type	Type of join. Outgoing: channel is source. Incoming: channel is destination.
Destination name	Destination channel or queue.
Target instance name	Name of the target instance. Required only if the channel or queue is in a remote Universal Messaging server instance.
Target instance URL	URL of the target Universal Messaging server instance that has the destination channel or queue for the join.
Filter	Type the filtering criteria, a string or a regular expression. Events will be routed to the destination channel based on the filtering criteria.
Hop count	Type the maximum number of subsequent joins.
Event ID	Last retrieved event ID.
Allow purge	Select to allow purging of events on the channel.
Archival	Select to enable archival join. Archival join is a join between a channel and a queue where events will not be checked for duplication.

Queue Configuration

You can view, create, update, and delete a queue using the Command Central web or command-line interface.

You can configure the following properties:

Queue Properties

Property...	Description..
Name	Required. Name of the queue to be created. You cannot edit the queue name.
Type	Type of queue. Universal Messaging queue types: ■ Transient ■ Simple ■ Reliable ■ Persistent ■ Mixed (default) ■ Off-heap ■ Paged
TTL (ms)	Specifies how long (in milliseconds) each event is retained in the queue after being published. For example, if you specify a TTL of 10000, the events on the queue will be automatically removed by the server after 10000 milliseconds. Specify 0 for events to remain on the queue indefinitely.
Capacity	Event capacity of the queue. Specifies the maximum number of events that can be on a queue once published. Specify 0 to store unlimited events. The maximum queue capacity is 2147483646.
Dead event store	Channel or queue to be used to store events that are purged before being consumed.
Cluster-wide	Selected automatically if the Universal Messaging server instance is part of a cluster.
Protobuf descriptor	Absolute path to the Protocol Buffer descriptor file that is stored on the machine where Software AG Platform Manager is installed.

Storage Properties

Property...	Description..
Auto-maintenance	Select to retain events till they reach their TTL. Cancel the selection to purge events from the queue storage file.
Honor capacity	Select to prevent publishing of data when the queue is full. Cancel the selection to purge the oldest published event.
Enable caching	Select for the events to be stored in the cache memory and reused. Cancel the selection to read and stored in the file store.
Cache on reload	Select to enable caching during reload.
Enable read buffering	Select to enable read buffering for the store on the Universal Messaging server.
Enable multicast	Select to enable multicast client to receive events over multitask connections.
Read buffer size	Read buffer size in bytes.
Sync each write	Select to sync each write to the file system.
Sync batch size	Configurable only when **Sync each write** is selected. Number of events that is to be synced with the file system at once.
Sync batch time	Configurable only when **Sync each write** is selected. Time in milliseconds (ms) between syncs with the file system.
Fanout archive target	Target number of events that are written to an archive after fanout.
Priority	Priority range. 0 (lowest) to 9 (highest).
Events per spindle	Maximum number of events allowed per file.
Stamp dictionary	Select to stamp events on the channel by the server.

Queue ACL

Column...	Description...
Name	User name in the format `user@host` or the name of an existing group.
Manage ACL	Select to allow the user or group to manage ACLs.
Full	Select to grant full privileges to the user or group.
Purge	Select to allow the user or group to purge events.
Peek	Select to allow the user or group to peek events.
Push	Select to allow the user or group to push events.

Column...	Description...
Pop	Select to allow the user or group to pop events.

Joins

You can view the following join properties:

Column...	Description...
Type	Type of join.
Destination name	Destination channel.
Target instance name	Name of the target instance. Required only if the target channel is in a remote Universal Messaging server instance.
Target instance URL	URL of the target Universal Messaging server instance that has the destination channel for the join.
Remote node	Destination channel.
Hop count	Maximum number of subsequent joins.
Event ID	Last retrieved event ID.
Allow purge	Allow purging of events in the queue.
Archival	Archival join is a join between a channel and a queue where events will not be checked for duplication.

Zones

Zones are a logical grouping of one or more Universal Messaging server instances (realms) that maintain active connections to each other. Each Universal Messaging server instance can be a member of zero or one zone, but a server instance cannot be a member of more than one zone. For more information about zones, see the Zones section in the Universal Messaging Concepts guide.

You can create, edit, or delete a zone using the Command Central web user interface. You can create **Zone with Realms** consisting one or more reams, or create **Zone with Clusters** consisting one or more Universal Messaging clusters. You can export the zone configuration using the **Export** option.

Important:
If the server instances that you want to add to the zone have only nsps or nhps interfaces, ensure that the keystore and truststore of each server instance contain the certificates required for communicating with the rest of the server instances. To do so for a server instance:

1. Create a keystore and a truststore.
2. Import the certificate of the server instance into the key store.
3. Import the certificates of all servers that you want to add to the zone into the truststore.

4. Add the paths to the keystore and truststore, as well as the keystore and truststore passwords, to the corresponding JSSE system properties in the Server_Common.conf file of the server instance. The file is located in the *Software AG_directory* \UniversalMessaging\server*instanceName*\bin directory.

The table describes the zone parameters required to create a zone:

Configure...	Specify...
Zone name	Required. Name of the zone, it is used to uniquely identify this zone.
Servers (for Zone with Realms) or Clustered servers (for Zone with Clusters)	Required. Provide server URL and name if you are creating a **Zone with Realms** or provide all or one of the cluster nodes if you are creating a **Zone with Clusters**. ■ Server URL: Universal Messaging server URL. ■ Server name: Name of the Universal Messaging server. ■ Cluster configuration: Automatically populated with the cluster name. **Note:** You can provide any one cluster node and all the running nodes of the cluster will be added to the zone.

Java System Properties

You can add, view, edit, and delete Java system properties from the Command Central web user interface. Restart the Universal Messaging instance for the changes to take effect.

Note:
You cannot delete the default Java system properties that are already present.

To export the Java system properties in XML format, click **Export**. Click **Edit** to add a custom property.

You can edit the following Java system properties:

Property	Description
LICENCE_FILE	Name of the license file.
LICENCE_DIR	Absolute path to the location of the license file.
DATADIR	Absolute path to the location of the data directory. **Note:** Modifying the DATADIR property does not copy the existing data directory to the new location.

You can delete a property value by passing an empty value for the property, this will revert the property to its default value.

JVM Options

You can add, view, edit, and delete the JVM options from the Command Central web user interface. Restart the Universal Messaging instance for the changes to take effect.

The JVM options are stored in the Server_ Common.conf file located in the <*InstallDir*>/UniversalMessaging/server/<*InstanceName*>/bin folder. If you add a custom JVM option, it is added to the Custom_Server_Common.conf file. These options are given to the Java Service Wrapper when it launches the JVM.

For more information about the Java Service Wrapper, see *Software AG Infrastructure Administrator's Guide*.

To export the JVM options in XML format, click **Export**. Click **Edit** to add a custom JVM option.

Note:
You can edit but cannot delete the default JVM options that are already present.

Cluster Management

Before You Create a Universal Messaging Cluster

Before you create or update a Universal Messaging cluster:

- Ensure that the server instances that you want to add to the cluster are running.
- Verify that the permissions on the server machines allow connections to the other servers in the cluster.
- Ensure that the /naming/defaultContext channel exists only on one or none of the nodes that will form the cluster. The Universal Messaging server instance used as a JNDI provider uses the /naming/defaultContext channel to store JMS references and JNDI objects. If channels exists on multiple nodes, you cannot create the cluster.
- Ensure that the keystore and truststore of each server instance in the cluster contain the certificates required for communicating with the rest of the server instances. This is required when the server instances that you want to add to the cluster have only nsps or nhps interfaces. Perform the following steps for a server instance:

  1. Create a keystore and a truststore.
  2. Import the certificate of the server instance into the keystore.
  3. Import the certificates of all servers that you want to add to the cluster into the truststore.
  4. Add the paths to the keystore and truststore, as well as the keystore and truststore passwords, to the corresponding JSSE system properties in the Server_Common.conf file

of the server instance. The file is located in the *Software AG_directory* \UniversalMessaging\server*instanceName*\bin directory.

Cluster Configuration Fields

Field	Specify...
Cluster Name	Unique cluster name.
Server URL	Server instances URL (for example, nsp://127.0.0.1:9002) of each server node. When you save the cluster details, the **Server Name** field is populated with the name of the server corresponding to the specified server URL.
Cluster Site	Name of the site (Optional) to which the server node belongs.
Prime Site	Name of the primary site (Optional), if you have configured sites in the cluster.

Cluster Configuration Tasks Supported

Cluster configuration tasks that you can perform:

- Create a cluster of two or more server instances. The cluster create operation converts the local stores to cluster-wide stores for the selected master node. For the other nodes, the stores must be empty for the cluster create operation to be successful.
- Add one or more server instances to the existing cluster
- Remove one or more server instances from the existing cluster
- Upgrade a cluster
- Create sites and assign server instances to the sites
- Assign a site as the prime site of a cluster
- Remove one or more server instances from a cluster site
- Remove sites from a cluster
- Delete a cluster

  Note:
  The cluster delete operation does not delete any stores on the nodes.

- Migrate a cluster

Cluster Migration

You can now automatically migrate Universal Messaging clusters using Command Central composite templates. Use Command Central composite templates to migrate all the Universal

Messaging server instances that are part of the cluster, and add the following property to resolve the cluster node connections successfully after migration:

- Add the following property to the `um-cluster` section of the composite template YAML file present in the following directory: `<InstallDir>/profiles/CCE/data/templates/composite` :

```
<ClusterSettings>
         ...
           <ExtendedProperties>
              <Property name="migrationType">SAME_HOST or CROSS_HOST</Property>

           </ExtendedProperties>
      </ClusterSettings>
```

Note:
A Universal Messaging server instance mapping file called remote_realms_bootstrap.conf containing information about the new Universal Messaging hosts is automatically generated in each of the Universal Messaging server instances. The mapping file is used to identify the new hosts, and to form the new cluster.

Universal Messaging Administration Types

Command Central lets you administer and monitor the following on a Universal Messaging server instance:

- Durable Subscribers
- Channels
- Queues

Durable Subscribers

You can search, monitor, and delete durable subscribers for a Universal Messaging server instance. You can browse and purge events on a specific durable subscriber.

Types of durable subscribers:

- Shared
- Shared - Queued
- Priority
- Serial
- Durable

Note:
You can purge events only for shared durable subscribers.

The table displays the following durable subscriber attributes:

Attribute	Description
Name	Name of the durable subscriber.
Channel	Name of the channel to which the durable subscriber belongs.
Durable type	Type of the durable subscriber (Shared, Shared - Queued, Priority, Serial, or Durable).
Last event ID	Event ID of the last successfully consumed event.
Outstanding events	The number of events outstanding for a particular durable subscriber. **Note:** The outstanding event count displayed for a non-shared durable is only an estimate.
Last read time	The last date and time when the durable subscriber read, committed, or rolled back an event.

Select a durable subscriber to view:

- Durable details
- Browse events
- Bulk purge

The **Durable details** page contains the following information about a durable subscriber:

Details

Attribute	Description
Name	Name of the durable subscriber.
Channel	Name of the channel to which the durable subscriber belongs.
Durable type	Type of the durable subscriber (Shared, Shared - Queued, Priority, Serial, or Durable).
Cluster-wide	Whether the durable subscriber is on a channel that is part of a Universal Messaging cluster.
Persistent	Whether the durable subscriber is persistent. Persistent durable subscribers retrieve the last event ID consumed before the Universal Messaging server instance was restarted.
Selector	Events are filtered based on the defined selector.

Status

Attribute	Description
Last event ID	Event ID of the last successfully consumed event.
Store size (KB)	The size of the store available for the Shared - Queued durable.
Total events	The number of outstanding events for the durable subscriber.
Pending events	The number of outstanding events waiting for a commit or a rollback.
Last read time	The last date and time when the durable subscriber read, committed, or rolled back an event.
Last write time	The last time the durable was written to. Typically, this is the last time an event was added to the durable subscriber.

Connections

Attribute	Description
ID	Connection ID.
Mode	Client subscription mode (subscription based or getNext).
Max pending	Window size specified by the client.
Acknowledged	Total acknowledged events.
Rolled back	Total rolled back events.
Pending	Event queues waiting to be acknowledged or rolled back.
Last read time	Last time the session acknowledged, rolled back, or read an event from the durable subscriber.

The **Browse events** page displays the event list and lets you to browse events for a durable subscriber. Events are displayed in the order of old to new. A maximum of 1000 events are displayed in the table with a maximum combined size of 10 MB. For example, if two events of size 10 MB and 100 MB are present for the durable subscriber, only the event of size 10 MB is displayed, and no other events are displayed. Click **Browse events** to refresh the events displayed in the table.

Attribute	Description
Event ID	Unique ID to identify the event.
TTL (ms)	Specifies how long (in milliseconds) each event is retained.
Tag	Shows the tag information of the event if an event tag exists.
Event data	Content of the event.

Attribute	Description
	Note: If the payload is Protocol Buffers (Protobuf), the event data will contain the message "Protobuf payload" and not the decoded payload.
Event size (bytes)	Size of the event in bytes.
	Note: Event size is the total size of the event that is the sum of event data and event properties.
Event properties	Event properties represented as key-value pairs.
Persistent	Shows whether the event is persistent or not.

You can select an event ID to view the event data and event properties.

The **Bulk purge** option allows you to purge events in bulk for a durable subscriber. You can purge events by providing an event range, event filter, or purge all the events.

Channels

You can search and monitor the following channel attributes:

Attribute	Description
Name	Name of the channel.
Event ID	Event ID of the last event that was consumed from the channel. Event ID is -1 if the channel is empty.
Events	Number of events in the channel that are yet to be consumed.
Current Connections	Number of current connections to the channel.
% Free	Percentage of free storage available in the channel.

Select a channel to view detailed information about the status of the channel and the durable subscribers (named objects) subscribed to the channel. You can also delete a durable subscriber subscribed to the channel.

Queues

You can search and monitor the following queue attributes:

Attribute	Description
Name	Name of the queue.
Event ID	Event ID of the last event that was popped from the queue. Event ID is -1 if the queue is empty.
Events	Number of events in the queue that are yet to be consumed.
Current Connections	Number of current connections to the queue.
% Free	Percentage of free storage available in the queue.

Select a queue to view detailed information about the status of the queue.

Snooping on Channels

Important:
The event snooping functionality works with Command Central 10.4 and higher.

When you select a channel on the Universal-Messaging-*instance_name* > Administration > Channels page in Command Central, you can perform the following operations:

- Start snooping on events on the channel.
- Stop snooping on events on the channel.
- View details about a snooped event.
- Purge one or more events from the channel.

Considerations When Snooping on Channels

Consider the following information when you want to snoop on events on a channel:

- Snooping on events on a channel is performed per user. Only the user who started snooping can stop it. However, Command Central allows more than one person to log in as Administrator at the same time, which might result in simultaneous attempts to perform various snooping operations on the same channel.
- When snooping is inactive for five minutes, for example, because the user logged off or navigated away from the channel details page, the snoop stops automatically.
- If the Universal Messaging server becomes unavailable after the snoop on the channel started, Command Central stops snooping on all channels on that Universal Messaging server for all Command Central users. If you try to start snooping while the Universal Messaging server is still unavailable, Command Central returns an error.

Starting the Channel Snoop

The event snooping functionality works with Command Central 10.4 and higher.

To start snooping on events on a channel

1. In Command Central, go to **Environments > Instances > All > Universal-Messaging-*instance_name* > Administration > Channels**.

2. Select the channel on which you want to start snooping on events, and then click the **Snoop** tab.

3. Do one of the following:

   - To snoop on all events published on the channel, click **Start**.
   - To snoop on a range of events, in the **From Event ID** field specify the ID of the first event in the range, and in the **To Event ID** field, specify the ID of the last event in the range. Click **Start**. You can also specify additional filtering criteria based on the properties of the event.

     Note:
     If you do not specify a value in the **From Event ID** field, the range of events starts with the first event on the channel and ends with the event specified in the **To Event ID** field. If you do not specify a value in the **To Event ID** field, the range of events starts with the event specified in the **From Event ID** field and ends with the last event on the channel.

Command Central populates the snooped events table with the events published on the channel.

Stopping the Channel Snoop

To stop snooping on events on a channel

1. In Command Central, go to **Environments > Instances > All > Universal-Messaging-*instance_name* > Administration > Channels**.

2. Select the channel on which you want to stop snooping.

3. Click the **Snoop** tab, and then click **Stop**.

Viewing Channel Event Details

After you start snooping on a channel in the Command Central web user interface, the snooped events table on the channel details page displays information about each event including the event ID, tag, time to live (TTL), and data, as well as whether the event is persistent.

When you select an event in the table, you can view additional details about the event including the type of event (persistent, transient, or protobuf), a hexadecimal view of the event data, an ASCII representation of the event data, the header and properties of the event, and the Protobuf descriptor for Protobuf events.

If the event is a Protobuf event and its Protobuf descriptor matches the name of a Protobuf file descriptor that has already been uploaded on the channel, the ASCII representation of the event is the decoded Protobuf content in JSON format.

Purging Snooped Events from a Channel

After you start snooping on a channel, you can purge snooped events from the channel. You can purge a single event, a range of events, or all events.

For information about how to start snooping, see "Starting the Channel Snoop" on page 391.

To purge events

1. In Command Central, go to **Environments > Instances > All > Universal-Messaging-*instance_name* > Administration > Channels**.
2. Select the channel on which you want to purge snooped events, and then click the **Snoop** tab.
3. (Optional) To purge a single event directly from the snooped events table, click .
4. Click **Purge Events** and on the Purge Events page, do one of the following:
   - To purge all events, click **Purge**.
   - To purge a range of events, in the **From Event ID** field specify the ID of the first event in the range, and in the **To Event ID** field, specify the ID of the last event in the range. Then click **Purge**.

     Note:
     If you do not specify a value in the **From Event ID** field, the range of events to purge starts with the first event on the channel and ends with the event specified in the **To Event ID** field. If you do not specify a value in the **To Event ID** field, the range of events to purge starts with the event specified in the **From Event ID** field and ends with the last event on the channel.

Snooping on Queues

Important:
The event snooping functionality works with Command Central 10.4 and higher.

When you select a queue on the Universal-Messaging-*instance_name* > Administration > Queues page in Command Central, you can perform the following operations:

- Start snooping on events on the queue.
- Stop snooping on events on the queue.
- View details about a snooped event.

- Purge all events from the queue.

Considerations When Snooping on Queues

Consider the following information when you want to snoop on events on a queue:

- Snooping on events on a queue is performed per user. Only the user who started snooping can stop it. However, Command Central allows more than one person to log in as Administrator at the same time, which might result in simultaneous attempts to perform various snooping operations on the same queue.
- When snooping is inactive for five minutes, for example, because the user logged off or navigated away from the queue details page, the snoop stops automatically.
- If the Universal Messaging server becomes unavailable after the snoop on the queue started, Command Central stops snooping on all queues on that Universal Messaging server for all Command Central users. If you try to start snooping while the Universal Messaging server is still unavailable, Command Central returns an error.

Starting the Queue Snoop

The event snooping functionality works with Command Central 10.4 and higher.

To start snooping on events on a queue

1. In Command Central, go to **Environments > Instances > All > Universal-Messaging-*instance_name* > Administration > Queues**.
2. Select the queue on which you want to start snooping on events, and then click the **Snoop** tab.
3. Do one of the following:
   - To snoop on all events published on the queue, click **Start**.
   - To snoop on a range of events, in the **From Event ID** field specify the ID of the first event in the range, and in the **To Event ID** field, specify the ID of the last event in the range. Click **Start**. You can also specify additional filtering criteria based on the properties of the event.

     Note:
     If you do not specify a value in the **From Event ID** field, the range of events starts with the first event on the queue and ends with the event specified in the **To Event ID** field. If you do not specify a value in the **To Event ID** field, the range of events starts with the event specified in the **From Event ID** field and ends with the last event on the queue.

Command Central populates the snooped events table with the events published on the queue.

Stopping the Queue Snoop

To stop snooping on events on a queue

1. In Command Central, go to **Environments > Instances > All > Universal-Messaging-*instance_name* > Administration > Queues**.

2. Select the queue on which you want to stop snooping.

3. Click the **Snoop** tab, and then click **Stop**.

Viewing Queue Event Details

After you start snooping on a queue in the Command Central web user interface, the snooped events table on the queue details page displays information about each event including the event ID, tag, time to live (TTL), and data, as well as whether the event is persistent.

When you select an event in the table, you can view additional details about the event including the type of event (persistent, transient, or protobuf), a hexadecimal view of the event data, an ASCII representation of the event data, the header and properties of the event, and the Protobuf descriptor for Protobuf events.

If the event is a Protobuf event and its Protobuf descriptor matches the name of a Protobuf file descriptor that has already been uploaded on the queue, the ASCII representation of the event is the decoded Protobuf content in JSON format.

Purging Snooped Events from a Queue

After you start snooping on a queue, you can purge all snooped events from the queue.

For information about how to start snooping, see "Starting the Queue Snoop" on page 394.

To purge events

1. In Command Central, go to **Environments > Instances > All > Universal-Messaging-*instance_name* > Administration > Queues**.

2. Select the queue on which you want to purge the snooped events, and then click **Purge All Events**.

Publishing Events

Publishing Events on a Channel or Queue

Important:
The event publishing functionality works with Command Central 10.4 and higher.

Use the following procedure to create a new event and publish it on a Universal Messaging channel or queue, using the Command Central web user interface.

To publish an event on a channel or queue

1. In Command Central, go to **Environments > Instances > All > Universal-Messaging-*instance_name* > Administration**.

2. Do one of the following:

   - To publish an event on a channel, from the drop-down menu, select **Channels** and then select the channel on which you want to publish the event.
   - To publish an event on a queue, from the drop-down menu, select **Queues** and then select the queue on which you want to publish the event.

3. Click the **Publish** tab and specify values for the following fields as required:

Property	Value
Event data	Required. The content of the event.
Tag	Optional. The tag of the event.
TTL (ms)	Optional. The time-to-live (TTL) of the event in milliseconds. Defines how long the event remains available on the channel or queue. If you specify a TTL of `0`, the event remains on the channel or queue indefinitely.
Persistent	Optional. Whether the event is persistent.
Transient	Optional. Whether the event is transient.
Properties	Optional. Click to add event properties. For each property, specify the name, type, and value.
Number of publishes	Optional. The number of times to publish the event. If you do not select the option, it defaults to `1`.

Property	Value
Publish as a Protobuf event	Optional. Whether to send the event as a Protobuf event. For more information about how to publish Protobuf events, see "Publishing Protobuf Events" on page 397.
Protobuf descriptor	Required only for Protobuf events. The Protobuf file descriptor that defines the message schema to be used for converting the event data into a Protobuf event.

4. Click **Publish**.

When you start snooping on events on the channel or queue, Command Central displays the event in the snooped events table.

For information about how to snoop on channels and queues, see "Snooping on Channels" on page 391 and "Snooping on Queues" on page 393.

Publishing Protobuf Events

Important:
The event publishing functionality works with Command Central 10.4 and higher.

Perform the following actions to publish a Protobuf event, using the Command Central web user interface:

1. Before publishing the Protobuf event, upload on the channel or queue the Protobuf file descriptor that defines the Protobuf schema, as part of a file descriptor set. For information about uploading a Protobuf file descriptor set on a channel or queue, see "Channel Configuration" on page 377 or "Queue Configuration" on page 381.
2. When you create the event on the **Publish** tab for a channel or queue, do the following:
   a. In the **Event data** field, specify a JSON string that represents the Protobuf event.
   b. Select the **Publish as a Protobuf event** option.
   c. In the **Protobuf descriptor** field, specify the name of the Protobuf file descriptor that defines the message schema.

      Important:
      If you do not specify a Protobuf descriptor or if the specified Protobuf descriptor value does not correspond to any Protobuf file descriptor on the channel or queue, the system returns an error that no Protobuf descriptor was found on the channel or queue and does not create a Protobuf event.

   For information about how to publish events, see "Publishing Events on a Channel or Queue" on page 396.

For more information about working with Protobuf events, see the "Google Protocol Buffers" section in the *Universal Messaging Concepts* guide.

Republishing Events on a Channel or Queue

Before republishing a snooped event, you must start snooping on the channel or queue. For information about how to start snooping, see "Starting the Channel Snoop" on page 391 and "Starting the Queue Snoop" on page 394.

To republish events

1. In Command Central, go to **Environments > Instances > All > Universal-Messaging-*instance_name* > Administration**.
2. Do one of the following:
   - To republish an event on a channel, from the drop-down menu, select **Channels** and then select the channel on which you want to republish the event.
   - To republish an event on a queue, from the drop-down menu, select **Queues** and then select the queue on which you want to republish the event.
3. Click the **Snoop** tab and then click the snooped event that you want to republish.
4. Click **Republish** and modify the event fields as required.
5. Optionally, select **Purge Original Event**.
6. Click **Publish**.

Universal Messaging Cloud Transformation

Universal Messaging Integration with LAR

The current on-premise asset deployment model uses a push-model which is supported in Deployer and Command Central Deployer API. On cloud-based deployment, the cloud and container deployments require a pull-model approach for asset deployment rather than push-model.

The pull-model is based on Landscape Asset Repository (LAR) and its change listeners. Universal Messaging needs to integrate with the LAR based update mechanism for deployment of its assets.

Configuration types supported on cloud deployment

The following table lists the configuration types supported on cloud deployments:

Configuration type	Description
JNDI connection factory	On JNDI look up for the connection factory, connection factory URL will be set to the value used while building the JNDI context (`java.naming.provider.url`)
JNDI destinations	For native messaging, Integration Server creates the channel or queue as soon as the publishable document type is created. This would include the protocol buffer definitions. Only JMS destination deployment is supported for cloud-based deployment as native ones are created by Integration Server. All the JNDI destinations are exported with the parameter `autoCreateDest` set to true, so that all the channels or queues are created automatically. The newly created channel or queues will have default Universal Messaging configurations. For more information about JNDI destinations, see "JNDI Asset Configuration Commands" on page 412. The durables will be created automatically after the Integration Server triggers are started for the first time.

Realm ACLs and Groups. The basic authentication is enabled by default and the default ACL setting will have full privileges.

Common Users. The Universal Messaging users are migrated and the users are created (stored in `users.txt` file) with custom passwords.

Note:
If you want to change the instance name, server name or runtime component ID, you have to manually edit the `template.yaml` file. The `template.yaml` file is generated by Designer and contains Universal Messaging configuration.

Universal Messaging Logs

You can view, download, and search all the Universal Messaging logs at one place in the Command Central web user interface. You can access the logs for a Universal Messaging server instance by selecting a Universal Messaging instance name and clicking the **Logs** tab.

Universal Messaging Inventory

When you view installations in an environment, Command Central displays the Universal Messaging server instances listed in the <*InstallDir*>/UniversalMessaging/server directory of an installation. Command Central lists all the folders (except the templates) in the server directory.

Universal Messaging Lifecycle Actions

You can start, stop, and restart a Universal Messaging server instance from the Command Central web user interface. The following legend describes what each one of the lifecycle operations.

- **Start**. Start a server instance that has stopped.
- **Stop**. Stop a running server instance.
- **Restart**. Restart a running server instance.

Universal Messaging KPIs

You can view the following key performance indicators (KPIs) to monitor the performance of the Universal Messaging servers:

KPI	Description
JVM Memory	Indicates the utilization of JVM memory. The marginal, critical, and maximum values for this KPI depend on the maximum memory size of the JVM. - Marginal is 80% of the maximum JVM memory. - Critical is 95% of the maximum JVM memory. - Maximum is 100% of the maximum JVM memory.
Fanout Backlog	Indicates the total number of events currently waiting to be processed by the fanout engine. If the fanout backlog is more than the critical value, there is a possibility that the subscribers receive the published events after some delay. The KPI uses the following marginal, critical, and maximum values: - Marginal is 80% of the maximum value. - Critical is 95% of the maximum value. - Maximum is 100% of the peak value (high-water mark) of fanout backlog. Default is 100.
Queued Tasks	Indicates the total number of tasks in the read, write, and common read/write pools. If the number of read and write tasks queued is more than the critical value, it indicates that the Universal Messaging server is unable to match the speed of the publishers and subscribers. The KPI uses the following marginal, critical, and maximum values: - Marginal is 80% of the maximum value. - Critical is 95% of the maximum value.

KPI	Description
	■ Maximum is 100% of the peak value (high-water mark) of read and write tasks queued. Default is 100.

Universal Messaging Run-time Monitoring Statuses

The Command Central instances page displays the run-time status of a Universal Messaging server instance in the status column. The Universal Messaging can have one of the following run-time status:

- Online : Server instance is online.
- Failed : Server instance failed. For example, the Universal Messaging server instance stopped unexpectedly due an error or system failure.
- Stopped : Server instance is not running.
- Stopping : Server instance is stopping.
- Unresponsive : Server instance is running, but it is unresponsive. When none of the server interfaces are connected to the server.
- Unknown : Server instance status cannot be determined.

When you have set up a Universal Messaging cluster, the run-time status indicates if a server instance is:

- Online Master : Server instance is online and it is the master node in the cluster.
- Online Slave : Server instance is online and it is the slave node in the cluster.
- Error : Server instance is part of a cluster that does not satisfy the requisite quorum.

Universal Messaging and the Command Line Interface

Universal Messaging supports the following Command Central commands:

- `sagcc create configuration data`
- `sagcc delete configuration data`
- `sagcc get configuration data`
- `sagcc update configuration data`

- `sagcc get configuration instances`
- `sagcc list configuration instances`
- `sagcc get configuration types`
- `sagcc list configuration types`
- `sagcc exec configuration validation create`
- `sagcc exec configuration validation delete`
- `sagcc exec configuration validation update`
- `sagcc create instances`
- `sagcc delete instances`
- `sagcc list instances`
- `sagcc list instances supportedproducts`
- `sagcc get inventory components`
- `sagcc exec lifecycle`
- `sagcc get diagnostics logs`
- `sagcc get diagnostic logs export file`
- `sagcc list diagnostics logs`
- `sagcc get monitoring`
- `sagcc exec administration product`

Important:
When a Universal Messaging server instance is running as a service, you cannot perform administrative tasks such as check the status, or start and stop the server instance.

For general information about using the commands, see *Software AG Command Central Help*.

Universal Messaging Instance Management Commands

Creating an Instance on the Command Line

The following table lists the parameters that you use when you create a Universal Messaging realm server, Enterprise Manager, or Template Applications instance using the Command Central instance management commands.

Command	Parameter	Description
`sagcc create instances`	`NUMRealmServer`, `NUMEnterpriseManager`, and `NUMTemplateApplications`	The product ID for the Universal Messaging realm server, Enterprise Manager, or Template Applications instance, respectively.
	`instance.name=`*`name`*	Required. A name for the new Universal Messaging instance.
	`instance.ip=`*`ipAddress`*	Optional. An IP address for the Universal Messaging server interface. If you do not specify a value, the default value is `0.0.0.0`.
	`instance.port=`*`port`*	Optional. A port number for the Universal Messaging server. If you do not specify a value, the default value is `9000`.
	`instance.dataDir`	Optional. Absolute path to the Universal Messaging server data directory. If you do not specify a path, the default directory location is used for creating the data directory. Location of the default directory is: *<InstallDir>*/UniversalMessaging/*server*/*<InstanceName>*
	`license.file`	Optional. Absolute path to the Universal Messaging license file. If you do not specify a path, the default license is used. Location of the default directory is: *<InstallDir>*/UniversalMessaging/*server*/*<InstanceName>*
	`instance.config.profile={wM|TC|CUSTOM}`	Optional. Initial configuration settings for the Universal Messaging server instance. Values are: ■ `wM`: webMethods suite use cases. ■ `TC`: Standalone use cases. ■ `CUSTOM`: Custom profile.
	`instance.config.file=`*`file_path`*	Required if the `instance.config.profile` parameter value is `CUSTOM`. Absolute path to the custom profile XML file.
	`install.service={true|false}`	Optional. Specify if a Windows service will be registered during instance creation. The default value is `false`.

Important:
You cannot rename a Universal Messaging instance.

Examples

- To check if Universal Messaging supports instance management operations through Command Central for a node with alias “messagingNode”:

```
sagcc list instances messagingNode supportedproducts
```

- To create a new instance for an installed Universal Messaging realm server with instance name “umserver” and port number “9000” on a node with alias “messagingNode”:

```
sagcc create instances messagingNode NUMRealmServer instance.name=umserver
instance.ip=0.0.0.0 instance.port=9000
```

- To create a new instance for an installed Universal Messaging server with instance name “umserver” and port number “9000”, and with a custom data directory path and license file, on a node with alias “messagingNode”:

```
sagcc create instances messagingNode NUMRealmServer instance.name=umserver
instance.ip=0.0.0.0 instance.port=9000
instance.dataDir=Data_Directory_Absolute_Path
license.file=absolute path to the license file
```

- To create a new instance for an installed Universal Messaging realm server with instance name "umserver" and port number “9000” on a node with alias “messagingNode”, and provide initial configuration settings:

```
sagcc create instances messagingNode NUMRealmServer instance.name=umserver
instance.ip=0.0.0.0 instance.port=9000 instance.config.profile=wM or TC
```

```
```

- To create a new instance of Universal Messaging server instance named "umserver" on port number “9000” of the node with alias “messagingNode”, and provide custom initial configuration settings:

```
sagcc create instances messagingNode NUMRealmServer instance.name=umserver
instance.ip=0.0.0.0 instance.port=9000 instance.config.profile=CUSTOM
```

```
instance.config.file=absolute path to the custom profile XML file
```

- To register a Windows service when creating the Universal Messaging server instance named “umserver” on port number “9000” of the node with alias “messagingNode”:

```
sagcc create instances messagingNode NUMRealmServer instance.name=umserver
install.service=true
```

- To read the following properties of Universal Messaging server named “umserver":
  - Port number
  - License path
  - Interface IP address
  - Server data directory path

- Service status

```
sagcc list instances messagingNode Universal-Messaging-umserver
```

Note:
The service associated with the instance is automatically deleted when the instance is deleted.

- To create an Enterprise Manager instance named "EM1":

```
sagcc create instances messagingNode NUMEnterpriseManager instance.name=em1
instance.ip=0.0.0.0 instance.port=9000
```

Use the Universal Messaging Instance Manager tool to delete the Enterprise Manager and Template Applications instances. You cannot use Command Central commands for deleting these instances as they are not listed in the product inventory.

Note:
You can create a Template Applications instance using the same command used to create an Enterprise Manager instance.

Updating an Instance on the Command Line

To update the "umserver" Universal Messaging server instance existing on the node with alias "messagingNode":

```
sagcc update instances messagingNode Universal-Messaging-umserver
install.service=true or false
```

Deleting an Instance on the Command Line

To delete the "umserver" Universal Messaging server instance existing on the node with alias "messagingNode":

```
sagcc delete instances messagingNode Universal-Messaging-umserver
```

Configuration Types That the Universal Messaging Server Supports

The Universal Messaging realm server run-time component supports creating instances of the configuration types listed in the following table.

Configuration Type	Use to...
COMMON-LOCAL-USERS	Configure and manage users of a Universal Messaging server instance. COMMON-LOCAL-USERS-*userId* supports configuring the user ID and password of each user. By default, the users have administrator privileges for the Universal Messaging server instance.

Configuration Type	Use to...
COMMON-LICENSE	Configure the Universal Messaging-specific SagLic license file.
COMMON-LICLOC	View the location of a Universal Messaging server instance's license file. You cannot change the location of the license file.
COMMON-JAVASYSPROPS	Extended Java system properties.
COMMON-JVM-OPTIONS	Extended JVM options.
UM-JNDI-CF	Configure the following JNDI connection factories: ■ Connection Factory ■ Topic Connection Factory ■ Queue Connection Factory ■ XA Connection Factory
UM-JNDI-DEST	Configure the following JNDI destinations: ■ Topics ■ Queues
COMMON-CLUSTER	Configure an active/active Universal Messaging cluster.
COMMON-MEMORY	Configure the initial memory size and maximum memory size of a Universal Messaging server instance.
COMMON-PORTS	Configure the Universal Messaging server interfaces. **Note:** ■ You cannot change the protocol, bind address, port number, or alias of a port of an existing server interface. ■ If you change the SSL certificates of a secured interface, you must restart the interface.
UM-ZONE	Create and manage Universal Messaging zones.
UM-CHANNELS	Create and manage Universal Messaging channels.
UM-QUEUES	Create and manage Universal Messaging queues.
UM-REALM-ACL	Manage Universal Messaging Access Control List (ACL).
UM-GROUPS	Manage Universal Messaging security groups.
COMMON-SYSPROPS	DEPRECATED. Use COMMON-JAVASYSPROPS.

User Configuration Commands

You can perform the following user configuration tasks on a Universal Messaging server instance from the command line interface:

- List the path and the users existing in the user repository using the `sagcc get configuration instances node_alias Universal-Messaging-umserver` command.
- Retrieve information about a specific user using the `sagcc get configuration instances node_alias Universal-Messaging-umserver COMMON-LOCAL-USERS-user name` command.
- Add new users to the user repository using the `sagcc create configuration data node_alias Universal-Messaging-umserver COMMON-LOCAL-USERS --input absolute path to the XML file containing the user ID and password` command.
- Update or change the password of an existing user using the `sagcc update configuration data node_alias Universal-Messaging-umserver COMMON-LOCAL-USERS-user name --input absolute path to the XML file containing new password` command.
- Delete existing users from the user repository using the `sagcc delete configuration data node_alias Universal-Messaging-umserver COMMON-LOCAL-USERS-user name` command.

Information to authenticate the users of a Universal Messaging server instance is stored in the user repository (users.txt file) of the Universal Messaging server instance. The users.txt file is generated only after you create a new internal user. While creating the user repository, if you specify a relative path in the jaas.conf file, the users.txt file will be created in a directory relative to the bin directory of the Universal Messaging server instance.

The path to the users.txt file is added in the jaas.conf file present in <*InstallDir*>/UniversalMessaging/server/<*InstanceName*>/bin/jaas.conf. If you specify a relative path in the jaas.conf file, the users.txt file will be created in a directory relative to the bin directory of the Universal Messaging server instance.

Examples

- To list the path of the user repository and the users of a Universal Messaging server instance:

  ```
  sagcc get configuration instances sag01 Universal-Messaging-umserver
  ```

  where `umserver` is the name of the Universal Messaging server instance and `sag01` is the alias name of the installation where `umserver` is running.

- To retrieve information of a Universal Messaging server instance user:

  ```
  sagcc get configuration instances sag01 Universal-Messaging-umserver
  COMMON-LOCAL-USERS-user1
  ```

where `umserver` is the name of the Universal Messaging server instance, `sag01` is the alias name of the installation where `umserver` is running, and `user1` is the user ID of the user.

- To add a user to a Universal Messaging server instance:

```
sagcc create configuration data sag01 Universal-Messaging-umserver
COMMON-LOCAL-USERS --input c:\inputxmls\user2.xml
```

where `umserver` is the name of the Universal Messaging server instance, `sag01` is the alias name of the installation where umserver is running, and `user2.xml` is file that contains the user ID and the password of the new user.

The user2.xml file has the following format:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<User id="user2">
<Password>test</Password>
</User>
```

- To update the password of a Universal Messaging server instance user:

```
sagcc update configuration data sag01 Universal-Messaging-umserver
COMMON-LOCAL-USERS-user2 --input c:\inputxmls\user2.xml
```

where `umserver` is the name of the Universal Messaging server instance, `sag01` is the alias name of the installation where `umserver` is running, and `user2.xml` is the file that contains the new password of the specified user.

- To delete a Universal Messaging server instance user:

```
sagcc delete configuration data sag01 Universal-Messaging-umserver
COMMON-LOCAL-USERS-user2
```

where `umserver` is the name of the Universal Messaging server instance and `sag01` is the alias name of the installation where `umserver` is running.

License Configuration Commands

You can perform the following license management tasks on a Universal Messaging server instance from the Command Central command-line interface:

- View the license details of a Universal Messaging server instance using the `sagcc get configuration data` *`node_alias`* `Universal-Messaging-`*`umserver`* `COMMON-LICENSE-Universal-Messaging` command.
- View the license location of a Universal Messaging server instance using the `sagcc get configuration data` *`node_alias`* `Universal-Messaging-`*`umserver`* `COMMON-LICLOC-Universal-Messaging` command.
- Add a Universal Messaging license key file with the specified alias to the Command Central license key manager using the `sagcc add license-tools keys` *`license key alias -i absolute path to the license file`* command.

- Update a license key file assigned to the specified license key alias using the `sagcc update configuration license` *`node_alias`* `Universal-Messaging-`*`umserver`* `COMMON-LICENSE-Universal-Messaging` *`license key alias`* command.

Examples

- To view the license details of a Universal Messaging server instance with "Universal-Messaging-umserver" component ID that runs in the installation with alias name “sag01”:

```
sagcc get configuration data sag01 Universal-Messaging-umserver
COMMON-LICENSE-Universal-Messaging
```

- To view the license file location of a Universal Messaging server instance with "Universal-Messaging-umserver" component ID that runs in the installation with alias name “sag01”:

```
sagcc get configuration data sag01 Universal-Messaging-umserver
COMMON-LICLOC-Universal-Messaging
```

- To add a Universal Messaging license key file with the license key alias "um_lic" to the Command Central license key manager:

```
sagcc add license-tools keys um_lic -i C:\umlicense\new_license.xml
```

- To update a license key file assigned to the license key alias "um_lic" with "Universal-Messaging-umserver" component ID that runs in the installation with alias name “sag01” :

```
sagcc update configuration license sag01 Universal-Messaging-umserver
COMMON-LICENSE-Universal-Messaging um_lic
```

Port Configuration Commands

You can create, update, and delete a port for a Universal Messaging server instance using the Command Central command-line interface.

- Create a port using the `sagcc create configuration data` *`node_alias`* `Universal-Messaging-`*`umserver`* `COMMON-PORTS --`*`input_port_configuration`*`.xml --password` *`Command Central password`* command.

  The input port configuration XML file should be of the following format:

```
<PortSettings>
    <Port alias="nhp1">
        <Enabled>true</Enabled>
        <Type>STANDARD</Type>
        <Number>9001</Number>
        <Protocol>NHP</Protocol>
        <Backlog>100</Backlog>
        <ExtendedProperties>
            <Property name="autostart">true</Property>
            <Property name="allowforinterrealm">true</Property>
            <Property name="authtime">1000</Property>
```

```
            <Property name="EnableNIO">true</Property>
            <Property name="acceptThreads">2</Property>
            <Property name="receivebuffersize">1310721</Property>
            <Property name="SelectThreads">4</Property>
            <Property name="advertise">true</Property>
            <Property name="allowclientconnections">true</Property>
            <Property name="Backlog">100</Property>
            <Property name="Alias"/>
            <Property name="keyAlias"/>
            <Property name="sendbuffersize">1310721</Property>
            <Property name="EnableHTTP11">true</Property>
            <Property name="EnableJavaScript">true</Property>
            <Property name="CORSAllowCredentials">true</Property>
            <Property name="CORSAllowedOrigins">*</Property>
            <Property name="AjaxLPActiveDelay">100</Property>
            <Property name="EnableWebSockets">true</Property>
            <Property name="EnableGZipLP">true</Property>
            <Property name="MinimumBytesBeforeGZIP">1000</Property>
            <Property name="AjaxLPIdleDelay">60000</Property>
            <Property name="header1Name">foo</Property>
            <Property name="header1Value">bar</Property>
            <Property name="header1UserAgent">mozilla</Property>
        </ExtendedProperties>
    </Port>
</PortSettings>
```

- Modify the configuration of a port using the `sagcc update configuration data` *`node_alias`* `Universal-Messaging-`*`umserver`* `COMMON-PORTS-`*`port name`* `--`*`input_port_configuration`*`.xml --password` *`Command Central password`* command.
- Delete a port using the `sagcc delete configuration data` *`node_alias`* `Universal-Messaging-`*`umserver`* `COMMON-PORTS-`*`port name`* `--`*`input_port_configuration`*`.xml --password` *`Command Central password`* command.

Examples

- To create a port for a Universal Messaging server instance with "Universal-Messaging-umserver" component ID that runs in the installation with alias name “sag01”, where input_port.xml is an XML file containing port configuration information:

```
sagcc create configuration data sag01 Universal-Messaging-umserver
 COMMON-PORTS --input_port.xml --password myccpassword
```

- To update or modify port configuration for a Universal Messaging server instance with "Universal-Messaging-umserver" component ID that runs in the installation with alias name “sag01”, where input_port.xml is an XML file containing the updated port configuration information, and nhp1 is the name of the port to be updated:

```
sagcc update configuration data sag01 Universal-Messaging-umserver
 COMMON-PORTS-nhp1 --input_port.xml --password myccpassword
```

- To delete a port for a Universal Messaging server instance with "Universal-Messaging-umserver" component ID that runs in the installation with alias name “sag01”, where input_port.xml is an XML file containing the updated port configuration information, and nhp1 is the name of the port to be deleted:

```
sagcc delete configuration data sag01 Universal-Messaging-umserver
 COMMON-PORTS-nhp1 --input_port.xml --password myccpassword
```

Security Group Configuration Commands

You can retrieve the security group configuration data for a Universal Messaging server instance using the `sagcc get configuration data` *`node_alias`* `Universal-Messaging-`*`serverName`* `UM-GROUPS-`*`groupName`* command.

To retrieve the security group configuration information for a Universal Messaging server instance with "Universal-Messaging-umserver" component ID that runs in the installation with alias name “sag01”:

```
sagcc get configuration data sag01 Universal-Messaging-umserver
UM-GROUPS-Everyone
```

Realm ACL Configuration Commands

You can view the realm ACL configuration information for a Universal Messaging server instance in XML format using the `sagcc get configuration data` *`node_alias`* `Universal-Messaging-`*`serverName`* `UM-REALM-ACL` command.

To view the realm ACL configuration information for a Universal Messaging server instance with "Universal-Messaging-umserver" component ID that runs in the installation with alias name “sag01”:

```
sagcc get configuration data sag01 Universal-Messaging-umserver
UM-REALM-ACL
```

General Properties Configuration Commands

- You can view configuration properties for a specific configuration group by using the `sagcc get configuration data` command.
- You can update the configuration properties by using the `sagcc update configuration data` command.

For information about the configuration properties that the Universal Messaging realm server supports, see “General Properties” on page 373.

Usage Notes

- Parameter names are case sensitive. For parameter values of enumeration type, set values ranging from 0 to n to map to the corresponding enumeration values in the Command Central user interface.
- The `sagcc update configuration data` command exits and displays an error at the first instance of a wrong parameter definition. For example, the command will exit displaying an error if the MonitorTimer property is assigned a string value.

Examples

- To view the cluster configuration properties for the `Universal-Messaging-umserver` server instance:

  ```
  sagcc get configuration data sag01 Universal-Messaging-umserver
  COMMON-SYSPROPS-Data_Stream_Config
  ```

  `sag01` is the alias name of the installation where `Universal-Messaging-umserver` server instance is running. `Data_Stream_Config` is the name of the configuration group.

- To update cluster configuration properties for `Universal-Messaging-umserver` server instance:

  ```
  sagcc update configuration data sag01 Universal-Messaging-umserver
  COMMON-SYSPROPS-Data_Stream_Config --input c:\datastreamconfig.properties
  ```

  `sag01` is the alias name of the installation where `Universal-Messaging-umserver` server instance is running.

  `COMMON-SYSPROPS-Cluster_Config` is the configuration instance ID where `COMMON-SYSPROPS` is the configuration type and `Data_Stream_Config` is the name of the configuration group.

  `datastreamconfig.properties` is the properties file containing the modified configuration parameters.

  The properties file should contain parameter values in the following format:

  ```
  MonitorTimer=10000
  OffloadMulticastWrite=false

  SendInitialMapping=true
  ```

JNDI Asset Configuration Commands

You can create, view, update, and delete a JNDI asset. Ensure that the Universal Messaging server instance is running before running the following commands.

- View all the configuration types for a Universal Messaging server instance using the `sagcc get configuration types` *`node_alias`* `Universal-Messaging-`*`umserver`* command.
- View all the configuration instances for a Universal Messaging server instance using the `sagcc get configuration instances` *`node_alias`* `Universal-Messaging-`*`umserver`* command.
- Create a new JNDI connection factory by passing parameters defined in an XML file using the `sagcc create configuration data` *`node_alias`* `Universal-Messaging-`*`umserver`* `UM-JNDI-CF -i` *`absolute path to the XML file`* command. You can create the following connection factory types: `ConnectionFactory`, `TopicConnectionFactory`, `QueueConnectionFactory`, and `XAConnectionFactory`.

  The XML file should contain the parameters in the following format:

  ```
  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  <connectionFactory>
  ```

```
 <name>connection_factory_name</name>
 <type>connection_factory_type</type>
 <url>connection_factory_binding_url</url>
 <durableType>type of durable</durableType>
</connectionFactory>
```

Note:
The parameters `name`, `type`, and `url` are required, and the `durableType` parameter is optional.

- Create a new JNDI destination by passing parameters defined in an XML file using the `sagcc create configuration data node_alias Universal-Messaging-umserver UM-JNDI-DEST -i absolute path to the XML file` command. You can create the following destination types: `Topic` and `Queue`.

  The XML file should contain the parameters in the following format:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<destination>
 <name>destination_name</name>
 <type>destination_type</type>
 <storeName>jms_channel_or_queue_name</storeName>
 <autoCreateDest>true/false</autoCreateDest>
</destination>
```

  `name` parameter can include upper and lower case alphabetic characters, digits (0-9), double colon (::), slash (/), and periods (.), for example, destination1. Use the double colon (::) for specifying nested name space, for example, destination1::destination2. A combination of special characters in a `name` is not allowed, for example, destination1::destination2/destination3. `storeName` parameter can include upper and lower case alphabetic characters, digits (0-9), double colon (::), slash (/), and underscores (_) but cannot include periods (.).

