

Universal Messaging Administration Guide

Version 9.8

April 2015

This document applies to Universal Messaging Version 9.8 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 © 2013-2015 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 <http://softwareag.com/licenses>.

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 <http://softwareag.com/licenses> and/or in the root installation directory of the licensed product(s).

Use, reproduction, transfer, publication or disclosure is prohibited except as specifically provided for in your License Agreement with Software AG.

Table of Contents

Overview.....	7
Universal Messaging Enterprise Manager.....	9
Introduction.....	10
Getting Started.....	10
Tab-by-Tab Overview.....	11
Administration Using Enterprise Manager.....	14
Enterprise View.....	14
Realm Administration.....	15
Connecting to Multiple Realms.....	15
Disconnecting from Realms.....	17
Editing Connection Information.....	19
Realm Profiles.....	21
Realm Federation.....	22
Realm Configuration.....	25
Zone Administration.....	53
Cluster Administration.....	56
Creating a Cluster.....	57
Deleting Clusters.....	61
Modifying Clusters.....	62
Cluster Channel Administration.....	64
Cluster Queue Administration.....	67
Viewing Cluster Information.....	70
Manage Inter-Cluster Connections.....	72
Creating and Managing Clusters with Sites.....	75
Channel Administration.....	79
Channel Creation.....	80
Channel Editing.....	85
Copying Channels.....	88
Creating Channel Joins.....	90
Channel Snoop.....	93
Channel Publishing.....	95
Channel Named Objects.....	98
DataGroup Administration.....	99
Creating DataGroups.....	100
Adding Existing DataGroups to DataGroups.....	104
Removing DataGroups from DataGroups.....	106
Deleting DataGroups.....	107
Queue Administration.....	109
Creating Queues.....	109
Editing Queues.....	115

Copying Queues.....	118
Queue Snoop.....	120
Security.....	122
Nirvana Enterprise Manager - Security Groups.....	123
Realm Entitlements.....	125
Channel Entitlements.....	127
Queue Entitlements.....	129
P2P Service.....	131
Interface VIA Rules.....	133
Scheduling.....	134
Universal Messaging Scheduling : Writing Schedule Scripts.....	135
Universal Messaging Scheduling : Calendar Triggers Schedules.....	139
Universal Messaging Scheduling : Conditional Triggers.....	141
Universal Messaging Scheduling : Tasks.....	154
Universal Messaging Scheduling: Editor.....	167
Scheduler Examples.....	173
Universal Messaging Scheduling : Example Realm Script.....	174
Universal Messaging Scheduling : Store Triggers Example.....	174
Universal Messaging Scheduling : Interface Triggers Example.....	176
Universal Messaging Scheduling : Memory Triggers Example.....	176
Universal Messaging Scheduling : Realm Triggers Example.....	176
Universal Messaging Scheduling : Cluster Triggers Example.....	177
Universal Messaging Scheduling : Counter Trigger Example.....	177
Universal Messaging Scheduling : Time Triggers Example.....	178
Universal Messaging Scheduling : Configuration Example.....	178
Integration with JNDI.....	179
Universal Messaging Enterprise Manager Comms: TCP Interfaces, IP Multicast and SHM.....	185
Creating Interfaces.....	187
Deleting Interfaces.....	190
SSL Interfaces.....	191
Stopping Interfaces.....	191
Starting Interfaces.....	192
Interface Configuration.....	192
JavaScript Interface Panel.....	195
Modifying Interfaces.....	198
Interface plugins.....	199
Interface VIA Rules.....	199
Multicast Configuration.....	201
Shared Memory Configuration.....	206
Creating an SSL network interface to a Universal Messaging Realm server.....	208
How to generate certificates for use.....	212
Plugins.....	215
File Plugin.....	216
XML Plugin.....	219

Proxy Passthrough Plugin.....	224
REST Plugin.....	226
SOAP Plugin.....	244
Servlet Plugin.....	248
XML Configuration: Overview.....	249
XML Configuration: Exporting To XML.....	250
XML Configuration: Importing From XML.....	251
XML Configuration: Sample XML File for EXPORT.....	252
Management and Monitoring Sections.....	256
Enterprise view.....	256
Management Information.....	258
Enterprise Summary.....	259
Clusters Summary.....	261
Clusters Status.....	262
Realms Summary.....	264
Realm Status.....	266
Realm Monitoring.....	268
Universal Messaging Enterprise Manager : Logs Panel.....	268
Realm Connections.....	272
Threads Status.....	275
Top.....	277
Audit Panel.....	280
Container Status.....	283
Container Monitor Panel.....	285
Channel Status.....	288
Data Group Status.....	290
Channel Connections.....	292
Queue Status.....	295
Interface Status.....	297
Scheduler view.....	299
Channel view.....	304
Queue view.....	310
Peer 2 Peer view.....	313
Universal Messaging Administration API.....	317
Introduction.....	318
Administration API Package Documentation.....	320
Namespace Objects.....	321
nRealmNode.....	321
nLeafNode (Channels and Queues).....	323
nServiceNode (P2P Services).....	325
Realm Federation.....	326
Channel Join.....	327
Realm Server Management.....	328
Interfaces.....	328

Scheduling.....	330
Config.....	331
Clustering.....	332
Multicast.....	334
Security.....	335
Access Control Lists.....	335
Nirvana Admin API - Nirvana Security Groups.....	336
Realm Access Control List (nACL).....	337
Channel Access Control List (nACL).....	338
Queue Access Control List.....	339
P2P Service Access Control List.....	339
Management Information.....	340
nRealmNode.....	340
nClusterNode.....	343
nLeafNode.....	344
nServiceNode.....	346
Connection Information.....	347

Overview

This administration guide covers the following areas:

- ["The Enterprise Manager" on page 9](#): a graphical user interface for management of your Universal Messaging environment.
- ["The Administration API" on page 317](#): a powerful API that allows you to build applications to manage your Universal Messaging environment programmatically.

1 Universal Messaging Enterprise Manager

■ Introduction	10
■ Administration Using Enterprise Manager	14
■ Management and Monitoring Sections	256

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

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

The Universal Messaging 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 enabled 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 Universal Messaging enterprise manager GUI.

Introduction

Getting Started

In order to start administering and monitoring your Universal Messaging Realm servers you need to launch the Universal Messaging 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 21).

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:

```
<UM_install_dir>\java\<UM_server_name>\bin\nenterprisemgr.exe
```

where `<UM_install_dir>` is the installation root location and `<UM_server_name>` 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 for information about setting up basic authentication.

Tab-by-Tab Overview

This document 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 left hand pane).

- ["Universal Messaging Enterprise" on page 11](#)
- ["Realm Nodes" on page 11](#)
- ["Container \(Folder\) Nodes" on page 13](#)
- ["Channel Nodes" on page 13](#)
- ["Queue Nodes" on page 13](#)

Universal Messaging Enterprise View

Highlighting the **Universal Messaging 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 GC stats, 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.
 - **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.

■ Named Objects

Enables the viewing and deletion of named objects (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.

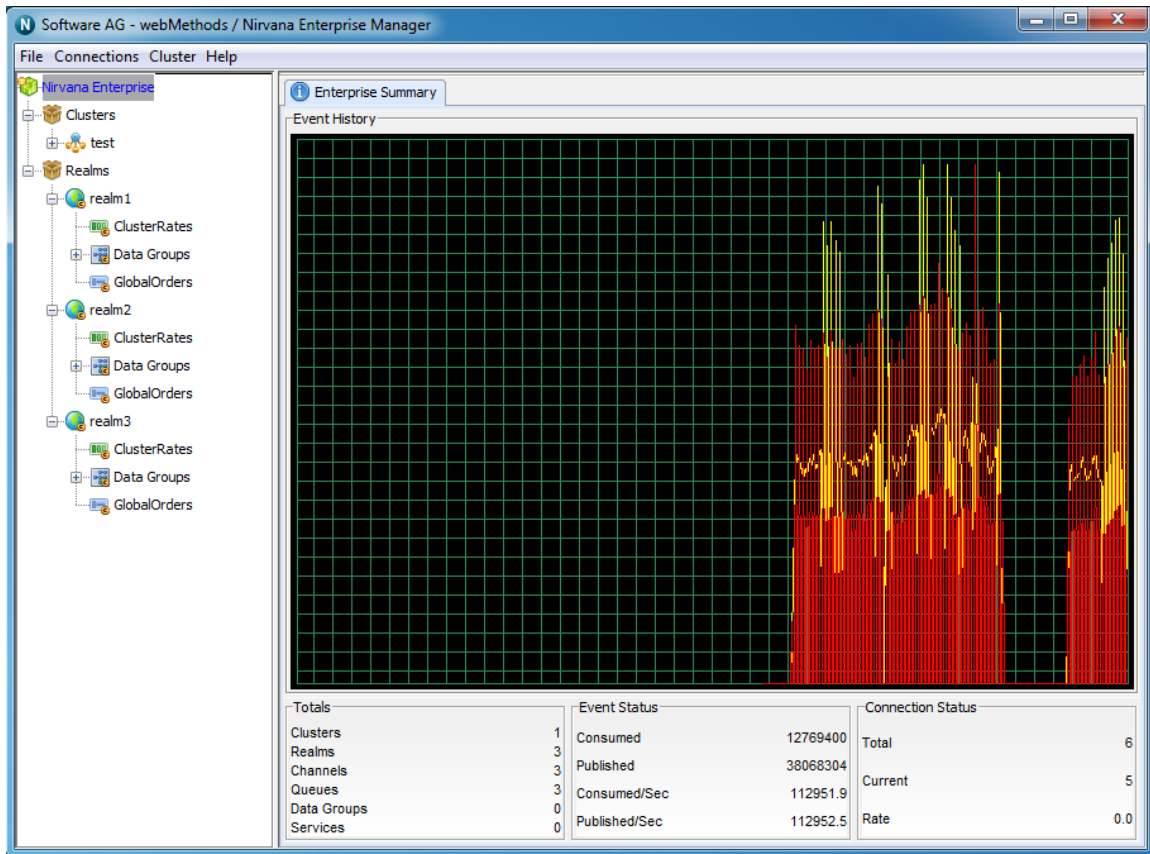
Administration Using Enterprise Manager

Enterprise View

The enterprise view is the first screen you see whenever the Universal Messaging 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 will include any type of Universal Messaging realm you have added to your connection information whether they are standalone development realms or production clustered realms. Adding or removing Universal Messaging realms to the enterprise manager's connection info will result in those realm's data to be also included in this view (see ["Connecting to Multiple Realms" on page 15](#) and ["Disconnecting from Realms" on page 17](#)).

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 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 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, resources and peer 2 peer services across all Universal Messaging realms. The **Event Status** panel displays the total number of events consumed, published as well as the current consume and publish rates. Finally the **Connection Status** panel displays the total number, the current number as well as the rate of connections (sessions), whether application or administrative, across all Universal Messaging realms.



Realm Administration

Connecting to Multiple Realms

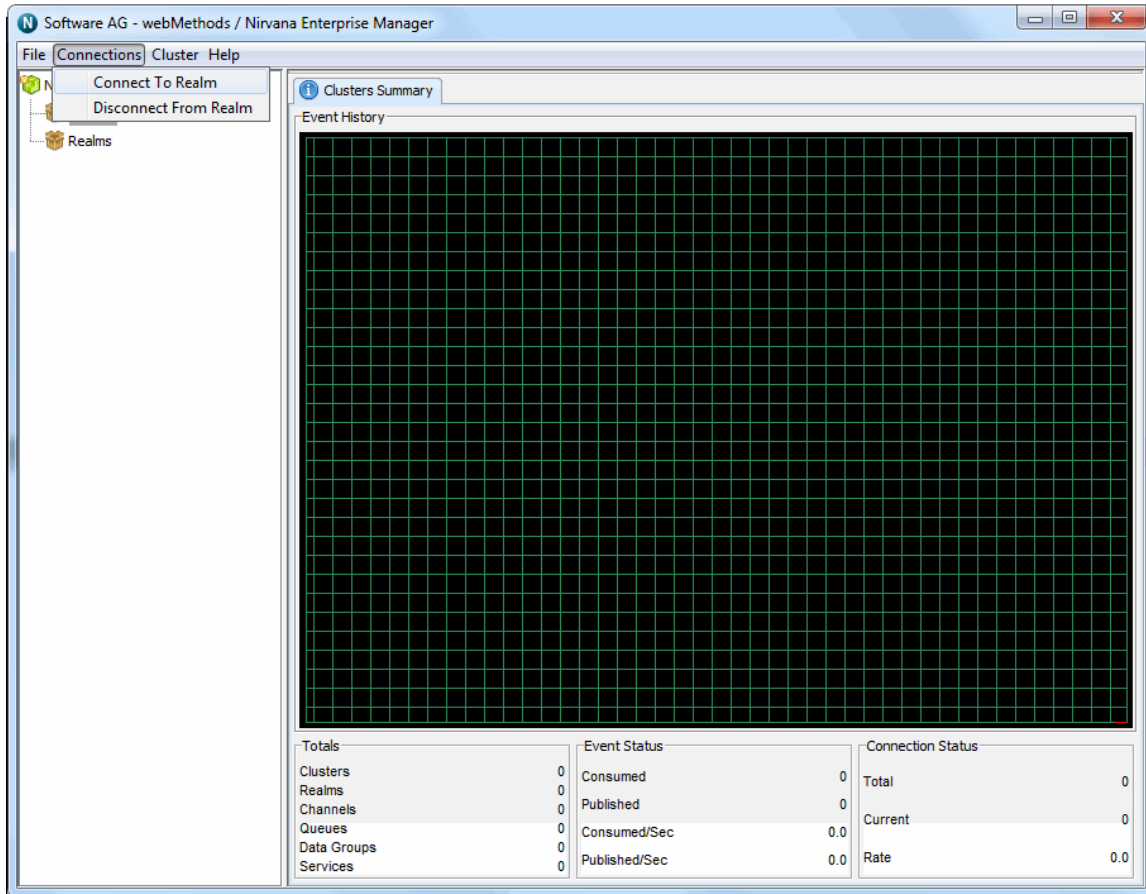
An Enterprise Manager has the ability to connect to multiple Universal Messaging realms at the same time. These realms can be standalone or clustered so developers and administrators can now manage and monitor the whole Universal Messaging enterprise infrastructure from a single instance of Enterprise Manager. Once connected to a set of Universal Messaging realms, it is possible to save (see ["Realm Profiles" on page 21](#)) 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.

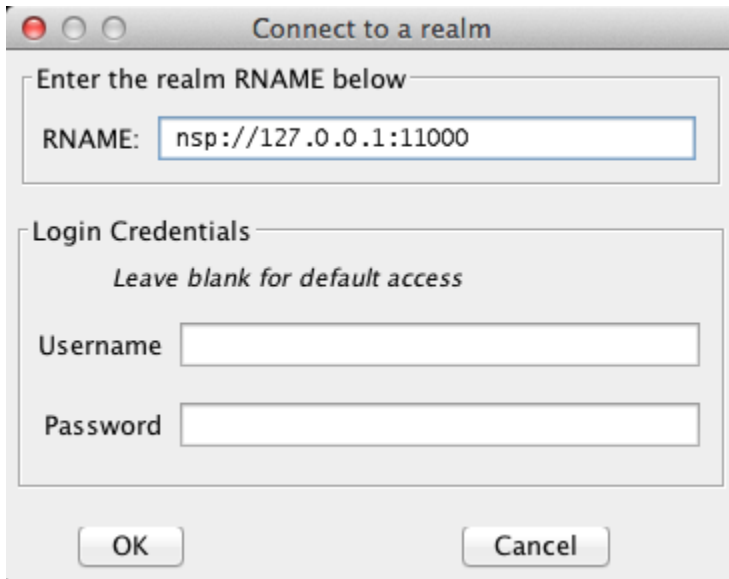
For more information on how to set the RNAME variable, see the section *Communication Protocols and RNAMEs* in the Developer Guide.

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:

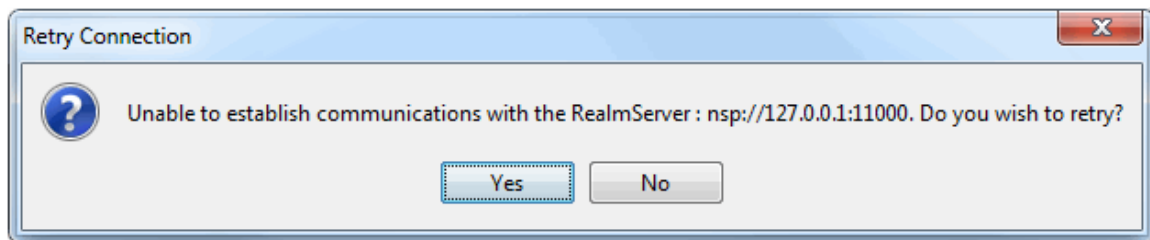


The menu option will open the Connect To Realm dialog as shown in the figure below:



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.

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:



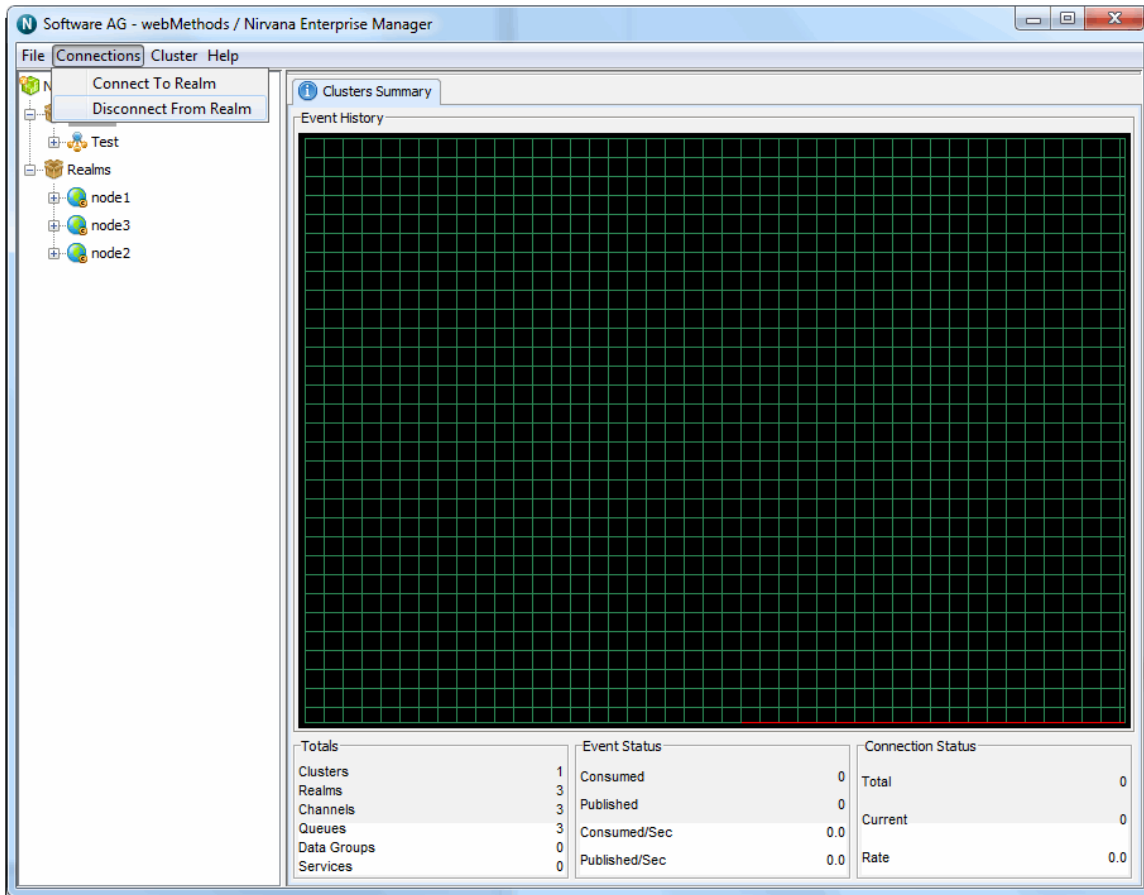
If you had typed the correct RNAME this gives you the opportunity to start the Universal Messaging realm or interface needed and click yes to retry the connection without entering the information again. If however the RNAME entered was wrong or you do not wish to retry then clicking no will close the 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

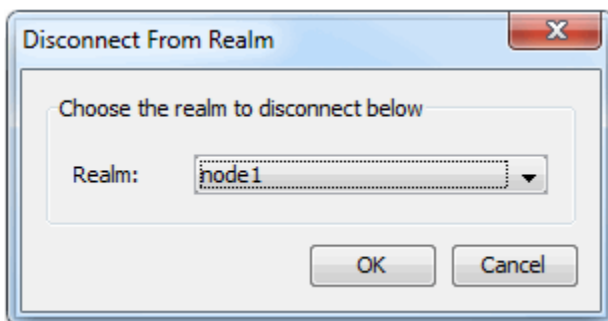
Using the multiple realm connection functionality, the startup time of the Enterprise manager is slightly increased each time you add a Universal Messaging realm to your connection list. If you connect from a different location or network, if the development phase of a Universal Messaging application 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 Enterprise manager will then

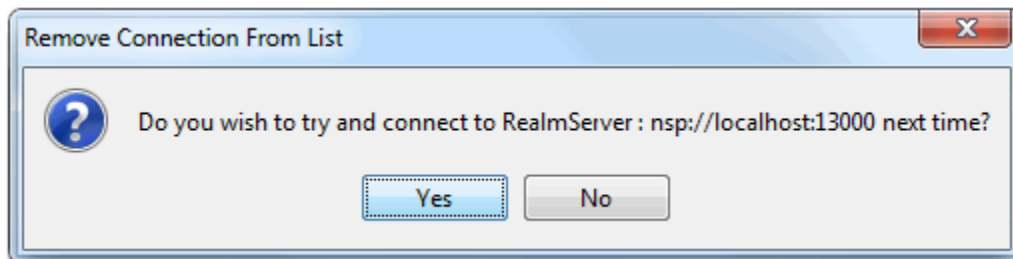
disconnect from that Universal Messaging realm and the realm node with all its sub nodes will disappear from the namespace tree.

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 21) your connection information after you disconnect.

Editing Connection Information

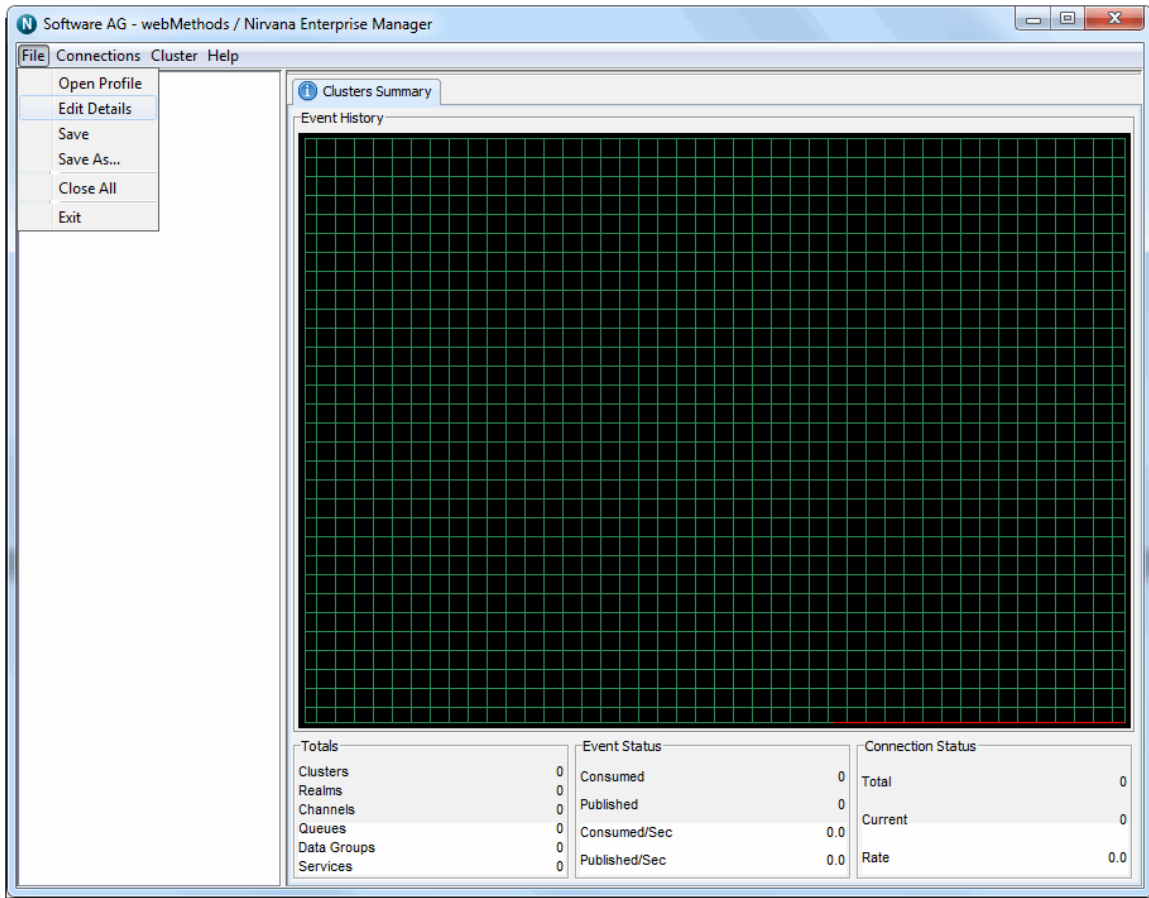
As mentioned in previous sections, Universal Messaging Enterprise manager can connect to multiple Universal Messaging realms at the same time and allows saving connection information in a configuration file. This configuration file can change in one of 3 ways:

1. By selecting the Save Connection Info menu option (see "[Realm Profiles](#)" on page 21) 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:

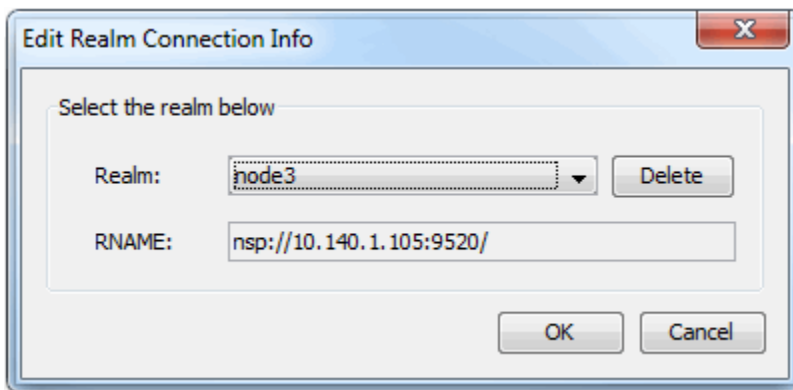


If the user clicks Yes, then the configuration file remains the same. However if the user chooses no, the failed connection is removed from the configuration file without any further action required. The Enterprise manager will never try to connect to that Universal Messaging realm again during startup.