- Retrieve information about a specific JNDI connection factory using the `sagcc get configuration data node_alias Universal-Messaging-umserver UM-JNDI-CF-connection_factory_name` command.
- Retrieve information about a specific JNDI destination using the `sagcc get configuration data node_alias Universal-Messaging-umserver UM-JNDI-DEST-destination_name` command.
- Update a JNDI connection factory by passing the new parameters defined in an XML file using the `sagcc update configuration data node_alias Universal-Messaging-umserver UM-JNDI-CF-connection_factory_name -i absolute path to the XML file` command.

  Important:
  You can update the URL and the `durableType` property, you cannot update the name of the connection factory.

- Delete a JNDI connection factory using the `sagcc delete configuration data node_alias Universal-Messaging-umserver UM-JNDI-CF-connection_factory_name` command.
- Delete a JNDI destination using the `sagcc delete configuration data node_alias Universal-Messaging-umserver UM-JNDI-DEST-destination_name` command.

  Note:

Deleting a JNDI destination will not delete the channel or queue that exists on the Universal Messaging server instance.

Usage Notes

- Updating JNDI destinations is not supported.
- Creating a connection factory and destination with the same name is not allowed for a Universal Messaging server instance.

Examples

- To view all the configuration types for a Universal Messaging server instance:

```
sagcc get configuration types sag01 Universal-Messaging-umserver
```

`sag01` is the alias name of the installation where `Universal-Messaging-umserver` server instance is running.

- To view all the configuration instances for a Universal Messaging server instance:

```
sagcc get configuration instances sag01 Universal-Messaging-umserver
```

`sag01` is the alias name of the installation where `Universal-Messaging-umserver` server instance is running.

- To create a new JNDI connection factory by passing parameters defined in an XML file:

```
sagcc create configuration data sag01 Universal-Messaging-umserver
UM-JNDI-CF -i C:\jndi\connecton_factory.xml
```

`sag01` is the alias name of the installation where `Universal-Messaging-umserver` server instance is running. `UM-JNDI-CF` is the configuration type and `C:\jndi\connecton_factory.xml` is the absolute path to the XML file in which the parameters are defined. Example of properties defined in the XML file:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<connectionFactory>
 <name>connectionfactory1</name>
 <type>ConnectionFactory</type>
 <url>nhp://124.597.890:9100</url>
 <durableType>Shared</durableType>
</connectionFactory>
```

- To create a new JNDI destination by passing parameters defined in an XML file:

```
sagcc create configuration data sag01 Universal-Messaging-umserver
UM-JNDI-DEST -i C:\jndi\destination.xml
```

`sag01` is the alias name of the installation where `Universal-Messaging-umserver` server instance is running. `UM-JNDI-DEST` is the configuration type and `C:\jndi\destination.xml` is the absolute path to the XML file in which the parameters are defined. Example of properties defined in the XML file:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

```
<destination>
 <name>topicLookup</name>
 <type>destination1</type>
 <storeName>topic1</storeName>
 <autoCreateDest>true</autoCreateDest>
</destination>
```

- To retrieve information about a specific JNDI connection factory:

```
sagcc get configuration data sag01 Universal-Messaging-umserver
UM-JNDI-CF-connectionfactory1
```

`sag01` is the alias name of the installation where `Universal-Messaging-umserver` server instance is running. `UM-JNDI-CF` is the configuration type and `connectionfactory1` is the name of the JNDI connection factory from which information is to be retrieved.

- To retrieve information about a specific JNDI destination:

```
sagcc get configuration data sag01 Universal-Messaging-umserver
UM-JNDI-DEST-destination1
```

`sag01` is the alias name of the installation where `Universal-Messaging-umserver` server instance is running. `UM-JNDI-DEST` is the configuration type and `destination1` is the name of the JNDI destination from which information is to be retrieved.

- To update a JNDI connection factory by passing the new parameters defined in an XML file:

```
sagcc update configuration data sag01 Universal-Messaging-umserver
UM-JNDI-CF-connectionfactory1 -i C:\jndi\update.xml
```

`sag01` is the alias name of the installation where `Universal-Messaging-umserver` server instance is running. `UM-JNDI-CF` is the configuration type, `onnectionfactory1` is the name of the connection factory to be updated, and `C:\jndi\update.xml` is the absolute path to the XML file in which the updated parameters are defined. Example of properties defined in the XML file:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<connectionFactory>
 <name>connectionfactoryupdated</name>
 <type>ConnectionFactory</type>
 <url>nhp://124.597.890:9100</url>
 <durableType>Serial</durableType>
</connectionFactory>
```

- To delete a JNDI connection factory:

```
sagcc delete configuration data sag01 Universal-Messaging-umserver
UM-JNDI-CF-connectionfactory1
```

`sag01` is the alias name of the installation where `Universal-Messaging-umserver` server instance is running. `UM-JNDI-CF` is the configuration type and `connectionfactory1` is the name of the JNDI connection factory.

- To delete a JNDI destination:

```
sagcc delete configuration data sag01 Universal-Messaging-umserver
UM-JNDI-DEST-destination1
```

`sag01` is the alias name of the installation where `Universal-Messaging-umserver` server instance is running. `UM-JNDI-DEST` is the configuration type and `destination1` is the name of the JNDI destination.

Channel and Queue Configuration Commands

Ensure that the Universal Messaging server instance is running before executing the following commands:

- View all the configuration instances for a Universal Messaging server instance using the `sagcc get configuration instances` *`node_alias`* `Universal-Messaging-`*`umserver`* command.
- Create a channel on a Universal Messaging server instance using the `sagcc create configuration data` *`node_alias`* `Universal-Messaging-`*`umserver`* `UM-CHANNELS -i` *`absolute path to the XML file containing channel properties`* command .
- Update a channel on a Universal Messaging server instance using the `sagcc update configuration data` *`node_alias`* `Universal-Messaging-`*`umserver`* `UM-CHANNELS-`*`channel_name`* `-f xml -i` *`absolute path to the XML file containing channel properties`* command.
- Create a queue on a Universal Messaging server instance using the `sagcc create configuration data` *`node_alias`* `Universal-Messaging-`*`umserver`* `UM-QUEUES -i` *`absolute path to the XML file containing queue properties`* command.
- Update a queue on a Universal Messaging server instance using the `sagcc update configuration data` *`node_alias`* `Universal-Messaging-`*`umserver`* `UM-QUEUES-`*`queue_name`* `-f xml -i` *`absolute path to the XML file containing channel properties`* command.
- Retrieve the configuration information of a specific channel using the `sagcc get configuration data` *`node_alias`* `Universal-Messaging-`*`umserver`* `UM-CHANNELS-`*`channel_name`* command.
- Retrieve the configuration information of a specific queue using the `sagcc get configuration data` *`node_alias`* `Universal-Messaging-`*`umserver`* `UM-QUEUES-`*`queue_name`* command.

Examples

- To view all the configuration instances for a Universal Messaging server instance:

```
sagcc get configuration instances sag01 Universal-Messaging-umserver
```

`sag01` is the alias name of the installation where the `Universal-Messaging-umserver` server instance is running.

- To create a channel on a Universal Messaging server instance:

```
sagcc create configuration data sag01 Universal-Messaging-umserver
UM-CHANNELS -i C:\Channels\channel_create.xml
```

`sag01` is the alias name of the installation where the `Universal-Messaging-umserver` server instance is running. `UM-CHANNELS` is the configuration type and `channel_create.xml` is an XML file containing the channel attributes of the new channel that is created.

Format of the channel_create.xml file:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Channel>
    <name>channel1</name>
    <type>Persistent</type>
    <ttl>0</ttl>
    <capacity>0</capacity>
 .......
</Channel>
```

- To update a channel on a Universal Messaging server instance:

```
sagcc update configuration data sag01 Universal-Messaging-umserver
UM-CHANNELS-channel1 -f xml -i C:\Channels\channel_edited.xml
```

`sag01` is the alias name of the installation where the `Universal-Messaging-umserver` server instance is running. `UM-CHANNELS-channel1` is the configuration type, in which `channel1` is the name of the channel. `channel_edited.xml` is an XML file containing the channel attributes to be updated.

Format of the channel_edited.xml file:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Channel>
    <name>channel1</name>
    <type>Persistent</type>
    <ttl>50000</ttl>
    <capacity>50000</capacity>
    <deadEventStore/>
    <engine>JMS Engine</engine>
......
</Channel>
```

- To create a queue on a Universal Messaging server instance:

```
sagcc create configuration data sag01 Universal-Messaging-umserver
UM-QUEUES -i C:\queues\queue_create.xml
```

`sag01` is the alias name of the installation where the `Universal-Messaging-umserver` server instance is running. `UM-QUEUES` is the configuration type and `queue_create.xml` is an XML file containing the attributes of the new queue.

Format of the queue_create.xml file:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Queue>
    <name>queue1</name>
    <type>Persistent</type>
    <ttl>78</ttl>
    <capacity>99</capacity>
    <parent>umserver</parent>
  ......
</Queue>
```

- To update a specific queue on a Universal Messaging server instance:

```
sagcc update configuration data sag01 Universal-Messaging-umserver
UM-QUEUES-queue1 -f xml -i C:\queues\queue_edited.xml
```

`sag01` is the alias name of the installation where the `Universal-Messaging-umserver` server instance is running. `UM-QUEUES-queue1` is the configuration type, in which `queue1` is the name of the queue to update. `queue_edited.xml` is an XML file containing the queue attributes to be updated.

Format of the queue_edited.xml file:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Queue>
    <name>queue1</name>
    <type>Persistent</type>
    <ttl>50000</ttl>
    <capacity>50000</capacity>
    <parent>umserver</parent>
    <deadEventStore/>
......
</Queue>
```

- To retrieve configuration information about a specific channel:

```
sagcc get configuration data sag01 Universal-Messaging-umserver
UM-CHANNELS-channel1
```

`sag01` is the alias name of the installation where the `Universal-Messaging-umserver` server instance is running. `UM-CHANNELS-channel1` is the configuration type, in which `channel1` is the name of the channel.

- To retrieve configuration information about a specific queue:

```
sagcc get configuration data sag01 Universal-Messaging-umserver
UM-QUEUES-queue1
```

`sag01` is the alias name of the installation where the `Universal-Messaging-umserver` server instance is running. `UM-QUEUES-queue1` is the configuration type, in which `queue1` is the name of the queue.

Zone Configuration Commands

Ensure that the Universal Messaging server instances are running before running the following commands.

- Retrieve zone configuration information for a Universal Messaging server instance using the `sagcc get configuration data` *`node_alias`* `Universal-Messaging-`*`umserver`* `UM-ZONE` command.
- Create a zone using the `sagcc create configuration data` *`node_alias`* `Universal-Messaging-`*`umserver`* `UM-ZONE -i=`*`path to the XML file containing the zone configuration`* command.

> **Note:**
> Provide the absolute path to the XML file if the XML file is not in the same directory from which the command is run.

- Update a zone using the `sagcc update configuration data node_alias Universal-Messaging-umserver UM-ZONE -i=path to the XML file containing the updated zone configuration` command.

  Note:
  Provide the absolute path to the XML file if the XML file is not in the same directory from which the command is run.

- Delete a zone using the `sagcc delete configuration data node_alias Universal-Messaging-umserver UM-ZONE` command. The delete command deletes the zone in which *umserver* belongs. You can remove specific server instances from a zone by removing the server instances in the updated zone configuration XML file, and then using the update zone command.

Examples

- To retrieve zone configuration information for a Universal Messaging server instance:

```
sagcc get configuration data sag01 Universal-Messaging-umserver UM-ZONE
```

  `sag01` is the alias name of the installation where `Universal-Messaging-umserver` server instance is running. UM-ZONE is the configuration type for zones.

- To create a zone:

```
sagcc create configuration data sag01 Universal-Messaging-umserver UM-ZONE
 -i=C:\zones\zone_create.xml
```

  `sag01` is the alias name of the installation where `Universal-Messaging-umserver` server instance is running. UM-ZONE is the configuration type for zones. And, c:\zones\zones_create.xml is the absolute path to the location where the zone configuration XML file is located.

  Format of the zone_create.xml file for zone with realms:

```
<?xml version="1.0" encoding="UTF-8"?>
<Zone>
 <name>RealmZone</name>
 <type>Realm</type>
 <realms>
  <server name="um_realm1">
   <url>nsp://localhost:9701</url>
  </server>
  <server name="um_realm2">
   <url>nsp://localhost:9702</url>
  </server>
 </realms>
 <clusters/>
</Zone>
```

  Format of the zone_create.xml file for a zone with clusters:

```
<?xml version="1.0" encoding="UTF-8"?>
<Zone>
 <name>ClusterZone</name>
 <type>Cluster</type>
```

```
 <realms />
 <clusters>
  <server name="um_cluster1">
   <url>nsp://localhost:9704</url>
   <clusterName />
   <status />
  </server>
  <server name="um_cluster2">
   <url>nsp://localhost:9705</url>
   <clusterName />
   <status />
  </server>
 </clusters>
</Zone>
```

- To update a zone:

```
sagcc update configuration data sag01 Universal-Messaging-umserver UM-ZONE
 -i=C:\zones\zone_update.xml
```

`sag01` is the alias name of the installation where `Universal-Messaging-umserver` server instance is running. UM-ZONE is the configuration type for zones. And, c:\zones\zone_update.xml is the absolute path to the location where the zone configuration XML file is located.

Note:
zone_update.xml file should be in the same format as the XML file used to create zones. You can substitute the updated configuration values in place of the old values.

- To delete a zone:

```
sagcc delete configuration data local Universal-Messaging-umserver UM-ZONE
```

`sag01` is the alias name of the installation where `Universal-Messaging-umserver` server instance is running. UM-ZONE is the configuration type for zones.

Java System Properties Configuration Commands

Restart the Universal Messaging server instance after running the commands for the configuration changes to take effect.

- Retrieve the Java system properties for a Universal Messaging server instance using the `sagcc get configuration data` *`node_alias`* `Universal-Messaging-`*`umserver`* `COMMON-JAVASYSPROPS` command.
- Update the Java system properties for a Universal Messaging server instance using the `sagcc get configuration data` *`node_alias`* `Universal-Messaging-`*`umserver`* `COMMON-JAVASYSPROPS -i` *`absolute path to the XML file containing the updated configuration`* command.

Examples

- To retrieve the Java system properties for a Universal Messaging server instance:

```
sagcc get configuration data local Universal-Messaging-umserver1
COMMON-JAVASYSPROPS
```

- To update the Java system properties for a Universal Messaging server instance:

```
sagcc update configuration data local Universal-Messaging-umserver1
COMMON-JAVASYSPROPS -i C:\javasysprop.xml
```

JVM Options Configuration Commands

Restart the Universal Messaging server instance after running the commands for the configuration changes to take effect.

- Retrieve the JVM options for a Universal Messaging server instance using the `sagcc get configuration data` *`node_alias`* `Universal-Messaging-`*`umserver`* `COMMON-JVM-OPTIONS` command.
- Update the JVM options for a Universal Messaging server instance using the `sagcc get configuration data` *`node_alias`* `Universal-Messaging-`*`umserver`* `COMMON-JVM-OPTIONS -i` *`absolute path to the XML file containing the updated configuration`* command.

Examples

- To retrieve the JVM options for a Universal Messaging server instance:

```
sagcc get configuration data local Universal-Messaging-umserver1
COMMON-JVM-OPTIONS
```

- To update the JVM options for a Universal Messaging server instance:

```
sagcc update configuration data local Universal-Messaging-umserver1
COMMON-JVM-OPTIONS -i C:\javasysprop.xml
```

  The XML file containing the updated JVM options has the following format:

```
<?xml version="1.0"?>
-<jvmOptions>
<option>-XX:+TraceClassLoading</option>
</jvmOptions>
```

Cluster Configuration Commands

Before You Create or Update a Universal Messaging Cluster

- Make sure the server instances that you want to add to the cluster are running.
- Verify that the permissions on the server machines allow connections to the other servers in the cluster.
- Make sure the /naming/defaultContext channel exists only on one or none of the nodes that will form the cluster. The Universal Messaging server instance used as a JNDI provider uses the /naming/defaultContext channel to store JMS references and JNDI objects. If the channel exists on multiple nodes, you cannot create the cluster.
- If you have created custom composite templates for Command Central 9.9 or earlier, ensure that you remove the Universal Messaging server instance name suffix from

COMMON-CLUSTER configuration type in the composite template when applying the composite template in Command Central 9.10 or later. For example:

In Command Central 9.9 and earlier:

```
um-cluster:
    description: Cluster configuration for two UM instances
    products:
      NUMRealmServer:
        ${node.host}:
          instance.port: ${um.instance.port}
          instance.ip:   ${um.host}
          runtimeComponentId: Universal-Messaging-${instance.name}
          configuration:
            Universal-Messaging-${instance.name}:
              COMMON-CLUSTER:
                COMMON-CLUSTER-${instance.name}: &umClusterConfig
                  Name: ${um.cluster}
                  Servers:                      # two UM instances cluster
                    Server:
                      -
                        "@name": ${um.host}
                        URL: "nsp://${um.host}:${um.instance.port}"
                      -
                        "@name": ${um.host2}
                        URL: "nsp://${um.host2}:${um.instance.port2}"
```

In Command Central 9.10 and later:

```
um-cluster:
    description: Cluster configuration for two UM instances
    products:
      NUMRealmServer:
        ${node.host}:
          instance.port: ${um.instance.port}
          instance.ip:   ${um.host}
          runtimeComponentId: Universal-Messaging-${instance.name}
          configuration:
            Universal-Messaging-${instance.name}:
              COMMON-CLUSTER:
                COMMON-CLUSTER: &umClusterConfig
                  Name: ${um.cluster}
                  Servers:                      # two UM instances cluster
                    Server:
                      -
                        "@name": ${um.host}
                        URL: "nsp://${um.host}:${um.instance.port}"
                      -
                        "@name": ${um.host2}
                        URL: "nsp://${um.host2}:${um.instance.port2}"
```

Creating a Universal Messaging Cluster

To create an active/active cluster of Universal Messaging server instances, specify the cluster configuration details in an XML input file and supply the input file as an argument of the `sagcc create configuration data` command. Specify the following settings in the XML file:

- Cluster name (required). A cluster name that is unique to the installation.

- Server instances (required). The name, URL, and port of each server node in the cluster.
- Sites (optional). The name of the site to which each server node belongs. `siteName` is a server level property.
- Primary site (optional). The name of the primary site, if you have configured sites in the cluster. `primeSite` is a cluster level property that holds the name of the site, which is flagged as `isPrime`.

Examples

To create a new cluster with the following configurations specified in the umSalesClusterConfig.xml file:

- Cluster name: umSales
- Cluster sites: site1 and site2
- Primary site: site1
- Server instances in site1: um9000, um9001
- Server instances in site2: um9002, um9003

```
sagcc create configuration data sag01 Universal-Messaging-um9001
COMMON-CLUSTER --input C:\inputxmls\umSalesClusterConfig.xml
```

where `sag01` is the alias name of the installation in which the `Universal-Messaging-um9001` server instance is running. The cluster configurations are specified in the umSalesClusterConfig.xml file as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<ClusterSettings>
  <Name>umSales</Name>
  <Servers>
  <Server name="um9000">
    <URL>nsp://127.0.0.1:9000</URL>
    <ExtendedProperties>
      <Property name="siteName">site1</Property>
    </ExtendedProperties>
  </Server>
  <Server name="um9001">
    <URL>nsp://127.0.0.1:9001</URL>
    <ExtendedProperties>
      <Property name="siteName">site1</Property>
    </ExtendedProperties>
  </Server>
  <Server name="um9002">
    <URL>nsp://127.0.0.1:9002</URL>
    <ExtendedProperties>
      <Property name="siteName">site2</Property>
    </ExtendedProperties>
  </Server>
  <Server name="um9003">
    <URL>nsp://127.0.0.1:9003</URL>
    <ExtendedProperties>
      <Property name="siteName">site2</Property>
    </ExtendedProperties>
```

```
  </Server>
  </Servers>
  <ExtendedProperties>
    <Property name="primeSite">site1</Property>
  </ExtendedProperties>
</ClusterSettings>
```

Usage Notes

A Universal Messaging server instance can be part of only one cluster.

If you remove all the server instances from a site, the site will be deleted. Server instance deletion is not allowed if the deletion operation leaves fewer than two server instances in the cluster.

Viewing Cluster Details

To retrieve the following details of the Universal Messaging cluster in an XML file, specify one of the server instances of the cluster in the `sagcc get configuration data` command.

- Name of the cluster
- Name, URL, and port of each Universal Messaging server instance in the cluster
- Site information, if sites are configured

To view the details of the cluster configuration of the um9001 Universal Messaging server instance:

```
sagcc get configuration data sag01 Universal-Messaging-um9001 COMMON-CLUSTER
```

where `sag01` is the alias name of the installation where `Universal-Messaging-um9001` server instance is running.

Updating a Universal Messaging Cluster

To update a Universal Messaging cluster:

The XML file used for configuring a cluster must contain all the specifications for the cluster. When you update a cluster, you only edit the parameters that specify the change; other specifications in the cluster configuration file should not be changed. You can make these Universal Messaging cluster configurations changes:

To...	Edit the cluster configuration XML file to...
Add one or more server instances to the existing cluster	Include the name, URL, and port of the server instances that you want to add to the cluster.
Remove one or more server instances from the existing cluster	Remove the specifications of the server instances that you want to remove from the existing cluster.
Create sites and assign server instances to sites	Set the siteName extended property of the corresponding server instances.

To...	Edit the cluster configuration XML file to...
Assign a site as the prime site of a cluster	Assign the name of the prime site to the `primeSite` cluster level property.
Remove one or more server instances from a cluster site	Remove the siteName extended property of the corresponding server instances.
Remove sites from a cluster	Remove the site definitions of all the server instances in the cluster.

To update the configuration of the cluster that contains the um9001 server instance:

```
sagcc update configuration data sag01 Universal-Messaging-um9001
COMMON-CLUSTER --input C:\inputxmls\umSalesClusterConfig.xml
```

where `sag01` is the alias name of the installation where `Universal-Messaging-um9001` server instance is running.

Usage Notes

A Universal Messaging server instance can be part of only one cluster.

If you remove all the server instances from a site, the site will be deleted. Server instance deletion is not allowed if the deletion operation leaves fewer than two server instances in the cluster.

Deleting a Universal Messaging Cluster

To delete Universal Messaging cluster:

Delete the cluster by specifying one of the server instances of the cluster using the `sagcc delete configuration data` command.

Example:

To delete the cluster that contains the um9001 server instance:

```
sagcc delete configuration data sag01 Universal-Messaging-um9001
COMMON-CLUSTER
```

where `sag01` is the alias name of the installation where `Universal-Messaging-um9001` server instance is running.

Migrating a Universal Messaging Cluster

You can migrate a Universal Messaging cluster automatically even if the nodes are installed on multiple hosts.

To automatically migrate a Universal Messaging cluster:

1. Add the following extended property to the source Universal Messaging cluster configuration XML file:

```
<ExtendedProperties>
      <Property name="crossHostMigration">true</Property>
    </ExtendedProperties>
```

2. Run the `sagcc update configuration data` command.

   A remote realms bootstrap configuration file is created in the *`Universal Messaging_directory`*`/bin`.

Channel and Queue Monitoring Commands

You can retrieve information about channels and queues using the following commands:

- View the list of administration namespaces using the `sagcc get administration component` *`node_alias`* `Universal-Messaging-`*`umserver`* command.
- Retrieve all the configuration information about a channel in XML format using the `sagcc get configuration data` *`node_alias`* `Universal-Messaging-`*`umserver`* `UM-CHANNELS-`*`channel name`* command.
- Retrieve information about the options available for monitoring channels using the `sagcc get administration component` *`node_alias`* `Universal-Messaging-`*`umserver`* `channels` command.
- List the channels on a Universal Messaging server instance using the `sagcc get administration component` *`node_alias`* `Universal-Messaging-`*`umserver`* `channels list` command.
- Retrieve the status of a specific channel in TSV format using `sagcc get administration component` *`node_alias`* `Universal-Messaging-`*`umserver`* `channels status name=`*`channel name`* command.
- Retrieve a list of durable subscribers on a specific channel in TSV format using the `sagcc get administration component` *`node_alias`* `Universal-Messaging-`*`umserver`* `channels durablesubscribers name=`*`channel name`* command.
- Delete a durable subscriber subscribed to a channel using `sagcc exec administration component` *`node_alias`* `Universal_Messaging-`*`umserver`* `channels deletedurablesubscriber name=`*`channel_name`* `durablesubscriber=`*`durable subscriber name`*

Examples

- To view the list of administration namespaces:

  ```
  sagcc get administration component sag01 Universal-Messaging-umserver
  ```

  `sag01` is the alias name of the installation where `Universal-Messaging-umserver` server instance is running.

- To retrieve all the configuration information about a channel:

  ```
  sagcc get configuration data sag01 Universal-Messaging-umserver
  UM-CHANNELS-channelname
  ```

`sag01` is the alias name of the installation where `Universal-Messaging-umserver` server instance is running. `UM-CHANNELS` is the configuration type and `channelname` is the name of the channel from which the information is to be retrieved.

- To retrieve information about the options available for monitoring channels:

```
sagcc get administration component sag01 Universal-Messaging-umserver channels
```

`sag01` is the alias name of the installation where `Universal-Messaging-umserver` server instance is running.

- To retrieve the list of channels on a Universal Messaging server instance:

```
sagcc get administration component sag01 Universal-Messaging-umserver channels
list
```

`sag01` is the alias name of the installation where `Universal-Messaging-umserver` server instance is running. `list` is the command to list all the channels in a server instance.

- To retrieve the status of a specific channel in TSV format:

```
sagcc get administration component sag01 Universal-Messaging-umserver channels
status name=channel_name
```

`sag01` is the alias name of the installation where `Universal-Messaging-umserver` server instance is running.`channel_name` is the channel name for which the status is to be retrieved.

- To retrieve a list of durable subscribers on a specific channel in TSV format:

```
sagcc get administration component sag01 Universal-Messaging-umserver channels
durablesubscribers name=channel_name
```

`sag01` is the alias name of the installation where `Universal-Messaging-umserver` server instance is running. `channel_name` is the name of the channel from which the durable subscribers list is to be retrieved.

- To delete a durable subscriber subscribed to a channel:

```
sagcc exec administration component sag01 Universal_Messaging-umserver channels
deletedurablesubscriber name=channel_name durablesubscriber=durable_subscriber_name
```

`sag01` is the alias name of the installation where `Universal-Messaging-umserver` server instance is running. `channel_name` is the name of the channel, and `durable_subscriber_name` is the name of the durable subscriber that has to be deleted from this channel.

Durable Subscribers Monitoring Commands

You can monitor durable subscribers using the following commands:

- View the list of administration namespaces using the `cc get administration component` *`node_alias`* `Universal-Messaging-`*`umserver`* command.
- Retrieve information about the options available for monitoring durable subscribers using the`cc get administration component` *`node_alias`* `Universal-Messaging-`*`umserver`* `durablesubscribers` command.

- Retrieve the list of durable subscribers of an Universal Messaging server instance using:

  `sagcc get administration component` *`node_alias`* `Universal-Messaging-`*`umserver`* `durablesubscribers list -f xml` command for XML format.

  `sagcc get administration component` *`node_alias`* `Universal-Messaging-`*`umserver`* `durablesubscribers list -f tsv` command for TSV format.

  `sagcc get administration component` *`node_alias`* `Universal-Messaging-`*`umserver`* `durablesubscribers list -f csv` command for CSV format.

- Retrieve the attributes of a specific durable subscriber using:

  `sagcc get administration component` *`node_alias`* `Universal-Messaging-`*`umserver`* `durablesubscribers details channel=`*`channel_name`* `name=`*`durable subscriber name`* `-f xml` command for XML format.

  `sagcc get administration component` *`node_alias`* `Universal-Messaging-`*`umserver`* `durablesubscribers details channel=`*`channel_name`* `name=`*`durable subscriber name`* `-f TSV` command for TSV format.

  `sagcc get administration component` *`node_alias`* `Universal-Messaging-`*`umserver`* `durablesubscribers details channel=`*`channel_name`* `name=`*`durable subscriber name`* `-f CSV` command for CSV format.

- Retrieve specific attributes of the durable subscribers using the `cc get administration component` *`node_alias`* `Universal-Messaging-`*`umserver`* `durablesubscribers list -f` *`tsv or csv`* `properties=`*`comma separated attribute list`* command.

- Retrieve the list of events for a durable subscriber using the `cc get administration component` *`node_alias`* `Universal-Messaging-`*`umserver`* `durablesubscribers getDurableEvents durableName=`*`durable name`* `chanName=`*`channel name`*

- Purge a specified range of events using the `cc get administration component` *`node_alias`* `Universal-Messaging-`*`umserver`* `durablesubscribers purgeStartEndID startEID=`*`start event ID`* `endEID=`*`end event ID`* `durableName=`*`durable subscriber name`* `chanName=`*`channel_name`*

- Purge all events using the `cc get administration component` *`node_alias`* `Universal-Messaging-`*`umserver`* `durablesubscribers purgeAll durableName=`*`durable subscriber name`* `chanName=`*`channel_name`* `command.`

- Purge events by filtering events using the `cc get administration component` *`node_alias`* `Universal-Messaging-`*`umserver`* `durablesubscribers purgeFilter durableName=`*`durable subscriber name`* `chanName=`*`channel_name`* `filter=`*`the filter expression`* `command.`

- Delete a durable subscriber using the `sagcc exec administration component` *`node_alias`* `Universal-Messaging-`*`umserver`* `durablesubscribers delete channel=`*`channel_name`* `name=`*`durable subscriber name`*

Examples

- To list all the administration namespaces:

```
sagcc get administration component sag01 Universal-Messaging-umserver
```

`sag01` is the alias name of the installation where `Universal-Messaging-umserver` server instance is running.

- To retrieve information about the options available for monitoring durable subscribers:

```
sagcc get administration component sag01 Universal-Messaging-umserver
durablesubscribers
```

`sag01` is the alias name of the installation where `Universal-Messaging-umserver` server instance is running.

- To retrieve the list of durable subscribers in a Universal Messaging server instance:

  XML format

```
sagcc get administration component sag01 Universal-Messaging-umserver
durablesubscribers list -f xml
```

  TSV format

```
sagcc get administration component sag01 Universal-Messaging-umserver
durablesubscribers list -f tsv
```

  CSV format

```
sagcc get administration component sag01 Universal-Messaging-umserver
durablesubscribers list -f csv
```

- To retrieve attributes of the durable subscribers:

  XML format

```
sagcc get administration component sag01 Universal-Messaging-umserver
durablesubscribers details channel=channelname name=durable_subscriber_name
-f xml
```

  TSV format

```
sagcc get administration component sag01 Universal-Messaging-umserver
durablesubscribers details channel=channelname name=durable_subscriber_name
-f tsv
```

  CSV format

```
sagcc get administration component sag01 Universal-Messaging-umserver
durablesubscribers details channel=channelname name=durable_subscriber_name
-f csv
```

  `sag01` is the alias name of the installation where `Universal-Messaging-umserver` server instance is running. `channelname` is the name of the channel, and `durable_subscriber_name` is the name of the durable subscriber from which the attributes are to be retrieved.

- To retrieve the attributes specific attributes of a durable subscriber:

```
sagcc get administration component sag01 Universal-Messaging-umserver
durablesubscribers list -f tsv properties=name,channel,lastEventID,
```

```
outStandingEvents
```

sag01 is the alias name of the installation where Universal-Messaging-umserver server instance is running. name, channel, lastEventID, and outStandingEvents are attribute values to be retrieved.

- To retrieve the list of events for a durable subscriber:

```
sagcc exec administration component sag01 Universal-Messaging-umserver
durablesubscribers getDurableEvents durableName=durablename
chanName=channelname
```

sag01 is the alias name of the installation where Universal-Messaging-umserver server instance is running. durablename is the name of the durable subscriber, and channelname is the name of the channel on which the durable subscriber exists.

- To purge a specified range of events:

```
sagcc exec administration component sag01 Universal-Messaging-umserver
durablesubscribers purgeStartEndID startEID=10 endEID=20
durableName=durablename chanName=channelname
```

sag01 is the alias name of the installation where Universal-Messaging-umserver server instance is running. startEID is the the start event ID for the range, endEID is the end event ID of the range, durablename is the name of the durable subscriber, and channelname is the name of the channel on which the durable subscriber exists.

- To purge all the events for a durable subscriber:

```
sagcc exec administration component sag01 Universal-Messaging-umserver
durablesubscribers purgeAll durableName=durablename chanName=channelname
```

sag01 is the alias name of the installation where Universal-Messaging-umserver server instance is running. durablename is the name of the durable subscriber, and channelname is the name of the channel on which the durable subscriber exists.

- To purge events by filtering events:

```
sagcc exec administration component sag01 Universal-Messaging-umserver
durablesubscribers purgeFilter durableName=durablename chanName=channelname
filter=size BETWEEN 10.0 AND 12.0
```

sag01 is the alias name of the installation where Universal-Messaging-umserver server instance is running. durablename is the name of the durable subscriber, and channelname is the name of the channel on which the durable subscriber exists.

- To delete a durable subscriber:

```
sagcc exec administration component sag01 Universal-Messaging-umserver
durablesubscribers delete channel=channelname name=DS3
```

sag01 is the alias name of the installation where Universal-Messaging-umserver server instance is running. channelname is the name of the channel on which the durable subscriber exists, and durablename is the name of the durable subscriber.

Commands for Snooping on Channels

Important:
The snoop commands work with Command Central 10.4 and higher.

You use the `sagcc exec administration component` command to snoop on events on a Universal Messaging channel. All snooping operations are performed for a particular user. You can do the following:

- Start snooping on events on a channel for a specific user.
- List the snooped events on a channel for a specific user.
- View details of a snooped event on a channel for a specific user.
- Purge snooped events on a channel for a specific user.
- Stop snooping on events on a channel for a specific user.

Start Snooping on a Channel

- To start snooping on events on a channel for a specific user:

```
sagcc exec administration component node_alias
Universal-Messaging-server_instance_name
channels startSnoop name=channel_name user=user_name
```

- To start snooping on events on a channel for a specific user with filtering criteria:

```
sagcc exec administration component node_alias
Universal-Messaging-server_instance_name
channels startSnoop name=channel_name user=user_name [fromeid=first_event_id]
[toeid=last_event_id] [filter=filter_string]
```

Arguments and Options

Argument or Option	Value
`Universal-Messaging-server_instance_name`	Required. The ID of the Universal Messaging server instance on which you want to snoop.
`name=channel_name`	Required. The name of the channel on which you want to snoop.
`user=user_name`	Required. The username of the user for whom you want to start snooping.
`[fromeid=first_event_id]`	Optional. The ID of the first event in the event range on which you want to start snooping.
`[toeid=last_event_id]`	Optional. The ID of the last event in the event range on which you want to start snooping.

Argument or Option	Value
`[filter=filter_string]`	Optional. Additional filtering criteria based on the properties of the event.

Usage Notes

When you want to start snooping on a range of events:

- If you do not specify `fromeid`, the range of events starts with the first event on the channel and ends with the event specified for `toeid`.
- If you do not specify `toeid`, the range of events starts with the event specified for `fromeid` and ends with the last event on the channel.

Examples

- To start snooping on events on channel "channel2", created on the server instance with ID "Universal-Messaging-umserver" that is installed in the installation with alias name "sag01", for user "Administrator":

```
sagcc exec administration component sag01 Universal-Messaging-umserver
channels startSnoop name=channel2 user=Administrator
```

- To start snooping on the events with IDs from "2" to "10" on channel "channel2", created on the server instance with ID "Universal-Messaging-umserver" that is installed in the installation with alias name "sag01", for user "Administrator":

```
sagcc exec administration component sag01 Universal-Messaging-umserver
channels startSnoop name=channel2 user=Administrator fromid=2 toid=10
```

List Snooped Events on a Channel

- To list the snooped events on a channel for a specific user in TSV format:

```
sagcc exec administration component node_alias
Universal-Messaging-server_instance_name
channels snoop name=channel_name user=user_name
```

- To list the snooped events on a channel for a specific user in XML format:

```
sagcc exec administration component node_alias
Universal-Messaging-server_instance_name
channels snoop name=channel_name user=user_name -f|--format xml
```

- To list the snooped events on a channel for a specific user in JSON format:

```
sagcc exec administration component node_alias
Universal-Messaging-server_instance_name
channels snoop name=channel_name user=user_name -f|--format json
```

Arguments and Options

Argument or Option	Value
`Universal-Messaging-`*`server_instance_name`*	Required. The ID of the Universal Messaging server instance for which you want to list snooped events.
`name=`*`channel_name`*	Required. The name of the channel for which you want to list snooped events.
`user=`*`user_name`*	Required. The username of the user for whom you want to list snooped events.
`[-f|--format {xml|json}]`	Optional. Whether to list the snooped events in XML or JSON format.

Usage Notes

If the Universal Messaging server becomes unavailable after the snoop on the channel started, Command Central stops snooping all channels on that Universal Messaging server for all Command Central users. If you run the command that lists snooped events while the Universal Messaging server is unavailable, the system returns "`snoopStarted=false`" and an empty list of events.

Examples

- To list the snooped events in TSV format on channel "channel2", created on the server instance with ID "Universal-Messaging-umserver" that is installed in the installation with alias name "sag01", for user "Administrator":

```
sagcc exec administration component sag01 Universal-Messaging-umserver
channels snoop name=channel2 user=Administrator
```

- To list the snooped events in JSON format on channel "channel2", created on the server instance with ID "Universal-Messaging-umserver" that is installed in the installation with alias name "sag01", for user "Administrator":

```
sagcc exec administration component sag01 Universal-Messaging-umserver
channels snoop name=channel2 user=Administrator --format json
```

View Details of a Snooped Event on a Channel

- To retrieve the details of a snooped event on a channel for a specific user in TSV format:

```
sagcc exec administration component node_alias
Universal-Messaging-server_instance_name
channels event name=channel_name user=user_name id=event_id
```

- To retrieve the details of a snooped event on a channel for a specific user in XML format:

```
sagcc exec administration component node_alias
Universal-Messaging-server_instance_name
```

```
channels event name=channel_name user=user_name id=event_id -f|--format xml
```

- To retrieve the details of a snooped event on a channel for a specific user in JSON format:

```
sagcc exec administration component node_alias
Universal-Messaging-server_instance_name
channels event name=channel_name user=user_name id=event_id -f|--format json
```

Arguments and Options

Argument or Option	Value
Universal-Messaging-*server_instance_name*	Required. The ID of the Universal Messaging server instance.
name=*channel_name*	Required. The name of the channel on which the snooped event is published.
user=*user_name*	Required. The username of the user for whom you want to retrieve the snooped event.
id=*event_id*	Required. The ID of the event that you want to view.
[-f|--format {xml|json}]	Optional. Whether to view the snooped event in XML or JSON format.

Usage Notes

The TSV format is tabular and does not display the header and properties of an event. To see the header and properties of an event, use the XML or JSON format.

Examples

- To view details of a snooped event with ID "2" in TSV format on channel "channel2", created on the server instance with ID "Universal-Messaging-umserver" that is installed in the installation with alias name "sag01", for user "Administrator":

```
sagcc exec administration component sag01 Universal-Messaging-umserver
channels event name=channel2 user=Administrator id=2
```

- To view details of a snooped event with ID "2" in XML format on channel "channel2", created on the server instance with ID "Universal-Messaging-umserver" that is installed in the installation with alias name "sag01", for user "Administrator":

```
sagcc exec administration component sag01 Universal-Messaging-umserver
channels event name=channel2 user=Administrator id=2 --format xml
```

Purge Snooped Events on a Channel

- To purge a snooped event from a channel and the snooped events list for a specific user:

```
sagcc exec administration component node_alias
Universal-Messaging-server_instance_name
channels purgeEvent name=channel_name user=user_name id=event_id
```

- To purge a range of snooped events from a channel and the snooped events list for a specific user:

```
sagcc exec administration component node_alias
Universal-Messaging-server_instance_name
channels purgeEvents name=channel_name user=user_name
[fromeid=first_event_id] [toeid=last_event_id]
```

Arguments and Options

Argument or Option	Value
`Universal-Messaging-`*`server_instance_name`*	Required. The ID of the Universal Messaging server instance on which you want to purge snooped events.
`name=`*`channel_name`*	Required. The name of the channel on which you want to purge events.
`user=`*`user_name`*	Required. The username of the user for whom you want to purge event.
`id=`*`event_id`*	Required with the `sagcc exec administration component purgeEvent` command. The ID of the event to purge.
`[fromeid=`*`first_event_id`*`]`	Optional. The ID of the first event in the event range that you want to purge.
`[toeid=`*`last_event_id`*`]`	Optional. The ID of the last event in the event range that you want to purge.

Usage Notes

When you want to purge a range of snooped events:

- If you do not specify `fromeid`, all events from the first one on the channel to the one with ID smaller than or equal to `toeid` are purged.
- If you do not specify `toeid`, all events from the one with ID greater than or equal to `fromeid` to the last one on the channel are purged.
- If you do not specify both `fromeid` and `toeid`, all events on the channel are purged.

Examples

- To purge a snooped event with ID "2" from channel "channel2", created on the server instance with ID "Universal-Messaging-umserver" that is installed in the installation with alias name "sag01", for user "Administrator":