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:

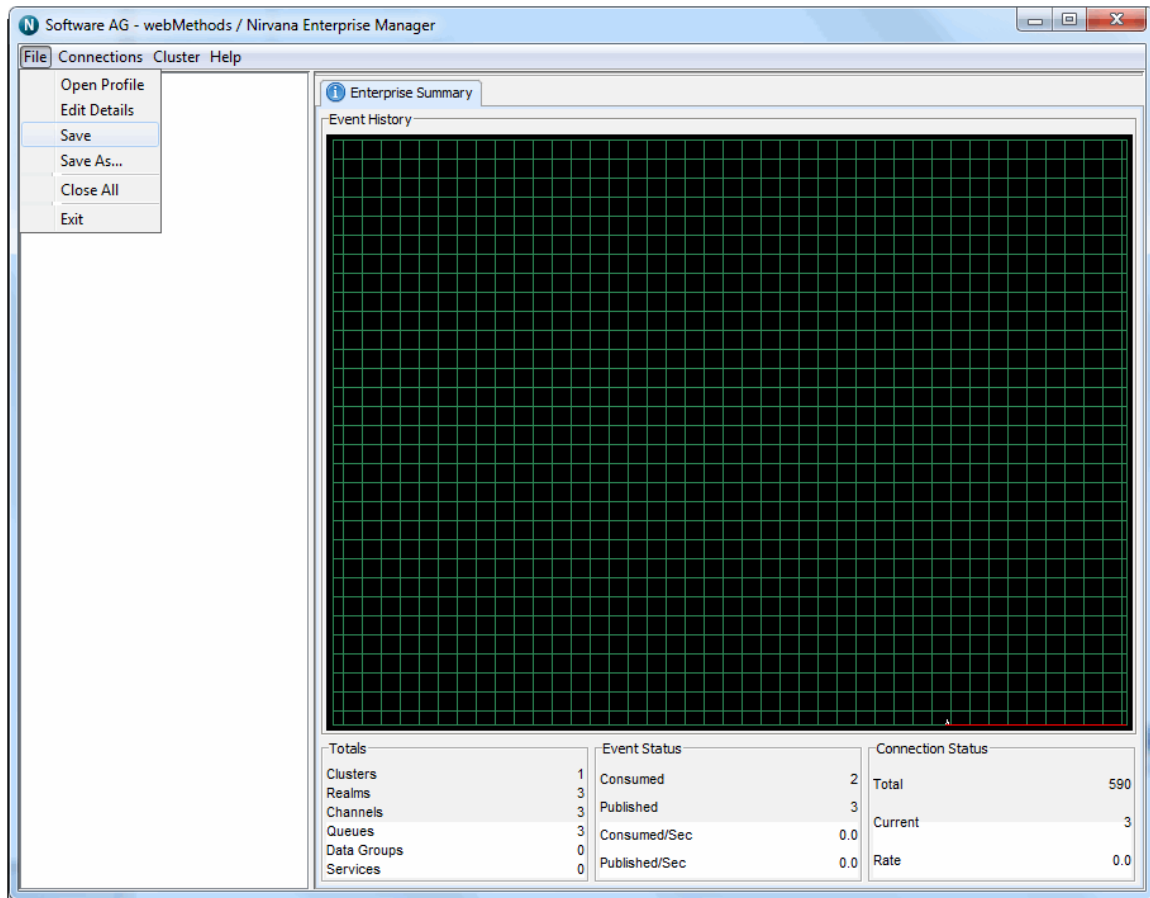


The Realm name combo box contains the complete list of configured Universal Messaging realms that had been connected during the last Save Connection Info operation. If you have connected to additional realms that had not been saved, these will not be included in this list. By selecting a particular Realm 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

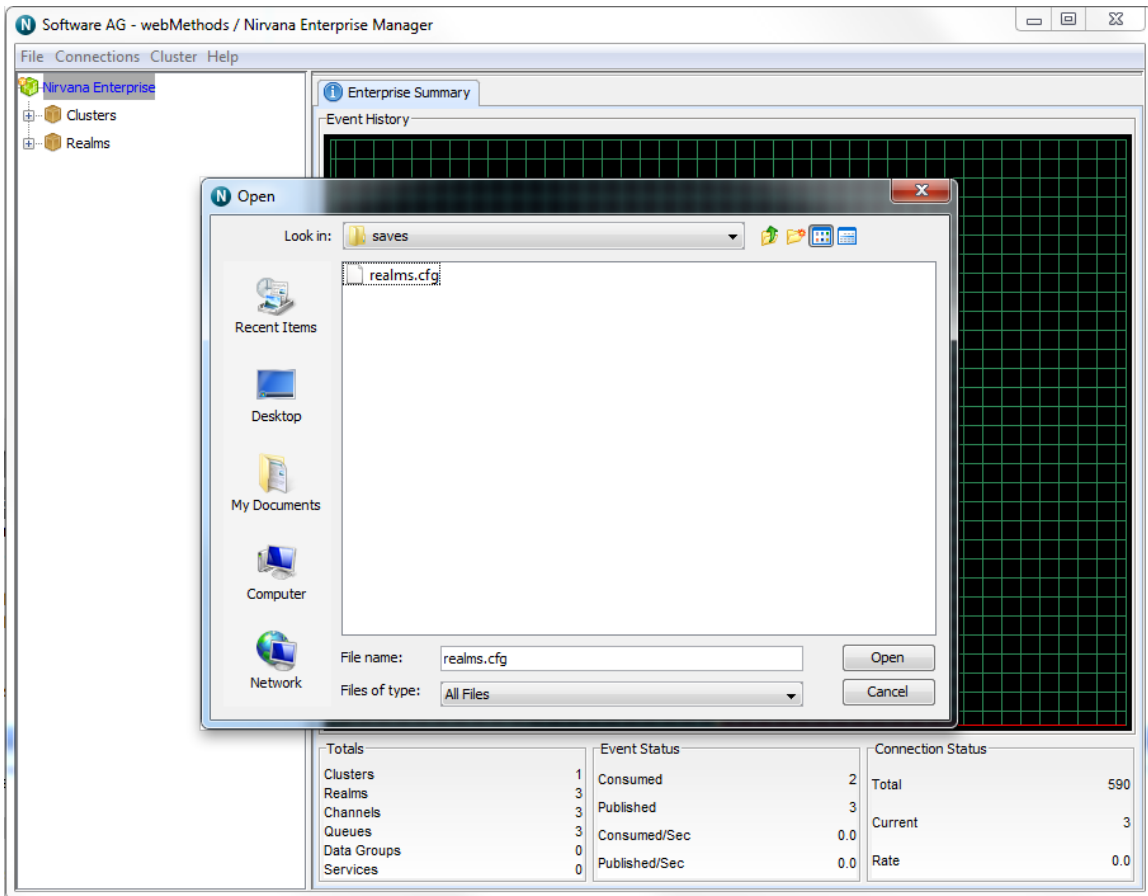
realms are present in the list. When this is done, click on the Save button to recreate the connection info file.

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.

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 1 of 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 join (see "[Channel Join](#)" on page 327) between these realms. The second option also makes a new communication connection, however if you specify a *'mount point'*, the realm you add will also be visible within the namespace of the realm you added it to.

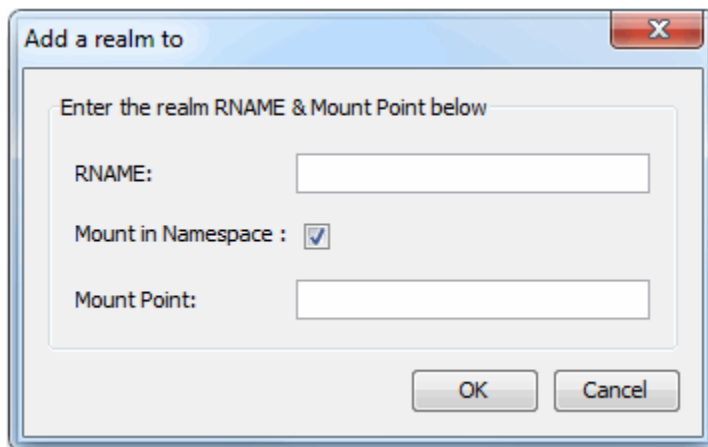
Mount Points

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, and 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

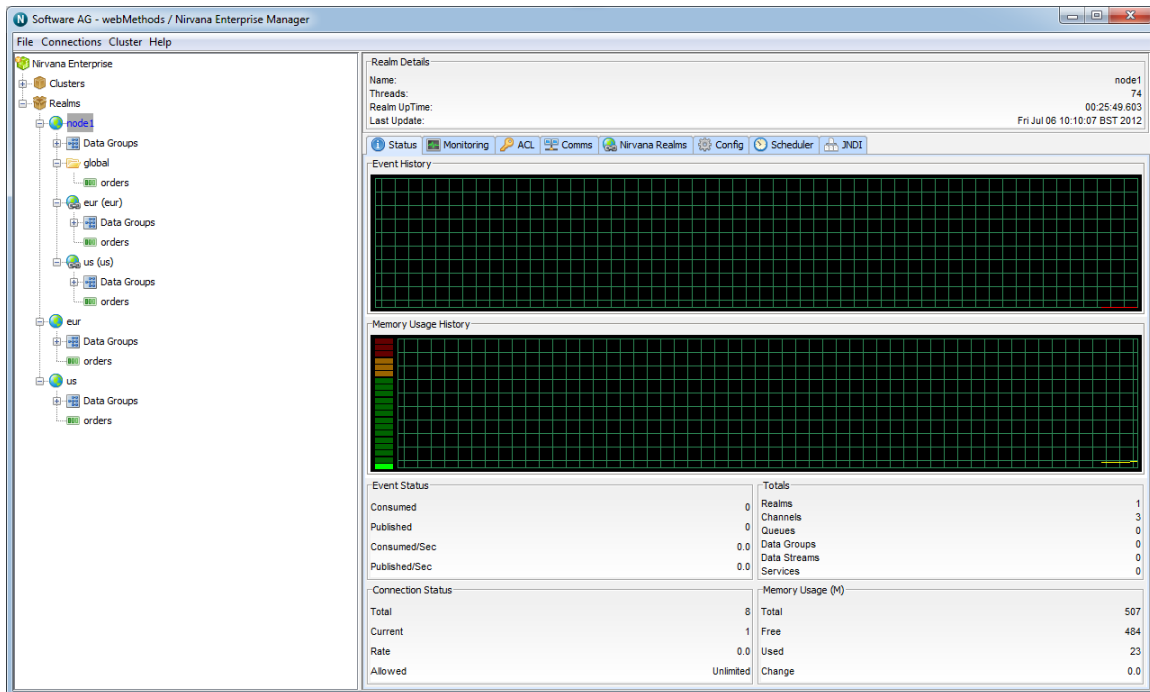
In order to add a realm to another realm, first of all you need to select the realm node from the namespace that you wish to add the realm to. Then, right-click on the realm node to display the menu options available for a realm node. One of the menu options is labelled *'Add Realm to Namespace'*, clicking 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 (see "[Channel Join](#)" on page 327) 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