```
sagcc exec administration component sag01 Universal-Messaging-umserver
channels purgeEvent name=channel2 user=Administrator id=2
```

- To purge the snooped events with IDs from "2" to "6" from channel "channel2", created on the server instance with ID "Universal-Messaging-umserver" that is installed in the installation with alias name "sag01", for user "Administrator":

```
sagcc exec administration component sag01 Universal-Messaging-umserver
channels purgeEvents name=channel2 user=Administrator fromid=2 toid=6
```

Stop Snooping on a Channel

To stop snooping on events on a channel for a specific user:

```
sagcc exec administration component node_alias Universal-Messaging-server_instance_name
channels stopSnoop name=channel_name user=user_name
```

Arguments and Options

Argument or Option	Value
`Universal-Messaging-server_instance_name`	Required. The ID of the Universal Messaging server instance on which you want to stop snooping.
`name=channel_name`	Required. The name of the channel on which you want to stop snooping.
`user=user_name`	Required. The username of the user for whom you want to stop snooping.

Examples

To stop snooping for events on channel "channel2", created on the server instance with ID "Universal-Messaging-umserver" that is installed in the installation with alias name "sag01", for user "Administrator":

```
sagcc exec administration component sag01 Universal-Messaging-umserver
channels stopSnoop name=channel2 user=Administrator
```

Commands for Snooping on Queues

Important:
The snoop commands work with Command Central 10.4 and higher.

You use the `sagcc exec administration component` command to snoop on events on a Universal Messaging queue. All snooping operations are performed for a particular user. You can do the following:

- Start snooping on events on a queue for a specific user.
- List the snooped events on a queue for a specific user.

- View details of a snooped event on a queue for a specific user.
- Purge all snooped events on a queue for a specific user.
- Stop snooping on events on a queue for a specific user.

Start Snooping on a Queue

- To start snooping on events on a queue for a specific user:

```
sagcc exec administration component node_alias
Universal-Messaging-server_instance_name
queues startSnoop name=queue_name user=user_name
```

- To start snooping on events on a queue for a specific user with filtering criteria:

```
sagcc exec administration component node_alias
Universal-Messaging-server_instance_name
queues startSnoop name=queue_name user=user_name [fromeid=first_event_id]
[toeid=last_event_id] [filter=filter_string]
```

Arguments and Options

Argument or Option	Value
`Universal-Messaging-`*`server_instance_name`*	Required. The ID of the Universal Messaging server instance on which you want to snoop.
`name=`*`queue_name`*	Required. The name of the queue on which you want to snoop.
`user=`*`user_name`*	Required. The username of the user for whom you want to start snooping.
`[fromeid=`*`first_event_id`*`]`	Optional. The ID of the first event in the event range on which you want to start snooping.
`[toeid=`*`last_event_id`*`]`	Optional. The ID of the last event in the event range on which you want to start snooping.
`[filter=`*`filter_string`*`]`	Optional. Additional filtering criteria based on the properties of the event.

Usage Notes

When you want to start snooping on a range of events:

- If you do not specify `fromeid`, the range of events starts with the first event on the queue and ends with the event specified for `toeid`.
- If you do not specify `toeid`, the range of events starts with the event specified for `fromeid` and ends with the last event on the queue.

Examples

- To start snooping on events on queue "queue1", created on the server instance with ID "Universal-Messaging-umserver" that is installed in the installation with alias name "sag01", for user "Administrator":

```
sagcc exec administration component sag01 Universal-Messaging-umserver
queues startSnoop name=queue1 user=Administrator
```

- To start snooping on the events with IDs from "2" to "10" on queue "queue1", created on the server instance with ID "Universal-Messaging-umserver" that is installed in the installation with alias name "sag01", for user "Administrator":

```
sagcc exec administration component sag01 Universal-Messaging-umserver
queues startSnoop name=queue1 user=Administrator fromid=2 toid=10
```

List Snooped Events on a Queue

- To list the snooped events on a queue for a specific user in TSV format:

```
sagcc exec administration component node_alias
Universal-Messaging-server_instance_name
queues snoop name=queue_name user=user_name
```

- To list the snooped events on a queue for a specific user in XML format:

```
sagcc exec administration component node_alias
Universal-Messaging-server_instance_name
queues snoop name=queue_name user=user_name -f|--format xml
```

- To list the snooped events on a queue for a specific user in JSON format:

```
sagcc exec administration component node_alias
Universal-Messaging-server_instance_name
queues snoop name=queue_name user=user_name -f|--format json
```

Arguments and Options

Argument or Option	Value
`Universal-Messaging-`*`server_instance_name`*	Required. The ID of the Universal Messaging server instance for which you want to list snooped events.
`name=`*`queue_name`*	Required. The name of the queue for which you want to list snooped events.
`user=`*`user_name`*	Required. The username of the user for whom you want to list snooped events.
`[-f|--format {xml|json}]`	Optional. Whether to list the snooped events in XML or JSON format.

Usage Notes

If the Universal Messaging server becomes unavailable after the snoop on the queue started, Command Central stops snooping on all queues on that Universal Messaging server for all Command Central users. If you run the command that lists snooped events while the Universal Messaging server is unavailable, the system returns "`snoopStarted=false`" and an empty list of events.

Examples

- To list the snooped events in TSV format on queue "queue1", created on the server instance with ID "Universal-Messaging-umserver" that is installed in the installation with alias name "sag01", for user "Administrator":

```
sagcc exec administration component sag01 Universal-Messaging-umserver
queues snoop name=queue1 user=Administrator
```

- To list the snooped events in JSON format on queue "queue1", created on the server instance with ID "Universal-Messaging-umserver" that is installed in the installation with alias name "sag01", for user "Administrator":

```
sagcc exec administration component sag01 Universal-Messaging-umserver
queues snoop name=queue1 user=Administrator --format json
```

View Details of a Snooped Event on a Queue

- To retrieve the details of a snooped event on a queue for a specific user in TSV format:

```
sagcc exec administration component node_alias
Universal-Messaging-server_instance_name
queues event name=queue_name user=user_name id=event_id
```

- To retrieve the details of a snooped event on a queue for a specific user in XML format:

```
sagcc exec administration component node_alias
Universal-Messaging-server_instance_name
queues event name=queue_name user=user_name id=event_id -f|--format xml
```

- To retrieve the details of a snooped event on a queue for a specific user in JSON format:

```
sagcc exec administration component node_alias
Universal-Messaging-server_instance_name
queues event name=queue_name user=user_name id=event_id -f|--format json
```

Arguments and Options

Argument or Option	Value
`Universal-Messaging-`*`server_instance_name`*	Required. The ID of the Universal Messaging server instance.

Argument or Option	Value
`name=queue_name`	Required. The name of the queue on which the snooped event is published.
`user=user_name`	Required. The username of the user for whom you want to retrieve the snooped event.
`id=event_id`	Required. The ID of the event that you want to view.
`[-f|--format {xml|json}]`	Optional. Whether to view the snooped event in XML or JSON format.

Usage Notes

The TSV format is tabular and does not display the header and properties of an event. To see the header and properties of an event, use the XML or JSON format.

Examples

- To view details of a snooped event with ID "2" in TSV format on queue "queue1", created on the server instance with ID "Universal-Messaging-umserver" that is installed in the installation with alias name "sag01", for user "Administrator":

```
sagcc exec administration component sag01 Universal-Messaging-umserver
queues event name=queue1 user=Administrator id=2
```

- To view details of a snooped event with ID "2" in XML format on queue "queue1", created on the server instance with ID "Universal-Messaging-umserver" that is installed in the installation with alias name "sag01", for user "Administrator":

```
sagcc exec administration component sag01 Universal-Messaging-umserver
queues event name=queue1 user=Administrator id=2 --format xml
```

Purge Snooped Events on a Queue

To purge all snooped events from a queue and the snooped events list for a specific user:

```
sagcc exec administration component node_alias Universal-Messaging-server_instance_name
queues purgeEvents name=queue_name user=user_name
```

Arguments and Options

Argument or Option	Value
`Universal-Messaging-server_instance_name`	Required. The ID of the Universal Messaging server instance on which you want to purge snooped events.
`name=queue_name`	Required. The name of the queue on which you want to purge events.

Argument or Option	Value
`user=user_name`	Required. The username of the user for whom you want to purge event.

Usage Notes

You cannot purge a single event or a range of events from a queue.

Examples

To purge all snooped events from queue "queue1", created on the server instance with ID "Universal-Messaging-umserver" that is installed in the installation with alias name "sag01", for user "Administrator":

```
sagcc exec administration component sag01 Universal-Messaging-umserver
queues purgeEvents name=queue1 user=Administrator
```

Stop Snooping on a Queue

To stop snooping on events on a queue for a specific user:

```
sagcc exec administration component node_alias Universal-Messaging-server_instance_name
queues stopSnoop name=queue_name user=user_name
```

Arguments and Options

Argument or Option	Value
`Universal-Messaging-server_instance_name`	Required. The ID of the Universal Messaging server instance on which you want to stop snooping.
`name=queue_name`	Required. The name of the queue on which you want to stop snooping.
`user=user_name`	Required. The username of the user for whom you want to stop snooping.

Examples

To stop snooping for events on queue "queue1", created on the server instance with ID "Universal-Messaging-umserver" that is installed in the installation with alias name "sag01", for user "Administrator":

```
sagcc exec administration component sag01 Universal-Messaging-umserver
queues stopSnoop name=queue1 user=Administrator
```

Event Publishing Commands

Important:
The event publishing commands work with Command Central 10.4 and higher.

You use the `cc exec administration component` command to publish and republish events on a Universal Messaging channel or queue.

- To publish an event on a channel:

```
cc exec administration component node_alias Universal-Messaging-server_instance_name
channels publish name=channel_name data=event_content [tag=event_tag] [ttl=ttl_value]
[persistent={true|false}] [transient={true|false}] [properties=properties_string]

[pubcount=publish_count] [sendasprotobuf={true|false}]
[protobufdescriptor=protobuf_descriptor]
```

- To republish a snooped event on a channel:

```
cc exec administration component node_alias Universal-Messaging-server_instance_name
channels publish name=channel_name data=event_content id=original_event_id
user=user_name republish=true [tag=event_tag] [ttl=ttl_value]
[persistent={true|false}] [transient={true|false}] [properties=properties_string]

[pubcount=publish_count] [sendasprotobuf={true|false}]
[protobufdescriptor=protobuf_descriptor] [purgeoriginal={true|false}]
```

- To publish an event on a queue:

```
cc exec administration component node_alias Universal-Messaging-server_instance_name
queues publish name=queue_name data=event_content [tag=event_tag] [ttl=ttl_value]
[persistent={true|false}] [transient={true|false}] [properties=properties_string]

[pubcount=publish_count] [sendasprotobuf={true|false}]
[protobufdescriptor=protobuf_descriptor]
```

- To republish a snooped event on a queue:

```
cc exec administration component node_alias Universal-Messaging-server_instance_name
queues publish name=queue_name data=event_content republish=true [tag=event_tag]
[ttl=ttl_value] [persistent={true|false}] [transient={true|false}]
[properties=properties_string] [pubcount=pub_count]
[sendasprotobuf=[{true|false}] [protobufdescriptor=protobuf_descriptor]
```

Arguments and Options

Argument or Option	Value
`Universal-Messaging-server_instance_name`	Required. The ID of the Universal Messaging server instance on which you want to publish or republish an event.
`name=channel_or_queue_name`	Required. The name of the channel or queue on which you want to publish or republish an event.

Argument or Option	Value
`data=event_content`	Required. The content of the event.
`[tag=event_tag]`	Optional. The tag of the event.
`[ttl=ttl_value]`	Optional. The time-to-live (TTL) of the event in milliseconds. Defines how long the event remains available on the channel or queue. If you specify a TTL of `0`, the event remains on the channel or queue indefinitely.
`[persistent={true|false}]`	Optional. Whether the event is persistent. Values are: ■ `true` ■ `false` (default)
`[transient={true|false}]`	Optional. Whether the event is transient. Values are: ■ `true` ■ `false` (default)
`id=original_event_id`	Required when republishing an event on a channel. The ID of the original event.
`user=user_name`	Required when republishing an event on a channel. The user name of the user who started snooping on the event on the channel.
`republish={true|false}`	Required when republishing an event on a channel or queue. Whether to republish the snooped event. Values are: ■ `true` ■ `false` (default)
`[purgeoriginal={true|false}]`	Optional. Whether to purge the original event from the channel. Values are: ■ `true` ■ `false` (default)
`[properties=properties_string]`	Optional. A string that contains event properties in the following JSON format: `[ { name: "property1_name", type: "property1_type", value: property1_value }, { name: "property2_name", type: "property2_type", value: property2_value }, … ]`

Argument or Option	Value
	where ■ `propertyX_name` is the name of the property. ■ `propertyX_type` is the type of the property and can have one of the following values: `int`, `byte`, `long`, `short`, `float`, `double`, `boolean`, `char`, `string`, `int[]`, `byte[]`, `long[]`, `short[]`, `float[]`, `double[]`, `boolean[]`, `char[]`, and `string[]`. ■ `propertyX_value` is the value of the property.
`[pubcount=pub_count]`	Optional. The number of times to republish an event. If you omit `pubcount`, the option defaults to `1`.
`[sendasprotobuf={true|false}]`	Optional. Whether to convert the event content in the `data` option to a Protobuf event that matches a Protobuf schema specified in the `protobufdescriptor` option and already uploaded on the channel or queue. Values are: ■ `true` ■ `false` (default) For more information about working with Protobuf events, see the "Google Protocol Buffers" section in the *Universal Messaging Concepts* guide.
`[protobufdescriptor=` `protobuf_descriptor]`	Required only for Protobuf events. The Protobuf file descriptor that defines the message schema to be used for converting the event content in the `data` option to a Protobuf event. For more information about working with Protobuf events, see the "Google Protocol Buffers" section in the *Universal Messaging Concepts* guide.

Usage Notes

- When you use the `properties` option, consider the following information:
  - Include the `-f json` option to specify the format of the properties string.
  - Enclose char and string property values, and values with spaces in double quotes ("). If the value contains double quotes, replace them with a backslash and double quotes (\").
  - For array values, specify a valid JSON array of the corresponding type.
- When you want to publish a Protobuf event, consider the following information:

- Before publishing a Protobuf event, you must upload on the channel or queue the Protobuf file descriptor that defines the Protobuf schema, as part of a file descriptor set. For information about uploading a Protobuf file descriptor set on a channel or queue, see "Channel Configuration" on page 377 or "Queue Configuration" on page 381.
- The value of the `data` option is a JSON string that represents the Protobuf event.
- The value of the `protobufdescriptor` option is the name of the Protobuf file descriptor that defines the message schema.

- Before republishing a snooped event on a channel or queue, you must start snooping on the channel or queue and obtain the event ID.

Examples

- To publish an event with event data "CustomerOrders" and event tag "COrders" on channel "channel2" that is created on the server instance with ID "Universal-Messaging-umserver", installed in the installation with alias name "sag01":

```
cc exec administration component sag01 Universal-Messaging-umserver channels
publish name=channel2 data=CustomerOrders tag=COrders
```

- To publish a persistent event with event data "CustomerOrders", event tag "COrders", and TTL "10000" three times on queue "queue1" that is created on the server instance with ID "Universal-Messaging-umserver", installed in the installation with alias name "sag01":

```
cc exec administration component sag01 Universal-Messaging-umserver queues
publish name=queue1 data=CustomerOrders tag=COrders ttl=10000
persistent=true pubcount=3
```

- To publish an event with event data "CustomerOrders", event tag "COrders", and custom properties in JSON format on queue "queue1" that is created on the server instance with ID "Universal-Messaging-umserver", installed in the installation with alias name "sag01":

```
cc exec administration component sag01 Universal-Messaging-umserver queues
publish name=queue1 data=CustomerOrders tag=COrders
properties="[ { name: \"orderNumber\", type: \"string\", value: \"F18LP\" },
{ name: \"items\", type: \"int\", value: 3 },
{ name: \"itemIds\", type:\"int[]\", value: [ 509, 19, 100 ] } ]" -f json
```

- To publish a Protobuf event on channel "channel2" that is created on the server instance with ID "Universal-Messaging-umserver", installed in the installation with alias name "sag01":

```
cc exec administration component sag01 Universal-Messaging-umserver channels
publish name=channel2
data="{ header: { id: 1, time: 1541163198345 },
contents: { intData: 124, doubleData: 3.141593, stringData: \"Software AG\" } }"
protobufdescriptor=Event sendasprotobuf=true
```

  The example assumes that a Protobuf file descriptor set containing the following descriptors has already been uploaded on "channel2". The Protobuf event in the example is created from the `"Event"` message type in the file descriptor set. The event data is a valid JSON string that represents a message of type `"Event"` and has the same fields as the `"Event"` message type.

```
file {
```

```
  name: "EventWrapper"
  package: "um"
  dependency: "HeaderWrapper"
  dependency: "ContentsWrapper"
  message_type {
    name: "Event"
    field {
      name: "header"
      number: 1
      label: LABEL_REQUIRED
      type: TYPE_MESSAGE
      type_name: "Header"
    }
    field {
      name: "contents"
      number: 2
      label: LABEL_REQUIRED
      type: TYPE_MESSAGE
      type_name: "Contents"
    }
  }
}
file {
  name: "HeaderWrapper"
  package: "um"
  message_type {
    name: "Header"
    field {
      name: "id"
      number: 1
      label: LABEL_REQUIRED
      type: TYPE_INT32
    }
    field {
      name: "time"
      number: 2
      label: LABEL_REQUIRED
      type: TYPE_SINT64
    }
  }
}
file {
  name: "ContentsWrapper"
  package: "um"
  message_type {
    name: "Contents"
    field {
      name: "intData"
      number: 1
      label: LABEL_REQUIRED
      type: TYPE_SINT32
    }
    field {
      name: "doubleData"
      number: 4
      label: LABEL_REQUIRED
      type: TYPE_DOUBLE
    }
    field {
      name: "stringData"
```

```
      number: 5
      label: LABEL_REQUIRED
      type: TYPE_STRING
    }
  }
}
```

- To republish an event with ID "3", modified event data "CancelledCustomerOrders", and modified event tag "CCOrders", snooped by user "Administrator", on channel "channel2" that is created on the server instance with ID "Universal-Messaging-umserver", installed in the installation with alias name "sag01":

```
cc exec administration component sag01 Universal-Messaging-umserver channels
publish name=channel2 data=CnacelledCustomerOrders tag=CCOrders
republish=true id=3 user=Administrator
```

Server Inventory Commands

`sagcc get inventory components` and `sagcc list inventory components` gets and lists Universal Messaging inventory:

The commands retrieve information about the Universal Messaging server instances configured in the <*InstallDir*>/UniversalMessaging/<*InstanceName*> directory in an installation. Information from all the folders under the server directory, except templates, is displayed.

Property	Value
Display name	Universal-Messaging-*ServerInstanceName*
Run-time component ID	Universal-Messaging-*ServerInstanceName*
Product ID	NUMRealmServer
Run-time component category	PROCESS

Note:
ServerInstanceName can include upper and lower case alphabetic characters, digits (0-9), and underscores (_) but cannot include hyphens (-), periods (.), and colons (:).

Server Instance Migration Commands

Ensure that the target Universal Messaging server has the migration utility installed. For information about the migration utility, see *Upgrading Software AG Products*.

Important:
You must run the commands in the context and order documented in *Upgrading Software AG Products*. Otherwise, you may experience unpredictable results, possibly including corruption of your installation and data.

- View the command line help for the migration utility using the `sagcc list administration product node_alias NUMRealmServer migration help` command.

- Migrate all Universal Messaging server instances present in a source installation using the `sagcc exec administration product node_alias NUMRealmServer migration migrate srcDir=SAG_Installation_directory` command.

  Note:
  Use this command when migrating from Universal Messaging server version 9.8 and later.

- Start migration by providing the source Universal Messaging instance name using the `sagcc exec administration product node_alias NUMRealmServer migration migrate srcDir=SAG_Installation_directory instanceName=instance_name[,instance_name,instance_name...]` command .

  Note:
  Use this command when migrating from Universal Messaging server version 9.8 and later.

- Start migration by passing arguments and using the migrate.dat file using the `sagcc exec administration product node_alias NUMRealmServer migration migrate srcDir=SAG_Installation_directory importFile=migrate.dat` command.

  Note:
  Use this command when migrating from Universal Messaging server 9.0 through 9.7. The argument `silent` is set to `true` and `continueOnError` is set to `false` by default.

- Start migration using the zip file from the old product installation using the `sagcc exec administration product node_alias NUMRealmServer migration migrate srcFile=old_installation.zip importFile=migrate.dat` command.

- Start migration using the zip file from the old product installation and specifying the source Universal Messaging instance name using the `sagcc exec administration product node_alias NUMRealmServer migration migrate srcFile=old_installation.zip instanceName=instance_name[,instance_name,instance_name...]` command.

- View APIs under the migration namespace using the `sagcc list administration product node_alias NUMRealmServer migration` command.

- View if Universal Messaging supports migration as a custom API using the `sagcc list administration product node_alias NUMRealmServer` command.

Examples

- To view the command line help:

```
sagcc list administration product sag01 NUMRealmServer migration help
```

  where `sag01` is the alias name of the installation where `NUMRealmServer` server instance is running. `migration` is the namespace for the custom Command Central API. `Help` is the command to view the migration tool's command line help.

- To migrate all Universal Messaging server instances present in a source installation:

```
sagcc exec administration product sag01 NUMRealmServer
migration migrate srcDir=C:\SoftwareAG
```

where `sag01` is the alias name of the installation where `NUMRealmServer` server instance is running. `migration` is the namespace for the custom Command Central API. `migrate` is the command to access the migration tool. `scrDir` is the source Software AG installation directory.

- To start migrating an older server instance to a new `NUMRealmServer` server instance by providing the source instance name:

```
sagcc exec administration product sag01 NUMRealmServer
migration migrate srcDir=C:\SoftwareAG
instanceName=umserver1,umserver2
```

where `sag01` is the alias name of the installation where `NUMRealmServer` server instance is running. `migration` is the namespace for the custom Command Central API. `migrate` is the command to access the migration tool. `scrDir` is the source Software AG installation directory. `instanceName` contains the comma-separated names of the Universal Messaging instances that will be migrated.

- To start migrating an older server instance to a new `NUMRealmServer` server instance using the migrate.dat file:

```
sagcc exec administration product sag01 NUMRealmServer
migration migrate srcDir=C:\SoftwareAG
silent=true importFile=migrate.dat
```

where `sag01` is the alias name of the installation where `NUMRealmServer` server instance is running. `migration` is the namespace for the custom Command Central API. `migrate` is the command to access the migration tool. `scrDir` is the source Software AG installation directory. `importFile` specifies the data file containing the migration settings.

- To start migration using the zip file from the old product installation:

```
sagcc exec administration product sag01 NUMRealmServer
migration migrate srcFile=99Src.zip importFile=migrate.dat
```

where `sag01` is the alias name of the installation where `NUMRealmServer` server instance is running. `migration` is the namespace for the custom Command Central API. `migrate` is the command to access the migration tool. `scrFile` argument is used to provide the name of the zip file from the source Universal Messaging instance. `importFile` specifies the archive file path containing migration settings.

- To view APIs under the migration namespace for the `NUMRealmServer` server instance:

```
sagcc list administration product sag01 NUMRealmServer migration
```

where `sag01` is the alias name of the installation where `NUMRealmServer` server instance is running.

- To view if Universal Messaging supports migration as a custom API:

```
sagcc list administration product sag01 NUMRealmServer
```

where `sag01` is the alias name of the installation where `NUMRealmServer` server instance is running.

Lifecycle Actions for Universal Messaging Server

The following table lists the actions that Universal Messaging supports with the `sagcc exec lifecycle` command and the operation taken against a Universal Messaging server when an action is executed.

Action	Description
`start`	Starts theUniversal Messaging server instance. When successful, the Universal Messaging server instance run-time status is set to ONLINE.
`stop`	Stops the Universal Messaging server instance. The Universal Messaging server run-time status is STOPPED.
`restart`	Stops, then restarts the Universal Messaging server instance. The Universal Messaging server run-time status is set to ONLINE.

Run-time Monitoring States for Universal Messaging Server

The `sagcc get monitoring runtimestate` and `sagcc get monitoring state` commands provide information about the following key performance indicators (KPIs) for a Universal Messaging server instance:

KPI	Description
JVM memory usage	Indicates the utilization of JVM memory. The KPI uses the following marginal, critical, and maximum values: ■ Marginal is 80% of the maximum JVM memory. ■ Critical is 95% of the maximum JVM memory. ■ Maximum is 100% of the maximum JVM memory.
Fanout backlog	Indicates the total number of events currently waiting to be processed by the fanout engine. If the fanout backlog is more than the critical value, there is a possibility that the subscribers receive the published events after some delay. The KPI uses the following marginal, critical, and maximum values: ■ Marginal is 80% of the maximum value. ■ Critical is 95% of the maximum value. ■ Maximum is 100% of the peak value (high-water mark) of fanout backlog. Default is 100.

KPI	Description
Tasks queued for read and write	Indicates the total number of tasks in the read, write, and common read/write pools. If the number of read and write tasks queued is more than the critical value, it indicates that the Universal Messaging server instance is unable to match the speed of the publishers and subscribers. The KPI uses the following marginal, critical, and maximum values: ■ Marginal is 80% of the maximum value. ■ Critical is 95% of the maximum value. ■ Maximum is 100% of the peak value (high-water mark) of read and write tasks queued. Default is 100.

Run-time Monitoring Statuses for Universal Messaging Server

The following table lists the run-time statuses that a Universal Messaging server instance can return in response to the `sagcc get monitoring state` command, and the meaning of each status.

Note:
Universal Messaging server instance does not return the STARTING and STOPPING statuses.

Run-time Status	Meaning
ONLINE	Universal Messaging server instance is running.
FAILED	Universal Messaging server instance is not running due to some failure. LOCK file exists.
STOPPING	Universal Messaging instance is being stopped. LOCK file exists.
STOPPED	Universal Messaging server instance is not running because it was shut down normally. LOCK file does not exist.
UNRESPONSIVE	Universal Messaging server instance does not respond to a ping operation. LOCK file exists and the Universal Messaging server instance is running.
UNKNOWN	The status of Universal Messaging server instance cannot be determined.
ONLINE_MASTER	Server instance is online and it is the master node in the cluster.
ONLINE_SLAVE	Server instance is online and it is the slave node in the cluster.

Run-time Status	Meaning
ERROR	Server instance is part of a cluster that does not satisfy the requisite quorum.

4 Comparison of Enterprise Manager and Command Central Features

The following table compares the availability of features of Enterprise Manager with their availability in Command Central. The Enterprise Manager features are listed in alphabetical order.

The table represents the status of available features at the time of the first release of version 10.3. Later re-releases of version 10.3 might contain features not available in the initial release.

Area	Enterprise Manager Feature	Supported in Command Central
Audit	View audit log entries: Order by user, type etc. Stream or archive audit file	No
Channel/Queue	Create a join form a channel ... set name source channel remote realm/cluster for inter-cluster joins filter hope count purge forward archival	Yes
Channel/Queue	View and/or delete *named objects* View outstanding events, configuration, etc. view messages purge messages view status e.g. last eventID	Yes

Area	Enterprise Manager Feature	Supported in Command Central
Channel/Queue	Create channel ...	Yes
	name, type, ttl, capacity, deadevent store	
	JMS engine and merge engine	
	protobuf descriptor	
	channel key (with name and depth)	
	full storage properties	
Channel/Queue	Update protocol buffer definition	Yes
Channel/Queue	View/modify incoming/outgoing joins	Yes
Channel/Queue	View and modify ACLs	Yes
Channel/Queue	Edit channel/queue	Partial
	Full storage properties dialogue	Properties can be changed but any messages on the channel/queue will not be retained
Channel/Queue	Publish to any channel/queue	No
	Full message properties dialogue	
Channel/Queue	Create a copy of any channel/queue with option to edit specific fields/settings	No
Channel/Queue	View store level connection lists and stats	Partial
	Bounce and redirect connections	*Find under:*
		durables stats page, along with other durable details
Channel/Queue	Bulk apply ACL changes to folders of channels or queues	No
Channel/Queue	Purge events	No
	from EID	
	to EID	
	filter	
	purge all	

Area	Enterprise Manager Feature	Supported in Command Central
Channel/Queue	Perform maintenance	No
Channel/Queue	Snoop events on store Edit and republish events with full message edit dialogue	No
Channel/Queue	Snoop (edit and republish) message content of events encoded as protocol buffers	No
Cluster	Create/delete clusters Set name of cluster at creation time	Yes
Cluster	Modify Cluster membership	Yes
Cluster	See Clusterwide stats: cluster matrix (state of each node as seen by other nodes) event rates connections etc.	Partial cluster members and their state can be viewed
Cluster	Optionally choose to migrate stores at cluster migration and warn of conflicts	No
Comms	Allow creating, deleting, starting, stopping of interfaces	Yes
Comms	View and edit all configuration options available on interfaces, including JavaScript configuration	Yes
Comms	View live per-interface level stats (connections, idle threads, etc.)	No
Comms	Set up via lists on interfaces	No
Comms	Configure, add, delete and view multicast configuration and state Multicast configuration: view configuration and state configure, add. delete	No
Comms	Shared memory	Yes

Area	Enterprise Manager Feature	Supported in Command Central
	View configuration	
	Configure, add and delete	
Comms	Bounce and redirect connections	No
Connection	View details per connection	No
DataGroups	Add datagroup and their properties:	No
	name	
	multicast	
	priority	
	conflation:merge/drop/interval	
DataGroups	Delete datagroup	No
DataGroups	Publish to any datagroup (including default datagroup)	No
	Full message properties dialogue	
DataGroups	Add/remove datagroups from other datagroups	No
Inter-cluster connections	Add, modify, view inter-cluster connections	No
Inter-cluster, joins	Add joins between stores in different clusters or on remote realms (unclustered)	No
Realm	Enterprise Manager tab *Config*	Yes
	View and change realm configuration options	
Realm	Connect to realms	Yes
	Auto-discover other realms in cluster	
Realm	Display IP/host and port of realms/clusters	Yes
Realm	View logs:	Yes, with some additional options
	UM log	
	stream log to file	
	filter log	
	force roll log	

Area	Enterprise Manager Feature	Supported in Command Central
Realm	Apply namespace filter to see only partial list of resources on realm	Yes ... but separate for channels and queues
Realm	View all channels/queues/datagroups	Partial *Exception:* DataGroups
Realm	View graphs of ... event history/rate heap memory usage direct memory usage	Partial *Limited to:* Fanout backlog JVM memory Queued tasks of a server No graphs
Realm	Event status consumed published consumed/published rates connection rates numbers current total numbers of channels datagroups data streams Memory usage total free used change direct total direct free	Partial *Find under:* channels memory config
Realm	Bulk apply ACLs to all channels or queues.	Partial

Area	Enterprise Manager Feature	Supported in Command Central
		... available using templates
Realms	View all known connected realms and their state	Partial Stats are only shown in the channels
Realm	View ... current connections rate of connections total connections And for each connection see ... protocol user host connection description, including ephemeral port, language and name	No
Realm	Enterprise Manager tab *JNDI* View and modify JNDI on the realm	Yes
Realm	Add/Remove realms from each other Mount realms in namespace	No
Realm	Import/Export full/partial realmXML	No ... but most configurations can be exported
Realm	Request ... maintenance thread dump release of cached memory roll of server log	No
Realm	Enterprise Manager tab *Monitoring > Top* View ...	Partial

Area	Enterprise Manager Feature	Supported in Command Central
	CPU usage	
	garbage collection	
	heap usage	
	per channel stats on disk and memory usage	
Security	Set resource specific ACLs on realm, channel and queue.	Yes
	Add/remove ACLs including security groups.	
Security	Define user security groups with name and member IP	Yes
	Add/remove members to groups, including other groups	
Sites	Add/remove, modify sites	Yes
	Modify prime site membership	
Threads	View all threadpool:	No
Zones	Zones:	Yes
	Add	
	Modify	
	Configure	

5 Setting up Active/Passive Clustering with Shared Storage

- About Active/Passive Clustering 462
- Overview of Active/Passive Clustering on Windows 466
- Overview of Active/Passive Clustering on UNIX 467
- Configuring a Universal Messaging Active/Passive Cluster on UNIX 469

About Active/Passive Clustering

Active/passive clustering is a solution that uses clustering software and special purpose hardware to minimize system downtime. Active/passive clusters are groups of computing resources that are implemented to provide high availability of software and hardware computing services. Active/passive clusters operate by having redundant groups of resources (such as CPU, disk storage, network connections, and software applications) that provide service when the primary system resources fail.

In a high availability active/passive clustered environment, one of the nodes in the cluster will be active and the other nodes will be inactive. When the active node fails, the cluster fails over to one of the inactive nodes automatically. As a part of this failover process, clustering software will start the resources on the redundant node in a predefined order (or resource dependency) to ensure that the entire node comes back up correctly.

Universal Messaging can run in an active/passive cluster environment, under Windows or UNIX. This approach does not provide load balancing or scalability.

Active/Passive Clustering Requirements

You need the following to configure a Software AG Universal Messaging active/passive cluster:

- Cluster control software to manage the clusters on Windows or UNIX.
- Shared Storage for sharing data files.
- IP address for running the Universal Messaging cluster service.
- Universal Messaging installed on the cluster nodes in the same directory path (for example, C:\SoftwareAG_UM). In the installations, the data directory path for the shared storage must be the same.

Important: Universal Messaging installation must be identical on all cluster nodes. All instances of Universal Messaging must point to the same Universal Messaging storage files on the shared storage.

Universal Messaging Capabilities for Active/Passive Clustering

The following capabilities of Universal Messaging enable the vendor-specific cluster control software to monitor and manage Universal Messaging in an active/passive cluster.

- Functionality to start, stop, and monitor the servers.
- Ability to store the server's state information and data on a shared disk.
- Ability to survive a crash and restart itself in a known state.
- Ability to meet license requirements and host name dependencies.

Virtual IP Address of an Active/Passive Cluster

A virtual IP address is like any other IP address except it does not have a specific host or node to resolve to. It resolves at run time to a server wherever the IP is physically bound and reachable on the network.

For client applications to access the services in an active/passive cluster in a transparent way, the virtual IP address of the cluster must be supplied to the client applications. This virtual IP address is usually referred to as the "logical host." This logical host identity is a network address (or host name) and is not tied to a single cluster server.

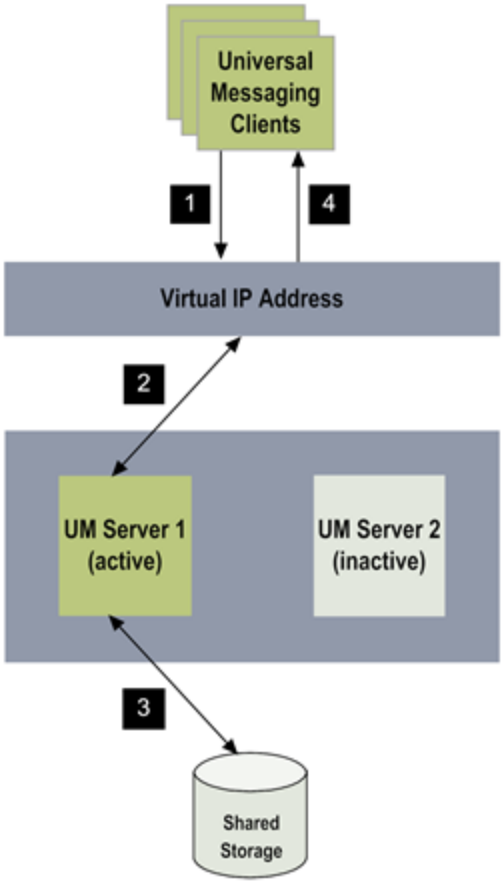
When there is a failover, the cluster control software will resolve the virtual IP address to the physical IP address of the current active server in the cluster. The client application is not affected in any way other than experiencing a brief outage of the services.

Failover Mechanism in an Active/Passive Cluster

Universal Messaging runs as a service in a cluster. Within an active/passive cluster, there only be a single instance of Universal Messaging server running at any given time. The other Universal Messaging servers are inactive.

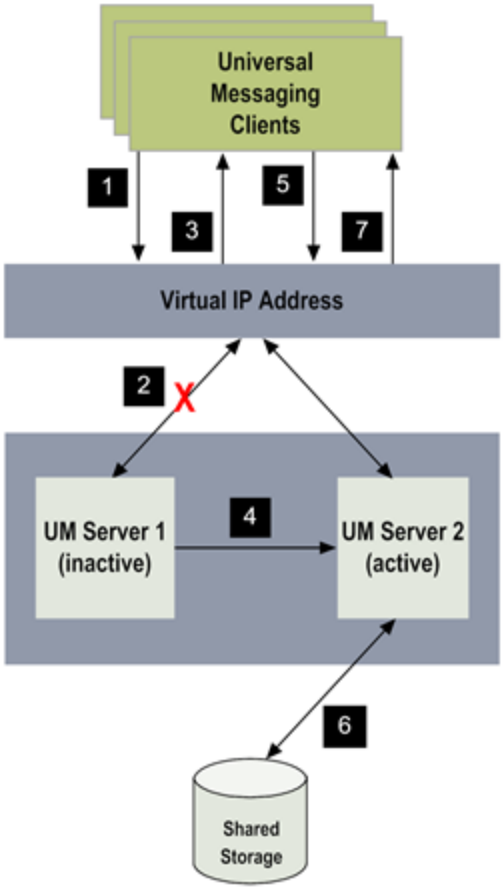
In a clustered environment, when a client makes a request to a server, the server handles the request much the same as in an unclustered environment, except that the server writes the client information to a shared disk instead of a private data store.

The following diagram illustrates the flow of documents through a typical clustered environment.



Steps	Description
1	Universal Messaging clients use the virtual IP address of the cluster to connect to the active/passive Universal Messaging cluster.
2	Cluster control software forwards the client request to the active server in the cluster.
3	The active server reads data from or writes data to the shared storage.
4	Universal Messaging returns the results to the client application.

The following diagram illustrates the failover in a clustered environment. If a server fails, subsequent requests for the session are redirected to a spare server in the cluster that is currently active and running.



Steps	Description
1	Universal Messaging clients use the virtual IP address of the cluster to connect to the active/passive Universal Messaging cluster.
2	The active server experiences failure and shuts down.
3	The cluster software returns the error code to the client.
4	Cluster control software marks the spare server as active.
5	Cluster control software forwards the client request to the active server in the cluster.
6	The active server reads data from or writes data to the shared storage.
7	Universal Messaging returns the results to the client application.

Cluster Verification

A cluster installation consultant will typically perform the cluster installation for you; however, verify the following to make sure that the cluster is installed properly:

- The shared drive can be accessed from the cluster nodes.
- The virtual IP address of the cluster is accessible on the public network.
- Only one Universal Messaging server instance in the cluster can access the shared drive at any given time.

Roles and Responsibilities for Configuring an Active/Passive Cluster

Configuring Universal Messaging in a high-availability cluster requires the efforts of the following people:

- System administrator
- Cluster vendor's installation consultant
- Universal Messaging administrator

Role	Responsibilities
System administrator	Perform system and network administration tasks.
Cluster vendor's installation consultant	Install cluster hardware and software (for example, Windows Server, Veritas, HP ServiceGuard, IBM HACMP, or Oracle Solaris Cluster) installation.
Universal Messaging administrator	Install Universal Messaging and high availability (HA) scripts.

Overview of Active/Passive Clustering on Windows

This section describes how to configure Universal Messaging with shared storage on Windows Server 2008 R2, Windows Server 2012 R2, and Windows Server 2016.

How Does Universal Messaging Run in a Windows Cluster?

In a Windows cluster environment, Universal Messaging runs as a service or as an application defined within a Windows cluster group. You use the Failover Cluster Manager to configure and monitor the Universal Messaging servers and all the associated resources. For more information about the settings in Failover Cluster Manager, see the Microsoft Windows Failover Cluster Manager manuals.

Active/Passive Cluster Configuration on Windows Server

Perform the following steps to configure Universal Messagingfor high availability.

1. Mount and configure the shared drive, and add the shared drive to the cluster. For more information about adding and configuring a shared drive, see the Microsoft Server documentation for your Microsoft Server version.

2. Install Universal Messaging on the cluster nodes.

   Use the same directory name on all cluster nodes. Ensure that the data directory path for the shared storage is the same in all the installations (for example, C:\SoftwareAG_UM). The data directory path in the `Server_Common.conf` configuration files must correctly refer to the same shared storage path. For example, the data directory path in all the nodes is specified as `wrapper.java.additional.4="-DDATADIR=H:\UMSharedStorage\Data"`.

   Note:
   If you want to make changes that will be automatically migrated in a future upgrade/migration, you need to set the corresponding properties in the `Custom_Server_Common.conf` file. This is described in the section "JVM Options" on page 385.

3. Create the Universal Messaging cluster in Windows Server. See the Microsoft Server documentation for instructions to create a failover cluster.

4. Create the Universal Messaging cluster group. Define all the resources and dependencies required to run Universal Messaging.

5. Configure Universal Messaging as a clustered service.

   You can run Universal Messaging as a service or an application.

6. Customize the Universal Messaging startup behavior. For instructions to configure the startup behavior, see the relevant Microsoft Server documentation.

   You can configure the number of possible attempts for starting the Universal Messaging server before failover.

7. Verify failover in the cluster using Windows Server tools.

   You or a system administrator can verify failover when there is a hardware failure.

8. Ensure that the installation and configuration enables the Universal Messaging server to failover correctly from one cluster node to the other.

Overview of Active/Passive Clustering on UNIX

This section describes how to configure Universal Messaging with shared storage on UNIX.

Cluster Monitoring Scripts

The cluster control software determines the health of the servers by periodically probing the servers using the monitor scripts. When the cluster control software determines that one of the servers in the cluster has failed, it will shut down that server and start the server on the spare node.

You must incorporate the UNIX shell scripts for starting, stopping, and monitoring the servers in the cluster control software's infrastructure. You might have to customize code to enable the cluster control software to invoke these UNIX shell scripts.

Summary of Active/Passive Cluster Configuration on UNIX

This section is written primarily to a Universal Messaging administrator to gain a better understanding of the configuration process.

To configure Universal Messaging in an active/passive cluster

1. Ask the cluster vendor's installation consultant to perform these tasks:

   a. Install the HA cluster environment.

   b. Configure the HA cluster environment including the shared disk storage.

2. Ask the system administrator to perform these tasks:

   a. Administer the HA cluster environment so it is ready for software installation.

   b. Configure the external network connection to the HA cluster and create the virtual host (virtual IP address) for the HA cluster.

3. Ask the cluster vendor's installation consultant and the system administrator to test the basic HA installation to ensure it functions properly.

4. Install and configure Universal Messaging on the cluster nodes with the help of the cluster vendor's installation consultant.

   For information about how to install the cluster nodes and configure the cluster, see "Configuring a Universal Messaging Active/Passive Cluster on UNIX" on page 469.

5. Verify that Universal Messaging runs on the cluster node. For instructions, see "Verify the Universal Messaging Server Installation" on page 469.

6. Make sure the cluster is installed properly and configured. For information, see "Verify Failover in the Cluster" on page 470.

7. Configure and test the scripts according to the cluster vendor's specification for starting, stopping, and monitoring the Universal Messaging servers. For instructions, see "Configure the Start, Stop, and Status Scripts" on page 470.

8. Verify failover in the cluster. For instructions, see "Verify Failover in the Cluster" on page 470.

Configuring a Universal Messaging Active/Passive Cluster on UNIX

Install Universal Messaging on Cluster Nodes

When you install Universal Messaging on cluster nodes, you must:

- Follow the instructions in the *Using Software AG Installer* guide.
- Work with the cluster vendor's installation consultant to prepare the cluster node to respond to the virtual IP address and have access to the storage files on the shared storage.

To install Universal Messaging on cluster nodes

1. Install Universal Messaging on the first cluster node and configure to use the shared storage.

2. Unmount the shared storage from cluster node 1 and mount it on cluster node 2.

3. Make the first cluster node inactive.

4. Install Universal Messaging on the second cluster node and configure it to use the shared storage.

5. Make the second cluster node active so that it will respond to the virtual IP address and have access to the storage files on the shared storage.

Verify the Universal Messaging Server Installation

Use the Universal Messaging Enterprise Manager to verify that the Universal Messaging server is properly installed and working.

To verify that the servers are installed properly

1. Start the Enterprise Manager.

2. Connect to the servers that are part of the cluster.

3. Verify the status of the servers in the cluster.

Configure the Start, Stop, and Status Scripts

Incorporate the scripts of each cluster node into the cluster control software with the help of the cluster vendor's installation consultant.

To configure the start, stop, and status scripts of a server

1. Provide the location of the start and stop scripts to the cluster vendor consultant.

   The scripts to start and stop a Universal Messaging server are located here:

   - *Universal Messaging_directory* /server/*server_name*/bin/nserver to start the server.
   - *Universal Messaging_directory* /server/*server_name*/bin/nstopserver to stop the server.

2. Implement a status script using the Universal Messaging API and provide the script to the cluster vendor consultant. For example, to return the server time stamp, you can use: *Universal Messaging_directory* /java/*server_name*/bin/ngetservertime.

   Important:
   When you have basic authentication enabled on the server, you must configure the UM_PASSWORD or PASS_PASSWORD_IN_CONSOLE property before running the ngetservertime command. For more information about UM_PASSWORD and PASS_PASSWORD_IN_CONSOLE, see the section *Running the Java Sample Applications when Basic Authentication is Enabled* in the *Developer Guide*.

   a. Modify env.sh and change last command from "$SHELL" to "$SHELL $*".

   b. Run this command to monitor the server status:

   ```
   ./env.sh -c "ngetservertime" | grep "Server Time"
   ```

Verify Failover in the Cluster

Test the entire cluster with an application to make sure that the cluster functions properly. With the help of the system administrator and the cluster vendor's installation consultant, you can verify the cluster configuration and installation by causing a failover.

To verify failover in a cluster

1. In the Enterprise Manager, provide the virtual IP address of the cluster to connect to the server and view the status.

2. Shut down the server on cluster node 1.

3. Start the server on cluster node 2 or let the cluster software start the server on cluster node 2.

4. Verify the status of the servers.

6 Command Line Administration Tools

- Introduction to the Administration Tools 474
- Starting the Tools using the Tools Runner Application 474
- Performing Standard Administration Tasks on Realms and Clusters 476
- Running a Configuration Health Check 482
- The "Realm Information Collector" Diagnostic Tool 492
- The ExportEventsFromOfflineMemFile Tool 499
- The RepublishEventsFromOfflineFile Tool 502
- Syntax reference for command line tools 503

Introduction to the Administration Tools

Universal Messaging provides a set of command line tools that allow you to perform many of the common actions available through Universal Messaging. Examples of how to use the tools are also provided.

The tools can in general be grouped into the following categories:

Category	Description
General administration tasks	This is a set of tools for performing many of the common administration actions available through Universal Messaging. For example, the `CreateChannel` tool allows you to create a channel on a specified realm, with a number of optional arguments - including TTL, ACLs, and many more - available through the parameters passed to the tool.
Configuration health checker	This tool allows you to check your configuration setup for either a single realm or for a cluster. The tool notifies you of any errors or inconsistencies in your setup. You can run the health check on a running system (realm or cluster). You can also run the health check offline on the basis of XML files containing the configuration of a realm or cluster (one XML configuration file per realm).

These tools are described in the following sections.

Starting the Tools using the Tools Runner Application

To run a tool, you start the *Tools Runner* application and pass the name of the required tool as a parameter to this application, as well as any additional parameters required by the tool.

The Tools Runner application is located in `<InstallDir>`/UniversalMessaging`/tools/runner`.

To start the Tools Runner application, use the appropriate command for Windows or Linux:

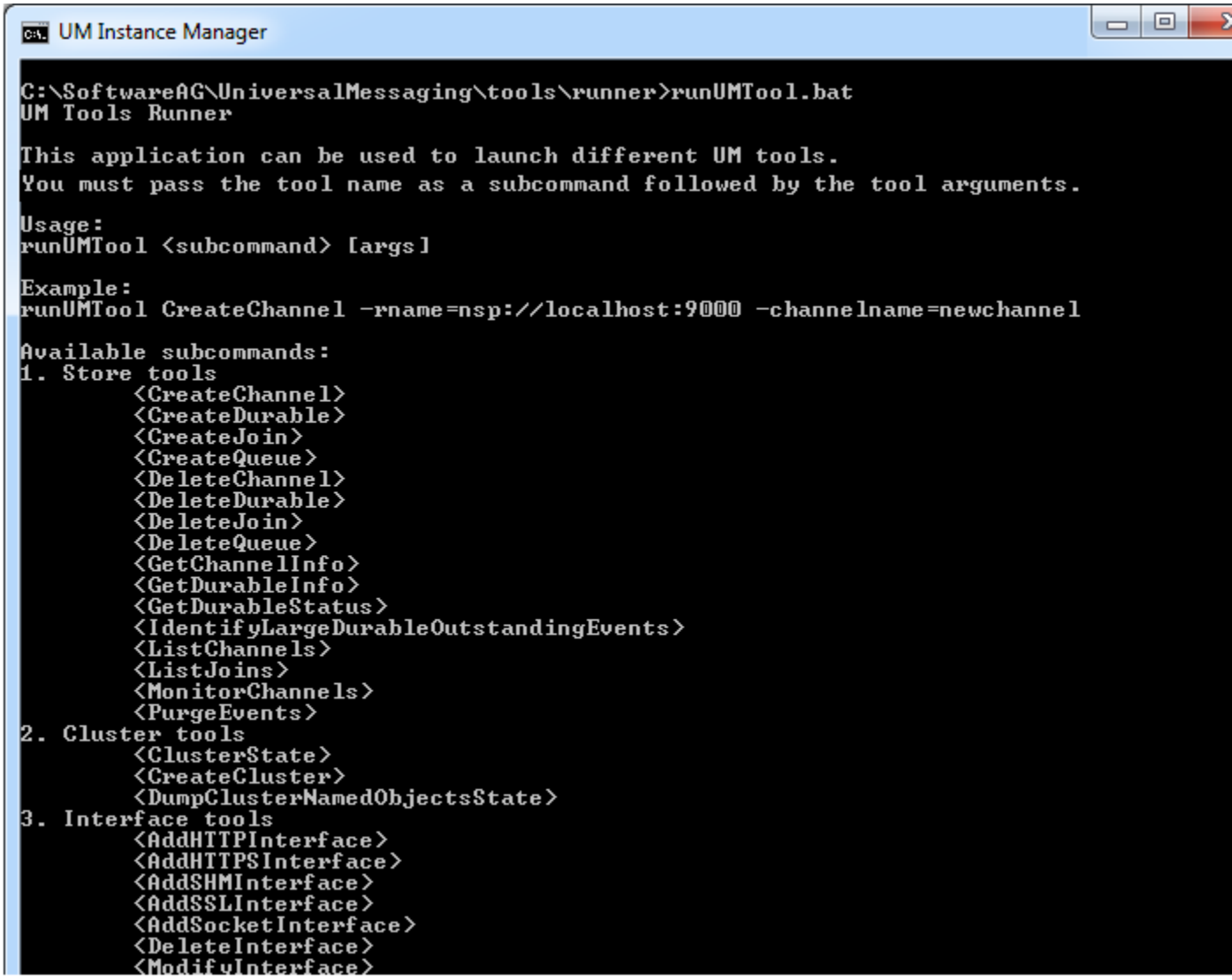
Windows:

```
runUMTool.bat
```

Linux:

```
runUMTool.sh
```

If you run Tools Runner with no arguments, this displays a list of installed tools, as well as instructions for using the Tools Runner, as shown in the following image.



Running a Tool

To run a specific tool, you pass the name of the tool as the first argument to the Tools Runner application. Doing so without any additional arguments will print the usage for the specific tool. For example, running

```
runUMTool.bat CreateChannel
```

will print the usage for the `CreateChannel` tool. The usage contains a description of the tool, and a list of the required and optional arguments that you can supply. Arguments which have a specific set of legal values will have these values displayed here. Also included in the usage are command line examples of running the tool.

To run a tool with additional arguments, each of the required arguments must be specified in the command. For example, the `CreateChannel` tool requires both a realm name and channel name to be specified:

```
runUMTool.bat  CreateChannel  -rname=nsp://localhost:9000  -channelname=channel
```

Additional optional arguments can be appended to the command in the same way; adding a channel type to the `CreateChannel` tool command would then be:

```
runUMTool.bat  CreateChannel  -rname=nsp://localhost:9000  -channelname=channel
-type=R
```

Using the Debug Logging option

You can use the optional `enableDebug` argument on the command line to create a log file that shows the progress of the tool while it is running. It will log most of the exceptions that can occur during the tool execution. For example:

```
runUMTool.bat DumpACL -rname=nhp://localhost:11000 -enableDebug
```

The log file is called `toolsLog.log` and is located in the same directory as the Tools Runner application.

Performing Standard Administration Tasks on Realms and Clusters

Using the Tools Runner application, you can launch command line tools for performing standard administration tasks on realms and clusters.

Tools are available to perform tasks such as:

- Creating, deleting and monitoring channels and queues
- Creating clusters
- Adding, modifying and deleting interfaces (HTTP, HTTPS, SSL, Sockets)
- Adding and deleting ACL entries for channels, queues and realms

For example, the `CreateChannel` tool allows you to create a channel on a specified realm, with a number of optional arguments - including TTL, ACLs, and many more - available through the parameters passed to the tool.

To see the complete set of administration tools available, start the `Tools Runner` application without any parameters, as described in the section "Starting the Tools using the Tools Runner Application" on page 474.

The following table lists the available tools. The tools are organized according into categories, according to the general purpose for which the tools are used.

Tool name / Category	Description
Category: Store tools	**For the command line syntax of the tools in this category, see the section "Syntax: Store Tools" on page 503.**

Tool name / Category	Description
CreateChannel	Creates a channel with the specified name on the specified server. A single permission can be set during channel creation using optional arguments. For adding a set of permissions use the client API.
CreateDurable	Creates a Durable with the specified name and type on the specified channel.
CreateJoin	Joins two channels.
CreateQueue	Creates a queue with the specified name on the specified server. A single permission can be set during queue creation using optional arguments. For adding a set of permissions use the client API.
DeleteChannel	Deletes a channel with the specified name on the specified session.
DeleteDurable	Deletes a Durable with the specified name on the specified channel.
DeleteJoin	Deletes a join between two channels.
DeleteQueue	Deletes a queue with the specified name on the specified session.
GetChannelInfo	Gets the attributes and storage properties of a specified channel in a specified realm.
GetDurablesInfo	Displays the durables details saved in a .nsb file.
GetDurableInfo	Gets the attributes of a specific Durable in a specific channel.
GetDurableStatus	Gets the current state of durables on a realm, sorted by a given field.
GetEventsInfo	Displays the event details present in the memory file(s).
IdentifyLargeDurableOutstandingEvents	Identifies channels containing Durables with a large number of outstanding events.
ListChannels	Lists details of the channels on the specified server.
ListJoins	Lists joins on a given realm.
MonitorChannels	Monitors the channels and queues in a realm and prints totals.
PurgeEvents	Purges events from a channel with the specified name on the specified session.

Tool name / Category	Description
Category: Cluster tools	**For the command line syntax of the tools in this category, see the section "Syntax: Cluster Tools" on page 516.**
ClusterState	Checks the cluster state by a given RNAME, which is part of a cluster.
CreateCluster	Creates a cluster with the specified name, consisting of the specified realms.
DumpClusterNamedObjectsState	Dumps the state of named objects (durables) on channels present on the specified cluster servers.
Category: Interface tools	**For the command line syntax of the tools in this category, see the section "Syntax: Interface Tools" on page 518.**
AddHTTPInterface	Adds an HTTP interface on the specified adapter and port, on the specified realm.
AddHTTPSInterface	Adds an HTTPS interface on the specified adapter and port, on the specified realm.
AddSHMInterface	Adds a shared memory interface with the specified path, buffer size and timeout, on the specified realm.
AddSSLInterface	Adds an SSL interface on the specified adapter and port, on the specified realm.
AddSocketInterface	Adds a socket interface on the specified adapter and port, on the specified realm.
DeleteInterface	Deletes the specified interface from the specified realm.
ModifyInterface	Modifies the specified interface on the specified realm.
Category: Data group tools	**For the command line syntax of the tools in this category, see the section "Syntax: Data Group Tools" on page 527.**
AddDataGroup	Adds a child data group to a parent data group. Both of these data groups must exist.
CreateDataGroup	Creates a data group with the specified name on the specified server. Additionally, conflation attributes and other options of the data group can be set.
DeleteDataGroup	Removes the data group with the specified name from the server.

Tool name / Category	Description
ListenDataGroup	Listens for data group events on a realm.
PublishDataGroup	Publishes messages to a data group.
Category: Publish tools	**For the command line syntax of the tools in this category, see the section "Syntax: Publish Tools" on page 530.**
PublishChannel	Publishes events to a channel.
PublishChannelXML	Publishes an XML document to a channel.
PublishCompressed	Publishes events to a store, using compression.
PublishQueue	Publishes events to a queue.
PublishTX	Publishes events, as a part of a transaction, to a channel or queue.
Category: Subscribe tools	**For the command line syntax of the tools in this category, see the section "Syntax: Subscribe Tools" on page 533.**
PeekQueue	Peeks all events on a queue and prints statistics for the bandwidth rates.
SubscribeChannel	Reads all the messages from a channel.
SubscribeChannelAsync	Listens for messages on a channel.
SubscribeChannelAsyncDurable	Listens for messages on a channel. Running the tool with the same "-name" argument will continue reading from the last unconsumed event.
SubscribeChannelDurable	Listens for messages on a channel. Running the tool with the same "-name" argument will continue reading from the last unconsumed event.
SubscribeCompressed	Listens for compressed messages on a channel.
SubscribeQueue	Reads all the messages from a queue.
SubscribeQueueAsync	Listens for messages on a queue.
Category: Security tools	**For the command line syntax of the tools in this category, see the section "Syntax: Security Tools" on page 538.**
AddChannelACLEntry	Adds an ACL entry on the specified channel for the specified user and host, on the specified session.

Tool name / Category	Description
AddContainerACLEntry	Adds an ACL entry on the specified container for the specified user and host.
AddQueueACLEntry	Adds an ACL entry on the specified queue for the specified user and host, on the specified session.
AddRealmACLEntry	Adds an ACL entry on the specified realm for the specified user and host.
AddSecurityGroup	Adds a security group to the specified realm with the specified name.
AddUserToSecurityGroup	Adds a specified user and host subject to a given security group on a specified realm.
DeleteChannelACLEntry	Deletes the ACL entry from the specified channel with the specified user and host.
DeleteContainerACLEntry	Removes an ACL entry from the specified container with the specified user and host.
DeleteQueueACLEntry	Deletes an ACL entry from the specified queue with the specified user and host.
DeleteRealmACLEntry	Removes an ACL entry from the specified realm with the specified user and host.
DeleteSecurityGroup	Removes a security group from the specified realm with the specified name.
DumpACL	Dumps all the ACL data for a realm.
ModifyChannelACLEntry	Updates an ACL entry on the specified channel for the specified user and host, on the specified session.
ModifyContainerACLEntry	AddContainerACLEntry adds an ACL entry on the specified container for the specified user and host.
ModifyQueueACLEntry	Updates an ACL entry on the specified queue for the specified user and host, on the specified session.
ModifyRealmACLEntry	Modifies an ACL entry on the specified realm for the specified user and host.
RemoveUserFromSecurityGroup	Removes a specified user from a given security group on the specified realm.
Category: Zone tools	**For the command line syntax of the tools in this category, see the section "Syntax: Zone Tools" on page 551.**

Tool name / Category	Description
AddMemberToZone	Adds a realm to a specified realm's zone.
CreateZone	Creates a zone with the specified name containing the specified realms.
DeleteZone	Deletes a zone with the specified name on the specified session.
RemoveMemberFromZone	Removes a realm from its current zone.
Category: JMS tools	**For the command line syntax of the tools in this category, see the section "Syntax: JMS Tools" on page 553.**
CreateConnectionFactory	Creates a JMS connection factory with the specified server.
CreateJMSQueue	Creates a JMS queue with the specified name on the specified session.
CreateJMSTopic	Creates a JMS topic with the specified name on the specified session.
JMSPublish	Publishes one or more messages to a JMS queue or topic.
JMSSubscribe	Reads messages arriving to a JMS destination.
ModifyConnectionFactory	Modifies settings of a JMS connection factory on the specified server.
ViewConnectionFactory	Views settings of a JMS connection factory on the specified server.
Category: Recovery tools	**For the command line syntax of the tools in this category, see the section "Syntax: Recovery Tools" on page 563.**
AddInterfaceOffline	Adds a new interface to an offline realm.
DeleteInterfaceOffline	Removes an interface from an offline realm using configuration data.
DumpInterfacesOffline	Dumps the list of interfaces for a specified offline realm.
ModifyInterfaceOffline	Modifies an interface of an offline realm.
ModifyPrimeFlagOffline	Modifies the prime flag of a site while the realm is offline.
Category: Durable tools	**For the command line syntax of the tools in this category, see the section "Syntax: Durable Tools" on page 568.**

Tool name / Category	Description
ViewDurableEvent	Gets all events for all durables or all events for a specific durable.
Category: Miscellaneous	**For the command line syntax of the tools in this category, see the section "Syntax: Miscellaneous Tools" on page 569.**
EditRealmConfiguration	Edits realm configuration parameters.
ExportRealmXML	Exports a selected realm to an XML file.
HealthChecker	Runs the Health Checker tool for analysing configuration items and highlighting robustness improvements. For more details, see the section "Running a Configuration Health Check" on page 482.
ImportRealmXML	Imports a realm from an XML file.
Category: Site tools	**For the command line syntax of the tools in this category, see the section "Syntax: Site Tools" on page 573.**
CreateSite	Creates a site with the specified name, consisting of the specified nodes.
DeleteSite	Deletes a site with the specified name from all the nodes associated with it.
SetPrimeSite	Toggles the specified site's prime status.
ShowSites	Displays the configuration of the sites.
Category: Diagnostic tools	**For the command line syntax of the tools in this category, see the section "Syntax: Diagnostic Tools" on page 575.**
RealmInformationCollector	Collects diagnostic information from a realm server installation and stores it in a zip archive. For more details, see the section "The "Realm Information Collector" Diagnostic Tool" on page 492.

Running a Configuration Health Check

Overview

The HealthChecker is a tool for checking the correctness of a realm or cluster configuration.

The tool is primarily intended for use by Software AG support staff for analyzing possible problems in customer configurations, but you might also find it useful for checking your configuration.

The tool can be used in the following ways:

- To check the configuration of a live realm (which can be a single entity or a node of a cluster) or a cluster. If the realm is a node of the cluster, the checks will also be automatically executed against all the other cluster members.
- To do an offline check of the configuration of a realm or cluster, based on configuration information that has been exported to XML files. Each such XML file contains the configuration data of a realm, regarding channels, queues, durables, datagroups, etc. The tool will only run the checks against all the cluster members if their XML paths are given explicitly in the call of the tool.

Typical configuration aspects that can be checked in a clustered realm are:

- **Datagroups:**

  Datagroups belonging to a cluster must be present on all nodes of the cluster and their attributes must be the same.

- **Durables:**

  Durables belonging to clusterwide channels should also be clusterwide. They must be present on all nodes of the cluster and their attributes must be the same.

- **Joins:**

  Joins between clusterwide channels must be present on all nodes of the cluster and their attributes have to be the same.

- **Stores:**

  Stores belonging to a cluster must be present on all nodes of the cluster and their attributes and properties must be the same.

Typical configuration aspects that can be checked in a non-clustered realm are:

- **Durables:**

  Durables belonging to a non-clustered realm must be non-clusterwide and must be attached to a non-clusterwide channel.

- **Stores:**

  Some store configurations may impact the performance of the system and they need to be highlighted.

Checks against a live realm

The checks that can be run against a live realm are the following:

Name of Check	Description	Default check?
DataGroupMismatchCheck	Check if datagroups are coherent across all nodes of the cluster.	Y
DurableMismatchCheck	Check if durables are coherent across all nodes of the cluster.	Y
DurableSubscriberLargeStoreCheck	Check the number of remaining events to be consumed in a shared durable. If the number is greater than the threshold a warning will be displayed. The default value for the threshold is 1000. This check takes an additional parameter `-threshold` that allows you to specify a custom value for the threshold.	Y
EnvironmentStateCheck	Check and display the status of a running environment. The HealthChecker first checks if the server configuration property `Enable Flow Control` in the configuration group `Server Protection` (see the note after this table) is set to true. If it is set to true then the HealthChecker will check what percentage of memory is taken by events from the whole heap memory. If the percentage is between 70 and 80, or between 80 and 90, or above 90, an appropriate warning will be displayed. The general idea is that Server Protection mechanism gradually slows the consumption of events from clients when a certain threshold is reached, and 70-80, 80-90 and >90 are these thresholds. The degree of slowing down is marked by these three server configuration properties: `FlowControlWaitTimeOne`, `FlowControlWaitTimeTwo` and `FlowControlWaitTimeThree`. These represent a period of time, measured in milliseconds, by which client publishing requests will be delayed when the corresponding threshold has been reached. Threshold 70%-80%: A warning message is displayed that client publishing requests will be delayed by `FlowControlWaitTimeOne` milliseconds.	Y

Name of Check	Description	Default check?
	Threshold 80%-90%: A warning message is displayed that client publishing requests will be delayed by `FlowControlWaitTimeTwo` milliseconds. Threshold 90%-100%: An error message is displayed that client publishing requests will be delayed by `FlowControlWaitTimeThree` milliseconds.	
FixLevelCheck	Check if the nodes of a given cluster have matching fix levels.	Y
JNDIStatusCheck	Check JNDI status and mismatches for stores.	Y
JoinMismatchCheck	Check if joins are coherent across all nodes of the cluster.	Y
ResourcesSafetyLimitsCheck	Check that channel/queue resources have either TTL or Capacity configured to a non-zero value. If both of these values are zero, this means that the channel/queue is not configured with any safety limits.	
ServerProtectionConsistencyCheck	Check if the server configuration properties in the configuration group `Server Protection` group (see the note after this table) are coherent across the nodes of a cluster against a running environment.	Y
StoreMemoryCheck	Check the memory usage of stores.	Y
StoreMismatchCheck	Check if stores are coherent across all nodes of the cluster.	Y
StoreWarningsCheck	Check store warnings on the specified realm.	

A "Y" in the column "Default check?" indicates that the check is included in the `-mode=default` setting (see the topic *The -mode parameter* below).

Important:
When the tool checks for outstanding durable events or event ID mismatches in a **live** environment, there is a chance of getting warning messages, even though the cluster is working correctly. This is because the check is not atomic for the live cluster, so a small synchronization discrepancy can be expected.

Note:
For further information about the server configuration parameters and the configuration group `Server Protection` mentioned in the table above, see the section "Realm Configuration" on page 49.

Checks against a realm's stored XML configuration

The checks that can be run against a stored XML configuration are the following:

Name of Check	Description	Default check?
XMLDataGroupMismatchCheck	Check if datagroups are coherent across all nodes of the cluster.	Y
XMLDurableMismatchCheck	Check if durables are coherent across all nodes of the cluster.	Y
XMLFixLevelCheck	Check if the nodes of a given cluster have matching fix levels.	Y
XMLJNDIStatusCheck	Check JNDI status and mismatches for stores.	Y
XMLJoinMismatchCheck	Check if joins are coherent across all nodes of the cluster.	Y
XMLResourcesSafetyLimitsCheck	Check that channel/queue resources have either TTL or Capacity configured to a non-zero value. If both of these values are zero, this means that the channel/queue is not configured with any safety limits.	
XMLServerProtectionConsistencyCheck	Check if the server configuration properties in the configuration group `Server Protection` are coherent across the nodes of a cluster against the exported configurations from the nodes.	Y
XMLStoreMismatchCheck	Check if stores are coherent across all nodes of the cluster.	Y
XMLStoreWarningsCheck	Check store warnings on the specified realm.	

Command Usage

The syntax is as follows:

```
runUMTool HealthChecker [-rname=<rname> | -xml=/path/to/xml1,...]
          [-check=<checktype>[,<checktype> ...] ]
          [-mode=<modetype>]
          [-include=<checktype>[,<checktype> ...] ]
          [-exclude=<checktype>[,<checktype> ...] ]
          [-<additionalParameter1>=<value>] [-<additionalParameter2>=<value>] ...
```

Displaying help text

To display a help text showing a summary of the command usage, call the HealthChecker without parameters:

```
runUMTool HealthChecker
```

Running a health check of a running realm

```
runUMTool HealthChecker -rname=-rname=nsp://localhost:11000
```

This will run the HealthChecker tool against the given running realm.

Running a health check of a stored realm configuration

```
runUMTool HealthChecker -xml=/path/to/xml1.xml, /path/to/xml2.xml
```

This will run the HealthChecker tool against the realm configurations stored in the given XML files.

The `-check` parameter

This parameter allows you to explicitly specify the check or checks that you want to be executed. No other checks will be included. This parameter can only be used together with the `-rname` or `-xml` parameter; the other additional parameters have no meaning in the context of `-check` so they can't be used.

Example - Execute only the `Store Warnings Check` check against the running realm:

```
runUMTool HealthChecker -rname=nsp://localhost:11000
                        -check=StoreWarningsCheck
```

Example - Execute only the `Store Warnings Check` and `Fix Level Check` checks against the running realm:

```
runUMTool HealthChecker -rname=nsp://localhost:11000
                        -check=StoreWarningsCheck, FixLevelCheck
```

The `-mode` parameter

This parameter allows you to select a predefined set of checks without having to name the checks explicitly. The `-mode` and `-check` parameters are mutually exclusive.

The mode parameter can take one of the following values:

- **default** - this value selects the recommended minimal subset of checks. This is the default option.
- **all** - this mode selects all checks.

If neither `-mode` nor `-check` is specified, the default set of checks will be executed.

The `-include` and `-exclude` parameters

You can use the `-include` and `-exclude` parameters to further refine the set of checks that have been selected by the `-mode` parameter. You can use `-include` and `-exclude` in the same call of the health checker, as long as they do not specify the same check.

- **include** - Run all checks from the set defined by the `-mode` parameter, and additionally include the check or checks specified by this parameter. The parameter may contain a single check or a comma-separated list of checks.

- **exclude** - Run all checks from the set defined by the `-mode` parameter, except the specified check or checks. The parameter may contain a single check or a comma-separated list of checks.

The `-<additionalParameter>` parameters

Some of the health checks allow you to specify one or more additional parameters when calling the HealthChecker. The name and purpose of each additional parameter is specific to the individual health check being run.

For example, the `DurableSubscriberLargeStoreCheck` check allows you to specify the additional parameter `-threshold=<value>`, which defines a threshold for the number of remaining events to be consumed in a shared durable.

The following general rules apply:

- Each additional parameter has a default value, so if you do not specify the additional parameters explicitly, the default values will be taken.
- If multiple additional parameters and multiple checks are specified, each individual check uses only its own additional parameters.
- The additional parameters can be given in any order.
- Checks that do not require additional parameters will ignore the additional parameters.

Syntax Examples

Example - Execute all available checks for live realm check:

```
runUMTool HealthChecker -rname=nsp://localhost:11000 -mode=all
```

Example - Execute all the available checks except the ones mentioned.

```
runUMTool HealthChecker -rname=nsp://localhost:11000
          -mode=all -exclude=JNDIStatusCheck, FixLevelCheck, JoinMismatchCheck
```

Example - Execute the default set of checks, adding the `StoreWarningsCheck` which is not part of the default set.

```
runUMTool HealthChecker -rname=nsp://localhost:11000
          -mode=default -include= StoreWarningsCheck
```

Example - Execute the default set of checks but excluding the `JNDIStatusCheck`, `FixLevelCheck` and adding the `StoreWarningsCheck`.

```
runUMTool HealthChecker -rname=nsp://localhost:11000
          -mode=default -include= StoreWarningsCheck
          -exclude=JNDIStatusCheck, FixLevelCheck
```

Note:
The previous examples are based on live checks using the `-realm` parameter. The same logic applies if you use the `-xml` parameter instead, but the names of the checks need to be adapted to the appropriate XML checks.

Full Example

The following example compares the XML configuration files of two realms in a cluster. The realms are named `realm0` and `realm1`, and their configuration files are named `clustered_realm0.xml` and `clustered_realm1.xml`.

XML configuration file `clustered_realm0.xml` for realm0:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<NirvanaRealm name="realm0" exportDate="2016-11-08Z"
    comment="Realm configuration from realm0" version="BuildIdentifier"
    buildInfo="BuildIdentifier">
    <ClusterSet>
        <ClusterEntry name="cluster1">
            <ClusterMember name="realm1" rname="nsp://localhost:11010/"
                canBeMaster="true"/>
            <ClusterMember name="realm0" rname="nsp://localhost:11000/"
                canBeMaster="true"/>
        </ClusterEntry>
    </ClusterSet>
    <ChannelSet>
       <ChannelEntry>
            <ChannelAttributesEntry name="channel1" TTL="0" capacity="5" EID="0"
                 clusterWide="true" jmsEngine="false" mergeEngine="false"
                 type="PERSISTENT_TYPE"/>
            <StorePropertiesEntry HonorCapacityWhenFull="false"
                 SyncOnEachWrite="false" SyncMaxBatchSize="0" SyncBatchTime="0"
                 PerformAutomaticMaintenance="false" EnableCaching="true"
                 CacheOnReload="true" EnableReadBuffering="true"
                 ReadBufferSize="10240" Priority="4" EnableMulticast="false"
                 StampDictionary="0" MultiFileEventsPerSpindle="50000"/>
            <ChannelJoinSet>
                <ChannelJoinEntry filter="" hopcount="50" to="channel2"
                    from="channel1" allowPurge="false" archival="false"/>
            </ChannelJoinSet>
        </ChannelEntry>
        <ChannelEntry>
            <ChannelAttributesEntry name="channel2" TTL="0" capacity="0" EID="0"
                clusterWide="true" jmsEngine="false" mergeEngine="false"
                type="RELIABLE_TYPE"/>
            <StorePropertiesEntry HonorCapacityWhenFull="false"
                SyncOnEachWrite="false"
                SyncMaxBatchSize="0" SyncBatchTime="0"
                PerformAutomaticMaintenance="false"
                EnableCaching="true" CacheOnReload="true"
                EnableReadBuffering="true"
                ReadBufferSize="10240" Priority="4" EnableMulticast="false"
                StampDictionary="0" MultiFileEventsPerSpindle="50000"/>
            <DurableSet>
                <durableEntry name="durable1" EID="-1" outstandingEvents="0"
                    clusterWide="true" persistent="true"
                    priorityEnabled="false" shared="true"/>
            </DurableSet>
        </ChannelEntry>
    </ChannelSet>
    <QueueSet>
        <QueueEntry>
```

```
            <ChannelAttributesEntry name="queue1" TTL="0" capacity="0" EID="0"
                clusterWide="true" jmsEngine="false" mergeEngine="false"
                type="RELIABLE_TYPE"/>
            <StorePropertiesEntry HonorCapacityWhenFull="false"
                SyncOnEachWrite="false" SyncMaxBatchSize="0" SyncBatchTime="0"
                PerformAutomaticMaintenance="true" EnableCaching="true"
                CacheOnReload="true" EnableReadBuffering="true"
                ReadBufferSize="10240" Priority="4" EnableMulticast="false"
                StampDictionary="0" MultiFileEventsPerSpindle="50000"/>
        </QueueEntry>
    </QueueSet>
</NirvanaRealm>
```

XML configuration file `clustered_realm1.xml` for realm1:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<NirvanaRealm name="realm1" exportDate="2016-11-16Z"
    comment="Realm configuration from realm1"
    version="BuildIdentifier" buildInfo="BuildIdentifier">
    <ClusterSet>
        <ClusterEntry name="cluster1">
            <ClusterMember name="realm1" rname="nsp://localhost:11010/"
                canBeMaster="true"/>
            <ClusterMember name="realm0" rname="nsp://localhost:11000/"
                canBeMaster="true"/>
        </ClusterEntry>
    </ClusterSet>
    <ChannelSet>
        <ChannelEntry>
            <ChannelAttributesEntry name="channel1" TTL="0" capacity="5" EID="0"
                clusterWide="true" jmsEngine="false" mergeEngine="false"
                type="RELIABLE_TYPE"/>
            <StorePropertiesEntry HonorCapacityWhenFull="false"
                SyncOnEachWrite="false" SyncMaxBatchSize="0" SyncBatchTime="0"
                PerformAutomaticMaintenance="false" EnableCaching="true"
                CacheOnReload="true" EnableReadBuffering="true"
                ReadBufferSize="10240" Priority="4" EnableMulticast="false"
                StampDictionary="0" MultiFileEventsPerSpindle="50000"/>
            <ChannelJoinSet>
                <ChannelJoinEntry filter="" hopcount="10" to="channel2"
                    from="channel1" allowPurge="false" archival="false"/>
            </ChannelJoinSet>
        </ChannelEntry>
        <ChannelEntry>
            <ChannelAttributesEntry name="channel2" TTL="0" capacity="0" EID="0"
                clusterWide="true" jmsEngine="false" mergeEngine="false"
                type="RELIABLE_TYPE"/>
            <StorePropertiesEntry HonorCapacityWhenFull="false"
                SyncOnEachWrite="false" SyncMaxBatchSize="0" SyncBatchTime="0"
                PerformAutomaticMaintenance="false" EnableCaching="true"
                CacheOnReload="true" EnableReadBuffering="true"
                ReadBufferSize="10240" Priority="4" EnableMulticast="false"
                StampDictionary="0" MultiFileEventsPerSpindle="50000"/>
        </ChannelEntry>
    </ChannelSet>
    <DataGroupSet>
        <DataGroupEntry>
            <DataGroupAttributesEntry name="dg1" id="3422373812" priority="1"
                multicastenabled="false"/>
```

```
        </DataGroupEntry>
    </DataGroupSet>
</NirvanaRealm>
```

From a first analysis we can say that these two realms belong to the same cluster (cluster1) and that they both contain various stores, joins and data groups. But let's see what happens when we run the HealthChecker tool specifying both XML files and running all the checks. Note that we need to exclude the `XMLServerProtectionConsistencyCheck` since the specified XML files do not contain the `RealmConfiguration` section.

Here is the call of the tool (using Windows syntax) and the result:

```
runUMTool.bat  HealthChecker  -xml=clustered_realm0.xml,clustered_realm1.xml
                              -exclude=XMLServerProtectionConsistencyCheck
HealthChecker Tool - Version: 1.0


XML JOIN MISMATCHES CHECK
ERROR: Join from (channel1) to (channel2) HopCount mismatch [realm1] does not
       equal [realm0]


XML JNDI PROPERTIES CHECK
WARN: Realm realm0: No JNDI entry for store channel1
WARN: Realm realm0: No JNDI entry for store channel2
WARN: Realm realm0: No JNDI entry for store queue1
WARN: Realm realm1: No JNDI entry for store channel1
WARN: Realm realm1: No JNDI entry for store channel2


XML DURABLE STATUS CHECK
ERROR: Could not find durable (durable1) on realm [realm1] but it is present
       on [realm0]


XML STORE MISMATCHES CHECK
WARN: Store (channel1) Type mismatch [realm1] does not equal [realm0]
ERROR: Could not find store (queue1) on realm [realm1] but it is present
       on realm [realm0]


XML DATAGROUP MISMATCHES CHECK
ERROR: Could not find Data Group (dg1) on realm [realm0] but it is present
       on realm [realm1]
```

These errors and warnings tell us:

- Joins: a join between two clusterwide channels has to be the same on all the nodes - in our case there is a mismatch in the HopCount;
- JNDIs: these simple warnings are saying: "Are you sure that you don't need any JNDI for these stores?";
- Durables: if a durable is clusterwide, then it has to be present on all the other nodes (and it has to be the same);

- Stores: if a store is clusterwide, then it has to be the same on all the other nodes.
- Datagroups: the same rule applies for datagroups, which are always clusterwide and have to be present on all the other nodes.

The "Realm Information Collector" Diagnostic Tool

Overview

`RealmInformationCollector` is a command-line diagnostic tool that gathers files and live data from one or more Universal Messaging realm servers. The tool makes it easier for you to collect information that Software AG support may require to diagnose issues with Universal Messaging, but the information collected may also be useful for internal support within your organization.

The tool can be executed in live and offline mode:

- **Live mode:** In live mode, the specified Universal Messaging realm server(s) must be running.

  The tool will collect files that contain operational data for each running realm server, but will also attempt to connect to and gather information directly from each running server process.
- **Offline mode:** In offline mode, the specified Universal Messaging realm server(s) must be offline.

  In this mode, the tool will only collect files that contain operational data for each realm server.

The mode of operation (either offline or live) is a mandatory argument and must be specified when running the tool. You must ensure that the specified Universal Messaging realm servers are stopped when `-mode=offline`, or running when `-mode=live`.

Depending on the mode, the tool will collect different files. For example, in live mode, it will not collect the content of realm server's "data" directory, because this might cause failures on the server.

The tool collects information by executing a list of *collectors*. Each collector is responsible for gathering a specific subset of the realm's information.

Collectors for a live realm server

The collectors that can be run against a live realm server are shown in the following table. Path names of files and directories given in the table are the installation defaults.

Name of Collector	Description	Default collector?
env	Collects environment information from a running realm server. This includes all JVM system properties and the list of Universal Messaging interfaces.	Y

Name of Collector	Description	Default collector?
healthchecker	Acquires health information from a running realm server using the HealthChecker tool. See the section "Running a Configuration Health Check" on page 482 for details.	Y
heapdump	Acquires a heap dump from a running realm server. **Note:** This collector is not available on all platforms. See the section *Operational Issues* below for related information.	
heapdumps	Collects the heap dump directory of a realm server. The location of this directory is `<InstallDir>/UniversalMessaging/server/<realmname>/heap_dumps`. See the section *The Dump file for Out-of-Memory Errors (OOME)* in the *Installation Guide* for related information about heap dumps.	
installlogs	Collects the product's installation log files. These files are located in `<InstallDir>/install/logs`.	Y
instancemgr	Collects the realm server's manager log. This file is located in `<InstallDir>/UniversalMessaging/tools/InstanceManager/instanceLog.txt`.	Y
jaas	Collects the JAAS configuration of a realm server. This file is located in `<InstallDir>/UniversalMessaging/server/<realmname>/bin/jaas.conf`.	Y
license	Collects the license file of a realm server. This file is located in `<InstallDir>/UniversalMessaging/server/<realmname>/licence.xml`.	Y
logs	Collects logs of a realm server. The location of the file is `<InstallDir>/UniversalMessaging/server/<realmname>/data/nirvana.log`. If there are any rolled log files in addition to the current log file, for example `nirvana.log_<timestamp1>` and `nirvana.log_<timestamp2>`, these are collected also.	Y
osinfo	Collects operating system and hardware information. The collector will gather hardware, processor, memory, file system, and network information from the operating system. The data	Y

Name of Collector	Description	Default collector?
	will be stored in the file generated/osinfo.txt in the generated archive. The `osinfo` tool runs on Windows, Apple macOS, Red Hat Enterprise Linux, and Solaris operating systems.	
plugins	Collects the `plugins` directory of a realm server. This directory is located at `<InstallDir>/UniversalMessaging/server/<realmname>/plugins`.	Y
psinfo	Collects information about processes in the operating system. The collector records the top 20 processes in terms of CPU consumption and stores the data in the generated/osinfo.txt file in the generated archive. The `psinfo` collector runs together with the `osinfo` collector, which is selected by default. If you try to run `psinfo` without `osinfo`, the system returns an error. The `psinfo` tool runs on Windows, Apple macOS, Red Hat Enterprise Linux, and Solaris operating systems.	Y
realmconfig	Collects the realm configuration properties of a realm server. The properties are listed in the section "Realm Configuration" on page 49. The collector delivers the configuration properties as a serialized XML file.	Y
secfile	Collects the security file of a realm server. This file is located at `<InstallDir>/UniversalMessaging/server/<realmname>/bin/secfile.conf`.	Y
tanukiconf	Collects the Tanuki service wrapper configuration of a realm server. This includes the files `nserverdaemon.conf`, `Server_Common.conf` and `Custom_Server_Common.conf`, located at `<InstallDir>/UniversalMessaging/server/<realmname>/bin`.	Y
tanukilogs	Collects Tanuki service wrapper logs of a realm server. This file is located at `<InstallDir>/UniversalMessaging/server/<realmname>/bin/UMRealmService.log`. If there are any rolled log files in addition to the current log file, for example `UMRealmService.log.1` and `UMRealmService.log.2`, these are collected also.	Y

Name of Collector	Description	Default collector?
threaddump	Generates three thread dumps of a realm server. The dumps are taken at 15-second intervals. Having three dumps instead of one can make it easier to analyze time-related thread issues.	Y
version	Collects the realm server version. The version will be stored in the file `UniversalMessaging/lib/nServer.jar_version.txt` in the generated archive.	Y

A "Y" in the column "Default collector?" indicates that the collector is included by default when you run the `RealmInformationCollector` tool.

Collectors for an offline realm server

The collectors that can be run against an offline realm server are the following (collectors that can be used also against a live realm server are indicated). Path names of files and directories given in the table are the installation defaults.

Name of Collector	Description	Default collector?
data	Collects the data directory of a realm server. The location of this directory is `<InstallDir>/UniversalMessaging/server/<realmname>/data`.	
heapdumps	(Same as the live collector)	
installlogs	(Same as the live collector)	Y
instancemgr	(Same as the live collector)	Y
jaas	(Same as the live collector)	Y
license	(Same as the live collector)	Y
logs	(Same as the live collector)	Y
osinfo	(Same as the live collector)	Y
plugins	(Same as the live collector)	Y
psinfo	(Same as the live collector)	Y
secfile	(Same as the live collector)	Y
tanukiconf	(Same as the live collector)	Y
tanukilogs	(Same as the live collector)	Y

Name of Collector	Description	Default collector?
version	(Same as the live collector)	Y

Command Usage

The syntax is as follows:

```
runUMTool RealmInformationCollector
          -mode=live|offline  [-username=<username> -password=<password>]
          -instance=*|<instanceName>[,<instanceName> ...]
          [-include=<collectorName>[,<collectorName> ...] ]
          [-exclude=<collectorName>[,<collectorName> ...] ]
          [-outputfile=<dir_or_file>]
```

Displaying help text

To display a help text showing a summary of the command usage, call the `RealmInformationCollector` tool without parameters:

```
runUMTool RealmInformationCollector
```

The `-mode` parameter

This parameter allows you to select the execution mode of the tool. The mode parameter is mandatory and can take one of the following values:

- **live** - the `RealmInformationCollector` tool will collect operational data files for each running realm server and also attempt to connect and gather information directly from each running realm server
- **offline** - the tool will collect operational data files only

In live mode, all specified realm servers (see the `-instance` parameter) must be running, whereas in offline mode, all specified realm servers must be stopped.

Also in live mode, the following collectors will connect to each specified running realm server to gather information, and will store the information in the following files under `UniversalMessaging/server/<InstanceName>/generated` in the generated archive:

Collector name	Generated file
env	`envinfo.txt`
realmconfig	`RealmConfig.xml`
healthchecker	`healthchecker.txt`
threaddump	Three thread dump files, generated at 15-second intervals, named `threaddump_<timestamp>.txt`

The `-username` and `-password` parameters

When establishing the connection to a live realm server, the `RealmInformationCollector` tool will authenticate using the current operating system user. It is therefore recommended to run the `RealmInformationCollector` tool using the same user as the one used to run the realm server.

You can specify a different user using the `-username` and `-password` arguments.

The `-instance` parameter

This parameter allows you to select the set of realm servers to collect information from. The parameter is mandatory and must contain either a single realm server name or a comma-separated list of realm server names. The specified realm servers must be available in the installation where the `RealmInformationCollector` tool is run from. You can specify `-instance=*` to select all installed realm servers.

The `-include` and `-exclude` parameters

You can use the `-include` and `-exclude` parameters to further refine the set of collectors that have been selected by the `-mode` parameter. You can use `-include` and `-exclude` in the same call of the `RealmInformationCollector` tool as long as they do not specify the same collector name.

- **include** - Run all default collectors available with the specified `-mode` parameter, and additionally include the collector or collectors specified by this parameter. The parameter may contain a single collector name or a comma-separated list of collector names.
- **exclude** - Run all default collectors available with the specified `-mode` parameter, except the specified collector or collectors. The parameter may contain a single collector name or a comma-separated list of collector names.

The `-output` parameter

Specifies the path where the generated zip archive will be stored.

If the path specifies a directory without a filename, the directory must already exist. The archive file will be generated in the specified directory using the following naming convention:

```
<InstallDir>_<mode>_<timestamp>.zip
```

For example, if the product installation directory is `C:\SoftwareAG` and the `RealmInformationCollector` tool is executed with `-mode=live`, the generated archive will be named for example `SoftwareAG_live_20171120100757940.zip`

If the path specifies a directory with a filename, the directory must already exist but the file must not already exist, and the tool will use the filename you specify.

If the parameter is not specified, the tool will generate an archive with a name corresponding to the naming convention mentioned above, and store the archive under the directory `<InstallDir>/UniversalMessaging/tools/runner`.

Syntax Examples

Example: Execute default collectors in offline mode against the `umserver` instance:

```
runUMTool RealmInformationCollector -mode=offline -instance=umserver
```

Example: Execute default collectors and also optional collectors `data` and `heapdumps` in offline mode against the `umserver` instance:

```
runUMTool RealmInformationCollector
    -mode=offline -instance=umserver -include=data,heapdumps
```

Example: Execute default collectors and the optional collectors `data` and `heapdumps`, excluding the `jaas` collector, in offline mode against all realm server instances:

```
runUMTool RealmInformationCollector
    -mode=offline -instance=* -include=data,heapdumps -exclude=jaas
```

Example: Execute default collectors in live mode against the `umserver` instance:

```
runUMTool RealmInformationCollector -mode=live -instance=umserver
```

Example: Execute the default collectors and optional collectors `heapdump` and `heapdumps` in live mode against the `umserver` instance:

```
runUMTool RealmInformationCollector
    -mode=live -instance=umserver -include=heapdump,heapdumps
```

Example: Execute the default collectors and optional collectors `heapdump` and `heapdumps`, excluding the `jaas` collector, in live mode against the `umserver` and `umserver2` instances:

```
runUMTool RealmInformationCollector
   -mode=live -instance=umserver,umserver2 -include=heapdump,heapdumps -exclude=jaas
```

Example: Execute the default collectors and optional collectors `heapdump` and `heapdumps` in live mode against the `umserver` instance and specify a custom location of the generated zip archive:

```
runUMTool RealmInformationCollector
    -mode=live -instance=umserver -include=heapdump,heapdumps
    -outputfile=C:/SoftwareAG_umserver_live.zip
```

Operational Issues

- On Windows, if the product installation directory path is too long, acquiring a live heap dump may fail with the error "CreateProcess error=267, The directory name is invalid". You can work around this error by configuring the `-outputFile` parameter to use a shorter directory/file path, for example `C:/SoftwareAG_live.zip`.
- The `RealmInformationCollector` tool does not support connecting via SSL-secured network interfaces to the realm server. If all realm server network interfaces are secured using SSL, live collectors which need to connect to the server (`env`, `realmconfig`, `healthchecker`, `threaddump`) will fail to connect to the server. You can work around this by configuring a temporary non-SSL secured network interface.
- Live heap dump generation using the `heapdump` collector is only available with the JVM that is delivered with the Universal Messaging distribution kit on Windows and Solaris machines. This feature is currently not available for use with other JVMs.
- The `RealmInformationCollector` tool might fail to acquire a live heap dump if the tool run with a different operating system user than the one used for running the realm server. It is

recommended to run the tool with the same operating system user that was used to run the realm server.

The ExportEventsFromOfflineMemFile Tool

Overview

The `ExportEventsFromOfflineMemFile` tool is a command-line recovery tool that dumps events from mem files of Universal Messaging realm server stores to XML format. Selector based event filtering can be applied while dumping with the tool.

The output file will contain a first element describing the export details and then a list of events. The `ExportDetails` element contains the tool version number and the protocol buffer file descriptor set in base64-encoded format, if the protocol buffer file descriptor set was specified by the user when starting the export tool.

Event structure in the output file

In XML output files, the `Event` element contains the following information:

```
id: event EID (nPublished.getKey());
size: event size (nPublished.getSize());
chanID: event ChannelAttributesId (nPublished.getChannelAttributesId());
ttl: event TTL (nPublished.getTTL());
tag: event tag (nPublished.getTag());
eventData: event data;
persistent: event isPersistent property (nPublished.isPersistant());
transient: event isTransientPropery(nPublished.isTransient());
headerProps: event header attributes;
dictionaryProps: event dictionary properties;
```

For now, the XML output file event `eventData` value is base64-encoded.

XML output file example

```
<?xml version="1.0" encoding="utf-8"?>
<EventsDetails>
    <ExportDetails>
        <ToolVersion>1.0</ToolVersion>

[illegible]
    </ExportDetails>
    <Events>
        <Event>
            <id>0</id>
            <size>306</size>
            <chanID>63565653663178134</chanID>
            <ttl>0</ttl>
            <tag>tag0</tag>
            <eventData>dGVzdGRhdGFib2R5MCBkYXRh</eventData>
            <persistent>true</persistent>
            <transient>false</transient>
            <dictionaryProps>
                <item>
```

```
                <name>string_key</name>
                <value>value0</value>
                <type>String</type>
            </item>
            <item>
                <name>boolean_key</name>
                <value>true</value>
                <type>Boolean</type>
            </item>
            <item>
                <name>int_key</name>
                <value>0</value>
                <type>Integer</type>
            </item>
            <item>
                <name>long_key</name>
                <value>0</value>
                <type>Long</type>
            </item>
            <item>
                <name>short_key</name>
                <value>0</value>
                <type>Short</type>
            </item>
            <item>
                <name>byte_key</name>
                <value>0</value>
                <type>Byte</type>
            </item>
            <item>
                <name>char_key</name>
                <value>0</value>
                <type>Character</type>
            </item>
            <item>
                <name>byte_arr_key</name>
                <value>dGVzdDA=</value>
                <type>byte[]</type>
            </item>
            <item>
                <name>float_key</name>
                <value>0.0</value>
                <type>Float</type>
            </item>
            <item>
                <name>double_key</name>
                <value>0.0</value>
                <type>Double</type>
            </item>
        </dictionaryProps>
        <headerProps>
            <item>
                <name>nrvpub.time</name>
                <value>1588946429779</value>
                <type>Long</type>
            </item>
            <item>
                <name>nrvpub.host</name>
                <value>127.0.0.1</value>
                <type>String</type>
```

```
                </item>
                <item>
                    <name>nrvpub.name</name>
                    <value>ekob</value>
                    <type>String</type>
                </item>
                <item>
                    <name>JMSDeliveryMode</name>
                    <value>PERSISTENT</value>
                    <type>String</type>
                </item>
                <item>
                    <name>JMSPriority</name>
                    <value>4</value>
                    <type>Byte</type>
                </item>
            </headerProps>
        </Event>
    </Events>
</EventsDetails>
```

Input parameters

-protobufdescriptor

The `-protobufdescriptor` input parameter is an optional parameter specifying the path to the protocol buffer file descriptor set. In this version, filtering events based on protobuf event data is not performed. The protobuf descriptor will be exported to the XML format file in the `ExportDetails` section just for information.

The protocol buffer file descriptor set can be received as:

```
protoc.exe <proto_file_name>.proto
  --descriptor_set_out=<protocol_buffer_file_descriptor_set_name>.fds
```

If the input parameter `protobufdescriptor` is specified and the mem file's store is a protobuf channel with the same protobuf descriptor, the protocol buffer descriptor will be exported to the output file as a base64-encoded string.

If the input parameter `protobufdescriptor` is specified but the mem file's store is a protobuf channel with a different protobuf descriptor, then dumping will not be performed.

If the input parameter `protobufdescriptor` is specified but the mem file's store is not a protobuf channel, then the event will be considered as `nPublished` events anyway. The protocol buffer descriptor will be exported to the output file as a base64-encoded string.

batchsize

This parameter can be optionally specified when running the tool. It defines the number of events which will be read/loaded to the memory/from the mem file, filtered and then written to the output file as a single batch. The default value of the batch size is 100.

The RepublishEventsFromOfflineFile Tool

The `RepublishEventsFromOfflineFile` tool is a command-line recovery tool that imports events into a Universal Messaging realm server store (channel or queue) from any of the following sources:

- An XML file

  Importing can be done from an XML file produced by the `ExportEventsFromOfflineMemFile` tool. The XML file is a copy of the store's persistent memory file (or multiple memory files for a multi-file store). Selector based event filtering can be applied while importing with the tool. Republishing is done as transaction event publishing.

- Input files

  Importing can be done from offline memory files of the store taken from the parent realm's data directory. The offline memory files have the filetype `*.mem`. When importing multi-file stores, you specify the folder that contains the `*.mem` files. When importing a mixed/persistent store, you specify a single `.mem` file.

When you invoke the tool, you can specify either a mem file (or mem folder name), or an XML file, but not a combination of these options.

The -protobufdescriptor input parameter

Filtering based on protobuf event data will be performed starting from v10.3 and not performed in earlier versions.

Import from the mem file of a protobuf store

If the import is performed from a protobuf store's mem file and the `protobufdescriptor` parameter specified is the same as for the "source" channel, and the "republish" channel has the protocol buffer file descriptor set, events will be republished as protobuf events.

If the import is performed from a protobuf store's mem file and the `protobufdescriptor` parameter specified is the same as for the "source" channel, but the "republish" channel has another protocol buffer file descriptor or is not a protobuf channel, then event republishing will not be done.

If the import is performed from a protobuf store's mem file but the `protobufdescriptor` parameter is not specified, events will be republished as non-protobuf events to a non-protobuf store and as protobuf events to a store with the protocol buffer file descriptor set.

Import from mem file of non-protobuf store

If the import is performed from a non-protobuf store's mem file and the `protobufdescriptor` parameter is not specified, events will be republished as non-protobuf events to any channel.

If the import is performed from a non-protobuf store's mem file and the `protobufdescriptor` parameter is specified and coincides with the descriptor of the "republish" channel, events will be republished as non-protobuf events.

But if the import is performed from a non-protobuf store's mem file and the `protobufdescriptor` parameter is specified and does not coincide with the descriptor of the "republish" channel, events will not be republished.

Import from an XML file

If the source file belongs to (was exported from) a protobuf channel's mem file, events will be republished as protobuf events to a store with the protobuf descriptor and as non-protobuf events to other channels.

If the `protobufdescriptor` parameter is specified, the tool will validate if the "republish" channel has the same protocol buffer file descriptor set.

The -batchsize input parameter

This parameter can be optionally specified when running the tool. It defines the number of events which will be read/loaded to the memory/ from the mem/XML file, filtered and then published to the "destination" store as a single batch.

The default batch size is 100.

Syntax reference for command line tools

Syntax: Store Tools

CreateChannel

```
Tool name:
  CreateChannel
Description:
  Creates a channel with the specified name on the specified server.
  A single permission can be set during channel creation using
  optional arguments.
  For adding a set of permissions use the client API.

Usage:
  runUMTool CreateChannel -rname=<rname> -channelname=<channelname>
    [optional_args]

Examples:
  CreateChannel -rname=nsp://localhost:8080 -channelname=channel0
    -maxevents=10

Required arguments:
  rname :
    URL of the realm to which the channel will be connected.
  channelname :
    Name of the channel to be created.

Optional Parameters:
  maxevents :
    Capacity of the new store (default 0).
  ttl :
```

```
  Time to Live for the new store (default 0).
type :
  Type of the new store (default S).
  R - Reliable (stored in memory), with persistent EIDs
  P - Persistent (stored on disk)
  S - Simple (stored in memory)
  T - Transient (no server-based storage)
  M - Mixed (allows both memory and persistent events)
  O - Off-Heap
  G - Paged (uses a memory-mapped file for storage)
publishkeys :
  Set of publish keys for the new store (default null).
  Multiple pairs - each pair is separated by a ';'
  Pairs - each name and depth is separated by a ','
    e.g. name,depth;name2,depth2
isclusterwide :
  Whether the new store is cluster-wide.
  Will only work if the realm is part of a cluster.
usejmsengine :
  Whether to use the JMS style fanout engine.
usemergeengine :
  Whether to use the merge style fanout engine.
isautodelete :
  Whether the store is auto-deleted upon disconnection of its creator.
isdurable :
  Whether the store is durable (restores after a server restart).
isautomaintenance :
  Whether the store will have automatic maintenance as events are
  being removed.
honourcapacity :
  Whether the store capacity setting will prevent publishing of any more
  data once full.
enablecaching :
  Whether the server will cache events in memory or will always refer back
  to the file-backed store.
cacheonreload :
  Whether the server will cache events in memory for fast replay upon restart.
enablereadbuffering :
  Whether reads will be buffered to optimise the I/O access to the
  file-based store.
readbuffersize :
  The size in bytes of the buffer to use when read buffering (default 10240).
enablemulticast :
  Whether multicast is supported on the new store.
synceachwrite :
  Whether each write to the store will also call sync on the file system
  to ensure all data is written to disk.
syncbatchsize :
 Maximum size of batch written to disk on sync.
syncbatchtime :
  Time for writing data to disk on sync.
fanoutarchivetarget :
  Name of fanout archive target to be configured.
priority :
  The default message priority for events on the new store.
stampdictionary :
  StampDictionary setting value for the new store.
subject :
  The subject in format user@host for which the permission
  will be set. For a group permission, this value will be set
```

```
    as a group name. If this parameter is missing, the other parameters
    related to the permission entry are considered invalid.
  group :
    Whether a group permission entry must be created. Such permissions
    can be applied during channel creation only for already existing
    security groups.
  manage :
    Whether the subject or group has permissions to manage ACLs
    (default is set to false).
  publish :
    Whether the subject or group has permissions to publish
    events to this channel (default is set to false).
  subscribe :
    Whether the subject or group has permissions to subscribe to the
    channel (default is set to false).
  purge :
    Whether the subject or group has permissions to purge events
    from the channel (default is set to false).
  fullprivileges :
    Whether the subject or group has full permissions for this channel
   (default is set to false).
  getlasteid :
    Whether the subject or group has permissions to get the last event ID
   (default is set to false).
  named :
    Whether the subject or group has permissions to use named
    subscription on the channel (default is set to false).
  multifileeventsperspindle :
    Number of events that will be stored per individual file for a store
   (default is 50000).
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

CreateDurable

```
Tool name:
  CreateDurable
Description:
  Creates a Durable (also known as Named Object) with the specified name and type
  on the specified channel.

Usage:
  runUMTool CreateDurable -rname=<rname> -channelname=<channelname>
    -durablename=<durablename> -durabletype=<durabletype> [optional_args]

Examples:
  CreateDurable -rname=nsp://localhost:8080 -channelname=channel0
    -durablename=durable0 -durabletype=N

Required arguments:

  rname :
    URL of the realm to list the details of all the channels within.

  channelname :
    Name of the channel on which the Durable will be created.
  durablename :
```

```
    Name of the Durable to be created.
  durabletype :
    Type of the new Durable.
    N - Named
    P - Priority
    SQ - Shared queue
    S - Shared
    SE - Serial

Optional Parameters:

  isclusterwide :
    Whether the durable should be created in the entire cluster.
  username :
    Your Universal Messaging server username
  password :
    Your Universal Messaging server password
```

CreateJoin

```
Tool name:
  CreateJoin
Description:
  Joins two channels.

Usage:
  runUMTool CreateJoin -rname=<rname> -channelhost=<channelhost>
    -channeldest=<channeldest> [optional_args]

Examples:
  CreateJoin -rname=nsp://localhost:8080 -rnamedest=nsp://localhost:8090
    -channelhost=source -channeldest=destination

Required arguments:
  rname :
    URL of the realm from which the source channel will be retrieved.
  channelhost :
    Name of the source channel.
  channeldest :
    Name of the destination channel.

Optional Parameters:
  rnamedest :
    URL of the realm from which the destination channel
    will be retrieved (default is set to be -rname).

  routed :
    Set routed parameter (default is set to be false).
  hopcount :
    Set maximum number of hops (default is set to be 10).
  selector :
    Set selector string (default is set to be null).
  allowpurge :
    Set allowPurge parameter when connecting to a channel
    (default is set to be true).
  createtwoway :
    Set createtwoway parameter to create a two way
    channel join (default is set to be false)
  username :
```

```
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

CreateQueue

```
Tool name:
  CreateQueue
Description:
  Creates a queue with the specified name on the specified server.
  A single permission can be set during queue creation using
  optional arguments. For adding a set of permissions use the client API.

Usage:
  runUMTool CreateQueue -rname=<rname> -queuename=<queuename> [optional_args]

Examples:
  CreateQueue -rname=nsp://localhost:8080 -queuename=queue0 -maxevents=10

Required arguments:
  rname :
    URL of the realm to which the queue will be connected.

  queuename :
    Name of the queue to be created.

Optional Parameters:

  maxevents :
    Capacity of the new store (default 0).

  ttl :
    Time to Live for the new store (default 0).

  type :
    Type of the new store (default S).
    R - Reliable (stored in memory), with persistent EIDs
    P - Persistent (stored on disk)
    S - Simple (stored in memory)
    T - Transient (no server-based storage)
    M - Mixed (allows both memory and persistent events)
    O - Off-Heap
    G - Paged (uses a memory-mapped file for storage)

  isclusterwide :
    Whether the new store is cluster-wide.
    Will only work if the realm is part of a cluster.

  usejmsengine :
    Whether to use the JMS style fanout engine.
  usemergeengine :
    Whether to use the merge style fanout engine.
  isautodelete :
    Whether the store is auto-deleted upon disconnection of its creator.
  isdurable :
    Whether the store is durable (restores after a server restart).
  isautomaintenance :
    Whether the store will have automatic maintenance as
    events are being removed.
```

```
  honourcapacity : Whether the store capacity setting will prevent
    publishing of any more data once full.
  enablecaching :
    Whether the server will cache events in memory
    or will always refer back to the file-backed store.
  cacheonreload :
    Whether the server will cache events in memory for fast replay
    upon restart.
  enablereadbuffering :
    Whether reads will be buffered to optimise the I/O access
    to the file-based store.
  readbuffersize :
    The size in bytes of the buffer to use when read buffering
    (default 10240).
  enablemulticast :
    Whether multicast is supported on the new store.
  synceachwrite :
    Whether each write to the store will also call sync on the file system
    to ensure all data is written to disk.
  syncbatchsize :
    Maximum size of batch written to disk on sync.
  syncbatchtime :
    Time for writing data to disk on sync.
  fanoutarchivetarget :
    Name of fanout archive target to be configured.
  priority :
    The default message priority for events on the new store.
  stampdictionary :
    StampDictionary setting value for the new store.
  subject :
    The subject in format user@host for which the permission
    will be set. For a group permission this value will be
    set as a group name. If this parameter is missing the other
    parameters related to the permission entry are
    considered invalid.
  group :
    Whether a group permission entry must be created.
    Such permissions can be applied during channel creation only
    for already existing security groups.
  manage :
    Whether the subject or group has permissions to manage
    ACLs (default is set to false).
  fullprivileges : Whether the subject or group has full permissions
    for this channel (default is set to false).
  purge :
    Whether the subject or group has permissions to purge
    events from the channel (default is set to false).
  pop :
    Whether the subject or group has permissions to pop events
    from the queue (default is set to false).
  peek :
    Whether the subject or group has permissions to peek events
    from this queue (default is set to false).
  push :
    Whether the subject or group has permissions to push events
    in the queue (default is set to false).
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

DeleteChannel

```
Tool name:
  DeleteChannel
Description:
  Deletes a channel with the specified name on the specified realm.

Usage:
  runUMTool DeleteChannel -rname=<rname> -channelname=<channelname>
    [optional_args]

Examples:
  DeleteChannel -rname=nsp://localhost:8080 -channelname=channel0

Required arguments:
  rname :
    URL of the realm to which the channel will be connected.

  channelname :
    Name of the channel to be deleted.

Optional Parameters:
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

DeleteDurable

```
Tool name:
  DeleteDurable
Description:
  Deletes a Durable with the specified name on the specified channel.

Usage:
  runUMTool DeleteDurable -rname=<rname> -channelname=<channelname>
    -durablename=<durablename> [optional_args]

Examples:
  DeleteDurable -rname=nsp://localhost:8080 -channelname=channel0
    -durablename=durable0

Required arguments:

  rname :
    URL of the realm to list the details of all the channels within.
  channelname :
    Name of the channel from which the Durable will be deleted.
  durablename :
    Name of the Durable to be deleted.

Optional Parameters:
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

DeleteJoin

```
Tool name:
  DeleteJoin
Description:
  Deletes a join between two channels.

Usage:
  runUMTool DeleteJoin -rname=<rname> -channelhost=<channelhost>
    -channeldest=<channeldest> [optional_args]

Examples:
  DeleteJoin -rname=nsp://localhost:8080 -rnamedest=nsp://localhost:8090
    -channelhost=source -channeldest=destination

Required arguments:
  rname :
    URL of the realm from which the source channel will be retrieved.
  channelhost :
    Name of the source channel.
  channeldest :
    Name of the destination channel.

Optional Parameters:
  rnamedest :
    URL of the realm from which the destination channel
    will be retrieved (default is set to be -rname).
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

DeleteQueue

```
Tool name:
  DeleteQueue
Description:
  Deletes a queue with the specified name on the specified realm.

Usage:
  runUMTool DeleteQueue -rname=<rname> -queuename=<queuename> [optional_args]

Examples:
  DeleteQueue -rname=nsp://localhost:8080 -queuename=queue0

Required arguments:
  rname :
    URL of the realm to which the queue is connected.
  queuename :
    Name of the queue to be deleted.

Optional Parameters:
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

ExportProtobufDefinitions

```
Tool Name:
  ExportProtobufDefinitions
Description:
  Exports the protobuf definitions from a store with the specified name on the specified
 server.
Usage:
  runUMTool ExportProtobufDefinitions -rname=<rname> -storename=<storename>
    -dirname=<dirname> [optional_args]
Examples:
  ExportProtobufDefinitions -rname=nsp://localhost:9000 -storename=store0
    -dirname=/../build/change-management/test/protobuf/
Required arguments:

  rname :
    URL of the session to which the store will be connected.

  storename :
    Name of the store from which to export the protobuf definitions.

  dirname :
    Directory in which to save the exported definition files.
    If the directory that you entered does not exist, the tool creates the directory.
Optional Parameters:

  username  :
    Your Universal Messaging server username.
  password  :
    Your Universal Messaging server password.
```

GetChannelInfo

```
Tool name:
  GetChannelInfo
Description:
  Gets the attributes and storage properties of a specified channel
  in a specified realm.

Usage:
  runUMTool GetChannelInfo -rname=<rname> -cname=<cname> [optional_args]

Examples:
  GetChannelInfo -rname=nsp://localhost:8080 -cname=channel0 -format=plaintext

Required arguments:
  rname :
    URL of the realm to which the channel will be connected.
  cname :
    Name of the channel to return info for.

Optional Parameters:
  format :
    Format to print output in (plaintext/xml/json).
  username :
    Your Universal Messaging server username.

  password :
```

```
    Your Universal Messaging server password.
```

GetDurableInfo

```
Tool name:
  GetDurableInfo
Description:
  Gets the attributes of a specific Durable in a specific channel.

Usage:
  runUMTool GetDurableInfo -rname=<rname> -channelname=<channelname>
    -durablename=<durablename> [optional_args]

Examples:
  GetDurableInfo -rname=nsp://localhost:8080 -channelname=channel0
    -durablename=durable0 -format=plaintext

Required arguments:
  rname :
    URL of the realm to list the details of all the channels within.
  channelname :
    Name of the channel from where to get the Durable.
  durablename :
    Name of the Durable to return info for.

Optional Parameters:
  format :
    Format to print output in (plaintext/xml/json).
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

GetDurablesInfo

```
Tool name:
  GetDurablesInfo
Description:
  Displays the durables details saved in a .nsb file.

Usage:
  runUMTool GetDurablesInfo -nsbfileloc=<nsbfileloc> [optional_args]

Examples:
  GetDurablesInfo -nsbfileloc=C:\filepath

Required arguments:
  nsbfileloc :
    Absolute path for the nsb files location. This can be a folder
    which consists of multiple nsb files or a single nsb file.

Optional Parameters:
  textfileexport :
    Path to a text file in which the .nsb content will be saved.
    Must be an absolute path to a text file or the name of a
    text file which will be created in the working directory.
  username :
    Your Universal Messaging server username.
```

```
  password :
    Your Universal Messaging server password.
```

GetDurableStatus

```
Tool name:
  GetDurableStatus
Description:
  Gets the current state of durables on a realm, sorted by a given field.

Usage:
  runUMTool GetDurableStatus -rname=<rname> [optional_args]

Examples:
  GetDurableStatus -rname=nsp://localhost:8080 -sort=storesize -v=true

Required arguments:

  rname :
    URL of the realm to find durables for.

Optional Parameters:
  sort :
    Field to sort objects by. May be depth, depthtx, storesize,
    lasteid, lastread or lastwrite (default lastread).
  v :
    Whether the final output includes all fields or only
    the one specified. May be true or false (default false).
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

GetEventsInfo

```
Tool name:
  GetEventsInfo
Description:
  Display the events details present in the memory file.

Usage:
  runUMTool GetEventsInfo -memfileloc=<memfileloc> -storetype=<storetype>
    [optional_args]

Examples:
  GetEventsInfo -memfileloc=C:\filename -storetype=mixed

Required arguments:
  memfileloc :
    Absolute path for the memory files location. This can be a folder
    which consists of multiple memory files or a single memory file.
  storetype :
    Store type of channel/queue. It will be either Mixed or Persistent.

Optional Parameters:
  eventfactory :
    Option to specify the type of the Event factory, by default
    nServerEventFactory is used.
```

```
  perfmaintenance :
    Option to remove the free memory in the memory file (yes or no).
    UM server must be down during maintenance.
  additionalevtinfo :
    Option to get the additional event details (yes or no).
  exportfileformat :
    Option to specify the file format to export the event data.
    File formats supported are txt, xml.
  exportfilepath :
    Option to specify the absolute file path to export the event data.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

IdentifyLargeDurableOutstandingEvents

```
Tool name:
  IdentifyLargeDurableOutstandingEvents
Description:
  Identifies channels containing Durable with a large number
  of outstanding events.

Usage:
  runUMTool IdentifyLargeDurableOutstandingEvents -rname=<rname>
    -threshold=<threshold> [optional_args]

Examples:
  IdentifyLargeDurableOutstandingEvents -rname=nsp://localhost:8080
    -threshold=100

Required arguments:
  rname :
    URL of the realm to list the details of all the channels within.

  threshold :
    Long value representing the tolerated number of outstanding events.

Optional Parameters:
  username :
    Your Universal Messaging server username.

  password : Your Universal Messaging server password.
```

ListChannels

```
Tool name:
  ListChannels
Description:
  Lists details of the channels on the specified server.

Usage:
  runUMTool ListChannels -rname=<rname> [optional_args]

Examples:
  ListChannels -rname=nsp://localhost:8080

Required arguments:
```

```
  rname :
    URL of the realm to list the details of all the channels within.

Optional Parameters:

  format :
    Format to print output in (plaintext/xml/json).
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

ListJoins

```
Tool name:
  ListJoins
Description:
  Lists joins on a given realm.

Usage:
  runUMTool ListJoins -rname=<rname> [optional_args]

Examples:
  ListJoins -rname=nsp://localhost:8080 -v=true

Required arguments:
  rname :
    URL of the realm to which we will connect.

Optional Parameters:
  v :
    Output additional information for each join.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

MonitorChannels

```
Tool name:
  MonitorChannels
Description:
  Monitors the channels and queues in a realm and prints totals.

Usage:
  runUMTool MonitorChannels -rname=<rname> [optional_args]

Examples:
  MonitorChannels -rname=nsp://localhost:8080 -channelname=channel0
    -format=plaintext
  MonitorChannels -rname=nsp://localhost:8080 -channelname=queue1
    -format=plaintext

Required arguments:
 rname :
    URL of the realm to monitor channels and queues for.
```

```
Optional Parameters:
  channelname :
    Name of a specific channel or queue to monitor
  format :
    Format to print output in (plaintext/xml/json)
  username :
    Your Universal Messaging server username.
  password : Your Universal Messaging server password.
```

PurgeEvents

```
Tool name:
  PurgeEvents
Description:
  Purges events from a channel with the specified name on the
  specified realm.

Usage:
  runUMTool PurgeEvents -rname=<rname> -channelname=<channelname>
    [optional_args]

Examples:
  PurgeEvents -rname=nsp://localhost:8080 -channelname=channel0

Required arguments:
  rname :
    URL of the realm to which the channel will be connected.
  channelname :
    Name of the channel to be created.

Optional Parameters:
  starteid :
    Starting event ID of range to purge.
  endeid :
    Ending event ID of range to purge.
  selector :
    Selector query to filter which events to purge.
  purgejoins :
    Whether to purge events from joined channels.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

Syntax: Cluster Tools

ClusterState

```
Tool name:
  ClusterState
Description:
  Checks the cluster state by a given RNAME, which is part of a cluster.

Usage:
  runUMTool ClusterState -rname=<rname> [optional_args]
```

```
Examples:
  ClusterState -rname=nsp://localhost:8080

Required arguments:
  rname :
    Name of a realm, which is part of a cluster.

Optional Parameters:
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

CreateCluster

```
Tool name:
  CreateCluster
Description:
  Creates a cluster with the specified name, consisting of the specified realms.

Usage:
  runUMTool CreateCluster -clustername=<clustername> -convertlocal=<convertlocal>
    -rnames=<rnames> [optional_args]

Examples:
  CreateCluster -clustername=cluster0 -convertlocal=true
    -rnames=nsp://localhost:8080,nsp://localhost:9090

Required arguments:
  clustername :
    Name of the cluster to be created.
  convertlocal :
    Whether the local stores of the master should be converted
    to cluster-wide stores.
  rnames :
    Server URLs to be included in the cluster. Can be more than one,
    separated by a comma.
    The proper format is [nsp/nhp/nsps/nhps]://[hostname]:[port] or
    shm://[path/to/file].

Optional Parameters:
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

DumpClusterNamedObjectsState

```
Tool name:
  DumpClusterNamedObjectsState
Description:
  Dumps the state of named objects (also called durable subscriptions)
  on channels present on the specified cluster servers.

Usage:
  runUMTool DumpClusterNamedObjectsState -rnames=<rnames>
    -verbosemode=<verbosemode> [optional_args]
```

```
Examples:
  DumpClusterNamedObjectsState -rnames=nsp://localhost:8080,nsp://localhost:9090
    -verbosemode=true

Required arguments:
  rnames :
    Comma-separated list of rNames of clustered nodes.
  verbosemode :
    Set true to see all node states; set false to see only those
    with mismatched node states.

Optional Parameters:
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

Syntax: Interface Tools

AddHTTPInterface

```
Tool name:
  AddHTTPInterface
Description:
  Adds a HTTP interface on the specified adapter and port,
  on the specified realm.

Usage:
  runUMTool AddHTTPInterface -rname=<rname> -adapter=<adapter> -port=<port>
    [optional_args]

Examples:
  AddHTTPInterface -rname=nsp://localhost:8080 -adapter=0.0.0.0 -port=9090
    -usewebsockets=true

Required arguments:
  rname :
    URL of the realm to which the realm node, on which the interface
    will be created, is connected.
  adapter :
    Adapter (network card) to which interface will bind.
  port : Port on which the interface will listen.

Optional Parameters:
  usehttp1.1 :
    Whether to use HTTP1.1.
  usewebsockets :
    Whether WebSockets are used.
  ajaxactivedelay :
    Time to wait (for additional events) before delivering to
    Long Poll style subscribers.
  ajaxidledelay :
    Time to wait before returning from a Long Poll call if no events
    have been received.
  isnativecomet :
    Whether JavaScript is enabled on the interface.
  allowedorigins :
```

```
    Set the Allowed Origins for CORS as a comma-separated list of origins
    (use '*' to allow all), e.g. origin1,origin2,origin3
  crossorigincredentials :
    Whether to allow credentials header to be sent with CORS requests.
  enablegzip :
    Whether or not GZIP compression is enabled for javascript Long Poll
    connections.
  minimumbytes :
    Set the minimum number of bytes in a packet before GZIP
    is enabled (default 1000).
  autostart :
    Whether this interface will automatically be started when
    the realm server starts.
  advertise :
    Set the current advertise status for this interface.
  allowinterrealm :
    Whether this interface is allowed to be used in
    inter realm/cluster communication.
  allowclientconnections :
    Whether this interface can accept client connections or not.
  allownio :
    Whether NIO is enabled on the interface.
  authtimeout :
    Set the number of milliseconds that the remote client
    has to authenticate with the server.
  backlog :
    Set the number of connections to queue before the Operating System
    will send rejects to the remote client.
  threads :
    Set the thread pool size handling the client connections.
  selectthreads :
    Set the number of select threads used by NIO.
  adapteralias :
    Set the interface's alias.
  receivebuffersize :
    Set the socket buffer size in bytes used by this
    interface when receiving data.
  sendbuffersize :
    Set the socket buffer size in bytes used by this
    interface when sending data.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

AddHTTPSInterface

```
Tool name:
  AddHTTPSInterface
Description:
  Adds a HTTPS interface on the specified adapter and port,
  on the specified server.

Usage:
  runUMTool AddHTTPSInterface -rname=<rname> -adapter=<adapter> -port=<port>
   [optional_args]

Examples:
  AddHTTPSInterface -rname=nsp://localhost:8080 -adapter=0.0.0.0 -port=9090
```

```
    -alias=myAlias

Required arguments:

  rname :
    URL of the realm to which the realm node, on which the interface
    will be created, is connected.
  adapter :
    Adapter (network card) to which interface will bind.
  port :
    Port on which the interface will listen.

Optional Parameters:
  alias :
    Set the certificate name/alias that this interface will use to
    select its certificate from a keystore with multiple entries.
  keystore :
    Set the keystore file that this interface uses to load the certificate.
  kspassword :
    Set the keystore password that this interface will use to
    access the keystore file specified.
  truststore :
    Set the truststore file against which this interface will
    validate the client certificate.
  tspassword :
    Sets the truststore password that the server uses to access the
    trust store.
  privatepassword :
    Private key password; used so that the key can be loaded
    from the key store.
  ciphers :
    Names of the ciphers enabled for use by this interface, as a
    comma-separated list of ciphers, e.g. cipher1,cipher2,cipher3.
  rndalg :
    Set the SecureRandom algorithm to use for this interface.
  rndprov :
    Set the SecureRandom provider to use for this interface.
  clientcertrequired :
    Whether this interface requires SSL client authentication.
  crl :
    Set the certificate revocation list file name that the interface
    should use to check incoming SSL connections.
  crlclassname :
    Name of the class used to validate a client connection.
  provider :
    Name of the JSSE provider to use for the interface.
  usehttp1.1 :
    Whether to use HTTP1.1.
  usewebsockets :
    Whether WebSockets are used.
  ajaxactivedelay :
    Time to wait (for additional events) before delivering
    to Long Poll style subscribers.
  ajaxidledelay :
    Time to wait before returning from a Long Poll call if
    no events have been received.
  isnativecomet :
    Whether JavaScript is enabled on the interface.
  allowedorigins :
    Set the Allowed Origins for CORS as a comma-separated list of origins
```

```
    (use '*' to allow all), e.g. origin1,origin2,origin3.
  crossorigincredentials :
    Whether to allow credentials header to be sent with CORS requests.
  enablegzip :
    Whether or not GZIP compression is enabled for javascript
    Long Poll connections.
  minimumbytes :
    Set the minimum number of bytes in a packet before GZIP
    is enabled (default 1000).
  autostart :
    Whether this interface will automatically be started when
    the realm server starts.
  advertise :
    Set the current advertise status for this interface.
  allowinterrealm :
    Whether this interface is allowed to be used in
    inter realm/cluster communication.
  allowclientconnections :
    Whether this interface can accept client connections.
  allownio :
    Whether NIO is enabled on the interface.
  authtimeout :
    Set the number of milliseconds that the remote client
    has to authenticate with the server.
  backlog :
    Set the number of connections to queue before the Operating System
    will send rejects to the remote client.
  threads :
    Set the thread pool size handling the client connections.
  selectthreads :
    Set the number of select threads used by NIO.
  adapteralias :
    Set the interface's alias.
  receivebuffersize :
    Set the socket buffer size in bytes used by this
    interface when receiving data.
  sendbuffersize :
    Set the socket buffer size in bytes used by this interface
    when sending data.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

AddSHMInterface

```
Tool name:
  AddSHMInterface
Description:
  Adds a shared memory interface with the specified path, buffer size and
  timeout, on the specified server.

Usage:
  runUMTool AddSHMInterface -rname=<rname> -path=<path> [optional_args]

Examples:
  AddSHMInterface -rname=nsp://localhost:11000 -path=/dev/shm -buffer=1024
    -timeout=2000 -autostart=true
```

```
Required arguments:
  rname :
    URL of the realm to which the realm node, on which the interface
    will be created, is connected.
  path :
    The path where the shared memory files will be stored.

Optional Parameters:
  buffer :
    The size of the shared memory buffer which will be used. If not
    provided a default value of 1024000 will be used.
  timeout :
    The timeout value that will be used for read / write. If not
    provided a default value of 20000 will be used.
  autostart :
    Whether this interface will be automatically started when
    the Realm Server starts. Default is set to true.
  interrealmallow :
    Sets whether this interface is allowed to be used
    in inter realm / cluster communication. Default is set to false.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

AddSSLInterface

```
Tool name:
  AddSSLInterface
Description:
  Adds a SSL interface on the specified adapter and port,
  on the specified server.

Usage:
  runUMTool AddSSLInterface -rname=<rname> -adapter=<adapter> -port=<port>
    [optional_args]

Examples:
  AddSSLInterface -rname=nsp://localhost:8080 -adapter=0.0.0.0 -port=9090
    -alias=myAlias

Required arguments:
  rname :
    URL of the realm to which the realm node, on which the interface
    will be created, is connected.
  adapter :
    Adapter (network card) to which interface will bind.
  port :
    Port on which the interface will listen.

Optional Parameters:
  alias :
    Set the certificate name/alias that this interface will use to
    select its certificate from a keystore with multiple entries.
  keystore :
    Set the keystore file that this interface uses to load the
    certificate.
  kspassword :
    Set the keystore password that this interface will use to
```

```
  access the keystore file specified.
truststore :
  Set the truststore file against which this interface will
  validate the client certificate.
tspassword :
  Sets the truststore password that the server uses to access
  the trust store.
privatepassword :
  Private key password; used so that the key can be loaded
  from the key store.
ciphers :
  Names of the ciphers enabled for use by this interface,
  as a comma-separated list of ciphers, e.g. cipher1,cipher2,cipher3.
rndalg :
  Set the SecureRandom algorithm to use for this interface.
rndprov :
  Set the SecureRandom provider to use for this interface.
clientcertrequired :
  Whether this interface requires SSL client authentication.
crl :
  Set the certificate revocation list file name that the interface
  should use to check incoming SSL connections.
crlclassname :
  Name of the class used to validate a client connection.
provider :
  Name of the JSSE provider to use for the interface.
autostart :
  Whether this interface will automatically be started when the
  realm server starts.
advertise :
  Set the current advertise status for this interface.
allowinterrealm :
  Whether this interface is allowed to be used in
  inter realm/cluster communication.
allowclientconnections :
  Whether this interface can accept client connections.
allownio :
  Whether NIO is enabled on the interface.
authtimeout :
  Set the number of milliseconds that the remote client has to
  authenticate with the server.
backlog :
  Set the number of connections to queue before the Operating System
  will send rejects to the remote client.
threads :
  Set the thread pool size handling the client connections.
selectthreads :
  Set the number of select threads used by NIO.
adapteralias :
  Set the interface's alias.
receivebuffersize :
  Set the socket buffer size in bytes used by this
  interface when receiving data.
sendbuffersize :
  Set the socket buffer size in bytes used by this interface
  when sending data.
username :
  Your Universal Messaging server username.
password :
  Your Universal Messaging server password.
```

AddSocketInterface

```
Tool name:
  AddSocketInterface
Description:
  Adds a socket interface on the specified adapter and port,
  on the specified server.

Usage:
  runUMTool AddSocketInterface -rname=<rname> -adapter=<adapter> -port=<port>
    [optional_args]

Examples:
  AddSocketInterface -rname=nsp://localhost:8080 -adapter=0.0.0.0 -port=9090
    -autostart=true

Required arguments:
  rname :
    URL of the realm to which the realm node, on which the interface
    will be created, is connected.
  adapter :
    Adapter (network card) to which interface will bind.
  port :
    Port on which the interface will listen.

Optional Parameters:
  autostart :
    Whether this interface will automatically be started when the
    realm server starts.
  advertise :
    Set the current advertise status for this interface.
  allowinterrealm :
    Whether this interface is allowed to be used in
    inter realm/cluster communication.
  allowclientconnections :
    Whether this interface can accept client connections.
  allownio :
    Whether NIO is enabled on the interface.
  authtimeout :
    Set the number of milliseconds that the remote client has
    to authenticate with the server.
  backlog :
    Set the number of connections to queue before the Operating System
    will send rejects to the remote client.
  threads :
    Set the thread pool size handling the client connections.
  selectthreads :
    Set the number of select threads used by NIO.
  adapteralias :
    Set the interface's alias.
  receivebuffersize :
    Set the socket buffer size in bytes used by this
    interface when receiving data.
  sendbuffersize :
    Set the socket buffer size in bytes used by this interface
    when sending data.
  username :
    Your Universal Messaging server username.
  password :
```

```
    Your Universal Messaging server password.
```

DeleteInterface

```
Tool name:
  DeleteInterface
Description:
  Deletes the specified interface from the specified server.

Usage:
  runUMTool DeleteInterface -rname=<rname> -interface=<interface>
    [optional_args]

Examples:
  DeleteInterface -rname=nsp://localhost:8080 -interface=interface0

Required arguments:
  rname :
    URL of the realm to which the realm node, from which the interface
    will be deleted, is connected.
  interface :
    Name of the interface to be deleted.

Optional Parameters:
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

ModifyInterface

```
Tool name:
  ModifyInterface
Description:
  Modifies the specified interface on the specified server .

Usage:
  runUMTool ModifyInterface -rname=<rname> -interface=<interface>
    -command=<command> [optional_args]

Examples:
  ModifyInterface -rname=nsp://localhost:9000 -interface=interface0
    -command=modify -usewebsockets=true

Required arguments:
  rname :
    URL of the realm to which the realm node, on which the interface
    will be modified, is connected.
  interface :
    Name of the interface to be modified.
  command :
    Whether the interface is to be stopped (STOP), started (START),
    or have its fields modified (MODIFY).

Optional Parameters:
  usehttp1.1 :
    Whether to use HTTP1.1.
  usewebsockets :
```

```
    Whether WebSockets are used.
  ajaxactivedelay :
    Time to wait (for additional events) before delivering
    to Long Poll style subscribers.
  ajaxidledelay :
    Time to wait before returning from a Long Poll call if no
    events have been received.
  isnativecomet :
    Whether JavaScript is enabled on the interface.
  allowedorigins :
    Set the Allowed Origins for CORS as a comma-separated list of origins
    (use '*' to allow all), e.g. origin1,origin2,origin3.
  crossorigincredentials :
    Whether to allow credentials header to be sent with CORS requests.
  enablegzip :
    Whether or not GZIP compression is enabled for javascript
    Long Poll connections.
  minimumbytes :
    Set the minimum number of bytes in a packet before GZIP is
    enabled (default 1000).
  alias :
    Set the certificate name/alias that this interface will use to
    select its certificate from a keystore with multiple entries.
  keystore :
    Set the keystore file that this interface uses to load the certificate.
  kspassword :
    Set the keystore password that this interface will use to
    access the keystore file specified.
  truststore :
    Set the truststore file against which this interface will
    validate the client certificate.
  tspassword :
    Sets the truststore password that the server uses to access
    the trust store.
  privatepassword :
    Private key password; used so that the key can be loaded
    from the key store.
  ciphers :
    Names of the ciphers enabled for use by this interface,
    as a comma-separated list of ciphers, e.g. cipher1,cipher2,cipher3.
  rndalg :
    Set the SecureRandom algorithm to use for this interface.
  rndprov :
    Set the SecureRandom provider to use for this interface.
  clientcertrequired :
    Whether this interface requires SSL client authentication.
  crl :
    Set the certificate revocation list file name that the interface
    should use to check incoming SSL connections.
  crlclassname :
    Name of the class used to validate a client connection.
  provider :
    Name of the JSSE provider to use for the interface.
  autostart :
    Whether this interface will automatically be started when the
    realm server starts.
  advertise :
    Set the current advertise status for this interface.
  allowinterrealm :
    Whether this interface is allowed to be used in
```

```
    inter realm/cluster communication.
  allowclientconnections :
    Whether this interface can accept client connections.
  allownio :
    Whether NIO is enabled on the interface.
  authtimeout :
    Set the number of milliseconds that the remote client has
    to authenticate with the server.
  backlog :
    Set the number of connections to queue before the Operating System
    will send rejects to the remote client.
  threads :
    Set the thread pool size handling the client connections.
  selectthreads :
    Set the number of select threads used by NIO.
  adapteralias :
    Set the interface's alias.
  receivebuffersize :
    Set the socket buffer size in bytes used by this
    interface when receiving data.
  sendbuffersize :
    Set the socket buffer size in bytes used by this interface
    when sending data.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

Syntax: Data Group Tools

AddDataGroup

```
Tool name:
  AddDataGroup
Description:
  Adds a child data group to a parent data group.
  Both of these data groups must exist.

Usage:
  runUMTool AddDataGroup -rname=<rname> -datagroupname=<datagroupname>
    -parentname=<parentname> [optional_args]

Examples:
  AddDataGroup -rname=nsp://localhost:9000 -datagroupname=mydatagroup01
    -parentname=mydatagroup02

Required arguments:
  rname :
    Connection URL to the realm where the data groups exist.
  datagroupname :
    Name of the child data group.
  parentname :
    Name of the parent data group.

Optional Parameters:
  username :
    Your Universal Messaging server username.
```

```
  password :
    Your Universal Messaging server password.
```

CreateDataGroup

```
Tool name:
  CreateDataGroup
Description:
  Creates a data group with the specified name on the specified server.
  Additionally, conflation attributes and other options of the data group
  can be set.

Usage:
  runUMTool CreateDataGroup -rname=<rname> -datagroupname=<datagroupname>
    [optional_args]

Examples:
  CreateDataGroup -rname=nsp://localhost:9000 -datagroupname=mydatagroup01
    -confinterval=2000 -enablemulticast=true

Required arguments:
  rname :
    Connection URL to the realm where the data group will be created.
  datagroupname :
    Name of the data group to be created.

Optional Parameters:
  enablemulticast :
    Whether multicast is supported on the new data group.
  priority :
    The default message priority for events on the new data group.
  dropexpired :
    Don't send events that are made obsolete by newer ones.
  confinterval :
    Interval at which all the events are sent.
  confaction :
    Action to take when multiple events arrive for this data group.
    0 = drop old events
    1 = merge events
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

DeleteDataGroup

```
Tool name:
  DeleteDataGroup
Description:
  Removes the data group with the specified name from the server.

Usage:
  runUMTool DeleteDataGroup -rname=<rname> -datagroupname=<datagroupname>
    [optional_args]

Examples:
  DeleteDataGroup -rname=nsp://localhost:9000 -datagroupname=mydatagroup01
```

```
Required arguments:
  rname :
    Connection URL to the realm from which the data group will be deleted.
  datagroupname :
    Name of the data group to be deleted.

Optional Parameters:
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

ListenDataGroup

```
Tool name:
  ListenDataGroup
Description:
  Listens for data group events on a Universal Messaging realm.

Usage:
  runUMTool ListenDataGroup -rname=<rname> [optional_args]

Examples:
  ListenDataGroup -rname=nsp://localhost:9000

Required arguments:
  rname :
    Connection URL to the realm from which messages will be received.

Optional Parameters:
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

PublishDataGroup

```
Tool name:
  PublishDataGroup
Description:
  Publishes messages to a data group.