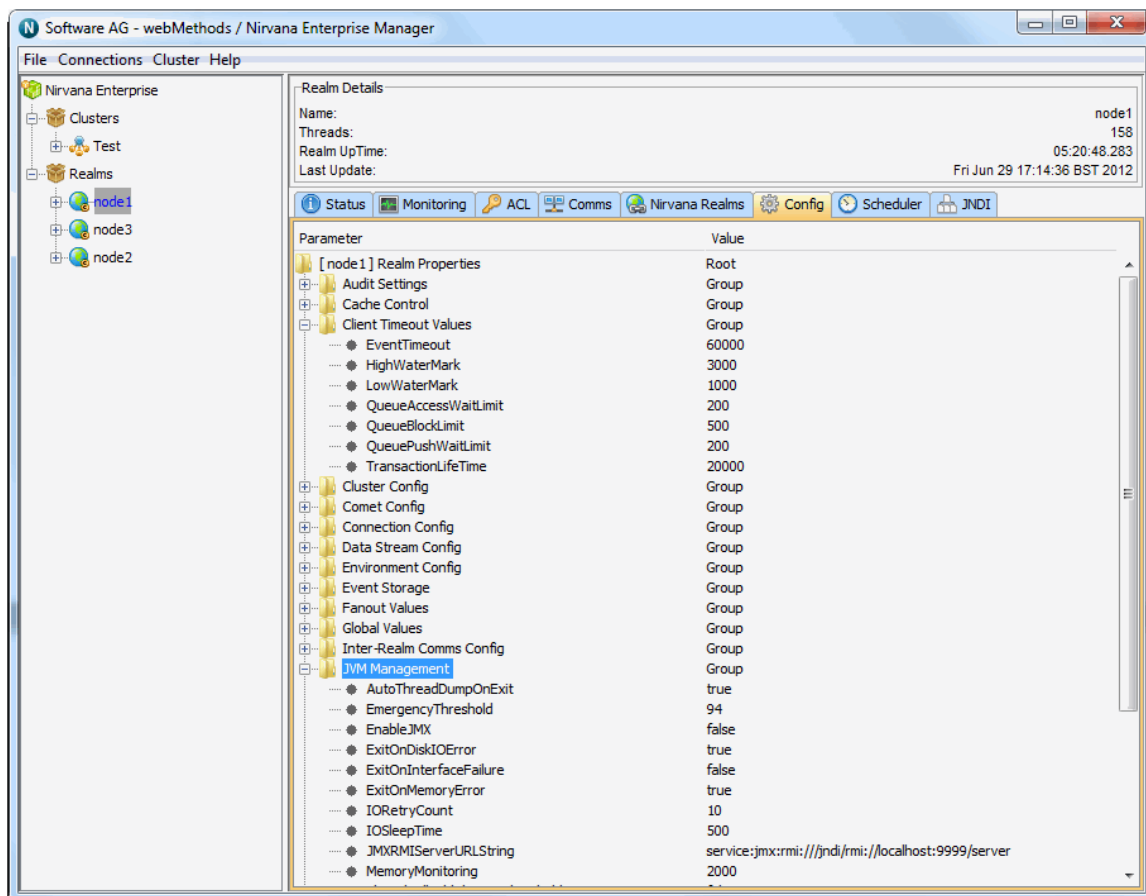
efficient in this respect, and ensure the data is available as close (physically) to the consumers as possible.

Realm Configuration

Universal Messaging Realms can be configured based on a number of parameters that are accessible both through the Universal Messaging Administration API as well as the Universal Messaging Enterprise Manager. Any changes made to the configuration parameters 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.

This section will describe the different configuration parameters 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 20 sets of configuration properties, with each set of properties relating to a specific area within the Universal Messaging Realm. Each set of properties contains different values for specific items. The image below shows the config panel for a selected realm within the namespace.



The 20 configuration groups are :

- **Audit Settings** - Values relating to what information is stored by the audit process
- **Client Timeout Values** - Values relating to client / server interaction
- **Cluster Config** - Values specific to the clustering engine
- **Comet Config** - Values relating to the configuration of Comet
- **Connection Config** - Values relating to the client server connection
- **Data Stream Config** - Values relating to the configuration of Data Streams
- **Environment Config** - Read only configuration values that relate to the system environment. These cannot be changed.
- **Event Storage** - Values specific to how events are stored and retrieved on the server
- **Fanout Values** - Values specific to the delivery of events to clients
- **Global Values** - Values specific to the realm process itself
- **Inter-Realm Comms Config** - Values relating to Inter-Realm communication
- **JVM Management** - Values relating to the JVM the Realm Server is using
- **Join Config** - Values specific to channel join management
- **Logging Config** - Values specific to logging
- **MQTT Config** - MQTT specific configs
- **Plugin Config** - Values relating to Realm Plugins
- **Protobuf Config** - Values relating to Protocol Buffers
- **RecoveryDaemon** - Values relating to clients that are in recovery (i.e. replaying large numbers of events)
- **Thread Pool Config** - Values specific to the servers thread pools.
- **TransactionManager** - Values specific to the transaction engine of the RealmServer

The table below describes each of the configuration groups and the items that can be changed within each group. It also shows valid ranges of values and a description of what each value actually represents.

Configuration Element	Valid Parameters	Description
Audit Settings		
ChannelACL	True or False	Log to the audit file any unsuccessful channel acl interactions. Default is true.

Configuration Element	Valid Parameters	Description
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.
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.
QueueSuccess	True or False	Log to the audit file any successful queue

Configuration Element	Valid Parameters	Description
		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.
ServiceACL	True or False	Log to the audit file any unsuccessful service acl interactions. Default is true.
ServiceFailure	True or False	Log to the audit file any unsuccessful service interactions. Default is true.
ServiceMaintenance	True or False	Log to the audit file any service maintenance activity. Default is true.
ServiceSuccess	True or False	Log to the audit file any successful realm interactions. Default is false.
Client Timeout Values		
EventTimeout	5000 to No Max	The amount of ms the client will wait for a

Configuration Element	Valid Parameters	Description
HighWaterMark	2 to No Max	<p>response from the server. Default is 60000.</p> <p>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 200.</p>
LowWaterMark	1 to No Max	<p>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 50.</p>
QueueAccessWaitLimit	200 to No Max	<p>The maximum number of milliseconds it should take to gain access to a queue to push events before notifying listeners. Default is 200</p>
QueueBlockLimit	500 to No Max	<p>The maximum number of milliseconds a queue will have reached HWM before notifying listeners, Default is 500.</p>
QueuePushWaitLimit	200 to No Max	<p>The maximum number of milliseconds it should take to gain access to a queue and to push events before notifying listeners. Default is 200.</p>
TransactionLifeTime	1000 to No Max	<p>The default amount of time a transaction is valid before being removed from the tx store. Default is 20000.</p>

Configuration Element	Valid Parameters	Description
Cluster Config		
BufferSize	1400 to 1048576	Size in bytes of the inter realm buffer to use
ClientQueueSize	10 to 10000	Size of the client request queue
ClientQueueWindow	10 to 1000	The number of events sent to an async queue reader before the realm will commit
ClientStateDelay	0 to 120000	The number of seconds to delay the cluster processing client requests when a cluster state change occurs
DisableHTTPConnections	True or False	Disable HTTP(s) connections between cluster nodes
DisconnectWait	1000 to 120000	Time to wait for the node to form in the cluster
DisconnectWhenNotReady	True or False	If the node has not formed in the cluster then disconnect the client
EnableMulticast	True or False	Enables multicast
EnableSites	True or False	If enabled then the master selection takes into account the Prime Site
EnableStoreRecoveryRetry	True or False	Enables/Disables the ability for the slave to reattempt a recovery of a store if it detects changes to the store during recovery
EnginePipelineSize	1 to 32	Size of the number of concurrent pipeline

Configuration Element	Valid Parameters	Description
		threads running within the cluster engine
FastSlaveMode	True or False	If enabled the realm will move to a slave state while finalizing the recovery of local stores
FilterEventsDuringRecovery	True or False	Only Applicable to JMS Engine Channels. Defines if we recover events that have already been consumed.
FormationTimeout	60000 to 300000	The time to wait for the state to move from recovery to slave or master
HeartBeatInterval	1000 to 120000	Heart Beat interval in milliseconds. Default is 120000.
InitialConnectionTimeout	5000 to 240000	The number of milliseconds that the server will wait while trying to establish a connection to a peer.
IsCommittedDelay	1000 to 30000	When a slave processes a IsCommitted request and it is still recovering the Transaction store, it will block the clients request for this timeout period.
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.
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

Configuration Element	Valid Parameters	Description
		master to reconnect. If the master fails to 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
QueueSize	100 to 1000	Number of events outstanding to be processed by the clusters internal queue before sending flow control requests back
SecureHandshake	True or False	If true, when peers connect they will perform a secure handshake to ensure the connection is valid. This is the preferred and secure option. Disabling this would only be recommended in debug mode.
SeperateLog	True or False	Create a separate log file for cluster events. Default is false.
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.
SyncPingSize	100 to 10000	Number of events sent before a cluster sync occurs
TransactionSync	True or False	Make all transactional based events sync across the cluster

Configuration Element	Valid Parameters	Description
Comet Config		
BufferSize	1024 to 102400	The buffer size for comet request.
EnableLogging	True or False	Enables logging of all comet queries
Timeout	10000 to No Max	The timeout for a comet connection.
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.
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.
BufferPoolSize	100 to 10000	The underlying Universal Messaging IO utilizes buffers from a pool. By default we preload 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, if too

Configuration Element	Valid Parameters	Description
BufferQueueSize	10 to 1000	<p>large we have a pool of buffers which are not used taking up memory.</p> <p>Number of buffers to queue before we stop reading from the socket.</p>
BufferSize	1024 to 1048576	<p>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 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).</p>
CometReadTimeout	1000 to 120000	<p>Specifies the time the server will wait for a client to complete sending the data</p>
ConnectionDelay	10 to 60000	<p>When the server has exceeded the connection count, how long to hold on to the connection before</p>

Configuration Element	Valid Parameters	Description
		disconnecting. Default is 60000.
FlexKeepAlive	5000 to 60000	The number of milliseconds the server will wait before sending a heartbeat for Adobe Flex clients. Default is 25000.
HandshakeTimeout	1000 to No Max	The number of milliseconds that the server will wait for the session to be established. Default is 1000, i.e. 1 second
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
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.
KeepAlive	5000 to No Max	The number of milliseconds the server will wait before sending a heartbeat. Default is 60000.
MaxBufferSize	1024 to No Max	The maximum buffer size that the server will accept. Default is 1048576.
MaxNoOfConnections	-1 to No Max	Sets the maximum concurrent connections to

Configuration Element	Valid Parameters	Description
MaxWriteCount	5 to 100	the server, -1 indicates no restriction, default is -1. 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 its 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.
NIOSelectArray	True or False	Specifies that the low level processing will use an array and not an iterator
NetworkMonitorThreads	2 to 100	The number of threads to allocate to flushing client data, Please note this will only take effect after a restart
OutputBlockSize	100 to No Max	The size of the application-level buffer used when streaming events. If the size is exceeded during streaming, the buffer is immediately emptied, and its contents transmitted over the network. Typically, each connection has its own buffer for outbound streaming. Default is 1400.
PriorityReadSpinLockMaxConnections	0 to 8	Maximum number of clients allowed to allocate high priority spin locks
PriorityReadSpinLockTime	1 to 10000	Maximum number of clients allowed to allocate high priority spin locks

Configuration Element	Valid Parameters	Description
PriorityReadType	0 to 2	If enabled then high priority sessions will be enabled to run spin locks waiting to read
QueueHighWaterMark	2 to No Max	The number of events in a client output queue before the server stops sending events. Default is 100.
QueueLowWaterMark	1 to No Max	The number of events in the clients queue before the server resumes sending events. Default is 50.
ReadCount	1 to 20	Number of times the thread will loop around waiting for an event to be delivered before returning
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
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
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

Configuration Element	Valid Parameters	Description
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
whMaxTimeBetweenFlush	1 to 1000	Total number of milliseconds to wait before a flush is done
whPeakTrailDelay	100 to 5000	When a peak is detected how long to stay in this state before returning to normal
Data Stream Config		
FanoutTaskQueueSize	32 to 1024	Sets the number of tasks that the FanOut Executor will have outstanding
FanoutTraversalType	0 to 2	The method to use when traversing connections.
MaxSessionIdSize	5 to 30	Maximum size of the session Id used to uniquely identify the clients
MonitorTimer	1000 to 120000	Time interval in milliseconds to scan the data group configuration looking for idle / completed streams
OffloadMulticastWrite	True or False	If true then all multicast writes will be performed by the parallel fanout engine.
ParallelFanoutThreshold	10 to 10000	Number of streams when the server will use parallel fanout

Configuration Element	Valid Parameters	Description
ParallelWorkers	1 to 64	Number of threads to allocate for the fanout executor,
SendInitialMapping	True or False	When any stream registered client connect sends the entire DataGroup Name to Id mapping
Environment Config		
AvailableProcessors	READ ONLY	Number of CPU's available
JavaVendor	READ ONLY	Vendor of Java Virtual Machine
JavaVersion	READ ONLY	Virtual Machine Version
NanosecondSupport	READ ONLY	Nano second 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
ServerVersion	READ ONLY	Universal Messaging Server Build Version
TimerAdjustment	READ ONLY	The size of the Operating Systems quatum
Event Storage		

Configuration Element	Valid Parameters	Description
ActiveDelay	100 to No Max	The time in milliseconds that an active channel will delay between scans. Default is 1000.
AutoDeleteScan	1000 to 500000	Specifies the number of milliseconds between scans on AutoDelete stores to see if they should be deleted
AutoMaintenanceThreshold	0 to 100	Sets the percentage free before an auto maintenance is performed, applies to internal stores only
CacheAge	1000 to No Max	The time in ms that cached events will be kept in memory for. Default is 86400000.
EnableStoreCaching	True or False	If true the server will try and cache events in memory after they have been written/read
EnableStoreReadBuffering	True or False	If true the server will buffer the reads from the store. This will increase replay performance greatly
IdleDelay	5000 to No Max	The time in milliseconds that an idle channel will delay between scans. Default is 60000.
JMSEngineAutoPurgeTime	5000 to 600000	Defines the interval between clean up of JMS Engine resources
PageSize	10 to 100000	The size in page size to use for the event store
QueueDeliveryPersistencePolicy	0 to 2	Sets the Queue Delivery Persistence Policy

Configuration Element	Valid Parameters	Description
StoreReadBufferSize	1024 to 2097152	Size of the buffer to use during reads from the store
SyncBatchSize	1 to 1000	Specifies the maximum size before the sync call is made
SyncServerFiles	True or False	If true the server will sync each file operation for its internal files
SyncTimeLimit	1 to 1000	Specifies the maximum time in milliseconds that will be allowed before the sync is called
ThreadPoolSize	1 to 4	The number of threads allocated to perform the management task on the channels. Default is 1.
Fanout Values		
ConnectionGrouping	True or False	If true allows the server to group connections with the same selector providing improved performance
JMSQueueMaxMultiplier	1 to 10	The multiplier used on the High Water mark when processing events from a JMS Engine Queue/Topic
MaximumDelayInWrite	1 to 5000	The number of milliseconds an event will wait in a queue before it will be processed
ParallelBatchSize	50 to 10000	Specifies the number of connections to process in one batch per parallel thread

Configuration Element	Valid Parameters	Description
ParallelThreadPoolSize	2 to 64	Specifies the number of threads to use within the thread pool. Restart Required
ParallelThreshold	1 to 10000	Specifies the number of connections to a channel before the server will use the parallel fanout engine
ParallelUseGlobalPool	True or False	If true all channels use a common pool else all channel have there own pool. Restart required
PeakPublishDelay	0 to No Max	When clients start to hit high water mark how long to delay the publisher to allow the client time to catch up
PublishDelay	0 to No Max	How long to delay the publisher when subscribers queue start to fill, in milliseconds. Default is 10.
PublishExpiredEvents	True or False	Publish expired events at server startup. Default is true.
RoundRobinDelivery	True or False	Use a round robin approach to event delivery. Default is false.
SendEndOfChannelAlways	True or False	Always send a End Of Channel, even if we find no matches within the topic
SendPubEventsImmediately	True or False	Send publish events immediately
SyncQueueDelay	10 to 3600000	Maximum number of milliseconds the queue publisher will be delayed

Configuration Element	Valid Parameters	Description
SyncQueuePublisher	True or False	If true then the queue publisher will be sync with the queue consumers
Global Values		
AllowRealmAdminFullAccess	True or False	Any subject with Admin ACL applied has full access to all objects within the realm.
CacheJoinInfoKeys	True or False	If enabled we cache join key information between events passed over joins. This reduces the number of objects created
DisableExplicitGC	True or False	If enabled the server will call the Garbage Collector at regular intervals to keep memory usage down
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 connection.
ExtendedMessageSelector	True or False	If true, allows the server to use the extended message selector syntax. Default is false.
HTTPCookieSize	14 to 100	The size in bytes to be used by nhp(s) cookies
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

Configuration Element	Valid Parameters	Description
NHPTimeout	2000 to No Max	The number of milliseconds the server will wait for client authentication. Default is 120000, i.e. 2 minutes.
NanoDelayBase	10000 to 1000000	This number represents the number of nano seconds in a millisecond. Typically this is 1,000,000 however, it can be used to increase or decrease the internal delays used by Universal Messaging.
OverrideEveryoneUser	True or False	Override the *@* permission for channels / queues with explicit acl entry permissions.
SendRealmSummaryStats	True or False	If true sends the realms status summary updates. Default is false.
ServerStateFlush	50 to 1000	Specifies the time in milliseconds between scans to save the servers state files
ServerTime	True or False	Allow the server to send the current time to the clients. Default is true.
StampDictionary	True or False	Place Universal Messaging details into the dictionary, default is false.
StampHost	True or False	Stamps the header with the publishing host (true/false)
StampTime	True or False	Stamps the header with the current time (true/false)
StampTimeUseHPT	True or False	If this is set to true then the server will use an accurate

Configuration Element	Valid Parameters	Description
		ms clock, if available, to stamp the dictionary
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)
StatusBroadcast	2000 to No Max	The number of ms between status events being published. Default is 5000, i.e. every 5 seconds.
StatusUpdateTime	2000 to No Max	The number of ms between status events being written to disk. Status events provide a history of the realm's state, the default for this is Long.MAX_VALUE, i.e. never written to disk.
SupportOlderClients	True or False	Allow the server to support older clients. Default is true.
Inter-Realm Comms Config		
EstablishmentTime	10000 to 120000	Time for an inter realm link to be initially established.
KeepAliveInterval	1000 to 120000	Time interval where if nothing is sent a Keep Alive event is sent
KeepAliveResetTime	10000 to 180000	If nothing has been received for this time the connection is deemed closed
MaximumReconnectTime	1000 to 50000	The maximum number of milliseconds to wait

Configuration Element	Valid Parameters	Description
		before trying to reestablish a connection.
MinimumReconnectTime	100 to 10000	The minimum time to wait before establishing a connection.
Timeout	60000 to 180000	If no events received within this time limit the link is assumed dead and will be closed.
WriteDelayOnFail	True or False	If true the comms will wait for the link to re establish.
WriteDelayTimeout	1000 to 60000	The maximum time to wait in a write of the link has dropped.
JVM Management		
AutoThreadDumpOnExit	True or False	Defines if a thread dump is produced when the server exits.
EmergencyThreshold	80 to 99	The memory threshold when the server starts to aggressively scan for objects to release. Default is 94, i.e. 94%
EnableJMX	True or False	Enable JMX beans within the server
ExitOnDiskIOError	True or False	If true, the server will exit if it gets a I/O Exception. Default is true
ExitOnInterfaceFailure	True or False	If true and for any reason an interface cannot be started when the realm initialises, the realm will shutdown.

Configuration Element	Valid Parameters	Description
ExitOnMemoryError	True or False	If true, the server will exit if it gets an out of memory exception. Default is true.
IORetryCount	2 to 100	Number of times an file I/O operation will be attemptd before aborting
IOSleepTime	100 to 60000	Time between disk I/O operations if it fails
JMXRMIServerURLString	String	JNDI Lookup URL for the JMX Server to use.
MemoryMonitoring	60 to 30000	Number of milliseconds between monitoing memory usage on the realm. Default is 2000.
ThrottleAllPublishersAtThreshold	True or False	Defines if publishers will be throttled back when memory emergency threshold is reached
WarningThreshold	70 to 95	The memory threshold when the server starts to scan for objects to release. 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. Default is 1.
MaxEventsPerSchedule	1 to No Max	Number of events that will be sent to the remote server in one run. Default is 50.

Configuration Element	Valid Parameters	Description
MaxQueueSizeToUse	1 to No Max	The maximum events that will be queued on behalf of the remote server. Default is 100.
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
Logging Config		
DefaultLogSize	No Min to No Max	The default size of the log in bytes
EmbedTag	True or False	Whether to include the type tag in the log message
EnableLog4J	True or False	If enabled will intercept log messages and pass to Log4J as well
LogManager	0 to 3	The Log manager to use
RolledLogFileDepth	No Min to No Max	The number of log files to keep on disk when using log rolling. Oldest entries will be deleted when new files are created.
customAuditTag	String	The tag to mark Audit log entries with
customFailureTag	String	The tag to mark Failure log entries with
customFatalTag	String	The tag to mark Fatal log entries with
customInformativeTag	String	The tag to mark Informative log entries with

Configuration Element	Valid Parameters	Description
customLogTag	String	The tag to mark Log log entries with
customSecurityTag	String	The tag to mark Security log entries with
customSuccessTag	String	The tag to mark successful log entries with
customWarningTag	String	The tag to mark Warning log entries with
fLoggerLevel	0 to 7	The server logging level, between 0 and 7 with 0, indicating very verbose, and 7 indicating very quiet. Default is 1.
MQTTConfig		
Timeout	1000 to 60000	The number of milliseconds over the timeout value before the server will close the connection
Plugin Config		
EnableAccessLog	True or False	Defines if plugin access log produced
EnableErrorLog	True or False	Defines if plugin error log produced
EnablePluginLog	True or False	Defines if plugin status log produced
MaxNumberOfPluginThreads	10 to 10000	Maximum number of threads to allocate to the plugin manager
PluginTimeout	1000 to 30000	Time in milliseconds that the plugin will read from a client

Configuration Element	Valid Parameters	Description
Protobuf Config		
CacheEventFilter	True or False	Hold the Protocol Buffer filter cache in memory. Default true
FilterProtobufEvents	True or False	Allows the server to filter on Protocol Buffers. Default is true
MaximumProtobufBuilders	No Min to No Max	The maximum amount of builders per descriptor file. Default 4
MinimumProtobufBuilders	No Min to No Max	The minimum amount of builders per descriptor file. Default 2
ProtobufDescriptorsInputDir	String	The folder to search for Protocol Buffer descriptor files to parse incoming messages.
ProtobufDescriptorsOutputDir	String	The folder for the server to put the combined Protocol Buffer descriptor file for serving out to clients.
UpdateDescriptorsInterval	1000 to No Max	The time in milliseconds between checking the Protocol Buffer directory for updates. Default is 60000
RecoveryDaemon		
EventsPerBlock	1 to No Max	The number of events to send in one block
ThreadPool	1 to No Max	Number of threads to use for client recovery
Thread Pool Config		

Configuration Element	Valid Parameters	Description
CommonPoolThreadSize	5 to 1000	Maximum number of thread to allocate to the common thread pool
ConnectionThreadPoolMaxSize	10 to No Max	The maximum number of threads allocated to establish client connections
ConnectionThreadPoolMinSize	4 to 100	The minimum number of threads allocated to establish client connections
ConnectionThreadWaitTime	10000 to 300000	The time for the thread to wait for the client to finalise the connection
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.
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 is expected to connect. If NIO is available then this number would be best to keep 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.

Configuration Element	Valid Parameters	Description
SchedulerPoolSize	1 to 100	The number of threads assigned to the scheduler, default is 2.
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. By specifying this number to be large enough to accommodate enough idle threads so that if any thread pool requires to expand then they can be reused.
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 is expected to connect. If NIO is available then this number would be best to keep 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.
TransactionManager		
MaxEventsPerTransaction	0 to No Max	The maximum number of events per transaction, a 0 indicates no limit

Configuration Element	Valid Parameters	Description
MaxTransactionTime	1000 to No Max	Time in milliseconds that a transaction will be kept active
TTLThreshold	1000 to 60000	The minimum time in milliseconds, below which the server will not store the Transaction ID

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

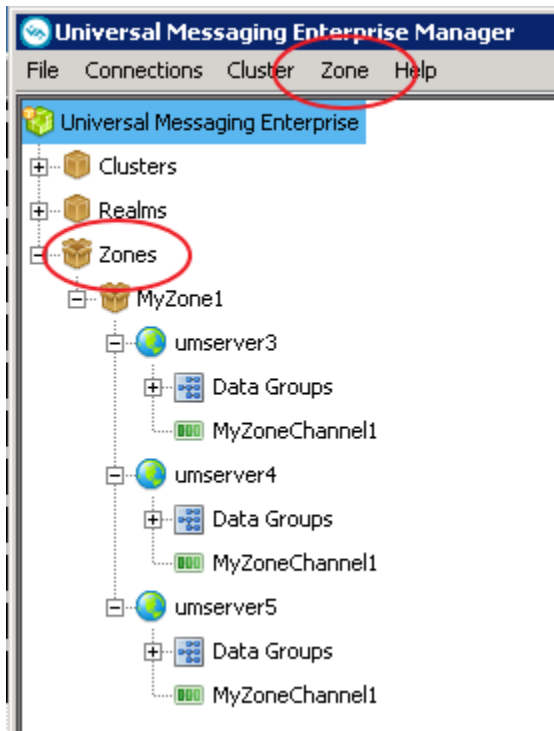
Zone Administration

Overview of Zone Administration

The Enterprise Manager provides menu items for performing the administrative functions on zones.

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/queue in a zone

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

To create a channel or queue 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** or **Create Queue** as required.
2. In the **Add Channel** or **Add Queue** dialog, specify the attributes of the channel or queue that you wish to create.
3. Click **OK** to complete the dialog and create the channel or queue.

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

Modifying a channel/queue in a zone

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

Note: Any changes you make to the channel/queue 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 or queue on one realm/cluster in a zone, proceed as follows:

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

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

This section describes the process of creating 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.

Creating a cluster of Universal Messaging realms ensures that applications either publishing / subscribing to channels, or pushing / popping events from queues can connect to any of the realms and view the same state. If one of the realms in the cluster is unavailable client applications can automatically reconnect to any of the other cluster realms and carry on from where they were.

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

- ["Creating Realm Clusters" on page 57](#)
- ["Deleting Realm Clusters" on page 61](#)

- ["Modifying Cluster Members" on page 62](#)
- ["Creating Cluster Channels" on page 64](#)
- ["Creating Cluster Queues" on page 67](#)
- ["Viewing and Monitoring Cluster Information" on page 70](#)
- ["Manage Inter-Cluster Connections" on page 72](#)

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 75](#)

Creating a Cluster

This section describes the process of creating a Universal Messaging Cluster . Clusters allow a group of Universal Messaging Realm Servers to replicate resources between them, and to maintain the state for those objects across all realms within the cluster.

Creating a cluster of Universal Messaging realms ensures that applications which publish/subscribe to channels, or which push/pop events from queues, can connect to *any* of the realms and view the same state. If one of the realms in the cluster is unavailable, client applications can automatically reconnect to any of the other cluster realms and carry on from where they were.