Usage:
  runUMTool PublishDataGroup -rname=<rname> -datagroupname=<datagroupname>
    [optional_args]

Examples:
  PublishDataGroup -rname=nsp://localhost:9000 -datagroupname=mydatagroup01
    -size=20

Required arguments:
  rname :
    Connection URL to the realm to which the messages will be published.
  datagroupname :
    Name of the data group to publish to.

Optional Parameters:
  message :
```

```
    Message to send. Put the message in quotes if it contains spaces.
    You can't use -size along with -message.
  size :
    Size of the message to send. Message will be generated.
    You can't use -message along with -size.
  count :
    How many times to send the event. Default 1.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

Syntax: Publish Tools

PublishChannel

```
Tool name:
  PublishChannel
Description:
  Publishes events to a Universal Messaging channel.

Usage:
  runUMTool PublishChannel -rname=<rname> -channelname=<channelname>
    [optional_args]

Examples:
  PublishChannel -rname=nsp://localhost:9000 -channelname=mychannel
    -message="hello world"

Required arguments:
  rname :
    Connection URL to the realm where the channel exists.
  channelname :
    Name of the channel on the Universal Messaging Realm.

Optional Parameters:
  message :
    Message to send. Put the message in quotes if it contains spaces.
    You can't use -size along with -message.
  count :
    How many times to send the event. Default 1.
  size :
    Size in bytes of the message to send. Message will be generated.
    You can't use -message along with -size.
  properties :
    Properties, if any, of the event. Expected syntax is
    "propertyName1:value1;propertyName2:value2",
    e.g. "shirt:green;price:80;sleeve:long".
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

PublishChannelXML

```
Tool name:
  PublishChannelXML
Description:
  Publishes an XML document to a Universal Messaging channel.

Usage:
  runUMTool PublishChannelXML -rname=<rname> -channelname=<channelname>
    -file=<file> [optional_args]

Examples:
  PublishChannelXML -rname=nsp://localhost:9000 -channelname=mychannel
    -file=C:\myDoc.xml

Required arguments:
  rname :
    Connection URL to the realm where the channel exists.
  channelname :
    Name of the channel on the Universal Messaging Realm.
  file :
    File path of the XML document to send.

Optional Parameters:
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

PublishCompressed

```
Tool name:
  PublishCompressed
Description:
  Publishes events to a store, using compression.

Usage:
  runUMTool PublishCompressed -rname=<rname> -storename=<storename>
    [optional_args]

Examples:
  PublishCompressed -rname=nsp://localhost:9000 -storename=mychannel
    -message="hello world"

Required arguments:
  rname :
    Connection URL to the realm where the store exists.
  storename :
    Name of the store or queue on the Universal Messaging Realm.

Optional Parameters:
  message :
    Message to send. Put the message in quotes if it contains spaces.
    You can't use -size along with -message.
  count :
    How many times to send the event. Default 1.
  size :
    Size in bytes of the message to send. Message will be generated.
```

```
    You can't use -message along with -size.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

PublishQueue

```
Tool name:
  PublishQueue
Description:
  Publishes events to a queue.

Usage:
  runUMTool PublishQueue -rname=<rname> -queuename=<queuename> [optional_args]

Examples:
  PublishQueue -rname=nsp://localhost:9000 -queuename=myqueue
    -message="hello world"

Required arguments:
  rname :
    Connection URL to the realm where the channel exists.
  queuename :
    Name of the queue on the Universal Messaging Realm.

Optional Parameters:
  message :
    Message to send. Put the message in quotes if it contains spaces.
    You can't use -size along with -message.
  count :
    How many times to send the event. Default 1.
  size :
    Size in bytes of the message to send. Message will be generated.
    You can't use -message along with -size.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

PublishTX

```
Tool name:
  PublishTX
Description:
  Publishes events, as a part of a transaction, to a Universal Messaging
  channel or queue.

Usage:
  runUMTool PublishTX -rname=<rname> -storename=<storename> [optional_args]

Examples:
  PublishTX -rname=nsp://localhost:9000 -storename=myStore
    -message="hello world" -count=20 -txsize=5
  PublishTX -rname=nsp://localhost:9000 -storename=myStore -size=2048

Required arguments:
  rname :
```

```
    Connection URL to the realm where the channel exists.
  storename :
    Name of the channel or queue on the Universal Messaging Realm.

Optional Parameters:
  message :
    Message to send. Put the message in quotes if it contains spaces.
    You can't use -size along with -message.
  count :
    How many times to send the event. Default 1.
  size :
    Size in bytes of the message to send. Message will be generated.
    You can't use -message along with -size.
  txsize :
    How many events to batch in a single transaction. Default 1.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

Syntax: Subscribe Tools

PeekQueue

```
Tool name:
  PeekQueue
Description:
  Peeks all events on a Universal Messaging queue and prints statistics for
  the bandwidth rates.

Usage:
  runUMTool PeekQueue -rname=<rname> -queuename=<queuename> [optional_args]

Examples:

  PeekQueue -rname=nsp://localhost:9000 -queuename=myqueue

Required arguments:
  rname :
    Connection URL to the realm where the channel exists.
  queuename :
    Name of a queue on the Universal Messaging Realm.

Optional Parameters:
  selector :
    Optional filter for the messages.
  statevents :
    How many events to peek before printing event statistics.
    Default 1000.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

SubscribeChannel

```
Tool name:
  SubscribeChannel
Description:
  Reads all the messages from a Universal Messaging channel.

Usage:
  runUMTool SubscribeChannel -rname=<rname> -channelname=<channelname>
    [optional_args]

Examples:
  SubscribeChannel -rname=nsp://localhost:9000 -channelname=channel

Required arguments:
  rname :
    Connection URL to the realm where the channel exists.
  channelname :
    Name of a channel on the Universal Messaging Realm.

Optional Parameters:
  selector :
    Optional filter for the messages.
  starteid :
    Starting EID of the messages to consume.
  statevents :
    How many events to peek before printing event statistics.
    Default 1000.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

SubscribeChannelAsync

```
Tool name:
  SubscribeChannelAsync
Description:
  Listens for messages on a Universal Messaging channel.

Usage:
  runUMTool SubscribeChannelAsync -rname=<rname> -channelname=<channelname>
    [optional_args]

Examples:
  SubscribeChannelAsync -rname=nsp://localhost:9000 -channelname=channel

Required arguments:
  rname :
    Connection URL to the realm where the channel exists.
  channelname :
    Name of a channel on the Universal Messaging Realm.

Optional Parameters:
  selector :
    Optional filter for the messages.
  starteid :
    Start event ID of the messages to consume.
```

```
  statevents :
    How many events to peek before printing event statistics.
    Default 1000.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

SubscribeChannelAsyncDurable

```
Tool name:
  SubscribeChannelAsyncDurable
Description:
  Listens for messages on a Universal Messaging channel.
  Running the tool with the same "-name" argument will continue reading
  from the last unconsumed event.

Usage:
  runUMTool SubscribeChannelAsyncDurable -rname=<rname>
    -channelname=<channelname> [optional_args]

Examples:
  SubscribeChannelAsyncDurable -rname=nsp://localhost:9000 -channelname=channel

Required arguments:
  rname :
    Connection URL to the realm where the channel exists.
  channelname :
    Name of a channel on the Universal Messaging Realm.

Optional Parameters:
  selector :
    Optional filter for the messages.
  name :
    Unique name of the durable subscriber. The name will be created
    if it doesn't exist. Default is "STGE".
  starteid :
    Start EID of the messages to consume.
  persistent :
    Whether the durable name will exist after Universal Messaging
    server reset. Default is false.
  clusterwide :
    Whether the durable name should be registered in the entire
    cluster. Default is false.
  autoack :
    Whether each event will be automatically acknowledged by the API.
    Default is true.
  statevents :
    How many events to peek before printing event statistics.
    Default 1000.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

SubscribeChannelDurable

```
Tool name:
```

```
  SubscribeChannelDurable
Description:
  Listens for messages on a Universal Messaging channel.
  Running the tool with the same "-name" argument will continue reading
  from the last unconsumed event.

Usage:
  runUMTool SubscribeChannelDurable -rname=<rname> -channelname=<channelname>
    [optional_args]

Examples:
  SubscribeChannelDurable -rname=nsp://localhost:9000 -channelname=channel

Required arguments:
  rname :
    Connection URL to the realm where the channel exists.
  channelname :
    Name of a channel on the Universal Messaging Realm.

Optional Parameters:
  selector :
    Optional filter for the messages.
  name :
    Unique name of the durable subscriber. The name will be created if
    it doesn't exist. Default is "STGE".
  starteid :
    Start event ID of the messages to consume.
  persistent :
    Whether the durable name will exist after Universal Messaging
    server reset. Default is false.
  clusterwide :
    Whether the durable name should be registered in the entire
    cluster. Default is false.
  autoack :
    Whether each event will be automatically acknowledged by the API.
    Default is true.
  statevents :
    How many events to peek before printing event statistics.
    Default 1000.
  timeout :
    Maximum wait time (milliseconds) when attempting to synchronously
    retrieve message from the server. Default is 1000ms.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

SubscribeCompressed

```
Tool name:
  SubscribeCompressed
Description:
  Listens for compressed messages on a Universal Messaging channel.

Usage:
  runUMTool SubscribeCompressed -rname=<rname> -storename=<storename>
    [optional_args]

Examples:
```

```
  SubscribeCompressed -rname=nsp://localhost:9000 -storename=channel

Required arguments:
  rname :
    Connection URL to the realm where the channel exists.
  storename :
    Name of a channel or queue on the Universal Messaging Realm.

Optional Parameters:
  selector :
    Optional filter for the messages.
  starteid :
    If the chosen store is a channel, only messages with ID greater
    than this will be consumed.
  statevents :
    How many events to peek before printing event statistics.
    Default 1000.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

SubscribeQueue

```
Tool name:
  SubscribeQueue
Description:
  Reads all the messages from a Universal Messaging queue.

Usage:
  runUMTool SubscribeQueue -rname=<rname> -queuename=<queuename> [optional_args]

Examples:
  SubscribeQueue -rname=nsp://localhost:9000 -queuename=myqueue

Required arguments:
  rname :
    Connection URL to the realm where the channel exists.
  queuename :
    Name of a queue on the Universal Messaging Realm.

Optional Parameters:
  selector :
    Optional filter for the messages.
  transacted :
    Set to true to use transacted subscriber. Default is false.
  statevents :
    How many events to peek before printing event statistics.
    Default 1000.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

SubscribeQueueAsync

```
Tool name:
  SubscribeQueueAsync
```

```
Description:
  Listens for messages on a Universal Messaging queue.

Usage:
  runUMTool SubscribeQueueAsync -rname=<rname> -queuename=<queuename>
   [optional_args]

Examples:
  SubscribeQueueAsync -rname=nsp://localhost:9000 -queuename=myqueue

Required arguments:
  rname :
    Connection URL to the realm where the channel exists.
  queuename :
    Name of a queue on the Universal Messaging Realm.

Optional Parameters:
  selector :
    Optional filter for the messages.
  transacted :
    Set to true to use transacted subscriber. Default is false.
  statevents :
    How many events to peek before printing event statistics.
    Default 1000.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

Syntax: Security Tools

AddChannelACLEntry

```
Tool name:
  AddChannelACLEntry
Description:
  Adds an ACL entry on the specified channel for the specified user and host,
  on the specified realm.

Usage:
  runUMTool AddChannelACLEntry -channelname=<channelname> -rname=<rname>
    -type=<type> [optional_args]

Examples:
  AddChannelACLEntry -rname=nsp://localhost:8080 -channelname=channel0
    -type=group -groupname=security_group0 -fullprivileges=true
  AddChannelACLEntry -rname=nsp://localhost:8080 -channelname=channel0
    -type=subject -user=username -host=127.0.0.1 -fullprivileges=true

Required arguments:
  channelname :
    Name of the channel to which the ACL entry is being applied.
  rname :
    URL of the server on which the channel exists.
  type :
    Type of ACL entry, either 'group' or 'subject'. If group is chosen,
    'groupname' must be set. Otherwise 'user' and 'host' must be set.
```

```
Optional Parameters:
  groupname :
    Name of the group for which ACL is being updated.
  user :
    User for which ACL is being updated.
  host :
    Host for which ACL is being updated.
  canlistacl :
    Specify that the 'list' ACL permission should be added.
  canmodifyacl :
    Specify that the 'modify' ACL permission should be added.
  fullprivileges :
    Specify that the 'full permissions' ACL permission should be added.
  cangetlasteid :
    Specify that the 'get last EID' ACL permission should be added.
  canread :
    Specify that the 'read' ACL permission should be added.
  canwrite :
    Specify that the 'write' ACL permission should be added.
  canpurge :
    Specify that the 'purge' ACL permission should be added.
  cannamed :
    Specify that the 'named' ACL permission should be added.
  canpop :
    Specify that the 'pop' ACL permission should be added.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

AddContainerACLEntry

```
Tool name:
  AddContainerACLEntry
Description:
  Adds an ACL entry on the specified container for the specified user and host.

Usage:
  runUMTool AddContainerACLEntry -containername=<containername> -rname=<rname>
    -type=<type> [optional_args]

Examples:
  AddContainerACLEntry -rname=nsp://localhost:8080 -containername=container0
    -type=group -groupname=security_group0 -fullprivileges=true

  AddContainerACLEntry -rname=nsp://localhost:8080 -containername=container0
    -type=subject -user=username -host=127.0.0.1 -fullprivileges=true

Required arguments:
  containername :
    Name of the container to which the ACL entry is being applied.
  rname :
    URL of the server from which to start searching for the container.
  type :
    Type of ACL entry, either 'group' or 'subject'. If group is chosen,
    'groupname' must be set. Otherwise 'user' and 'host' must be set.

Optional Parameters:
```

```
  groupname :
    Name of the group for which ACL is being updated.
  user :
    User for which ACL is being updated.
  host :
    Host for which ACL is being updated.
  canlistacl :
    Specify that the 'list' ACL permission should be added.
  canmodifyacl :
    Specify that the 'modify' ACL permission should be added.
  fullprivileges :
    Specify that the 'full permissions' ACL permission should be added.
  cangetlasteid :
    Specify that the 'get last EID' ACL permission should be added.
  canread :
    Specify that the 'read' ACL permission should be added.
  canwrite :
    Specify that the 'write' ACL permission should be added.
  canpurge :
    Specify that the 'purge' ACL permission should be added.
  cannamed :
    Specify that the 'named' ACL permission should be added.
  canpop :
    Specify that the 'pop' ACL permission should be added.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

AddQueueACLEntry

```
Tool name:
  AddQueueACLEntry
Description:
  Adds an ACL entry on the specified queue for the specified user and host,
  on the specified realm.

Usage:
  runUMTool AddQueueACLEntry -queuename=<queuename> -rname=<rname> -type=<type>
    [optional_args]

Examples:
  AddQueueACLEntry -rname=nsp://localhost:8080 -queuename=queue0 -type=group
    -groupname=security_group0 -fullprivileges=true
  AddQueueACLEntry -rname=nsp://localhost:8080 -queuename=queue0 -type=subject
    -user=username -host=127.0.0.1 -fullprivileges=true

Required arguments:
  queuename :
    Name of the queue to which the ACL entry is being applied.
  rname :
    URL of the server on which the queue exists.
  type :
    Type of ACL entry, either 'group' or 'subject'. If group is chosen,
   'groupname' must be set. Otherwise 'user' and 'host' must be set.

Optional Parameters:
  groupname :
    Name of the group for which ACL is being updated.
```

```
  user :
    User for which ACL is being updated.
  host :
    Host for which ACL is being updated.
  canlistacl :
    Specify that the 'list' ACL permission should be added.
  canmodifyacl :
    Specify that the 'modify' ACL permission should be added.
  fullprivileges :
    Specify that the 'full permissions' ACL permission should
    be added.
  canread :
    Specify that the 'read' ACL permission should be added.
  canwrite :
    Specify that the 'write' ACL permission should be added.
  canpurge :
    Specify that the 'purge' ACL permission should be added.
  canpop :
    Specify that the 'pop' ACL permission should be added.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

AddRealmACLEntry

```
Tool name:
  AddRealmACLEntry
Description:
  Adds an ACL entry on the specified realm for the specified user and host.

Usage:
  runUMTool AddRealmACLEntry -rname=<rname> -type=<type> [optional_args]

Examples:
  AddRealmACLEntry -rname=nsp://localhost:8080 -type=group
    -groupname=security_group0 -fullprivileges=true
  AddRealmACLEntry -rname=nsp://localhost:8080 -type=subject -user=username
    -host=127.0.0.1 -fullprivileges=true

Required arguments:

  rname :
    URL of the realm to which the ACL entry is being applied.
  type :
    Type of ACL entry, either 'group' or 'subject'. If group is chosen,
   'groupname' must be set. Otherwise 'user' and 'host' must be set.

Optional Parameters:
  groupname :
    Name of the group for which ACL is being updated.
  user :
    User for which ACL is being updated.
  host :
    Host for which ACL is being updated.
  canlistacl :
    Specify that the 'list' ACL permission should be added.
  canmodifyacl :
    Specify that the 'modify' ACL permission should be added.
```

```
  fullprivileges :
    Specify that the 'full permissions' ACL permission should be added.
  canuseadminapi :
    Specify that the 'use admin api' ACL permission should be added.
  canmanagerealms :
    Specify that the 'manage realms' ACL permission should be added.
  canmanagejoins :
    Specify that the 'manage joins' ACL permission should be added.
  canmanagechannels :
    Specify that the 'manage channels' ACL permission should be added.
  canaccess :
    Specify that the 'access' ACL permission should be added.
  canoverrideconnectioncount :
    Specify that the 'override connection count' ACL permission should be added.
  canconfigure :
    Specify that the 'configuration' ACL permission should be added.
  canmanagedatagroups :
    Specify that the 'manage data groups' ACL permission should be added.
  canpublishglobaldatagroups :
    Specify that the 'publish global data groups' ACL permission should
    be added.
  cantakeownershipdatagroups :
    Specify that the 'take ownership of data groups' ACL permission should
    be added.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

AddSecurityGroup

```
Tool name:
  AddSecurityGroup
Description:
  Adds a security group to the specified realm with the specified name.

Usage:
  runUMTool AddSecurityGroup -rname=<rname> -groupname=<groupname> [optional_args]

Examples:

  AddSecurityGroup -rname=nsp://localhost:8080 -groupname=security_group0

Required arguments:
  rname :
    URL of the realm to which the security group is being added.
  groupname :
    Name of the security group to be added.

Optional Parameters:
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

AddUserToSecurityGroup

```
Tool name:
```

```
  AddUserToSecurityGroup
Description:
  Adds a specified user and host subject to a given security group on a
  specified realm.

Usage:
  runUMTool AddUserToSecurityGroup -rname=<rname> -groupname=<groupname>
    -user=<user> -host=<host> [optional_args]

Examples:

  AddUserToSecurityGroup -rname=nsp://localhost:8080 -groupname=security_group0
    -user=username -host=127.0.0.1

Required arguments:
  rname :
    URL of the realm on which is the security group.
  groupname :
    Name of the security group to which the user is being added.
  user :
    User of the subject being added to security group.
  host :
    Host of the subject being added to security group.

Optional Parameters:
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

DeleteChannelACLEntry

```
Tool name:
  DeleteChannelACLEntry
Description:
  Deletes the ACL entry from the specified channel with the specified user
  and host.

Usage:
  runUMTool DeleteChannelACLEntry -channelname=<channelname> -rname=<rname>
    -type=<type> [optional_args]

Examples:
  DeleteChannelACLEntry -rname=nsp://localhost:8080 -channelname=channel0
    -type=group -groupname=security_group0
  DeleteChannelACLEntry -rname=nsp://localhost:8080 -channelname=channel0
    -type=subject -user=username -host=127.0.0.1

Required arguments:

  channelname :
    Name of the channel from which the ACL entry is being removed.
  rname :
    URL of the server on which the channel exists.
  type :
    Type of ACL entry, either 'group' or 'subject'. If group is chosen,
    'groupname' must be set. Otherwise 'user' and 'host' must be set.

Optional Parameters:
```

```
  groupname :
    Name of the group for which the ACL entry is being removed.
  user :
    User for which the ACL entry is being removed.
  host :
    Host for which the ACL entry is being removed.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

DeleteContainerACLEntry

```
Tool name:
  DeleteContainerACLEntry
Description:
  Removes an ACL entry from the specified container with the specified user
  and host.

Usage:
  runUMTool DeleteContainerACLEntry -containername=<containername> -rname=<rname>
    -type=<type> [optional_args]

Examples:
  DeleteContainerACLEntry -rname=nsp://localhost:8080 -containername=container0
    -type=group -groupname=security_group0
  DeleteContainerACLEntry -rname=nsp://localhost:8080 -containername=container0
    -type=subject -user=username -host=127.0.0.1

Required arguments:
  containername :
    Name of the container from which the ACL entry is being removed.
  rname :
    URL of the server from which to start searching for the container.
  type :
    Type of ACL entry, either 'group' or 'subject'. If group is chosen,
    'groupname' must be set. Otherwise 'user' and 'host' must be set.

Optional Parameters:
  groupname :
    Name of the group for which the ACL entry is being removed.
  user :
    User for which the ACL entry is being removed.
  host :
    Host for which the ACL entry is being removed.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

DeleteQueueACLEntry

```
Tool name:
  DeleteQueueACLEntry
Description:
  Deletes the ACL entry from the specified queue with the specified user and host.
```

```
Usage:
  runUMTool DeleteQueueACLEntry -queuename=<queuename> -rname=<rname>
   -type=<type> [optional_args]

Examples:
  DeleteQueueACLEntry -rname=nsp://localhost:8080 -queuename=queue0
    -type=group -groupname=security_group0
  DeleteQueueACLEntry -rname=nsp://localhost:8080 -queuename=queue0
    -type=subject -user=username -host=127.0.0.1

Required arguments:
  queuename :
    Name of the queue from which the ACL entry is being removed.
  rname :
    URL of the server on which the queue exists.
  type :
    Type of ACL entry, either 'group' or 'subject'. If group is chosen,
    'groupname' must be set. Otherwise 'user' and 'host' must be set.

Optional Parameters:
  groupname :
    Name of the group for which the ACL entry is being removed.
  user :
    User for which the ACL entry is being removed.
  host :
    Host for which the ACL entry is being removed.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

DeleteRealmACLEntry

```
Tool name:
  DeleteRealmACLEntry
Description:
  Removes an ACL entry from the specified realm with the specified user and host.

Usage:
  runUMTool DeleteRealmACLEntry -rname=<rname> -type=<type> [optional_args]

Examples:
  DeleteRealmACLEntry -rname=nsp://localhost:8080 -type=group
    -groupname=security_group0
  DeleteRealmACLEntry -rname=nsp://localhost:8080 -type=subject -user=username
    -host=127.0.0.1

Required arguments:
  rname :
    URL of the realm from which the ACL entry is being removed.
  type :
    Type of ACL entry, either 'group' or 'subject'. If group is chosen,
    'groupname' must be set. Otherwise 'user' and 'host' must be set.

Optional Parameters:
  groupname :
    Name of the group for which the ACL entry is being removed.
  user :
    User for which the ACL entry is being removed.
```

```
  host :
    Host for which the ACL entry is being removed.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

DeleteSecurityGroup

```
Tool name:
  DeleteSecurityGroup
Description:
  Removes a security group from the specified realm with the specified name.

Usage:
  runUMTool DeleteSecurityGroup -rname=<rname> -groupname=<groupname>
    [optional_args]

Examples:
  DeleteSecurityGroup -rname=nsp://localhost:8080 -groupname=security_groupp

Required arguments:
  rname :
    URL of the realm from which the security group is being removed.
  groupname :
    Name of the security group to be removed.

Optional Parameters:
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

DumpACL

```
Tool name:
  DumpACL
Description:
  Dumps all the ACL data for a realm.

Usage:
  runUMTool DumpACL -rname=<rname> [optional_args]

Examples:
  DumpACL -rname=nsp://localhost:8080
  DumpACL -rname=nsp://localhost:8080 -format=XML
  DumpACL -rname=nsp://localhost:8080 -format=JSON

Required arguments:
  rname :
    URL of the realm for which to dump the ACL data.

Optional Parameters:
  format :
    Which format to output ACL data. Defaults to plaintext, other
    options are: plaintext, xml, json.
  username :
    Your Universal Messaging server username.
```

```
  password :
    Your Universal Messaging server password.
```

ModifyChannelACLEntry

```
Tool name:
  ModifyChannelACLEntry
Description:
  Updates an ACL entry on the specified channel for the specified user and
  host, on the specified realm.

Usage:
  runUMTool ModifyChannelACLEntry -channelname=<channelname> -rname=<rname>
    -type=<type> [optional_args]

Examples:
  ModifyChannelACLEntry -rname=nsp://localhost:8080 -channelname=channel0
    -type=group -groupname=security_group0 -fullprivileges=true
  ModifyChannelACLEntry -rname=nsp://localhost:8080 -channelname=channel0
    -type=subject -user=username -host=127.0.0.1 -fullprivileges=true

Required arguments:
  channelname :
    Name of the channel on which the ACL entry is being updated.
  rname :
    URL of the server on which the channel exists.
  type :
    Type of ACL entry, either 'group' or 'subject'. If group is chosen,
    'groupname' must be set. Otherwise 'user' and 'host' must be set.

Optional Parameters:
  groupname :
    Name of the group for which ACL is being updated.
  user :
    User for which ACL is being updated.
  host :
    Host for which ACL is being updated.
  canlistacl :
    Specify that the 'list' ACL permission should be added.
  canmodifyacl :
    Specify that the 'modify' ACL permission should be added.
  fullprivileges :
    Specify that the 'full permissions' ACL permission should be added.
  cangetlasteid :
    Specify that the 'get last EID' ACL permission should be added.
  canread :
    Specify that the 'read' ACL permission should be added.
  canwrite :
    Specify that the 'write' ACL permission should be added.
  canpurge :
    Specify that the 'purge' ACL permission should be added.
  cannamed :
    Specify that the 'named' ACL permission should be added.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

ModifyContainerACLEntry

```
Tool name:
  ModifyContainerACLEntry
Description:
  AddContainerACLEntry adds an ACL entry on the specified container for the
  specified user and host.

Usage:
  runUMTool ModifyContainerACLEntry -containername=<containername> -rname=<rname>
    -type=<type> [optional_args]

Examples:
  ModifyContainerACLEntry -rname=nsp://localhost:8080 -containername=container0
    -type=group -groupname=security_group0 -fullprivileges=true
  ModifyContainerACLEntry -rname=nsp://localhost:8080 -containername=container0
    -type=subject -user=username -host=127.0.0.1 -fullprivileges=true

Required arguments:
  containername :
    Name of the container to which the ACL entry is being applied.
  rname :
    URL of the server from which to start searching for the container.
  type :
    Type of ACL entry, either 'group' or 'subject'. If group is chosen,
    'groupname' must be set. Otherwise 'user' and 'host' must be set.

Optional Parameters:
  groupname :
    Name of the group for which ACL is being updated.
  user :
    User for which ACL is being updated.
  host :
    Host for which ACL is being updated.
  canlistacl :
    Specify that the 'list' ACL permission should be added.
  canmodifyacl :
    Specify that the 'modify' ACL permission should be added.
  fullprivileges :
    Specify that the 'full permissions' ACL permission should be added.
  cangetlasteid :
    Specify that the 'get last EID' ACL permission should be added.
  canread :
    Specify that the 'read' ACL permission should be added.
  canwrite :
    Specify that the 'write' ACL permission should be added.
  canpurge :
    Specify that the 'purge' ACL permission should be added.
  cannamed :
    Specify that the 'named' ACL permission should be added.
  canpop :
    Specify that the 'pop' ACL permission should be added.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

ModifyQueueACLEntry

```
Tool name:
  ModifyQueueACLEntry
Description:
  Updates an ACL entry on the specified queue for the specified user and host,
  on the specified realm.

Usage:
  runUMTool ModifyQueueACLEntry -queuename=<queuename> -rname=<rname> -type=<type>
    [optional_args]

Examples:
  ModifyQueueACLEntry -rname=nsp://localhost:8080 -queuename=queue0 -type=group
    -groupname=security_group0 -fullprivileges=true
  ModifyQueueACLEntry -rname=nsp://localhost:8080 -queuename=queue0 -type=subject
    -user=username -host=127.0.0.1 -fullprivileges=true

Required arguments:
  queuename :
    Name of the queue on which the ACL entry is being updated.
  rname :
    URL of the server on which the queue exists.
  type :
    Type of ACL entry, either 'group' or 'subject'. If group is chosen,
    'groupname' must be set. Otherwise 'user' and 'host' must be set.

Optional Parameters:
  groupname :
    Name of the group for which ACL is being updated.
  user :
    User for which ACL is being updated.
  host :
    Host for which ACL is being updated.
  canlistacl :
    Specify that the 'list' ACL permission should be added.
  canmodifyacl :
    Specify that the 'modify' ACL permission should be added.
  fullprivileges :
    Specify that the 'full permissions' ACL permission should be added.
  canread :
    Specify that the 'read' ACL permission should be added.
  canwrite :
    Specify that the 'write' ACL permission should be added.
  canpurge :
    Specify that the 'purge' ACL permission should be added.
  canpop :
    Specify that the 'pop' ACL permission should be added.
  username :
    Your Universal Messaging server username.
  password : Your Universal Messaging server password.
```

ModifyRealmACLEntry

```
Tool name:
  ModifyRealmACLEntry
Description:
  Modifies an ACL entry on the specified realm for the specified user and host.
```

```
Usage:
  runUMTool ModifyRealmACLEntry -rname=<rname> -type=<type> [optional_args]

Examples:
  ModifyRealmACLEntry -rname=nsp://localhost:8080 -type=group
    -groupname=security_group0 -fullprivileges=true
  ModifyRealmACLEntry -rname=nsp://localhost:8080 -type=subject -user=username
    -host=127.0.0.1 -fullprivileges=true

Required arguments:
  rname :
    URL of the realm on which ACL is being updated.
  type :
    Type of ACL entry, either 'group' or 'subject'. If group is chosen,
    'groupname' must be set. Otherwise 'user' and 'host' must be set.

Optional Parameters:
  groupname :
    Name of the group for which ACL is being updated.
  user :
    User for which ACL is being updated.
  host :
    Host for which ACL is being updated.
  canlistacl :
    Specify that the 'list' ACL permission should be added.
  canmodifyacl :
    Specify that the 'modify' ACL permission should be added.

  fullprivileges :
    Specify that the 'full permissions' ACL permission should be added.
  canuseadminapi :
    Specify that the 'use admin api' ACL permission should be added.
  canmanagerealms :
    Specify that the 'manage realms' ACL permission should be added.
  canmanagejoins :
    Specify that the 'manage joins' ACL permission should be added.
  canmanagechannels :
    Specify that the 'manage channels' ACL permission should be added.
  canaccess :
    Specify that the 'access' ACL permission should be added.
  canoverrideconnectioncount :
    Specify that the 'override connection count' ACL permission should be added.
  canconfigure :
    Specify that the 'configuration' ACL permission should be added.
  canmanagedatagroups :
    Specify that the 'manage data groups' ACL permission should be added.
  canpublishglobaldatagroups :
    Specify that the 'publish global data groups' ACL permission should be added.
  cantakeownershipdatagroups :
    Specify that the 'take ownership of data groups' ACL permission should be added.

  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

RemoveUserFromSecurityGroup

```
Tool name:
  RemoveUserFromSecurityGroup
Description:
  Removes a specified user from a given security group on the specified realm.

Usage:
  runUMTool RemoveUserFromSecurityGroup -rname=<rname> -groupname=<groupname>
    -user=<user> -host=<host> [optional_args]

Examples:
  RemoveUserFromSecurityGroup -rname=nsp://localhost:8080
    -groupname=security_group0 -user=username -host=127.0.0.1

Required arguments:
  rname :
    URL of the realm on which the security group resides.
  groupname :
    Name of the security group user is being removed from.
  user :
    User being removed from security group.
  host :
    Host of subject being removed from security group.

Optional Parameters:
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

Syntax: Zone Tools

AddMemberToZone

```
Tool name:
  AddMemberToZone
Description:
  Adds a realm to a specified realm's zone.

Usage:
  runUMTool AddMemberToZone -rname=<rname> -zonememberrname=<zonememberrname>
    [optional_args]

Examples:
  AddMemberToZone -rname=nsp://localhost:8080 -zonememberrname=nsp://localhost:9090

Required arguments:
  rname :
    URL of the realm you want to add to the zone.
  zonememberrname :
    URL of a realm in the zone you want to expand.

Optional Parameters:
  username :
    Your Universal Messaging server username.
```

```
  password :
    Your Universal Messaging server password.
```

CreateZone

```
Tool name:
  CreateZone
Description:
  Creates a zone with the specified name containing the specified realms.

Usage:
  runUMTool CreateZone -rnames=<rnames> -zonename=<zonename> [optional_args]

Examples:
  CreateZone -rnames=nsp://localhost:8080,nsp://localhost:9090 -zonename=zone0

Required arguments:
  rnames :
    Comma separated list of URLs of the realms which the zone will contain.
  zonename :
    Name of the zone to be created.

Optional Parameters:
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

DeleteZone

```
Tool name:
  DeleteZone
Description:
  Deletes a zone with the specified name on the specified realm.

Usage:
  runUMTool DeleteZone -rname=<rname> [optional_args]

Examples:
  DeleteZone -rname=nsp://localhost:8080
  DeleteZone -rname=nsp://localhost:8080 -removejoins=true

Required arguments:
  rname :
    URL of a realm which belongs to the zone to be deleted.

Optional Parameters:
  removejoins :
    Whether to remove intra-zone connections when the zone
    is deleted. Defaults to false.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

RemoveMemberFromZone

```
Tool name:
  RemoveMemberFromZone
Description:
  Removes a realm from its current zone.

Usage:
  runUMTool RemoveMemberFromZone -rname=<rname> [optional_args]

Examples:
  RemoveMemberFromZone -rname=nsp://localhost:8080
  RemoveMemberFromZone -rname=nsp://localhost:8080 -removejoins=true

Required arguments:
  rname :
    URL of the realm you want to remove from the zone.

Optional Parameters:
  removejoins :
    Whether or not to remove realm links with the former zone member.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

Syntax: JMS Tools

Note:
The JMS Client for Universal Messaging supports JMS 1.1. Currently we do not support JMS 2.0.

CreateConnectionFactory

```
Tool name:
  CreateConnectionFactory
Description:
  Creates a JMS connection factory with the specified server.

Usage:
  runUMTool CreateConnectionFactory -rname=<rname> -factoryname=<factoryname>
    -destinationname=<destinationname> [optional_args]

Examples:
  CreateConnectionFactory -rname=nsp://localhost:8080 -factoryname=factory0
    -destinationname=channel0

Required arguments:
  rname :
    URL of the realm to which the ConnectionFactory is attached.
  factoryname :
    Name of the connection factory to create.
  destinationname :
    JMS Destination, only required for AMPQ connections.
```

```
Optional Parameters:
  factorytype :
    Connection factory type to be created. By default, it is ConnectionFactory
    if no parameter is passed.
    Possible values are:
      -factorytype=default (Creates a ConnectionFactory)
      -factorytype=queue (Creates a QueueConnectionFactory)
      -factorytype=topic (Creates a TopicConnectionFactory)
      -factorytype=xa (Creates an XAConnectionFactory)

  providerurl :
    URL of the local realm from which JNDI entries will be looked up.
    When using an AMPQ connection this should be changed to:
      amqp://localhost:9000
        when using a plain AMPQ connection, connecting to
        a standard nsp interface
     amqps://localhost:9000
        when using a plain AMPQ connection over a
        secure socket connection, connecting to a nsps interface
    Hint: AMPQ connections work only over nsp and nsps interfaces, they won't
    work over nhp or nhps .
  contextfactory :
    The name of the ContextFactory class to use
    (default: com.pcbsys.nirvana.nSpace.NirvanaContextFactory).
     When using an AMPQ connection this should be changed to:
     org.apache.qpid.jms.jndi.JmsInitialContextFactory
       to use the QPID Proton JMS Client libraries;
     org.apache.qpid.amqp_1_0.jms.jndi.PropertiesFileInitialContextFactory
       to use the QPID Legacy JMS Client libraries
    Hint: AMPQ connections work only over nsp and nsps interfaces, they won't
    work over nhp or nhps.
  autocreateresource :
    If set will create resources on the server when performing a lookup
    on a queue or channel.
  synctopicacks :
    Specifies for a specific connection whether the topic acknowledgements
    will be sent synchronously.
  syncqueueacks :
    Specifies for a specific connection whether the queue acknowledgements
    will be sent synchronously.
  writehandler :
    Specifies for a specific connection the write handler type to use.
    Ignored unless between 1-4.
  adapterbuffer :
    Specifies for a specific connection the adapter send / receive buffer size.
    If the value is 0 or less it is ignored .
  syncnamedtopicacks :
    Specifies for a specific connection whether the durable topic
    acknowledgements will be sent synchronously.
  permittedkeepalivesmissed :
    Set the number of keep server keep alives the client is allowed to miss
    before detecting a network issue and terminating the connection.
  connectiontimeout :
    Set the timeout used for connection / reconnection to realms.
    If the connection fails to establish within this time-frame it will fail.
  disconnectafterclusterfailure :
    If connected to a cluster of realms, and cluster quorum is lost,
    this flag determines whether the client will be disconnected.
  usejmsengine :
    JMS engine ensures no events are available for topic replay
```

```
  unless durable subscriptions are being used. Non JMS engine fanout enables
  events to be stored even after events are delivered. Default is true.
reconnectinterval :
  If a client is disconnected, and Immediate Reconnect is set to true,
  this value represents the interval between reconnect attempts.
immediatereconnect :
  If a client is disconnected, this flag will indicate whether the client
  will immediately reconnect and attempt to reconnect as fast as possible,
  rather than rely on a back off period.
autoreconnectafteracl :
  If a client is disconnected because of a security change, this flag
  will indicate whether the automatic session reconnection logic will kick in.
threadpoolsize :
  Gets the maximum number of threads used by the client for delivery of
  all messages to listeners.
redeliveredsize :
  Specifies the maximum number of messages that the client will keep
  reference to if they are marked as redelivered.
unackedsize :
  The client will keep a list of messages that have not been
  acknowledged. This value sets the maximum size of this list.
useinfinitewindowsize :
  When set to true, the consumer can consume as many events as required
  before committing.
windowsize :
  When synchronously consuming messages from the server, they will be
  delivered in batches (windows). This property sets the size of that window.
autoackcount :
  With AUTO acknowledgement mode, in order to improve performance,
  the acknowledgement of messages can be batched so that not every message
  consumed results in communication with the server. This value determines
  how many events can be consumed before an acknowledgement is sent to the
  server.
globalstorecapacity :
  Each topic or queue store can have a maximum number of messages that
  can exist before no more messages are allowed to be published.
synctime :
  When file sync is set, you can buffer the sync calls into batches
  in order to prevent the underlying system from being overloaded during
  busy periods. This value specifies the maximum time in milliseconds between
  sync calls. The smaller the value, the more frequent the sync will be
  called on the physical file system.
syncbatchsize :
  When file sync is set, you can buffer the sync calls into batches
  in order to prevent the underlying system from being overloaded
  during busy periods. This value specifies the number of messages in
  each batch. The smaller the value, the more frequent the sync will
  be called on the physical file system.
initialconnectionretrycount :
  When a connection is first established, the default number of
  connection attempts is 2. This allows this value to be overridden.
syncsendpersistent :
  For each persistent message written to the server, ensure the send
  is a synchronous call.
syncwritestodisc :
  For each persistent message written to the server, perform a
  file system sync to ensure the OS has written the data.
enabledurablepriority :
  If enabled, durable subscriptions of the same name can exist on
  the same topic, but only the first in will consume the events for
```

```
    that subscription.
  enablesinglequeueack :
    If enabled, message acknowledgements on a queue consumer will
    only acknowledge that specific message rather than all messages
    consumed prior to that message, on that queue.
  enablesinglesharedddurableack :
    If enabled, message acknowledgements on a shared durable consumer
    will only acknowledge that specific message rather than all messages
    consumed prior to that message, on that shared durable.
  enablesharedddurable :
    If enabled, durable subscriptions of the same name can exist on
    the same topic, and events will be distributed in a round robin
    fashion to each subscriber using that name (i.e. once and once only per
    durable name).
  maxreconnectattempts :
    When getConxExceptionOnFailure() is enabled, this value is used to
    prevent the disconnection exceptions being thrown via the
    ExceptionListener on the JMS Connection. The default value is -1, which
    represents infinite retries.
  conxexceptiononretryfailure :
    When enabled, any disconnections from the JMS Connection will not result
    in an Exception being generated through the ExceptionListener.
    An exception will only be thrown to the ExceptionListener when the
    getMaxReconAttempts() value is reached.
  retrycommit :
    Determines whether the commit call to a transacted session will
    retry if any exceptions are detected, rather than simply throw an exception.
  randomrnames :
    Allows the list of RNAME urls to be randomised to provide simple load
    balancing across a list of servers.
  enablemultiplexedconnections :
    Support the use of a shared physical connection by multiple sessions
    when the same topic or queue is used by multiple receivers.
  durabletype :
    Type of the new Durable:
    N - Named
    P - Priority
    SQ - Shared queue
    S - Shared
    Serial - Serial
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

CreateJMSQueue

```
Tool name:
  CreateJMSQueue
Description:
  Creates a JMS queue with the specified name on the specified realm.

Usage:
  runUMTool CreateJMSQueue -rname=<rname> -factoryname=<factoryname>
   -queuename=<queuename> [optional_args]

Examples:
  CreateJMSQueue -rname=nsp://localhost:8080 -factoryname=factory0
   -queuename=queue0
```

```
Required arguments:
  rname :
    URL of the realm to which the queue will be connected.
  factoryname :
    Name of the connection factory to locate.
  queuename :
    Name of the queue to be created.

Optional Parameters:
  maxevents :
    Capacity of the new store (default 0).
  synceachwrite :
    Whether each write to the store will also call sync on the
    file system to ensure all data is written to disk.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

CreateJMSTopic

```
Tool name:
  CreateJMSTopic
Description:
  Creates a JMS topic with the specified name on the specified realm.

Usage:
  runUMTool CreateJMSTopic -rname=<rname> -factoryname=<factoryname>
    -channelname=<channelname> [optional_args]

Examples:
  CreateJMSTopic -rname=nsp://localhost:8080 -factoryname=factory0
    -channelname=channel0

Required arguments:
  rname :
    URL of the realm to which the channel will be connected.
  factoryname :
    Name of the connection factory to locate.
  channelname :
    Name of the channel to be created.

Optional Parameters:
  maxevents :
    Capacity of the new store (default 0).
  synceachwrite :
    Whether each write to the store will also call sync on the
    file system to ensure all data is written to disk.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

JMSPublish

```
Tool name:
  JMSPublish
```

```
Description:
  Publishes one or more messages to a JMS queue or topic.

Usage:
  runUMTool JMSPublish -rname=<rname> -connectionfactory=<connectionfactory>
    -destination=<destination> [optional_args]

Examples:
  JMSPublish -rname=nsp://localhost:9000 -connectionfactory=factory
    -destination=topic -message=hello

Required arguments:
  rname :
    Connection URL to the realm where the data group will be created.
  connectionfactory :
    Name of the connection factory in the Universal Messaging Realm's
    JNDI namespace. Must exist.
  destination :
    Name of the JMS destination (queue or topic). Must exist.

Optional Parameters:
  size :
    Size of the message to send. Message will be generated. You can't use
    -message along with -size.
  message :
    Message to send. Put the message in quotes if it contains spaces.
    You can't use -size along with -message.
  count :
    How many times to send the message. Default is 1.
  transacted :
    If the session is transacted. Default is false.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

JMSSubscribe

```
Tool name:
  JMSSubscribe
Description:
  Reads messages arriving to a JMS destination.

Usage:
 runUMTool JMSSubscribe -rname=<rname> -connectionfactory=<connectionfactory>
   -destination=<destination> [optional_args]

Examples:
 JMSSubscribe -rname=nsp://localhost:9000 -connectionfactory=factory
   -destination=topic

Required arguments:
  rname :
    Connection URL to the realm where the data group will be created.
  connectionfactory :
    Name of the connection factory in the Universal Messaging Realm's
    JNDI namespace. Must exist.
  destination :
    Name of the JMS destination (queue or topic). Must exist.
```

```
Optional Parameters:
  transacted :
    If the session is transacted. Default is false.
  selector :
    Optional JMS message selector.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

ModifyConnectionFactory

```
Tool name:
  ModifyConnectionFactory
Description:
  Modifies settings of a JMS connection factory on the specified server.

Usage:
  runUMTool ModifyConnectionFactory -rname=<rname> -factoryname=<factoryname>
   -destinationname=<destinationname> [optional_args]

Examples:
  ModifyConnectionFactory -rname=nsp://localhost:8080 -factoryname=factory0
   -destination=channel0

Required arguments:
  rname :
    URL of the realm to which the ConnectionFactory is attached.
  factoryname :
    Name of the connection factory to locate.
  destinationname :
    JMS Destination, only required for AMPQ connections.

Optional Parameters:
  providerurl :
    URL of the local realm from which JNDI entries will be looked up.
    When using an AMPQ connection this should be changed to:
      amqp://localhost:9000
        when using a plain AMPQ connection, connecting to a
       standard nsp interface.
     amqps://localhost:9000
        when using a plain AMPQ connection over a secure socket connection,
        connecting to a nsps interface.
    Hint: AMPQ connections work only over nsp and nsps interfaces, they won't
    work over nhp or nhps.
  contextfactory :
    The name of the ContextFactory class to use
    (default: com.pcbsys.nirvana.nSpace.NirvanaContextFactory)
    When using an AMPQ connection this should be changed to:
      org.apache.qpid.jms.jndi.JmsInitialContextFactory
        to use the QPID Proton JMS Client libraries;
     org.apache.qpid.amqp_1_0.jms.jndi.PropertiesFileInitialContextFactory
        to use the QPID Legacy JMS Client libraries.
    Hint: AMPQ connections work only over nsp and nsps interfaces, they won't
    work over nhp or nhps.
  autocreateresource :
    If set will create resources on the server when performing a lookup
    on a queue or channel.
```

```
synctopicacks :
  Specifies for a specific connection whether the topic
  acknowledgements will be sent synchronously.
syncqueueacks :
  Specifies for a specific connection whether the queue
  acknowledgements will be sent synchronously.
writehandler :
  Specifies for a specific connection the write handler type to use.
  Ignored unless between 1-4.
adapterbuffer :
  Specifies for a specific connection the adapter send / receive
  buffer size. If the value is 0 or less it is ignored.
syncnamedtopicacks :
  Specifies for a specific connection whether the durable
  topic acknowledgements will be sent synchronously.
permittedkeepalivesmissed :
  Set the number of keep server keep alives the client is allowed
  to miss before detecting a network issue and terminating the connection.
connectiontimeout :
  Set the timeout used for connection / reconnection to realms.
  If the connection fails to establish within this time-frame it will
  fail.
disconnectafterclusterfailure :
  If connected to a cluster of realms, and cluster quorum is lost,
  this flag determines whether the client will be disconnected.
usejmsengine :
  JMS engine ensures no events are available for topic replay
  unless durable subscriptions are being used. Non JMS engine fanout enables
  events to be stored even after events are delivered. Default is true.
reconnectinterval :
  If a client is disconnected, and Immediate Reconnect is set to true,
  this value represents the interval between reconnect attempts.
immediatereconnect :
  If a client is disconnected, this flag will indicate whether the client
  will immediately reconnect and attempt to reconnect as fast as possible,
  rather than rely on a back off period.
autoreconnectafteracl :
  If a client is disconnected because of a security change, this flag
  will indicate whether the automatic session reconnection logic will
  kick in.
threadpoolsize :
  Gets the maximum number of threads used by the client for
  delivery of all messages to listeners.
redeliveredsize :
  Specifies the maximum number of messages that the client
  will keep reference to if they are marked as redelivered.
unackedsize :
  The client will keep a list of messages that have not been
  acknowledged. This value sets the maximum size of this list.
useinfinitewindowsize :
  When set to true, the consumer can consume as many events as required
  before committing.
windowsize :
  When synchronously consuming messages from the server, they
  will be delivered in batches (windows). This property sets the size of that
  window.
autoackcount :
  With AUTO acknowledgement mode, in order to improve performance,
  the acknowledgement of messages can be batched so that not every message
  consumed results in communication with the server. This value determines
```

```
  how many events can be consumed before an acknowledgement is sent to the
  server.
globalstorecapacity :
  Each topic or queue store can have a maximum number of messages that
  can exist before no more messages are allowed to be published.
synctime :
  When file sync is set, you can buffer the sync calls into batches
  in order to prevent the underlying system from being overloaded during
  busy periods. This value specifies the maximum time in milliseconds between
  sync calls. The smaller the value, the more frequent the sync will be
  called on the physical file system.
syncbatchsize :
  When file sync is set, you can buffer the sync calls into
  batches in order to prevent the underlying system from being overloaded
  during busy periods. This value specifies the number of messages in each
  batch. The smaller the value, the more frequent the sync will be called on
  the physical file system.
initialconnectionretrycount :
  When a connection is first established, the default number of
  connection attempts is 2. This allows this value to be overridden.
syncsendpersistent :
  For each persistent message written to the server, ensure the send
  is a synchronous call.
syncwritestodisc :
  For each persistent message written to the server, perform a file system
  sync to ensure the OS has written the data.
enabledurablepriority :
  If enabled, durable subscriptions of the same name can exist on the
  same topic, but only the first in will consume the events for that subscription.

enablesinglequeueack :
  If enabled, message acknowledgements on a queue consumer will only
  acknowledge that specific message rather than all messages consumed prior
  to that message, on that queue.
enablesinglesharedurableack :
  If enabled, message acknowledgements on a shared durable consumer will
  only acknowledge that specific message rather than all messages
  consumed prior to that message, on that shared durable.
enablesharedurable :
  If enabled, durable subscriptions of the same name can
  exist on the same topic, and events will be distributed in a round robin
  fashion to each subscriber using that name (ie once and once only per
  durable name).
maxreconnectattempts :
  When getConxExceptionOnFailure() is enabled, this value is used to
  prevent the disconnection exceptions being thrown via the
  ExceptionListener on the JMS Connection. The default value is -1, which
  represents infinite retries.
conxexceptiononretryfailure :
  When enabled, any disconnections from the JMS Connection will not
  result in an Exception being generated through the ExceptionListener.
  An exception will only be thrown to the ExceptionListener when the
  getMaxReconAttempts() value is reached.
retrycommit :
  Determines whether the commit call to a transacted session will
  retry if any exceptions are detected, rather than simply throw an exception.
randomrnames :
  Allows the list of RNAME urls to be randomised to provide simple
  load balancing across a list of servers.
enablemultiplexedconnections :
```

```
    Support the use of a shared physical connection by multiple sessions
    when the same topic or queue is used by multiple receivers.
  durabletype :
    Type of the new Durable:
    N - Named
    P - Priority
    SQ - Shared queue
    S - Shared
    Serial - Serial
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

ViewConnectionFactory

```
Tool name:
  ViewConnectionFactory
Description:
  Views settings of a JMS connection factory on the specified server.

Usage:
  runUMTool ViewConnectionFactory -rname=<rname> -factoryname=<factoryname>
    -destinationname=<destinationname> [optional_args]

Examples:
  ViewConnectionFactory -rname=nsp://localhost:8080 -factoryname=factory0
    -destinationname=channel0

Required arguments:
  rname :
    URL of the realm to which the ConnectionFactory is attached.
  factoryname :
    Name of the connection factory to locate.
  destinationname :
    JMS Destination, only required for AMPQ connections.

Optional Parameters:
  providerurl : URL of the local realm from which JNDI entries will be looked up
     When using an AMPQ connection this should be changed to:;
     amqp://localhost:9000 - when using a plain AMPQ connection, connecting to
        a standard nsp interface
     amqps://localhost:9000 - when using a plain AMPQ connection over a
        secure socket connection, connecting to a nsps interface
  Hint: AMPQ connections work only over nsp and nsps interfaces, they won't
     work over nhp or nhps.
  contextfactory :
    The name of the ContextFactory class to use
    (default: com.pcbsys.nirvana.nSpace.NirvanaContextFactory).
    When using an AMPQ connection this should be changed to:
      org.apache.qpid.jms.jndi.JmsInitialContextFactory
        to use the QPID Proton JMS Client libraries;
     org.apache.qpid.amqp_1_0.jms.jndi.PropertiesFileInitialContextFactory
        to use the QPID Legacy JMS Client libraries.
    Hint: AMPQ connections work only over nsp and nsps interfaces, they won't
    work over nhp or nhps.
  username :
    Your Universal Messaging server username.
  password :
```

```
    Your Universal Messaging server password.
```

Syntax: Recovery Tools

AddInterfaceOffline

```
Tool name:
  AddInterfaceOffline
Description:
  Adds a new interface to an offline realm.

Usage:
  runUMTool AddInterfaceOffline -dirname=<dirname> -protocol=<protocol>
    -adapter=<adapter> -port=<port> [optional_args]

Examples:
  AddInterfaceOffline -dirname=~/realmDirectories/realm0/data/ -protocol=socket
    -adapter=0.0.0.0 -port=11000

Required arguments:
  dirname :
    Data directory of the realm to add interface to.
  protocol :
    Protocol for the interface to use.
  adapter :
    Adapter the interface wants setting to.
  port :
    Port that the interface will listen on.

Optional Parameters:
  interface :
    Name of the interface to be created.
  autostart :
    Whether or not the interface should be autostarted when the realm
    starts.
  canadvertise :
    Whether or not the interface will be advertised.
  authtimeout :
    Number of milliseconds for authorisation timeout.
  interrealm :
    Whether or not this interface should be used for inter-realm
    communication.
  clientconnections :
    Whether or not this interface should be used for client
    connections.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

DeleteInterfaceOffline

```
Tool name:
  DeleteInterfaceOffline
Description:
  Removes an interface from an offline realm using config data.
```

```
Usage:
  runUMTool DeleteInterfaceOffline -dirname=<dirname> -interface=<interface>
    [optional_args]

Examples:
  DeleteInterfaceOffline -dirname=~/realmDirectories/realm0/data/ -interface=nhp0

Required arguments:
  dirname :
    Data directory of the realm to dump interfaces for.
  interface :
    Name of the interface to be removed.

Optional Parameters:
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

DumpInterfacesOffline

```
Tool name:
  DumpInterfacesOffline
Description:
  Dumps the list of interfaces for a specified offline realm.

Usage:
  runUMTool DumpInterfacesOffline -dirname=<dirname> [optional_args]

Examples:
  DumpInterfacesOffline -dirname=~/realmDirectories/realm0/data/

Required arguments:
  dirname :
    Data directory of the realm to dump interfaces for.

Optional Parameters:
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

ExportEventsFromOfflineMemFile

```
Tool name:
  ExportEventsFromOfflineMemFile
Description:
  Dumps events from offline mem file to XML file optionally using selector based
filtering.
Usage:
  runUMTool ExportEventsFromOfflineMemFile
    -memfileloc=<memfileloc>  -xmlfilename=<xmlfilename>  [optional_args]
Examples:
  ExportEventsFromOfflineMemFile
    -memfileloc=C:\source_folder\protobuf_channel94fab9fe6d3a92
    -dumpdata=true
    -xmlfilename=C:\destination_folder\file_name.xml
```

```
    -protobufdescriptor="C:\folder\School.fds"
  ExportEventsFromOfflineMemFile
    -memfileloc=C:\source_folder\channel_name.mem
    -dumpdata=true
    -xmlfilename=C:\destination_folder\file_name.xml
    -starteid=10 -endeid=30
  ExportEventsFromOfflineMemFile
    -memfileloc=C:\source_folder\mixed_channel_name231859db942796.mem
    -xmlfilename=C:\destination_folder\file_name.xml
    -selector="EVENTDATA.AS-STRING(0, 8) = 'data2702'"
    -batchsize=1000
Required arguments:
  memfileloc :
    Required parameter specifying the absolute path for the location of the
    memory file. The path can also be the location of a folder that contains
    multiple mem files.
    A folder with mem files can be specified only for multi-file storage.
    A single memory file can be specified only for mixed/persistent store.
  xmlfilename :
    Required parameter specifying XML file path to export events to.
Optional Parameters:

  selector :
    Optional parameter specifying the selector to filter the events.
  dumpdata :
   Optional parameter, when set to true the tool will dump event data (base64 encoded).
    Default value is true.
  starteid :
    Optional parameter specifying the startEID of event from mem file
    to start filtering/dumping from (default = 0).
  endeid :
    Optional parameter specifying the endEID of event from mem file to filter/dump to

    (if not specified, filtering/dumping will be done till the last storage event id).

  protobufdescriptor:
    Optional parameter specifying the path to protocol buffer file descriptor.
    Specified protocol buffer file descriptor will be exported to XML format file
    to ExportDetails section.
  batchsize :
    Optional parameter specifying the events batch size to read from mem file
    and dump to output file, default value is 100.
```

ModifyInterfaceOffline

```
Tool name:
  ModifyInterfaceOffline
Description:
  Modifies and interface of an offline realm.

Usage:
  runUMTool ModifyInterfaceOffline -dirname=<dirname> -interface=<interface>
    [optional_args]

Examples:
  ModifyInterfaceOffline -dirname=~/realmDirectories/realm0/data/
    -interface=nhp0 -port=11000

Required arguments:
```

```
  dirname :
    Data directory of the realm to dump interfaces for.
  interface :
    Name of the interface to be modified.

Optional Parameters:
  adapter :
    Adapter the interface wants setting to.
  port :
    Port that the interface will be set to.
  autostart :
    Whether or not the interface should be autostarted when the
    realm starts.
  canadvertise :
    Whether or not the interface will be advertised.
  authtimeout :
    Number of milliseconds for authorisation timeout.
  interrealm :
    Whether or not this interface should be used for inter-realm
    communication.
  clientconnections :
    Whether or not this interface should be used for client
    connections.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

ModifyPrimeFlagOffline

```
Tool name:
  ModifyPrimeFlagOffline
Description:
  Modifies the prime flag of a site while a realm is offline.

Usage:
  runUMTool ModifyPrimeFlagOffline -datadirectory=<datadirectory> [optional_args]

Examples:
  ModifyPrimeFlagOffline -datadirectory=~/realmDirectories/realm0/data/
  (This will show current state of prime flag per cluster sites)

  ModifyPrimeFlagOffline -datadirectory=~/realmDirectories/realm0/data/
    -Site1=true -Site2=false
  (This will set prime flag to true on site1 and to false on site 2 in
   directory ~/realmDirectories/realm0/data/ )


Required arguments:
  datadirectory :
    Data directory of the realm to be modified.

Optional Parameters:

  force :
    Skip confirmation step before modifying cluster sites.
 NOTE - if you want to edit the prime flag for sites inside a cluster you need to
   specify it like an optional parameter :
     -<site_name>=<new_value>.
```

```
   You need to specify the value of the flag for all sites inside the cluster.
   For example, if you want to set the prime flag to true on Site1 and you have
   another site called Site2, the optional parameter will look like :
     -Site1=true -Site2=false

  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

RepublishEventsFromOfflineFile

```
Tool Name:
  RepublishEventsFromOfflineFile
Description:
  Imports events from an XML or .mem file, optionally using selector based filter,
  and republishes events to specified store.
Usage:
  runUMTool RepublishEventsFromOfflineFile -realm=<realm>
    -channelname=<channelname>
    -memfileloc/xmlfilename=<source_file_path> [optional_args]
Examples:
  RepublishEventsFromOfflineFile
    -memfileloc=C:\source_folder\proto_channelb31238f42a49b
    -selector="EVENTDATA.AS-STRING(0, 8) = 'data2702'"
    -realm="nhp://0.0.0.0:11000"
    -channelname="destination_queue_name" -starteid=0 -endeid=90
  RepublishEventsFromOfflineFile
    -xmlfilename=C:\source_folder\filename.xml
    -selector="EVENTDATA.TAG = '3' or EVENTDATA.TAG = '33'"
    -realm="nhp://0.0.0.0:11000" -channelname="destination_channel_name"
    -endeid=50 -batchsize=1000
Required arguments:
  realm :
    Required parameter specifying realm server name/address for republishing events.

  channelname :
     Required parameter specifying destination store name for republishing events.
  memfileloc :
    Parameter specifying absolute path for the memory file location.
    The path can also be the location of a folder that contains
    multiple mem files.
    A folder with mem files can be specified only for multi-file storage,
    and a single memory file can be specified only for mixed/persistent store.
    One source file should be specified: memfileloc or xmlfilename.
  xmlfilename :
    Parameter specifying an XML file path to import events from.
    One source file should be specified: memfileloc or xmlfilename.

Optional Parameters:
  protobufdescriptor:
    Optional parameter specifying the path to protocol buffer file descriptor.
    When specified then republish tool will check if destination channel has
    the same protocol buffer file descriptor.
    No event filtering based on protobuf event data will be performed in
    versions earlier than 10.3.

  selector :
    Optional parameter specifying the selector to filter the events.
```

```
  started :
    Optional parameter specifying the startEID of event from source file
    to start filtering/importing from (default = 0).
  endeid :
   Optional parameter specifying the endEID of event from source file to filter/import
 to.
    If not specified, filtering/importing will be done till last storage event id).
  batchsize :
   Optional parameter specifying the event batch size used for import and republishing
 events
    to the store (default = 100).
```

Syntax: Durable Tools

ViewDurableEvent

```
Tool name:
  ViewDurableEvent
Description:
  Gets all events for all durables or all events for a specific durable.
  The tool has two required parameters (rname , channelname) and two optional
  parameters (durablename , maxevents , startid). By default, if no optional
  parameters are added it will list the events on all durables. Default number
  of events is 1000 per durable.

Usage:
  runUMTool ViewDurableEvent -rname=<rname> -channelname=<channelname>
    [optional_args]

Examples:
  ViewDurableEventWith required parameters: -rname=nhp://localhost:11000
    -channelname=testchan With optional parameters:
    -rname=nhp://localhost:11000 -channelname=testchan
    -durablename=testdurable -maxevents=100 -startid=50 -displayanydata=true

Required arguments:
  rname :
    Name of the realm.
  channelname :
    Channel which durable is subscribed to.

Optional Parameters:
  durablename :
    The name of the durable to browse events.(Optional Parameter).
  maxevents :
    The number of maximum events to display.(Optional Parameter).
  startid :
    The EID of the starting event to display events from.
    (Optional Parameter)
  displayanydata :
    If the data displayed should be of any kind. By default
    only UTF-8 encoded data is shown. (Optional Parameter)
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

Syntax: Miscellaneous Tools

EditRealmConfiguration

```
Tool name:
  EditRealmConfiguration
Description:
  Edits realm configuration parameters.

Usage:
  runUMTool EditRealmConfiguration -rname=<rname> [optional_args]

Examples:
  EditRealmConfiguration -rname=nsp://localhost:9000 -listgroupconfiguration=all
  (This will show all realm configuration parameters and their current value)

  EditRealmConfiguration -rname=nsp://localhost:9000
    -listgroupconfiguration=Thread_Pool_Config
  (This will show Thread Pool Config parameters and their current values)

  EditRealmConfiguration -rname=nsp://localhost:9000
   -Audit_Settings.ChannelACL=false -Join_Config.MaxQueueSizeToUse=50
  (This will set channelACL to false and MaxQueueSizeToUse to 50)


Required arguments:
  rname :
    Connection URL to the realm you want to edit configuration.

Optional Parameters:

  NOTE - If you want to edit a realm configuration parameter you should
    specify it like an optional parameter :
    -<group_name>.<parameter>=<new_value>
    where space is escaped in <group_name> by using an underscore("_").
    For example, if you want to change the parameter ChannelACL in the
    group Audit Settings to "true" the optional parameter will look like:
    -Audit_Settings.ChannelACL=true

  listgroupconfiguration :
    The configuration group for which you want to see values of parameters.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

ExportRealmXML

```
Tool name:
  ExportRealmXML
Description:
  Exports selected realm to an XML file.

Usage:
  runUMTool ExportRealmXML -rname=<rname> -filename=<filename> [optional_args]
```

```
Examples:
  ExportRealmXML -rname=nsp://localhost:9000 -filename=test.xml -exportall=true
  (This will export all the information)

  ExportRealmXML -rname=nsp://localhost:9000 -filename=test.xml -realms=true
    -realmconfiguration=true -channels=true -queues=true
  (This will export information about realm set, realm configuration, channels
   and queues)


Required arguments:
  rname :
    Connection URL to the realm you want to export.
  filename :
    File name where the information will be exported.

Optional Parameters:
  exportall :
    Export all information for the chosen realm.
  clusters :
    Include Cluster information in the export file.
  datagroups :
    Include DataGroups information in the export file.
  realmall :
    Export all information for realm including RealmSet,
    RealmConfiguration, RealmSchedulerSet and RealmACLS(RealmPermissionSet)
  realms :
    Include RealmSet information in the export file.
  realmconfiguration :
    Include RealmConfiguration information in the export file.
  realmschedule :
    Include RealmSchedulerSet information in the export file.
  realmacls :
    Include RealmACLS (RealmPermissionSet) information in the export file.
  channelsall :
    Export all information for channels in the chosen realm
    including ChannelEntry, ChannelACLs (ChannelPermissionSet), ChannelJoins,
    DurableSet and JNDI Configuration.
  channels :
    Include ChannelEntry information in the export file.
  channelacls :
    Include ChannelACLs (ChannelPermissionSet) information in
    the export file.
  channeljoins :
    Include ChannelJoins information in the export file.
  durables :
    Include DurableSet information in the export file.
  jndiconfig :
    Include JNDI Configuration information in the export file.
  interfacesall :
    Export all information for interfaces in the chosen realm
    including Interfaces, InterfaceVIA (ACLs) and Interface Plugins.
  interfaces :
    Include Interfaces information in the export file.
  interfacevia :
    Include InterfaceVIA (ACLs) information in the export file.
  plugins :
    Include Interface Plugins information in the export file.
  queuesall :
```

```
    Export all information for interfaces in the chosen realm
    including QueueEntry and QueueACLs (QueuePermissionSet).
  queues :
    Include QueueEntry information in the export file.
  queueacls :
    Include QueueACLs (QueuePermissionSet) information in the export file.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

HealthChecker

```
Tool name:
  HealthChecker
Description:
  Tool for analysing configuration items and highlighting robustness improvements.

Usage:
  runUMTool HealthChecker [optional_args]

Examples:
  HealthChecker -rname=nsp://localhost:11000
  HealthChecker -xml=/path/to/xml1,/path/to/xml2,...
  HealthChecker -rname=nsp://localhost:11000 -check=checkname1,checkname2,...
  HealthChecker -xml=/path/to/xml1,/path/to/xml2 -check=checkname1,checkname2,...
  HealthChecker -rname=nsp://localhost:11000 -exclude=checkname1,checkname2,...
  HealthChecker -xml=/path/to/xml1,/path/to/xml2
   -exclude=checkname1,checkname2,...
  HealthChecker -rname=nsp://localhost:11000 -include=checkname1,checkname2,...
  HealthChecker -xml=/path/to/xml1,/path/to/xml2
   -include=checkname1,checkname2,...
  HealthChecker -rname=nsp://localhost:11000 -mode=default
   -include=checkname1,checkname2,...
  HealthChecker -xml=/path/to/xml1,/path/to/xml2 -mode=default
   -include=checkname1,checkname2,...
  HealthChecker -rname=nsp://localhost:11000 -mode=all
   -exclude=checkname1,checkname2,...
  HealthChecker -xml=/path/to/xml1,/path/to/xml2 -mode=all
   -exclude=checkname1,checkname2,...
  HealthChecker -rname=nsp://localhost:11000 -mode=all
   -exclude=checkname1,checkname2,... -additionalArg1=... -additionalArg2=...
  HealthChecker -xml=/path/to/xml1,/path/to/xml2 -mode=all
   -exclude=checkname1,checkname2,... -additionalArg1=... -additionalArg2=...

Required arguments:

Optional Parameters:
  mode :
    Defines the initial set of 'HealthChecker' checks that the user can
    manipulate (with 'exclude' or 'include' options).
    There are two modes :
      default:
        This option gives access to the recommended minimal subset of
        checks. This is the default option if  mode is not specified.
      all:
        This option gives access to all checks. Executed without
        'exclude' or 'check' it will execute all HealthChecker checks.
```

```
  check :
    Run only the specified check or checks. It should not be used together
    with 'mode','include' or 'exclude arguments.
  exclude :
    Run all checks from the specified set (see 'mode') except the
    specified check or checks. The parameter may contain a single check
    or a comma-separated list of checks.
  include :
    Run all checks available with the given mode and additionally
    include the check(s) specified via this parameter. The parameter may
    contain a single check to include or a comma-separated list of checks.
  additionalArg<n>:
    Some of the health checks allow you to specify one or more additional
    parameters when calling the HealthChecker. The name and purpose of each
    additional parameter is specific to the individual health check being run.
    For example, the DurableSubscriberLargeStoreCheck check allows you
    to specify the additional parameter -threshold=<value>, which defines
    a threshold for the number of remaining events to be consumed in a
    shared durable.
    All additional parameters are passed to all the HealthChecker checks;
    if any given check has the capability to process any of the additional
    arguments then it will, and the given check will ignore any
    additional parameters that it cannot process.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

ImportRealmXML

```
Tool name:
  ImportRealmXML
Description:
  Imports selected realm from an XML file

Usage:
  runUMTool ImportRealmXML -rname=<rname> -filename=<filename> [optional_args]

Examples:
  ImportRealmXML -rname=nsp://localhost:9000 -filename=test.xml -importall=true
  (This will import all the information present in selected file)

  ImportRealmXML -rname=nsp://localhost:9000 -filename=test.xml -realms=true
    -realmconfiguration=true -channels=true -queues=true
  (This will import information about realm set, realm configuration, channels and
   queues if present in selected file)


Required arguments:
  rname :
    Connection URL to the realm you want to import configuration.
  filename :
    File name from which the information will be imported.

Optional Parameters:
  importall :
    Import all information for the chosen realm.
  clusters :
    Import Cluster information if present in file.
```

```
  datagroups :
    Import Data Groups information if present in file.
  realmall :
    Import all information for realm including RealmSet,
    RealmConfiguration, RealmSchedulerSet and RealmACLS(RealmPermissionSet)
    if present in file.
  realms :
    Import RealmSet information if present in file.
  realmconfiguration :
    Import RealmConfiguration information if present in file.
  realmschedule :
    Import RealmSchedulerSet information if present in file.
  realmacls :
    Import RealmACLS(RealmPermissionSet) information if present in file.
  channelsall :
    Import all information for channels including ChannelEntry,
    ChannelACLs (ChannelPermissionSet), ChannelJoins, DurableSet and
    JNDI Configuration if present in file.
  channels :
    Import ChannelEntry information if present in file.
  channelacls :
    Import ChannelACLS information if present in file.
  channeljoins :
    Import ChannelJoins information if present in file.
  durables :
    Import DurableSet information if present in file.
  jndiconfig :
    Import JNDI Configuration information if present in file.
  interfacesall :
    Import all information for interfaces including Interfaces,
    InterfaceVIA (ACLs) and Interface Plugins if present in file.
  interfaces :
    Import Interfaces information if present in file.
  interfacevia :
    Import InterfaceVIA (ACLs) information if present in file.
  plugins :
    Import Interface Plugins information if present in file.
  queuesall :
    Import all information for interfaces including QueueEntry and
    QueueACLs (QueuePermissionSet) if present in file.
  queues :
    Import QueueEntry information if present in file.
  queueacls :
    Import QueueACLs (QueuePermissionSet) information if present in
    file.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

Syntax: Site Tools

CreateSite

```
Tool name:
  CreateSite
Description:
```

```
  Creates a site with the specified name, consisting of the specified nodes.

Usage:
  runUMTool CreateSite -sitename=<sitename> -rnames=<rnames> [optional_args]

Examples:
  CreateSite -sitename=site0 -rnames=nsp://localhost:11000,nsp://localhost:11010

Required arguments:
  sitename :
    Name of the site to be created.
  rnames :
    Server URLs to be considered for the site. Can be more than one URL,
    separated by a comma.
    The proper format is [nsp/nhp/nsps/nhps]://[hostname]:[port] or
    shm://[path/to/file].

Optional Parameters:
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

DeleteSite

```
Tool name:
  DeleteSite
Description:
  Deletes a site with the specified name from all the nodes associated with it.

Usage:
  runUMTool DeleteSite -sitename=<sitename> -rname=<rname> [optional_args]

Examples:
  DeleteSite -sitename=site0 -rname=nsp://localhost:11000

Required arguments:
  sitename :
    Name of the site to be deleted.
  rname :
    Server URL to be considered for the site. Can be more than one URL,
    separated by a comma.
    The proper format is [nsp/nhp/nsps/nhps]://[hostname]:[port] or
    shm://[path/to/file].

Optional Parameters:
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

SetPrimeSite

```
Tool name:
  SetPrimeSite
Description:
  Toggles the specified site's prime status.
```

```
Usage:
  runUMTool SetPrimeSite -sitename=<sitename> -rname=<rname>
    -setprime=<setprime> [optional_args]

Examples:
  SetPrimeSite -sitename=site0 -rname=nhp://localhost:11000 -setprime=true

Required arguments:
  sitename :
    Name of the site to be configured.
  rname :
    Server URL to be considered for the site.
  setprime :
    True/False flag to set/unset a site being the prime site.

Optional Parameters:
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

ShowSites

```
Tool name:
  ShowSites
Description:
  Displays the configuration of the sites.

Usage:
  runUMTool ShowSites -rname=<rname> [optional_args]

Examples:
  ShowSites -rname=nhp://localhost:11000

Required arguments:
  rname :
    Server URL to be considered for the site.

Optional Parameters:
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

Syntax: Diagnostic Tools

RealmInformationCollector

```
Tool name:
  RealmInformationCollector
Description:
  Collects diagnostic information from a Universal Messaging realm server
  installation and stores it in a zip archive.

Usage:
  runUMTool RealmInformationCollector -mode=<mode> -instance=<instance>
```

```
    [optional_args]

Examples:
  RealmInformationCollector -mode=offline -instance=umserver
  RealmInformationCollector -mode=offline -instance=umserver
    -include=data,heapdumps
  RealmInformationCollector -mode=offline -instance=*
  RealmInformationCollector -mode=live -instance=umserver,umserver2
    -include=heapdump
  RealmInformationCollector -mode=live -instance=umserver,umserver2
    -exclude=jaas,plugins
  RealmInformationCollector -mode=live -instance=umserver,umserver2
    -outputfile=/path/to/outputfile.zip

Required arguments:
  mode : Operating mode, either 'offline' or 'live'.
    The chosen mode determines what information is collected.
    If 'offline' is specified, the tool will ensure that all instances to collect
    information from are not running.
    If 'live' is specified, the tool will ensure that all instances are running.

    In live mode, certain Universal Messaging directories/files are not collected,
    because reading them may cause failures on the server.
    For example, if the content of the data directory is needed, it can be
    collected only in offline mode.

    The following collectors will be executed by default in live mode:
      tanukilogs    - Collects Tanuki wrapper logs of an UM server instance.
      secfile       - Collects the security file of an UM server instance.
      installlogs   - Collects SoftwareAG installer logs.
      tanukiconf    - Collects Tanuki wrapper configuration of an UM server
                      instance.
      env           - Collects environment information from a running UM server.
      license       - Collects the license file of an UM server instance.
      realmconfig   - Exports realm configuration from a running UM server instance.

      instancemgr   - Collects Universal Messaging instance manager logs.
      healthchecker - Acquires health information from a running UM server instance
                      using UM HealthChecker tool
      jaas          - Collects JAAS configuration of an UM server instance.
      threaddump    - Generates 3 thread dumps of a running UM server instance.
      plugins       - Collects plugins directory of an UM server instance.

    Collectors not enabled per default in live mode (need to be explicitly included):

      heapdump      - Acquires heap dump from a running UM server instance.
                      Note: This collector is not available on all platforms.
      heapdumps     - Collects heap dumps directory of an UM server instance.

    The following collectors will be executed by default in offline mode:
      tanukilogs    - Collects Tanuki wrapper logs of an UM server instance.
      secfile       - Collects the security file of an UM server instance.
      installlogs   - Collects SoftwareAG installer logs.
      tanukiconf    - Collects Tanuki wrapper configuration of an UM server instance.

      license       - Collects the license file of an UM server instance.
      instancemgr   - Collects Universal Messaging instance manager logs.
      jaas          - Collects JAAS configuration of an UM server instance.
      logs          - Collects logs of an UM server instance.
      plugins       - Collects plugins directory of an UM server instance.
```

```
    Collectors not enabled per default in offline mode (need to be explicitly
    included):
      data          - Collects data directory of an UM server instance.
      heapdumps     - Collects heap dumps directory of an UM server instance.

  instance :
    Specifies a comma-separated list of realm server instance names to
    collect information from. Specify '*' to include all available instances.

Optional Parameters:
  outputfile :
    The directory or file to write generated archive to.
    If a directory is specified, it must exist.
    If a file is specified and it is already present, the tool will fail.
    If this argument is omitted, the tool will generate the archive in the
    current working directory.
  exclude :
    Specifies a comma-separated list of collector names to exclude.
    See 'mode' for list of available collectors.
  include :
    Specifies a comma-separated list of collector names to include.
    See 'mode' for list of available collectors.
  username :
    Your Universal Messaging server username.
  password :
    Your Universal Messaging server password.
```

7 Universal Messaging Administration API

- Introduction 580
- Administration API Package Documentation 583
- Namespace Objects 583
- Realm Server Management 589
- Security 597
- Management Information 601

Universal Messaging provides a feature rich Administration API capable of capturing all metrics, management and audit information from Universal Messaging realms. The API allows you to control and administer all aspects of any Universal Messaging realm or clusters of realms.

Universal Messaging's Enterprise Manager GUI has been written entirely using the Universal Messaging Administration API as a means of demonstrating how useful the API can be for the management of your messaging infrastructure.

Some example code showing how to use the Universal Messaging management API can be found in the examples section.

The Administration API is available in the following languages:

- Java
- C#.NET
- C++

Note:
The Administration APIs for C# and C++ are deprecated and will be removed from the product distribution in the next official release.

Introduction

Getting Started

The Universal Messaging Admin API (see the Package Documentation) allows management, configuration, audit and monitoring of all aspects of a Universal Messaging realm server.

The starting point for the Admin API is connecting to a realm. In order to connect to a realm using the Admin API, you need to ensure you are familiar with the concept of an RNAME. Once you have the RNAME that corresponds to your realm, you can then connect to the realm.

The way you connect to a realm is by constructing an nRealmNode object. The nRealmNode object is the main object you need to access all of the objects you wish to configure, monitor and manage:

```
String[] RNAME={"nsp://127.0.0.1:9000"};
nSessionAttributes nsa=new nSessionAttributes(RNAME);
nRealmNode realm = new nRealmNode(nsa);
```

Universal Messaging namespace

Access to resources on a Universal Messaging realms, or indeed objects in a multi Universal Messaging realm server namespace, is based on a simple tree structure, where the nRealmNode is the root of the tree. All nodes within the tree are subclasses of a base class nNode. From the root, it is possible to obtain references to all child nodes. Child nodes may be other realm nodes, containers (folders containing other realms, channels etc), channels and queues.

For example, to obtain an enumeration of all child nodes within a realm node, simply call the following:

Java:

```
Enumeration children = realm.getNodes();
```

C#:

```
System.Collections.IEnumerator children = realm.getNodes();
```

C++:

```
fSortedList nodes = pNode->getNodes();
```

From this enumeration you can then perform operations on the child nodes. For example, if you have a realm with 1 channel and 1 queue, and wanted to find the number of events currently on each, the following code would do that:

Example: Finding out how many events are on a channel / queue

Java:

```
  while (children.hasMoreElements()) {
    nNode child = (nNode)children.nextElement();
    if (child instanceof nLeafNode) {
      nLeafNode leaf = (nLeafNode)child;
      System.out.println("Leaf node contains "+leaf.getCurrentNumberOfEvents());
    }
  }
```

C#:

```
while (children.MoveNext()){
  nNode child = (nNode)children.Current;
    if (child is nLeafNode) {
      nLeafNode leaf = (nLeafNode)child;
      Console.WriteLine("Leaf node contains "+leaf.getCurrentNumberOfEvents());
    }
  }
```

C++:

```
void searchNodes(fSortedList nodes)
    for (fSortedList::iterator iterator = nodes.begin(); iterator != nodes.end();
iterator++)
    {
        nNode *pNode = iterator->second;
        int type = pNode->getType ();
        if (type == fBase::LEAFNODE)
        {
           printf("Leaf node contains %ll events",pNode->getCurrentNumberOfEvents());
        }
    }
}
```

The namespace structure is dynamic and is managed asynchronously for you, so as and when objects are created, deleted modified, stopped or started, the namespace will manage those state changes and keep the structure up to date automatically.

Management / Configuration / Security

As well as the namespace nodes, there are also other objects that can be obtained from the nodes but which are not part of the namespace tree structure.

For example, from an nRealmNode it is possible to obtain the following objects:

- **nClusterNode** - The cluster node that this realm may be part of, allowing the administration of Universal Messaging realm clusters
- **nACL** - The realm acl object (see "Realm Entitlements" on page 214), allowing control of the ACL permissions (see "Access Control Lists" on page 597)
- **nInterfaceManager** - The realm interface manager, allows me to add, remove, stop, start interfaces on a realm (see "Interfaces" on page 589)
- **nSchedulerManager** - the scheduler manager allows me to control scheduled tasks (see "Scheduling" on page 591) on the realm
- **nConfigGroup** - an enumeration of these corresponds to all configuration (see "Config" on page 592) and tuning parameters for a given realm.

From an nLeafNode which could be a channel or a queue, the following objects are available:

- **nACL** - The leaf node acl object, allows me to control acl permissions (see "Channel Entitlements" on page 216) for resources
- **nJoinInfo** - All join information associated with a channel or queue

Monitoring

As well access to the channel resources as described above, there are also many monitoring tools available to developers that provide information asynchronously as and when events occur on a realm. This can be extremely useful in ongoing real time management of one or more Universal Messaging Realm servers.

For example, for a realm node you can provide listeners for the following :

- **Connections** - get notified as new connections (see "Connection Information" on page 607) to the realm occur, showing connection information
- **Creation / Deletions / Stop / Start** - get notified when new objects are created, deleted, modified, stopped or started (see "nRealmNode" on page 601) (for example new channels being created, acls being changed etc)
- **State Changes** - get notified when changes occur to any of the objects in the namespace (see "nLeafNode" on page 605), such as events being published / consumed. All updates are asynchronously received from the realm server and the API manages those changes for you.
- **Audit / Logging** - when security or state changes occur, get notified of audit events, as well as remotely receiving log file information from the server.

The following sections in this guide will work through in more detail, each of what has been discussed above.

Administration API Package Documentation

The Administration API is provided in the package com.pcbsys.nirvana.nAdminAPI

The API documentation is available in the *Universal Messaging Reference Guide* section of the documentation.

Namespace Objects

nRealmNode

Universal Messaging's namespace contains objects that can be administered, monitored and configured. The nRealmNode object in the nAdminAPI, corresponds to a Universal Messaging Realm server process. The nRealmNode is used to make an admin connection to a realm.

In order to connect to a realm you need to ensure you are familiar with the concept of an RNAME. Once you have the RNAME that corresponds to your realm, you can then construct the nRealmNode and connect to the corresponding realm. This is achieved by the following calls:

Java:

```
String[] RNAME={"nsp://127.0.0.1:9000"};
nSessionAttributes nsa=new nSessionAttributes(RNAME);
nRealmNode realm = new nRealmNode(nsa);
```

C++:

```
std::string rName = "nsp://127.0.0.1:9000";
nSessionAttributes* nsa=new nSessionAttributes(rName);
nRealmNode* realm = new nRealmNode(nsa);
```

By constructing an nRealmNode, and connecting to a realm, the realm node will automatically begin receiving status information from the realm periodically, as well as when things occur.

nRealmNode

The nRealmNode is the root of a Universal Messaging Realm's namespace, which is a tree like structure that contains child nodes. The tree nodes are all subclasses of a base class nNode. Each node corresponds to one of the following node subclasses:

- **nRealmNode** - other realm nodes that have been added to this realm's namespace
- **nContainer** - folders, if there was a channel called /eur/uk/rates, there would be a child nContainer node called, 'eur' which would have a child called 'uk' etc.
- **nLeafNode** - these correspond to channels and queues

The nRealmNode itself is a subclass of the nContainer class. To obtain an enumeration of all child nodes within a realm node, simply call the following:

Java:

```
Enumeration children = realm.getNodes();
```

C#:

```
System.Collections.IEnumerator children = realm.getNodes();
```

C++:

```
fSortedList nodes = pNode->getNodes();
```

Once you have this enumeration of nodes, you can then perform the various operations on those nodes available through the nAdminAPI.

If you know the name of the child node you wish to obtain a reference to, you can use the following method:

Java:

```
nNode found = realm.findNode("/eur/uk/rates");
```

C++:

```
nNode* found = realm->findNode("/eur/uk/rates");
```

Which should return you an nLeafNode that corresponds to the channel called '/eur/uk/rates'.

As well as obtaining references to existing nodes, it is also possible to create and delete channels and queues using the nRealmNode. For example, to create a channel called '/eur/fr/rates', we would write the following code:

```
nChannelAttributes cattrib = new nChannelAttributes();
cattrib.setMaxEvents(0);
cattrib.setTTL(0);
cattrib.setType(nChannelAttributes.SIMPLE_TYPE);
cattrib.setName(“/eur/fr/rates”);
nLeafNode channel = realm.createChannel(cattrib);
```

C++:

```
nChannelAttributes* cattrib = new nChannelAttributes();
cattrib->setMaxEvents(0);
cattrib->setTTL(0);
cattrib->setType(nChannelAttributes.SIMPLE_TYPE);
cattrib->setName(“/eur/fr/rates”);
nLeafNode* channel = realm->createChannel(cattrib);
```

To remove channel or a queue, you can simply call the following method on your realm node (using the channel created above):

```
realm.delLeafNode(channel);
```

C++:

```
realm->delLeafNode(channel);
```

For more information on Universal Messaging Administration, please see the API documentation, and the "Enterprise Manager Guide" on page 9.

nLeafNode (Channels and Queues)

Once you are familiar with the concept of the Universal Messaging Namespace, as discussed in the nRealmNode guide (see "nRealmNode" on page 583), you can then begin to use the other administration objects associated with a realm's Namespace.

In this section the nLeafNode is discussed. It is assumed you are aware of how to create an nRealmNode for this section, and have a general understanding of Universal Messaging's publish / subscribe and message queue technologies

nLeafNode

The nLeafNode is either a channel or a queue, and is, as its name suggests, an end point of a branch of the namespace tree. An nLeafNode's parent is always an instance of nContainer. Since nRealmNode is a subclass of nContainer, sometimes an nLeafNode's parent is also an instance of an nRealmNode. For example, consider the following 2 channels within the namespace:

```
/eur/uk/rates
/rates
```

The nLeafNode that corresponds to the channel '/eur/uk/rates' will have a parent which is an instance of nContainer, and is called 'uk', whereas the nLeafNode that corresponds to the channel '/rates' has a parent which is also an instance of nContainer, however is is also an instance of an nRealmNode (i.e. the namespace root), since it does not contain any folder information in its name.

As channels and queues are created, they are added to the nRealmNode's tree structure as nLeafNodes. This is all managed for you and does not require you to modify the structure. However it is possible to be notified when changes to the namespace occur so that your application can handle it as you see fit. This is discussed in more detail in the Management Information section of this guide.

To determine if an nLeafNode is a channel or a queue, there are 2 simple methods you can use. The following code snippet search the namespace and print out whether each leaf node it finds is a channel or a queue.

Example : Find channels and queues in the namespace

Java:

```
public void searchNodes(nContainer container)
  Enumeration children = container.getNodes();
  while (children.hasMoreElements()) {
    nNode child = (nNode)children.nextElement();
    if (child instanceof nContainer) {
      searchNodes((nContainer)child);
    } else if (child instanceof nLeafNode) {
      nLeafNode leaf = (nLeafNode)child;
      if (leaf.isChannel) {
```

```
        System.out.println("Leaf Node "+leaf.getName()+" is a channel");
      } else if (leaf.isQueue()) {
        System.out.println("Leaf Node "+leaf.getName()+" is a queue");
      }
    }
  }
}
```

C#:

```
public void searchNodes(nContainer container)
System.Collections.IEnumerator children = realm.getNodes();
while (children.MoveNext()){
  nNode child = (nNode)children.Current;
    if (child is nContainer) {
      searchNodes((nContainer)child);
    } else if (child is nLeafNode) {
      nLeafNode leaf = (nLeafNode)child;
      if (leaf.isChannel) {
        Console.WriteLine("Leaf Node "+leaf.getName()+" is a channel");
      } else if (leaf.isQueue()) {
        Console.WriteLine("Leaf Node "+leaf.getName()+" is a queue");
      }
    }
  }
}
```

C++:

```
void searchNodes(fSortedList nodes)
    for (fSortedList::iterator iterator = nodes.begin(); iterator != nodes.end();
iterator++)
    {
        nNode *pNode = iterator->second;
        int type = pNode->getType ();
        if (type == fBase::LEAFNODE)
        {
            if(iterator->second->isChannel()){
                printf("Leaf Node %s is a Channel");
            } else if(iterator->second->isQueue()){
                printf("Leaf Node %s is a Queue");
            }
        }
        else if (type == fBase::CONTAINER)
        {
            searchNodes(((nContainer*)pNode)->getNodes());
        }
    }
}
```

In the above code example, by the searchNodes(realm) method searches the namespace from the realm node, and this isChannel() and isQueue() methods are used to determine whether each leaf node is a queue or a channel.

Associated with each leaf node, is the nChannelAttributes for the queue or channel, this is obtained by using the getAttributes() method, so it is possible to determine the characteristics of each leaf node.

Each leaf node also has an associated nACL object that can be modified to change security permissions for users. This is discussed in more detail in the security section of this guide.

For more information on Universal Messaging Administration, please see the API documentation, and the Enterprise Manager Guide.

Realm Federation

A Universal Messaging Realm is an instance of the server and a container for resources. Each Universal Messaging Realm defines a namespace of its own but it is possible to merge the namespaces of multiple Realms into one large one. This is known as *realm federation*.

Note:
Clustering and Realm Federation are mutually exclusive. If a realm is a member of a cluster, you cannot use the realm for federation. Similarly, if a realm is part of a federation, the realm cannot be used for clustering.

While adding a Universal Messaging Realm into the namespace of another, there is one compulsory options and two optional. The compulsory option is the RNAME of that Realm. The optional parameter is the mount point that the Realm should be added in the existing Realm.

If you are specifying the name of the Realm you are adding it should be specified exactly as it appears in the Enterprise Manager. It appears adjacent to the globe icon specifying the realm to which this realm is being added.

A Universal Messaging Realm can also be added to another Realm's namespace using the Enterprise Manager (see "Realm Federation" on page 587).

A Realm is added into the namespace of another programmatically as follows.

Java, C#:

```
//Create an instance of the Universal Messaging Realm object to be added
String rname = "nsp://remoteHost:9002";
nRealm nr = new nRealm( realmName, rname);
//Set the mountpoint in the local realm's Namespace
nr.setMountPoint( mountPnt );
//Add the remote realm to the local one.
//assuming mySession has already been connected to your local realm
mySession.addRealm( nr );
```

C++

```
//Create an instance of the Universal Messaging Realm object to be added
string rname = "nsp://remoteHost:9002";
nRealm* nr = new nRealm( realmName, rname);
//Set the mountpoint in the local realm's Namespace
nr->setMountPoint( mountPnt );
//Add the remote realm to the local one.
//assuming mySession has already been connected to your local realm
mySession->addRealm( nr );
```

Example Usage of a Federated Universal Messaging Namespace

You can then provide filters for channel joins across the multiple realms you have added to the namespace. This allows you to ensure that events are routed to the correct channel based on the content of the event.

Note:
For a description of the general principles involved in creating channel joins, see the section *Creating Channel Joins*. The description details the usage based on the Enterprise Manager, but the same general principles apply if you are using the API.

For example, if channel1 on Realm1 is joined to channels channel2, channel3, channel4, channel5 on realms Realm2, Realm3, Realm4, Realm5, and each event is published using an nEventProperties dictionary that contains a key called 'DESTINATION'.

If each channel join from channel1 is created with a filter, for example for the join from channel1 to channel2 on Realm2 the filter would be:

```
DESTINATION='realm2'
```

This guarantees only those events that are published to channel1 and that contain 'realm2' in the 'DESTINATION' key will be published to channel2 on Realm2.

For further example code demonstrating adding Universal Messaging Realms to a names space please see the addRealm example.

Channel Join

Joining a channel to another allows you to set up content routing such that events on the source channel will be passed on to the destination channel also. Joins also support the use of filters thus enabling dynamic content routing.

Channels can be joined using the Universal Messaging Enterprise Manager GUI or programmatically.

In joining two Universal Messaging channels there is one compulsory option and two optional ones. The compulsory option is the destination channel. The optional parameters are the maximum join hops and a JMS message selector to be applied to the join.

Note:
For a description of the general principles involved in creating channel joins, see the section *Creating Channel Joins* in the *Administration Guide*. The description details the usage based on the Enterprise Manager, but the same general principles apply if you are using the API.

Channel joins can be created using the nmakechanjoin join sample application which is provided in the `<InstallDir>`/UniversalMessaging/`server/<InstanceName>/bin` directory of the Universal Messaging installation. For further information on using this example please see the nmakechanjoin example page.

Universal Messaging joins are created as follows:

Java, C#:

```
//Obtain a reference to the source channel
nChannel mySrcChannel = mySession.findChannel( nca );
//Obtain a reference to the destination channel
nChannel myDstChannel = mySession.findChannel( dest );
//create the join
mySrcChannel.joinChannel( myDstChannel, true, jhc, SELECTOR );
```

C++:

```
//Obtain a reference to the source channel
    nChannel* mySrcChannel = mySession->findChannel( nca );
    //Obtain a reference to the destination channel
    nChannel* myDstChannel = mySession->findChannel( dest );
    //create the join
    mySrcChannel->joinChannel( myDstChannel, true, jhc, SELECTOR );
```

Realm Server Management

Interfaces

Universal Messaging Realm servers provide the ability for connections to be made using any available physical network interface on the server machine. For example, if a machine has 4 physical network interfaces, Universal Messaging provides the ability to bind specific network interface addresses to specific ports and different protocols. This provides the ability to run segment the communication between client and server. There is no limit to the number of separate interfaces that can be run on a Universal Messaging realm server.

For example, a Realm Server that is visible to Internet users may have 4 Network cards, each one having its own physical IP address and hostname. Two of the network interfaces may be externally visible, while the other 2 may be only visible on internal sub-nets.

The 2 external interfaces may be specified as using nhp, and nhps on ports 80 and 443 respectively, since for firewall purposes, these ports are the most commonly accessible ports to external clients connecting to the realm. The remaining internal interfaces, visible to internal client connections do not have the same restrictions, and so could be defined as using nsp and nsps protocols on other ports, say 9000 and 9002 respectively.

What this guarantees is separation of internal and external connections based on network interface and protocol.

nInterfaceManager

When you have connected to a realm, and have a reference to an nRealmNode object (see "nRealmNode" on page 583), you can access an object called nInterfaceManager, which provides the ability to add, modify, delete, stop and start interfaces on the Universal Messaging realm. To get access to this object, you can call the following method from a realm node:

Java, C#:

```
nInterfaceManager iMgr = realm.getInterfaceManager();
```

C++:

```
nInterfaceManager* iMgr = realm->getInterfaceManager();
```

Using the nInterfaceManager object you can then obtain a list of known interfaces for that realm:

Java:

```
Vector ifaces = iMgr.getInterfaces();
```

C#:

```
List ifaces = iMgr.getInterfaces();
```

C++:

```
int numInterfaces; nInterfaceStatus** pTemp = iMgr->getInterfaces(numInterfaces);
```

All interfaces extend a base class called nInterface. There are 4 types of interface object that correspond to the different types of protocols that an interface can use. These are:

- nSocketInterface - standard socket interface, Universal Messaging protocol is nsp
- nHTTPInterface - http interface, Universal Messaging protocol is nhp
- nSSLInterface - ssl socket interface, Universal Messaging protocol is nsps
- nHTTPSInterface - https interface, Universal Messaging protocl is nhps

Each of these interface objects contain standard configuration information and allows the same operations to be performed on them. For example, if there is an interface called 'nsp1', and you wanted to change the 'autostart' property to true (i.e. make the interface start automatically when the realm is started) this can be achieved with the following code:

Java, C#:

```
nInterface iface = iMgr.findInterface("nsp0");
 iface.setAutostart(true);
 iMgr.modInterface(iface);
```

C++:

```
nInterface* iface = iMgr->findInterface("nsp0");
iface->setAutostart(true);
iMgr->modInterface(iface);
```

Which will modify the interface configuration at the server, stop and restart the interface. When performing a modInterface operation, if you are modifying the interface that your nRealmNode is connected to, you will be disconnected and reconnected when the interface restarts. This is important to remember when using the stop method of an interface too, since if you stop the interface you are connected to, you cannot start it again, since your connection needs to be active, and the stop operation will close your connection. If you wish to restart an interface you should therefore do it from a connection which has been made via another interface.

Example: creating an NHPS interface

You can create an NHPS interface using code such as the following:

```
nRealmNode rnode = ...;
nHTTPSInterface nhps = new nHTTPSInterface("0.0.0.0", 9443,
autoStart);
nhps.setKeyStore(keystore);
nhps.setKeyStorePassword(kpass);
nhps.setPrivateKeyPassword(kpass);
nhps.setTrustStore(tstore);
nhps.setTrustStorePassword(tpass);
rnode.getInterfaceManager().addInterface(nhps);
```

Scheduling

Universal Messaging Realm servers provide the ability for scheduling tasks. Tasks can be scheduled to execute based on certain conditions being met.

These conditions can be either time based (scheduling) or event based (triggers).

Universal Messaging scheduling is achieved through the creation of numerous scheduling scripts. Each script can contain multiple definitions of triggers and tasks.

The Universal Messaging server parses these scripts and sets up the triggers and tasks accordingly. For more information on the script grammar, there is a section in the enterprise manager guide which deals with writing scheduling scripts.

nSchedulerManager

When you have connected to a realm, and have a reference to an nRealmNode object (see "nRealmNode" on page 583), you can access an object called nSchedulerManager, which provides you with the ability to add, modify, delete scheduling scripts. To get access to this object, you can call the following method from a realm node:

Java, C#:

```
nSchedulerManager sMgr = realm.getSchedulerManager();
```

C++:

```
nSchedulerManager* sMgr = realm->getSchedulerManager();
```

Using the nSchedulerManager object you can then obtain a list of scheduler objects for the realm:

Java:

```
Enumeration schedulers = sMgr.getNodes();
```

C#:

```
System.Collections.IEnumerator schedulers = sMgr.getNodes();
```

C++:

```
fSortedList nodes = pNode->getNodes();
```

This method returns an enumeration of nScheduler objects. The nScheduler objects each correspond to a particular scheduling script.

The following code shows you how to construct a new scheduler object using a sample script that will log a message to the realm server log every hour, signified by the 'every 60' condition: {Please Note: typically this script would be read from a script file or it could be entered directly into the realm enterprise manager GUI.}

Java, C##:

```
String source = "scheduler myScheduler {\n";
 String logString = "Sample script : ";
 source += "\n";
 source += "\n";
 source += " initialise{\n";
 source += " Logger.setlevel(0);\n";
 source += " }\n";
 source += " every 60"{\n";
 source += " Logger.report(\""+logString+"\");\n";
 source += " }\n";
 source += "}\n";
 sMgr.add(source, "user@localhost", false);
```

C++:

```
stringstream s;
 s<<"scheduler myScheduler {\n";
 string logString = "Sample script : ";
 s<<"\n";
 s<<"\n";
 s<<"initialise{\n";
 s<<"Logger.setlevel(0);\n";
 s<<"}\n";
 s<<"every 60"{\n";
 s<<"fLogger::report(\""+logString+"\");\n";
 s<<"}\n";
 s<<"}\n";
 sMgr->add(source, "user@localhost", false);
```

For more information on Universal Messaging Administration, please see the API documentation, and the Enterprise Manager Guide.

Config

Universal Messaging Realm servers contain a large number of configurable parameters These parameters can be modified using the nAdminAPI.

The Universal Messaging Realm config can also be managed via the Realm enterprise manager (see "Realm Configuration" on page 49). This also provides a useful guide to the configuration groups and their specific config entities.

nConfigGroup

When connected to a realm, and using a reference to an nRealmNode object (see "nRealmNode" on page 583), you can access configuration objects that correspond to a group of configuration entries. To get access to the config groups, call the following method from a realm node:

Java, C#

```
Enumeration children = realm.getNodes();
```

C++

```
fSortedList nodes = pNode->getNodes();
```

The enumeration will contain a number of nConfigGroup objects. Each nConfigGroup contains a number of nConfigEntry objects, each one corresponds to a specific configurable parameter in the realm server.

For example, to change the log level of the realm server, we need to obtain the config group called 'Logging Config' and set the 'fLoggerLevel' property:

Java, C#:

```
nConfigGroup grp = realm.getConfigGroup("Logging Config");
nConfigEntry entry = grp.find("fLoggerLevel");
entry.setValue("0");
```

C++

```
nConfigGroup* grp = realm->getConfigGroup("Logging Config");
nConfigEntry* entry = grp->find("fLoggerLevel");
entry->setValue("0");
```

For a definitive list of available configuration groups and their specific properties please see "Realm Configuration" on page 49 in the enterprise manager guide.

For more information on Universal Messaging Administration, please see the API documentation, and the Enterprise Manager Guide.

Clustering

Universal Messaging provides the ability to group Realm servers together to form a cluster. A cluster is a logical group of realm servers that share common resources. The resources and any operations performed on then are replicated across all cluster members. Clients connecting to 'Realm A' in cluster 1, are able to access the same logical objects as clients connecting to Realms B or C in cluster1.

The state of these objects is fully replicated by each realm in the cluster. For example, if you create a queue (queue1) within cluster 1, it is physically created in realms A, B and C. If there are 3 consumers on queue1, say one on each of realms A, B and C respectively, each realm in the cluster will be aware as each message is consumed and removed from the different physical queue1 objects in the 3 realms.

If one of the realms within cluster1 stops, due to a hardware or network problems, then clients can automatically reconnect to any of the other realms and start from the same point in time on any of the other realms in the cluster.

This ensures a number of things:

- Transparency - Any client can connect to any Universal Messaging realm server within a cluster and see the same cluster objects with the same state. Clients disconnected from one realm will automatically be reconnected to another cluster realm.
- 24 x 7 Availability - If one server stops, the other realms within the cluster will take over the work, providing an always on service

nClusterNode

Using the nAdmin API, if you wish to create a cluster that contains 3 realms, and you know the RNAME values for all 3, then the following call will create the cluster.

Java, C#, C++:

```
String[] RNAME= {"nsp://127.0.0.1:9000",
"nsp://127.0.0.1:10000","nsp://127.0.0.1:11000"};
nRealmNode realms[] = new nRealmNode[RNAME.length];
nClusterMemberConfiguration[] config = new nClusterMemberConfiguration[RNAME.length];
for (int x = 0; x < RNAME.length; x++) {
  // you don't have to create the realm nodes
  // here, since the member configuration will create
  // them for you from the RNAME values
  realms[x] = new nRealmNode(new nSessionAttributes(RNAME[x]));
  config[x]=new nClusterMemberConfiguration(realms[x], true);
}
nClusterNode cluster = nClusterNode.create("cluster1", config);
```

Once the cluster node is created, each realm node within the cluster will know of the other realms within the cluster, and be aware of the cluster they are part of. For example, calling the following method:

Java, C#, C++:

```
nClusterNode cluster = realms[0].getCluster();
```

will return the cluster node just created with the realm with nsp://127.0.0.1:9000 for an RNAME.

Cluster nodes contain information about the member realms (nRealmNode objects) as well as the current state of the cluster members. This information can be found by calling the getClusterConnectionStatus() method on the cluster node, which returns a vector of nClusterStatus objects, each of which corresponds to a realm.

nRealmlNode

Once a realm becomes part of a cluster, channels and queues can be created that are part of the cluster, as well as standard local resources within the realms. For example, if you were to us the following calls:

Java, C#, C++:

```
nChannelAttributes cattrib = new nChannelAttributes();
cattrib.setMaxEvents(0);
cattrib.setTTL(0);
cattrib.setType(nChannelAttributes.PERSISTENT_TYPE);
cattrib.setClusterWide(true);
cattrib.setName(“clusterchannel”);
nLeafNode=.createChannel(cattrib);
realms[0].createChannel(cattrib);
```

This would create a channel that is visible to all realms within a cluster. Any administrative changes made to this channel such as ACL modifications will also be propagated to all cluster members in order for the channel to be kept in sync across all realms.

Inter-Cluster Connections

Inter-cluster connections can be created programmatically through the Administration API. To do this, connect to a realmNode in each cluster and then do the following:

Java, C#, C++:

```
cluster1realm1.getCluster().registerRemoteCluster(cluster2realms1.getCluster());
```

Similarly, the inter-cluster connection can be removed programmatically:

Java, C#, C++:

```
cluster1realm1.getCluster().deregisterRemoteCluster(cluster2realm1.getCluster());
```

For more information on Universal Messaging Administration, please see the API documentation, and the Enterprise Manager Guide.

Multicast

A common way to add a multicast configuration is via the Enterprise Manager (see “Multicast Configuration” on page 289) but you can also do this programmatically.

Creating the nMulticastConfiguration

In order to create an nMulticastConfiguration object you need to specify two parameters:

- multicastAddress - Multicast IP address to use
- adapter - Network adapter address of your multicast configuration

Java, C#:

```
String multicastAddress = "227.0.0.98";
    String adapter = "10.150.12.1";
    nMulticastConfiguration mConf = new nMulticastConfiguration(multicastAddress,
adapter);
```

C++:

```
std::string multicastAddress = "227.0.0.98";
    std::string adapter = "10.150.12.1";
    nMulticastConfiguration* mConf = new nMulticastConfiguration(multicastAddress,
adapter);
```

Enabling multicast for cluster communication

In order to use multicast for intra-cluster communication you need to set a flag on the nMulticastConfiguration:

Java, C#:

```
mConf.setUseForCluster(true);
```

C++:

```
mConf->setUseForCluster(true);
```

Enabling multicast on DataGroups

When you create a DataGroup you have the option to enable multicast delivery. However you also need to enable multicast for DataGroups on the multicast configuration:

Java, C#:

```
mConf.setUseForDataGroups(true);
```

C++:

```
mConf->setUseForDataGroups(true);
```

Then (after the configuration has been applied) when you create a DataGroup you need to set the enableMulticast flag to true:

Java, C#:

```
boolean enableMulticast = true;
    String name = "newGroup";
    mySession.createDataGroup(name,enableMulticast);
```

C++:

```
bool enableMulticast = true;
    std::string name = "newGroup";
    mySession->createDataGroup(name,enableMulticast);
```

Applying the multicast configuration

In order to register the new configuration on the server you will need to connect to a Universal Messaging Realm and establish an nRealmNode (see "nRealmNode" on page 583). You can then get a reference to the nMulticastManager:

Java, C#:

```
nMulticastManager mMgr = realm.getMulticastManager();
```

C++:

```
nMulticastManager* mMgr = realm->getMulticastManager();
```

You can now use the nMulticastManager to send the new configuration to the server:

Java, C#:

```
mMgr.addMulticastConfiguration(mConf);
```

C++:

```
mMgr->addMulticastConfiguration(mConf);
```

Security

Access Control Lists

The Universal Messaging Administration API allows Access Control Lists (ACLs) to be set using the nACL object defines a set of nACLEntry objects that consist of a user subject and a value that corresponds to the operations permitted for that subject. With an nACL object, it is possible to added, delete and modify acl entries for specific subjects.

The nACL Object

There are subclasses of the base nACLEntry object. These are :

- nRealmACLEntry - defines permissions for a specific subject on the Universal Messaging Realm server itself
- nChannelACLEntry - defines permissions for a subject on a channel or queue

ACL Lists can contain any combination and number of user@host entries, along with Security Groups (see "Nirvana Admin API - Nirvana Security Groups" on page 597).

Nirvana Admin API - Nirvana Security Groups

The Administration API allows groups of users to be defined. These groups can then be used in ACL lists in-place of individual ACL entries for each user.

Security Groups can contain any number of users (user@host pairs), and may also include other Security Groups.

A new security group can be registered as follows:

Java, C#, C++:

```
nSecurityGroup grp = new nSecurityGroup("mySecurityGroup");
grp.add(add(new nSubject("user@host");
```

```
realmNode.getSecurityGroupManager.registerSecurityGroup(grp);
```

The SecurityGroupManager can be used to edit memberships of multiple groups at the same time, for example:

Java, C#, C++:

```
nSecurityGroupManager mgr = realmNode.getSecurityGroupManager();
mgr.registerGroupMembers(group,members);
//Members can be a single subject(user@host), a group, or a collection
//containing many subjects, groups or a combination of these.
```

Once a security group has been registered, it can be added into ACL lists as you would normally add a user@host entry. Subsequent changes to the membership of the group will be reflected in which users have permissions for the corresponding resources.

Java, C#, C++:

```
nSecurityGroup grp = securityGroupManager.getGroup("myGroupName");
nChannelACLEntry aclEntry = new nChannelACLEntry(grp);
aclEntry.setFullPrivileges(true);
leafNode.addACLEntry(aclEntry);
```

Groups can also be deregistered from the realm. This will remove the group and will remove the group reference from all ACL lists where the group currently appears. As with the other examples, this can be done via the nSecurityGroupManager:

Java, C#, C++:

```
mgr.deregisterSecurityGroup(grp);
```

As with all ACLs in Universal Messaging, privileges are cumulative. This means that, for example, if a user is in a group which has publish permissions on a channel, but not subscribe permissions, the user will no be able to subscribe on the channel. Then, if an ACL entry is added on the channel for his specific username/host pair, with subscribe but no publish permissions, the user will then be able to both subscribe (from the non-group ACL permission), and publish (from the group ACL permission).

Deeply nested Security Groups hierarchies are generally discouraged, since this type of configuration can negatively impact the speed of checking ACLs, and may result in worse performance than a shallow hierarchy.

Realm Access Control List (nACL)

When you have connected to a realm, and have a reference to an nRealmNode object (see "nRealmNode" on page 583), you can access an the realm's acl object. This object contains a list of nRealmACLEntry objects that represent a subject and a set permissions for various operations on a realm.

You can also, add, delete and modify acl entry objects. To obtain the realm acl object, simply call the following method from a realm node:

Java, C#:

```
nACL acl = realm.getACLs();
```

C++:

```
nACL* acl = realm->getACLs();
```

nRealmACLEntry

Once you have the acl object, you can then add, remove or modify acl entries:

To find a specific acl entry from the realm acl, you can search the acl using the subject. For example, if I wished to change the default permissions for the *@* subject (i.e. the default permission for a realm), I could use the following code:

```
nRealmACLEntry entry = acl.find("Everyone");
 entry.setFullPrivileges(false);
 acl.replace(entry);
 realm.setACLs(acl);
```

C++:

```
nRealmACLEntry* entry = acl->find("Everyone");
 entry->setFullPrivileges(false);
 acl->replace(entry);
 realm->setACLs(acl);
```

which would set the full privileges flag to false for the default subject.

For more information on Universal Messaging Administration, please see the API documentation, and the Enterprise Manager Guide.

Channel Access Control List (nACL)

When connected to a Universal Messaging realm server ,with a reference to an nRealmNode object (see "nRealmNode" on page 583) it is possible to get a reference to an nLeafNode (see "nLeafNode (Channels and Queues)" on page 585) that corresponds to a channel. This can then be used to get access the node's nACL . This object contains a list of nChannelACLEntry objects that represent a subject and a set permissions for various operations on a channel. There is a separate nChannelACLEntry object for each subject that has been permissioned on the nLeafNode.

You can also, add, delete and modify ACL entry objects.

In order to obtain a reference to the correct channel ACL object for a channel called "/products/prices", simply call the following method from a realm node:

Java, C#, C++:

```
nLeafNode chan = realm.findNode("/products/prices");
 nACL acl = chan.getACLs();
```

C++:

```
nLeafNode* chan = realm->findNode("/products/prices");
nACL* acl = chan->getACLs();
```

nChannelACLEntry

Once you have the ACL object, you can then add, remove or modify acl entries:

To find a specific ACL entry from the channel ACL, the ACL object can be searched using the subject.

For example, to change the default permissions for the *@* subject (i.e. the default permission for the channel), the following code can be used:

Java, C#:

```
nChannelACLEntry entry = acl.find("Everyone");
 entry.setFullPrivileges(false);
 acl.replace(entry);
 chan.setACLs(acl);
```

C++:

```
nChannelACLEntry* entry = acl->find("Everyone");
 entry->setFullPrivileges(false);
 acl->replace(entry);
 chan->setACLs(acl);
```

which would set the full privileges flag to false for the default subject.

Click here to see example of how to modify channel ACLs programmatically or to see example of modifying ACLs using the enterprise manager.

For more information on Universal Messaging Administration, please see the API documentation, and the Enterprise Manager Guide.

Queue Access Control List

When you have connected to a realm, and have a reference to an nRealmNode object (see "nRealmNode" on page 583), and an nLeafNode (see "nLeafNode (Channels and Queues)" on page 585) that corresponds to a queue, you can access the node's ACL object. This object contains a list of nChannelACLEntry objects that represent a subject and a set permissions for various operations on a queue.

You can also, add, delete and modify acl entry objects. To obtain the queue ACL object, simply call the following method from a realm node:

Java, C#:

```
nLeafNode queue = realm.findNode("/eur/uk/orders");
 nACL acl = queue.getACLs();
```

C++:

```
nLeafNode* queue = realm->findNode("/eur/uk/orders");
 nACL* acl = queue->getACLs();
```

Once you have the acl object, you can then add, remove or modify acl entries:

nChannelACLEntry

To find a specific ACL entry from the queue ACL, you can search the ACL using the subject. For example, if I wished to change the default permissions for the *@* subject (i.e. the default permission for the queue), I could use the following code:

Java, C#:

```
nChannelACLEntry entry = acl.find("Everyone");
entry.setFullPrivileges(false):
acl.replace(entry);
queue.setACLs(acl);
```

C++:

```
nChannelACLEntry* entry = acl.find("Everyone");
entry->setFullPrivileges(false):
acl->replace(entry);
queue->setACLs(acl);
```

which would set the full privileges flag to false for the default subject.

Click here to see example of how to add queue ACLs programmatically or to see example of modifying ACLs using the enterprise manager.

For more information on Universal Messaging Administration, please see the API documentation, and the Enterprise Manager Guide.

Management Information

nRealmNode

The Universal Messaging admin API provides real time asynchronous management information on all objects within a realm server. By creating an nRealmNode (see "nRealmNode" on page 583), and connecting to a realm, information is automatically delivered to the nRealmNode object from the realm. This information is delivered periodically in summary form, and also as and when the state changes for one or all of the objects managed within a realm.

Before reading this section it may be useful to look at the management information available via the Universal Messaging enterprise manager. A full description of all Realm management screens is available in the enterprise manager guide. All functionality seen in the enterprise manager can be easily added to bespoke admin and monitoring processes as it is written entirely using the Universal Messaging Admin API.

This section discusses the following different types of information that can be obtained through the nAdmin API for the nRealmNode object:

Status Information

The nRealmNode extends nContainer, that extends nNode which is a subclass of Observable, so when the status information is received for a realm node, (by default this is every 5 seconds

although it is configurable (see "Realm Configuration" on page 49) by setting the StatusBroadcast property under the Global Values config group) the nRealmNode will trigger the update callback on any known Observers. For example, if you write a class that implements the Observer interface, it can be added as an observer as follows:

Java, C#, C++:

```
realm.addObserver(this);
```

Assuming 'this' is the instance of the class implementing Observer, then the implementation of the update(Observable obs, Object obj) will be notified that the realm node has changed.

When regular status events are sent, the Observable object referenced in the update method will be the realm node that you added your observer to, and the Object will be null.

State Change Events

When events occur on a realm node that you have added an observer to, the Observable/Observer mechanism will notify you of the details of that event. For example, the following implementation of the update method of the Observer interface demonstrates how to detect that a new channel or queue has been created or deleted :

Java, C#:

```
public void update(Observable obs, Object obj){
  if (obs instanceof nContainer) {
    if (obj instanceof nLeafNode) {
      nLeafNode leaf = (nLeafNode)obj;
      nContainer cont = (nContainer)obs;
      if (cont.findNode(leaf) == null) {
        // node has been deleted
        System.out.println("Node "+leaf.getName()+" removed");
      } else {
        // node has been added
        System.out.println("Node "+leaf.getName()+" added");
      }
    }
  }
}
```

C++:

```
void ObservableMapping::update(Observable *pObs, void *pObj)
{
 if (obs->getType() == fBase::CONTAINER) {
        if (obj->getType() == fBase::LEAFNODE) {
          nLeafNode leaf = (nLeafNode*)obj;
          nContainer cont = (nContainer*)obs;
          if (cont->findNode(leaf)) {
            // node has been deleted
            printf("Node %s removed",leaf->getName());
            System.out.println("Node "+leaf.getName()+" removed");
          } else {
            // node has been added
            printf("Node %s added",leaf->getName());
      }
    }
```

```
  }
}
```

Any changes to the realm ACL will also use the same notification mechanism. For example, if an ACL entry was changed for a realm, the update method would be fired calling with the realm node object and the nACLEntry that had been modified.

Logging and Audit

An nRealmNode allows you to asynchronously receive realm log file entries as well as audit file entries as they occur.

Firstly, for receiving asynchronous log file entries, there is an interface called nLogListener which your class must implement. This interface defines a callback method called report(String) that will deliver each new log entry as a string. Once implemented, the following call will add your log listener to the realm node:

Java, C#, C++:

```
realm.addLogListener(this);
```

Assuming 'this' is the instance of the class implementing the nLogListener interface.

The following is an example of the report(String) method implementation:

Java, C#:

```
public void report(String msg) {
  System.out.println("LOG "+msg);
}
```

C++:

```
printf("Log : %s\n", msg);
```

Secondly, realm servers provide an audit file that tracks object creations and deletions, acl changes, connection attempts and failures. This information can be very useful for tracking who has created ACL entries for example and when they were done.

This information, as with log file entries can be asynchronously received by implementing an interface called nAuditListener. This interface defines a callback method called audit(nAuditEvent) that delivers contains the details of the audit entry. Once implemented, the following call will add your log listener to the realm node:

Java, C#, C++:

```
realm.addAuditListener(this);
```

Assuming 'this' is the instance of the class implementing the nAuditListener.

For more information on Universal Messaging Administration, please see the API documentation, and the Enterprise Manager Guide.

nClusterNode

Universal Messaging's admin API provides real time asynchronous information on all objects within a realm server. By creating an nRealmNode (see "nRealmNode" on page 583), and connecting to a realm, information is automatically delivered to the realm node from the realm. This information is delivered periodically in summary form, and also as and when the state changes for one or all of the objects managed within a realm.

Before reading this section it may be useful to look at the management information available via the Universal Messaging enterprise manager. A full description of all Realm management screens is available in the enterprise manager guide. All functionality seen in the enterprise manager can be easily added to bespoke admin and monitoring processes as it is written entirely using the Universal Messaging Admin API.

This section discusses the following different types of information that can be obtained through the nAdmin API for the nClusterNode object. The nClusterNode corresponds to a cluster that 2 or more realms are members of. Each nRealmNode will have access to its cluster node object once it has been added to a new or existing cluster:

Status Information

Firstly, in order to detect that a cluster node has been created, one has to observer the realm to which you are connected. When the realm is added to a cluster, the Observer/Observable mechanism will notify you of the cluster creation.

As well as implementing the Observer interface to detect new clusters, there is an interface that can be used to be notified of specific cluster events when clusters already exist. This interface is the nClusterEventListener. The interface defines various methods that enable your program to receive callbacks for specific cluster events. When the status changes for a cluster node, this will trigger an callback on any known listeners of the nClusterNode. For example, when you have constructed your nRealmNode, if your class implements the nClusterEventListener interface, then we can do the following:

Java, C#:

```
realm.addObserver(this);
nClusterNode cluster = realm.getCluster();
if (cluster != null) {
  cluster.addListener(this);
}
```

C++:

```
pRealm->addObserver(this);
nClusterNode *pCluster = pRealm->getCluster();
pCluster->addListener(this);
```

If the realm is not part of a cluster, then the getCluster() method will return null. However, by adding an observer to the realm, if a cluster is created that contains the realm you are connected to, the update() method of the Observer implementation will notify you that a cluster has been created. For example, the following code demonstrates how to detect if a cluster has been created with the realm you are connected to as a member:

Java, C#:

```
public void update(Observable o, Object arg) {
  if (arg instanceof nClusterNode) {
    System.out.println("New cluster formed, name = "+( (nClusterNode)arg).getName());
    ((nClusterNode)arg).addListener(this);
  }
}
```

C++:

```
nNode *pNode = iterator->second;
int type = pNode->getType ();
   if (type == fBase::LEAFNODE)
   {
      ((nLeafNode*)pNode)->addListener(new nChannelWatch((nLeafNode*)pNode, this));
   }
```

For more information on how to monitor cluster nodes programmatically please see the appropriate code example.

For more information on how to monitor cluster nodes using the enterprise manager please see the enterprise manager guide.

For more information on Universal Messaging Administration, please see the API documentation and the Enterprise Manager Guide.

nLeafNode

Universal Messaging's admin API provides real time asynchronous information on all objects within a realm server. By creating an nRealmNode (see "nRealmNode" on page 583), and connecting to a realm, information is automatically delivered to the realm node from the realm. This information is delivered periodically in summary form, and also as and when the state changes for one or all of the objects managed within a realm.

Before reading this section it may be useful to look at the management information available via the Universal Messaging enterprise manager. A full description of all Realm management screens is available in the enterprise manager guide. All functionality seen in the enterprise manager can be easily added to bespoke admin and monitoring processes as it is written entirely using the Universal Messaging Admin API.

This section will discuss the basic information that can be obtained through the nAdmin API for the nLeafNode object:

Status Events

The nLeafNode extends nNode which is a subclass of Observable, so when the status information is received for a leaf node, (this occurs only when things change on the channel or queue, i.e. acl, connections, events published / consumed etc) the nLeafNode will trigger the update callback on any known Observers. For example, if you write a class that implements the Observer interface, then we can do the following:

Java, C#:

```
Enumeration children = realm.getNodes();
while (children.hasMoreElements();
  nNode child = (nNode)children.nextElement();
  if (child instanceof nLeafNode) {
    child.addObserver(this);
  }
}
```

C++:

```
pNode->addObserver(this);
pNode->addConnectionListener(new nRealmWatch(this));
fSortedList nodes = registerNodes(pNode->getNodes());
for (fSortedList::iterator iterator = nodes.begin(); iterator != nodes.end();
iterator++)
    {
            if (type == fBase::LEAFNODE)
        {
            ((nLeafNode*)pNode)->addListener(new nChannelWatch((nLeafNode*)pNode,
this));
        }
    }
```

Assuming 'this' is the instance of the class implementing Observer, then the implementation of the update(Observable obs, Object obj) will be notified that the leaf node has changed.

When events occur on a leaf node that you have added an observer to, the Observable/Observer mechanism will notify you of the details of that event. For example, the following implementation of the update method of the Observer interface demonstrates how to detect that a channel or queue acl has been added or deleted:

Java, C#:

```
public void update(Observable obs, Object obj){
  if (obs instanceof nLeafNode) {
    if (obj instanceof nACLEntry) {
      nLeafNode leaf = (nLeafNode)obs;
      nACLEntry entry = (nACLEntry)obj;
      if (leaf.isChannel()) {
        // acl modified / added / deleted
        System.out.println("Channel "+leaf.getName()+" acl event for
"+entry.getSubject());
      } else {
        // acl modified / added / deleted
        System.out.println("Queue "+leaf.getName()+" acl event for
"+entry.getSubject());
      }
    }
  }
}
```

C++:

```
void ObservableMapping::update(Observable *pObs, void *pObj)
{
 if (obs->getType() == fBase::LEAFNODE) {
        if (obj->getType() == fBase::ACLENTRY) {
          nLeafNode leaf = (nLeafNode*)obs;
          nACLEntry entry = (nACLEntry*)obj;
```

```
          if (leaf->isChannel()) {
            // acl modified / added / deleted
          printf("Channel %s acl event for %s",leaf->getName(),+entry->getSubject());
          } else {
            // acl modified / added / deleted
           printf("Queue %s acl event for %s",leaf->getName(),+entry->getSubject());
          }
    }
  }
}
```

For more information on Universal Messaging Administration, please see the API documentation, and the Enterprise Manager Guide.

Connection Information

Universal Messaging's admin API provides real time asynchronous information on all objects within a realm server. By creating an nRealmNode (see "nRealmNode" on page 583), and connecting to a realm, information is automatically delivered to the realm node from the realm. This information is delivered periodically in summary form, and also as and when the state changes for one or all of the objects managed within a realm.

Before reading this section it may be useful to look at the management information available via the Universal Messaging enterprise manager. A full description of all Realm management screens is available in the enterprise manager guide. All functionality seen in the enterprise manager can be easily added to bespoke admin and monitoring processes as it is written entirely using the Universal Messaging Admin API.

This section will discuss the connection information that is available through the nAdmin API for the nRealmNode and the nLeafNode objects:

nRealmNode Connections

The nRealmNode provides the ability to be notified of connections to the realm, and when connections are closed. When a client attempts a connection, a callback will be made that gives the details of the connection, such as the user name, hostname, protocol and connection id. When a user connection is closed, again, you will receive notification. This information can be useful for monitoring activity on a realm.

In order to receive this kind of information, you need to implement the nConnectionListener class. This class defines 2 methods, newConnection and delConnection. To receive notifications, you can use the following method:

Java, C#, C++:

```
realm.addConnectionListener(this);
```

Assuming 'this' is the instance of the class implementing nConnectionListener, then the implementation of the newConnection and delConnection methods will be notified when connections are made or closed with the realm.

nLeafNode Connections

Universal Messaging provides the ability to issue notifications of connections to leaf nodes. Connections to leaf nodes correspond to subscriptions on a channel, so when a user subscribes to a channel or removes the subscription, you can be notified. Notification is via a callback that contains the details of the connection, such as the user name, hostname, protocol, connection id, durable name and subscription filter.

In order to receive this kind of information, you need to implement the nConnectionListener class. This class defines 2 methods, newConnection and delConnection. To receive notifications, you can use the following method:

Java, C#:

```
leafaddListener(this);
```

C++:

```
leaf->addListener(this);
```

Assuming 'this' is the instance of the class implementing nConnectionListener, then the implementation of the newConnection and delConnection methods will be notified when channel subscriptions are made or removed.

8 Thread Pool Monitoring

In addition to the thread dump generation provided by the Java Service Wrapper, you can configure the Universal Messaging realm servers to monitor the thread pool for slow-running threads and generate thread dumps when certain events occur. The thread dumps and messages generated from the user-defined monitoring of the thread pool are logged into *Software AG_directory* \UniversalMessaging\server*InstanceName*\data\nirvana.log.

The thread pool monitoring generates thread dumps for stalled or slow-moving tasks, and reports reduced thread availability. Stalled tasks are tasks that run longer than the specified time. Slow-moving tasks are tasks that run slower than the timeout for the task execution. You can also set a threshold for pending tasks to monitor thread availability. You can then use the thread dump entries in the log to troubleshoot the task execution.

To ensure that the logs are not too big, you can configure the interval at which the server generates a thread dump.

Thread-Pool Monitoring Configuration Properties

You configure the thread-pool monitoring properties for a realm in the **Thread Pool Config** group on the Config tab in the Enterprise Manager.

For information about working with Universal Messaging configuration properties in the Enterprise Manager, see .

StalledTaskWarningTime
The time in milliseconds before reporting a stalled task. The system writes the information at the WARNING log level and generates a thread dump. When you change this configuration, the thread pool monitor interval is updated to monitor at the same time interval as the value you specify for this property. Valid values range from `10000` to `60000`. Default is `60000`.

SlowTaskWarningTime
The time in milliseconds before reporting a slow-running task. The server logs the information at the WARNING log level and generates a thread dump. Valid values range from `1000` to `30000`. Default is `5000`.

PendingTaskWarningThreshold
The threshold at which the server starts to warn about the number of pending tasks. When the number of pending tasks is below the threshold, but over `100`, the server logs a WARNING message. When the number is above the threshold, the server logs an ERROR message. When

the server does not find available threads, it logs a message that the thread pool is exhausted. Valid values range from `100` to `100000`. Default is `1000`.

ThreadDumpOnSlowTask
Whether to generate a thread dump when the server reports a slow task. Valid values are `true` - generate a thread dump, or `false` - do not generate a thread dump. Default is `false`.

ThreadDumpInterval
The interval in milliseconds at which a thread dump is generated when the system reports slow or stalled tasks, or when the number of pending tasks exceeds the value of PendingTaskWarningThreshold. The thread dump interval applies across all thread pools in the JVM instance. Valid values range from `1000` to `600000`. Default is `60000`.

Examples

In the following example, the slow-moving task warning timeout is set to 1000ms and the server is configured to generate a thread dump for a slow task. The task completed in 1060ms and the server reports that the task execution time exceeds 1000ms and generates a thread dump. The following entries will show in the log:

```
[Wed Feb 17 07:39:32.790 IST 2021] [ThreadPoolTest-Slow:9] ThreadPool:
<ThreadPool-SlowTasksTest> Slow moving task detected. ThreadPool-SlowTasksTest:9 has
 been active
for over 1060(ms) running task class
com.pcbsys.foundation.threads.fThreadPoolTaskReportTest$TestTask, Idle Threads 7,
Allocated
Threads 10, Queued Tasks 0, Task Executed 10
[Wed Feb 17 07:39:32.790 IST 2021] [ThreadPoolTest-Slow:0] ThreadPool:
<ThreadPool-SlowTasksTest>
Slow moving task detected. ThreadPool-SlowTasksTest:0 has been active for over 1060(ms)
 running
task class com.pcbsys.foundation.threads.fThreadPoolTaskReportTest$TestTask, Idle
Threads 7,
Allocated Threads 10, Queued Tasks 0, Task Executed 10
```

```
[Wed Feb 17 07:39:41.000 IST 2021] [ThreadPoolTest-Slow:7] ThreadPool:
<TThreadPool-SlowTasksTest> Slow moving task detected. ThreadPool-SlowTasksTest:7 has
 been active
for over 1020(ms) running task class
com.pcbsys.foundation.threads.fThreadPoolTaskReportTest$TestTask, Idle Threads 7,
Allocated
Threads 10, Queued Tasks 0, Task Executed 10
[Wed Feb 17 07:39:41.000 IST 2021] [ThreadPoolTest-Slow:7] Producing thread dump.
Reason : Slow
moving task detected on thread pool: ThreadPool-SlowTasksTest
```

In the following example, the threshold for pending tasks is set to 200. Because the number of pending tasks in the thread pool is 209, the server logs an error message and generates a thread dump. The following entries will show in the log:

```
[Wed Feb 17 07:39:39.949 IST 2021] [Time-limited test] ThreadPool:
<ThreadPoolTest-Pending> Pending tasks are above the threshold 200 pending tasks 209,
 Idle Threads
0, Allocated Threads 1, Queued Tasks 209, Task Executed 210
[Wed Feb 17 07:39:39.949 IST 2021] [Time-limited test] Producing thread dump. Reason
 : Pending
```

```
tasks are above the threshold: 200 pending tasks: 209
```

Troubleshooting Task Execution

To correct or improve the task execution, you can take the following actions:

- In the thread pool configuration, check whether you have allocated enough threads. If the logs report a large number of queued tasks, allocate more threads to the pool.
- Check for overall system slowdown, such as disk speed, network speed, CPU speed and allocation, and JVM garbage collection.
- Check for product behavior that might cause a slow performance.

9 Data Protection and Privacy

Introduction

Legislation in various parts of the world – such as the General Data Protection Regulation (GDPR) of the European Union (EU) - specifies that personal data cannot be collected and processed without a person's consent or other legitimate basis, and that organizations are responsible for protecting personal data that is entrusted to them. The concept of "personal data" typically covers details that can be used to identify a person, such as the person's name, email address or IP address.

Note:
In the different countries of the EU, the GDPR may be known under another, language-specific name. For example, it known as the Datenschutz-Grundverordnung (DSGVO) in Germany and as Règlement Général sur la Protection des Données (RGPD) in France.

Universal Messaging includes personal data such as user names, and client IP addresses / host names in the logs. Universal Messaging includes personal data in logs for purposes of auditing, monitoring activity with the server, and diagnosing and correcting problems.

Universal Messaging is a middleware platform on which customers build their own applications. Most of the data handled by Universal Messaging is arbitrary customer-defined data whose meaning is defined by the customer who developed the application. Some of that customer-defined data may qualify as "personal data", so if you are developing applications on the Universal Messaging platform, you should be careful to ensure compliance with laws related to that data.

If Software AG support personnel request you to send diagnostic data such as operational logs for the purposes of diagnosing product issues, and if this diagnostic data contains personal data, you should be aware that Software AG has GDPR processes in place to ensure that data is held securely and deleted when no longer needed.

Summary of Log Files used by Universal Messaging

Universal Messaging uses the log files described in the following table. The log files can contain personal data associated with a current activity, such as a user ID and client IP address. The length of time that a Universal Messaging server stores log data depends on the log.

Log	
standard server log file	The log file is named `nirvana.log` and resides in the `server/<RealmServerName>/data` directory. The data remains there for as long as the log file is retained. When using the default Universal Messaging logger, the log file policy is defined by the `DefaultLogSize` realm server property which defines the maximum size of the log file, and the `RolledLogFileDepth` realm server property, which defines the number of log files to keep if log rolling is activated. The personal data can be removed by either manually removing lines from the file, or deleting a log file altogether.
audit log file	The audit log file is named `NirvanaAudit.mem` and resides in the `server/<RealmServerName>/data/RealmSpecific` directory. The data remains in the audit log file for as long as the file exists. There is no mechanism to partially remove data from this log file. The only way to remove the personal data is by deleting the `NirvanaAudit.mem` file.

Ad-hoc creation of data collections

In addition to standard operational data that is collected by Universal Messaging, some data can be collected on an ad-hoc basis by the Universal Messaging administrator. Such ad-hoc data is typically written to a location on your file system.

Examples are:

Realm Information Collector
The files collected by the Realm Information Collector tool can include files that may contain personal data related to messages that are being handled by the server.

Exported Realm Configuration File
When you export a realm's configuration to an XML file for a later re-import, the XML file can contain personal data, such as user IDs and client IPs related to ACL permissions for accessing realm components.

Heap Dump
A heap dump (which Software AG may request you to generate for the purpose of diagnosing problems) may contain personal data related to messages that are being handled by the server, or the server's log files.

Protecting and erasing data from log files

As there are many situations in which user names, IP addresses or events containing personal data may be logged, including by customer-provided plug-ins and third-party libraries, it is not

practical to enumerate all of the log messages that may contain such data, or the set of categories they may be logged under.

Log files are formatted for reading by human system administrators (not machines), so rectification of data contained within them does not make sense, and erasure of data for individual persons is not practical. The retention of complete information in log files also serves an important and legitimate purpose, in providing a security audit trail, and the ability to diagnose and fix accidental or unlawful events compromising the availability, integrity or confidentiality of the application and personal data it contains.

For these reasons, the recommended approach to protecting personal data in log files is to regularly rotate the logs (also termed log rolling) in cases where log rotation is activated, and archive the old log files to a secured location protected by encryption.