Universal Messaging's clustering technology provides an unsurpassed level of reliability. Client applications can seamlessly switch between any cluster realm if any problems - such as network or hardware failures - occur with the realm to which they are connected. This also provides an exceptional ability to load balance clients between realm servers.

Viewing Clusters in Enterprise Manager

The Enterprise Manager's top level view shows a tree node labelled "Universal Messaging Enterprise" (see ["Enterprise Summary" on page 259](#)). 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 15](#))).

Preparing to Create a Cluster

Firstly, before a cluster can be created, the Enterprise Manager need to connect to those realms (see ["Connecting to Multiple Realms" on page 15](#)) 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 125](#)) 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
```

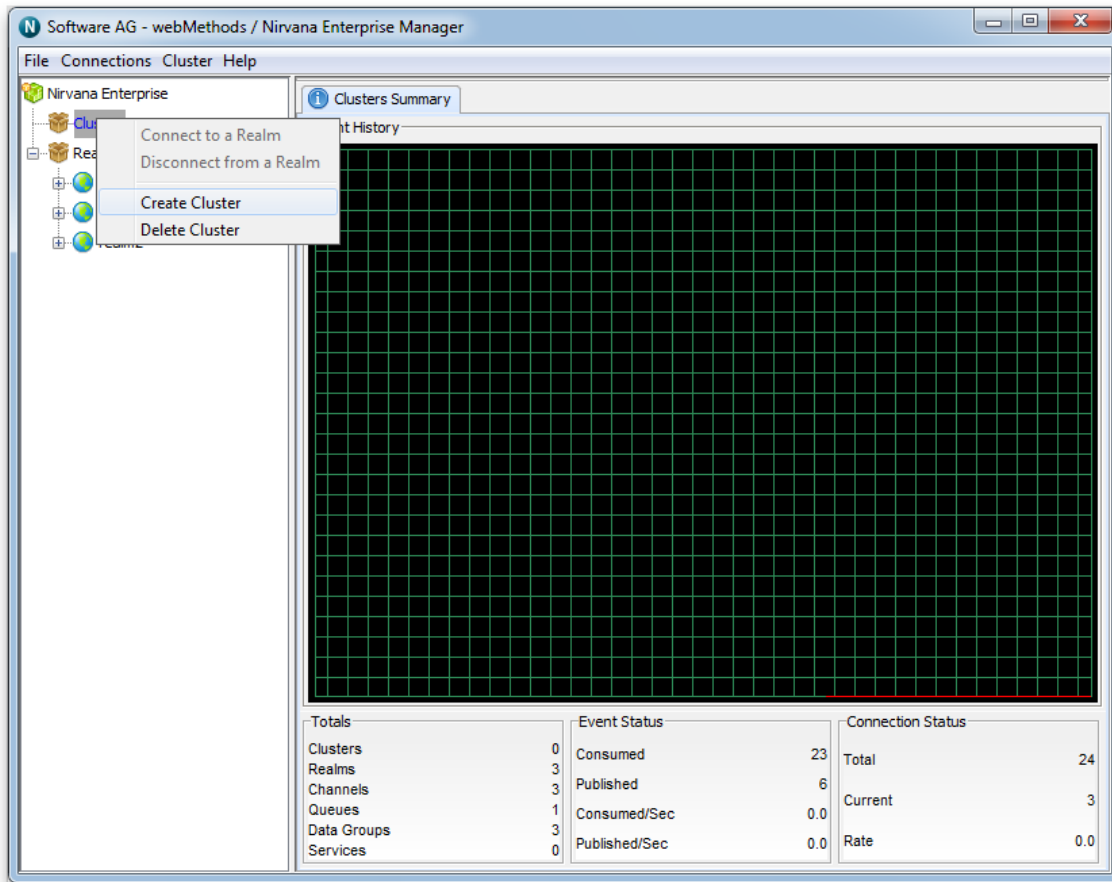
For example, in the following example, there are 3 realms that will form part of a cluster, each realm subject needs to exist in the other realms ACLs, for example, the following realm subjects need to be added to the acl for each realm in our example:

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

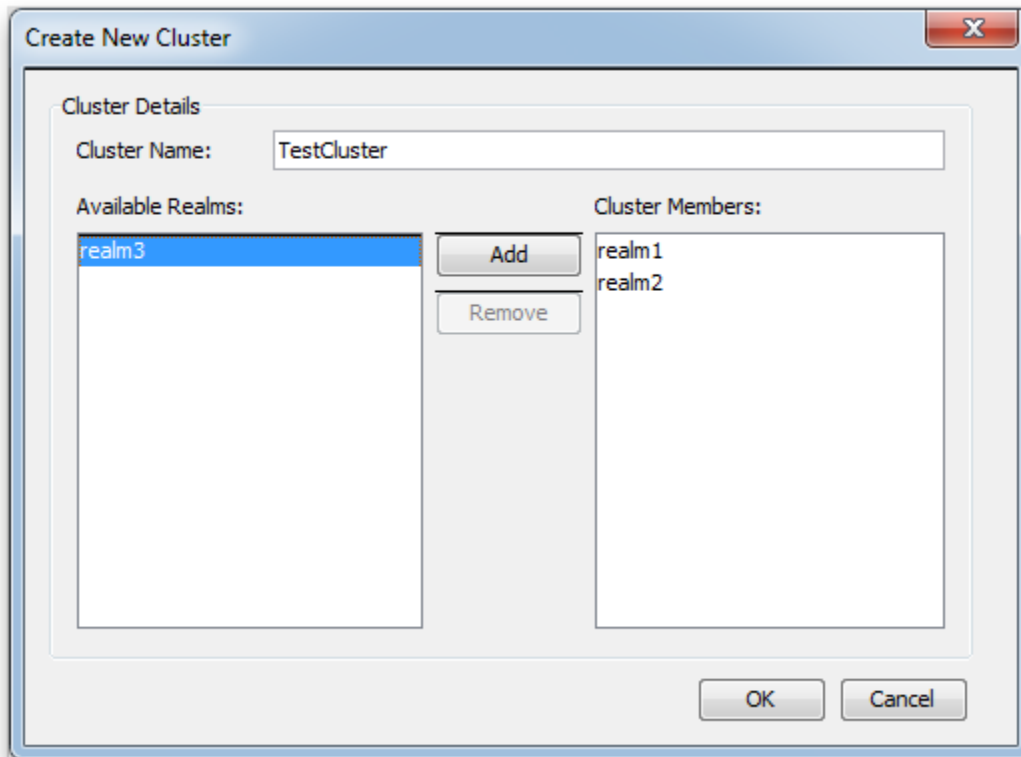
The permissions given for each realm need to be 'Access Realm'. As well as this, each realm must have a valid entry for the user@host that corresponds to the user that will create the cluster using the Enterprise Manager. The permissions for this user must be sufficient in order to create the cluster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

Creating a cluster

If you select the 'Clusters' node under the 'Universal Messaging Enterprise' node, you will be shown a pop up menu with a number of options. One of the options is to create a cluster. The image below shows this menu option as described.



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, 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 either 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 in the Cluster dialog will create the cluster. A new cluster node will appear under the 'Clusters' node and the realms that have been selected as members will be rendered 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.

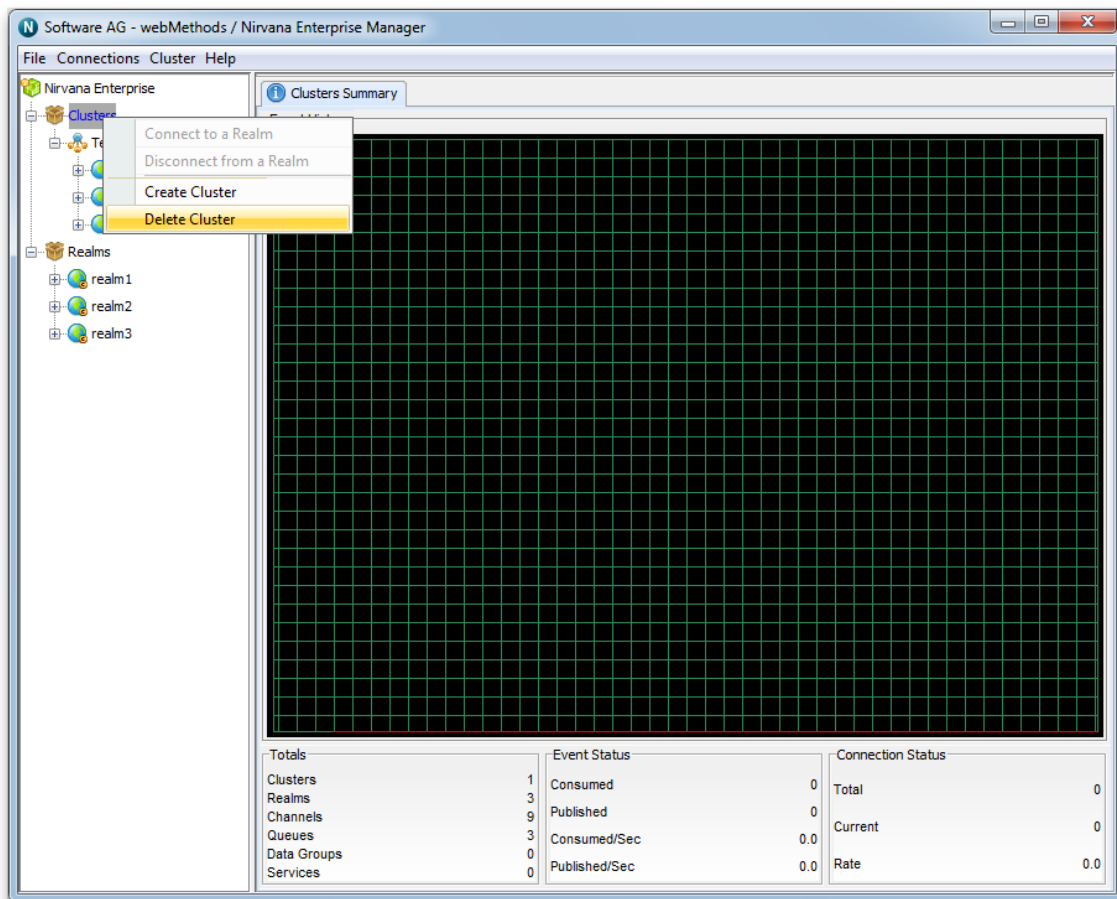
Name	State	Master	realm1	realm3	realm2
realm1	Master	realm1	Local	OnLine	OnLine
realm2	Slave	realm1	OnLine	OnLine	Local
realm3	Slave	realm1	OnLine	Local	OnLine

Creating cluster channels (see "[Cluster Channel Administration](#)" on page 64) and cluster queues (see "[Cluster Queue Administration](#)" on page 67) is not permitted if any of the cluster realms are offline.

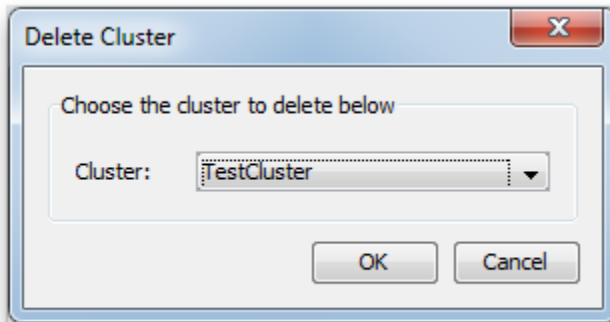
Deleting Clusters

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.



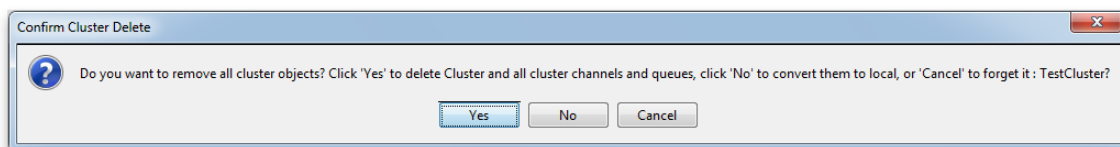
Clicking on the 'OK' button once you have selected the cluster you wish to delete will prompt you to answer a question. This question gives you 2 choices with regard to the cluster resources that may exist within the cluster. These are:

- 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.

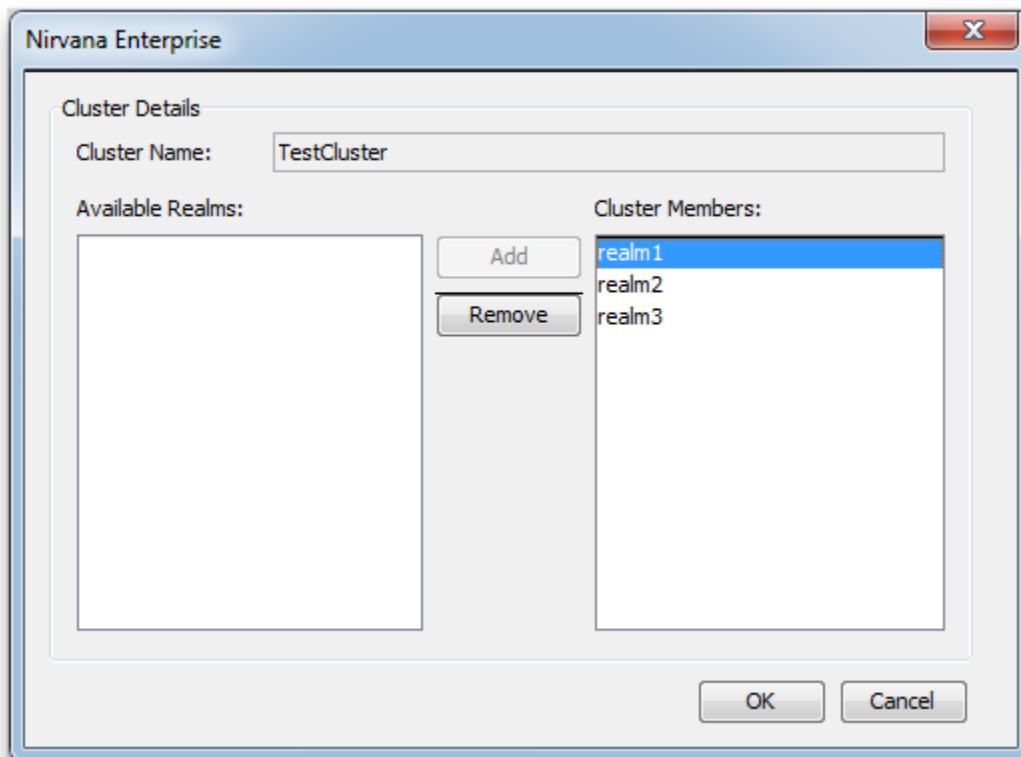
To add a new realm to a cluster, you must first of all ensure that you have connected to (see "[Connecting to Multiple Realms](#)" on page 15) the realm you wish to add. Removing realms is accomplished by selecting the realm you wish to remove from the cluster. This will be discussed in more detail further 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 57](#)). Once you have a cluster, then also ensure you have connected to (see ["Connecting to Multiple Realms" on page 15](#)) 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.

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 consists of a physical object within each Universal Messaging realm within the cluster as well as its logical reference within each realm's namespace.

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.

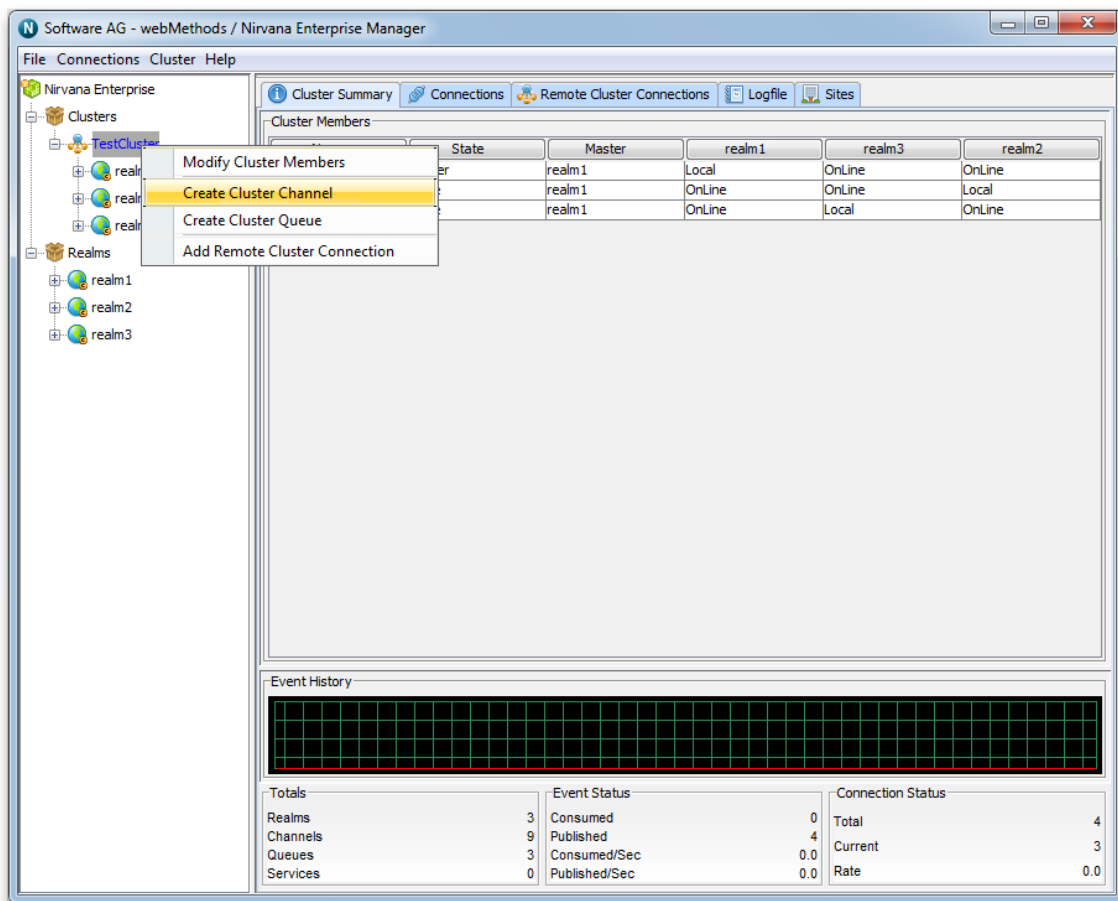
When you connect to a Universal Messaging realm in the Enterprise Manager, all resources and services found within each realm's namespace are displayed in a tree

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 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 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 on all clusters simultaneously.

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

The screenshot shows a dialog box titled "Add channel to cluster TestCluster". The dialog is divided into two main sections: "Channel Attributes" and "Channel Keys".

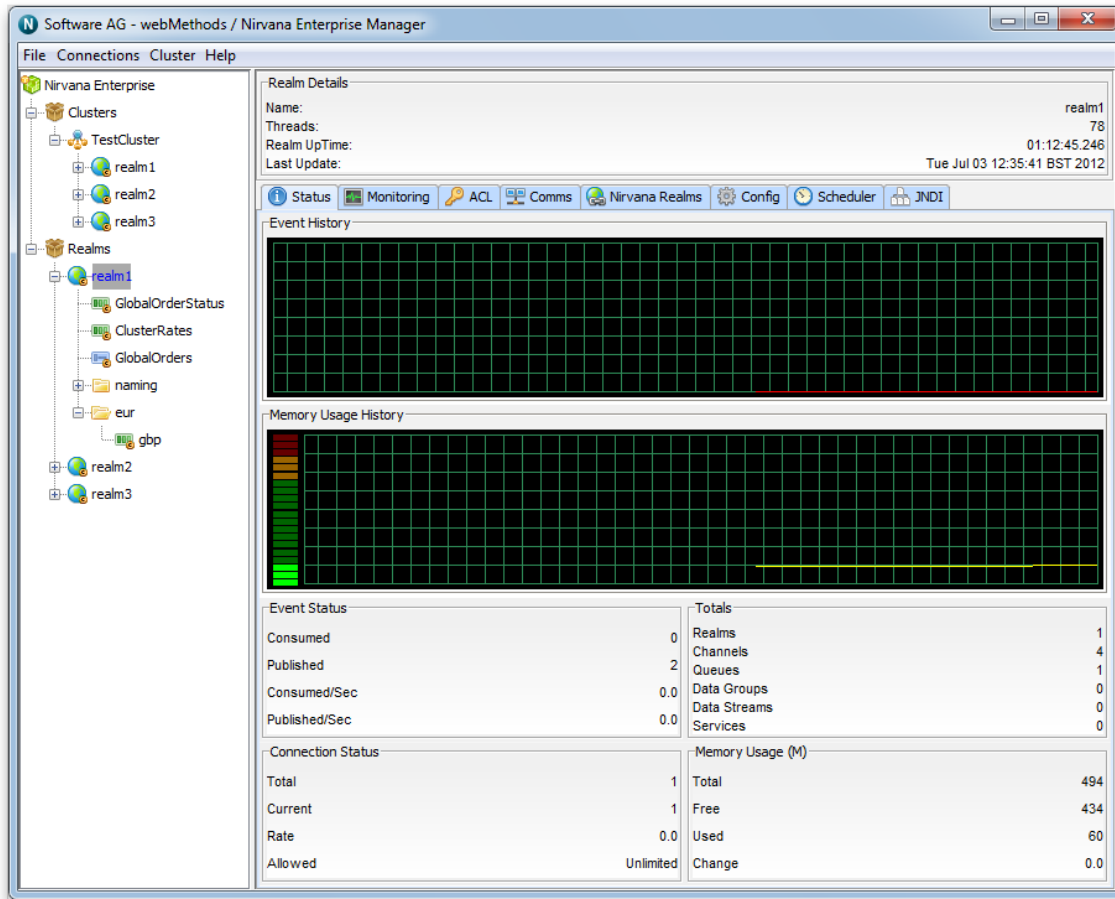
Channel Attributes:

- Channel Name: /eur/gbp
- Channel Type: Transient
- Channel TTL: (empty)
- Channel Capacity: (empty)
- Parent Realm: realm1
- Dead Event Store: (empty)
- Use JMS Engine:
- Use Merge Engine:
- Storage Properties: Edit...

Channel Keys:

- Select Key To Edit: (empty dropdown)
- New: (button)
- Key Properties:**
- Key Name: (empty)
- Depth: 1 (spinner)
- Save: (button)
- Delete: (button)
- OK: (button)
- Cancel: (button)

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, fully expanded.



As you can see from the image above, each realm node now contains the 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 to 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.

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.

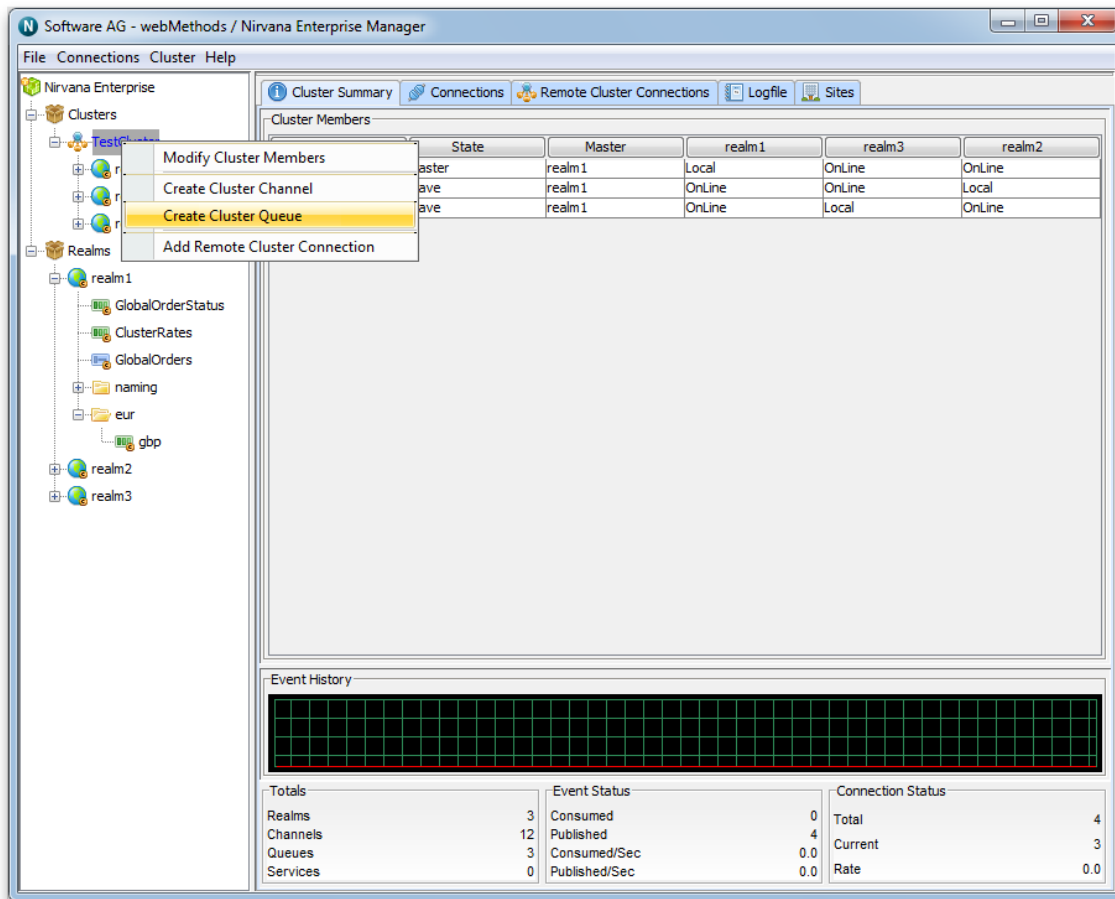
When you connect to a Universal Messaging realm in the Enterprise Manager, all resources and services found within each realm's namespace are displayed in a tree

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 57](#)) 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 i have a cluster called 'TestCluster', which contains 3 realms called 'realm1', 'realm2' and 'realm3' and i want to create a queue called /eur/orders within that cluster of realms, i would need to first of all click on the cluster node called 'TestCluster'. Then, by right-clicking on cluster node a pop-up menu will be displayed that shows a number of menu items (as shown in the image below).

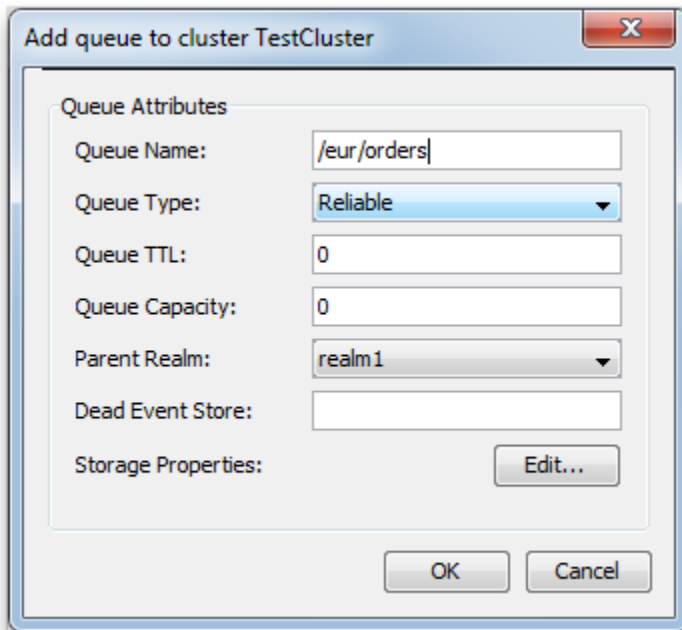


By clicking on the menu item 'Create Cluster Queue', you will be prompted with a dialog box that allows you to enter the details of the cluster queue you wish to create.

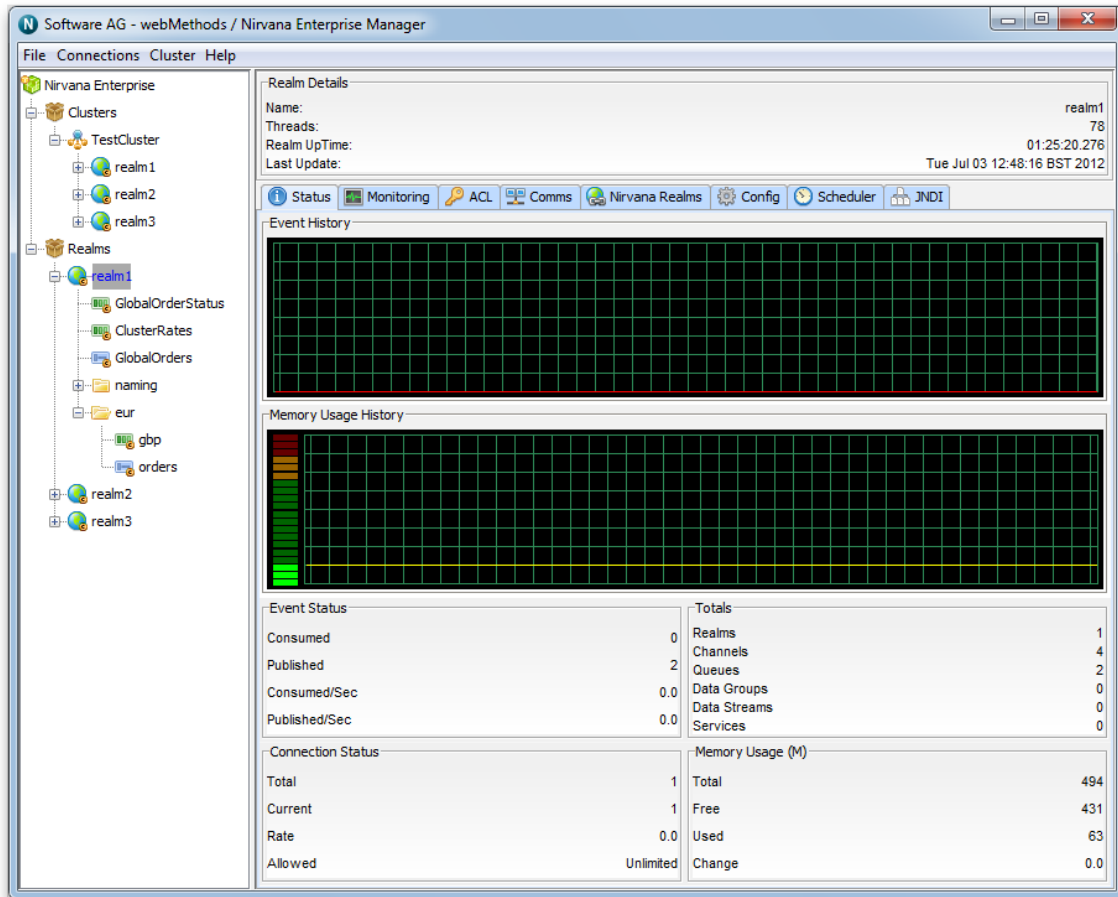
Cluster queues have exactly the same set of attributes assigned to them as normal queues when they are created. The create queue dialog for cluster queues allows you to input values for each of these attributes. The only difference will be that the queue

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 under each realm in the cluster as well as each realm underneath the realms container node. The image below shows how the namespace tree looks after the cluster queue has been created, fully expanded.



As you can see from the image above, each realm node now contains the 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 to that of normal queue and is denoted by a 'c' in the icon, whereas the normal queue icon does not have a 'c'. queue.

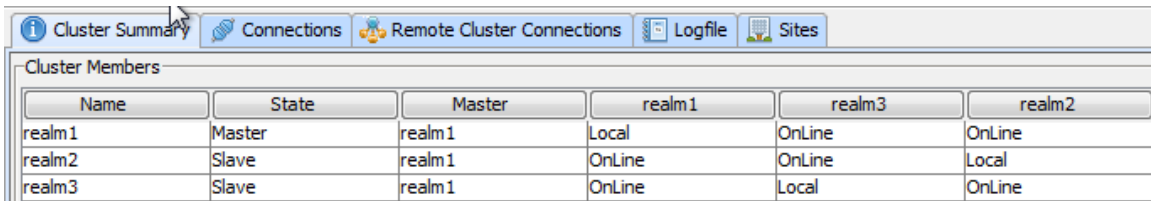
Viewing Cluster Information

The Enterprise Manager provides Cluster information through the following four tabs:

- Cluster Summary
- 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.

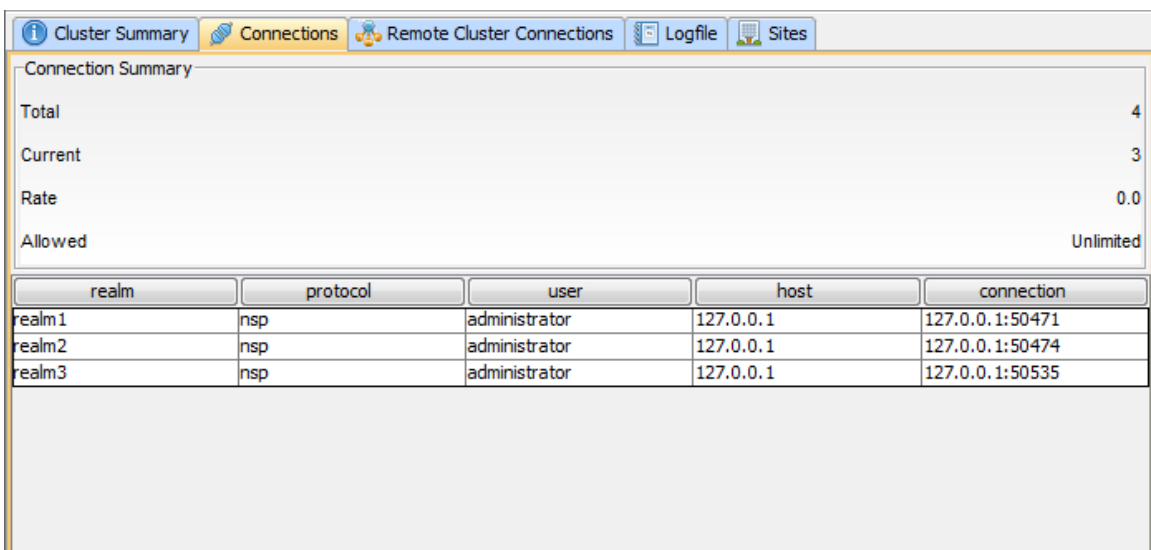


Name	State	Master	realm1	realm3	realm2
realm1	Master	realm1	Local	OnLine	OnLine
realm2	Slave	realm1	OnLine	OnLine	Local
realm3	Slave	realm1	OnLine	Local	OnLine

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:

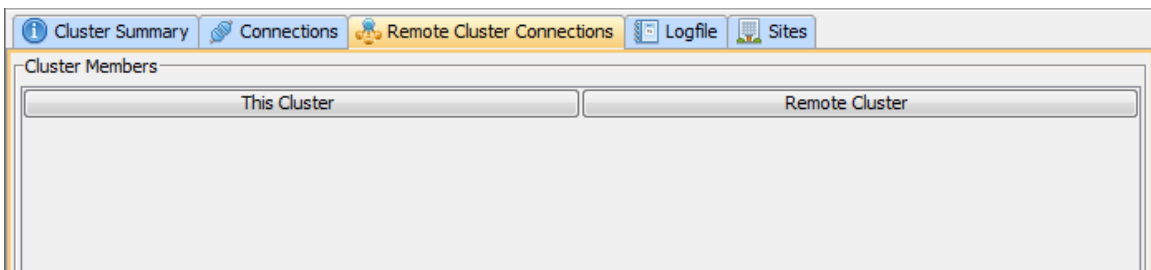


realm	protocol	user	host	connection
realm1	nsp	administrator	127.0.0.1	127.0.0.1:50471
realm2	nsp	administrator	127.0.0.1	127.0.0.1:50474
realm3	nsp	administrator	127.0.0.1	127.0.0.1:50535

The Cluster Connections Tab.

Remote Cluster Connections

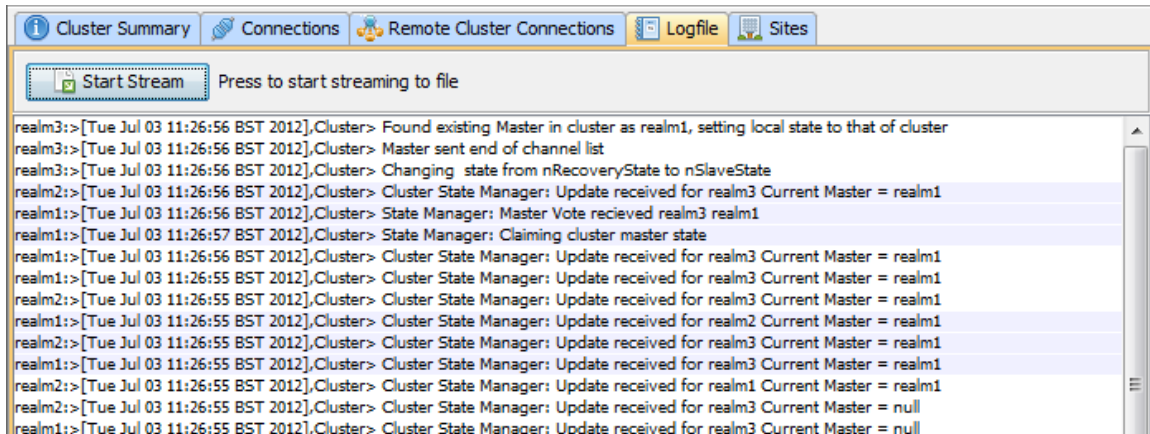
The Remote Cluster Connections tab shows all remote cluster connections for this Cluster. Clusters can be remotely connected together providing the ability to create joins between channels in different clusters:



The Remote Cluster Connections Tab.

Logfile

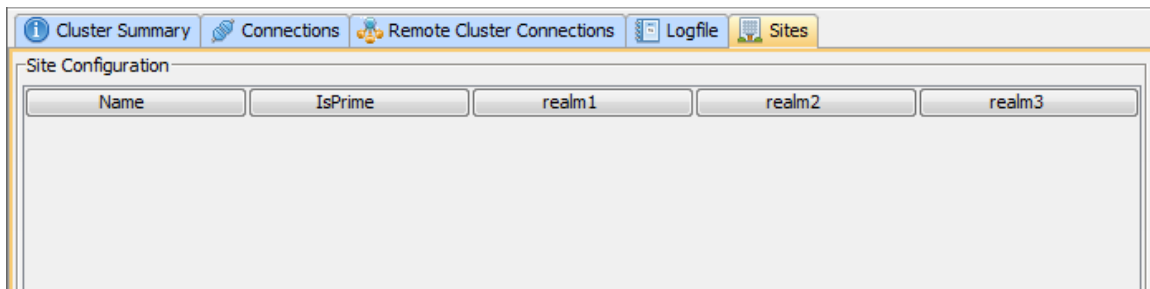
The Logfile tab shows a real-time Cluster-specific log, and provides the option to stream the log output to a file:



The Cluster Logfile Tab.

Sites

The Sites tab shows any site configurations (see "[Creating and Managing Clusters with Sites](#)" on page 75) 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*):

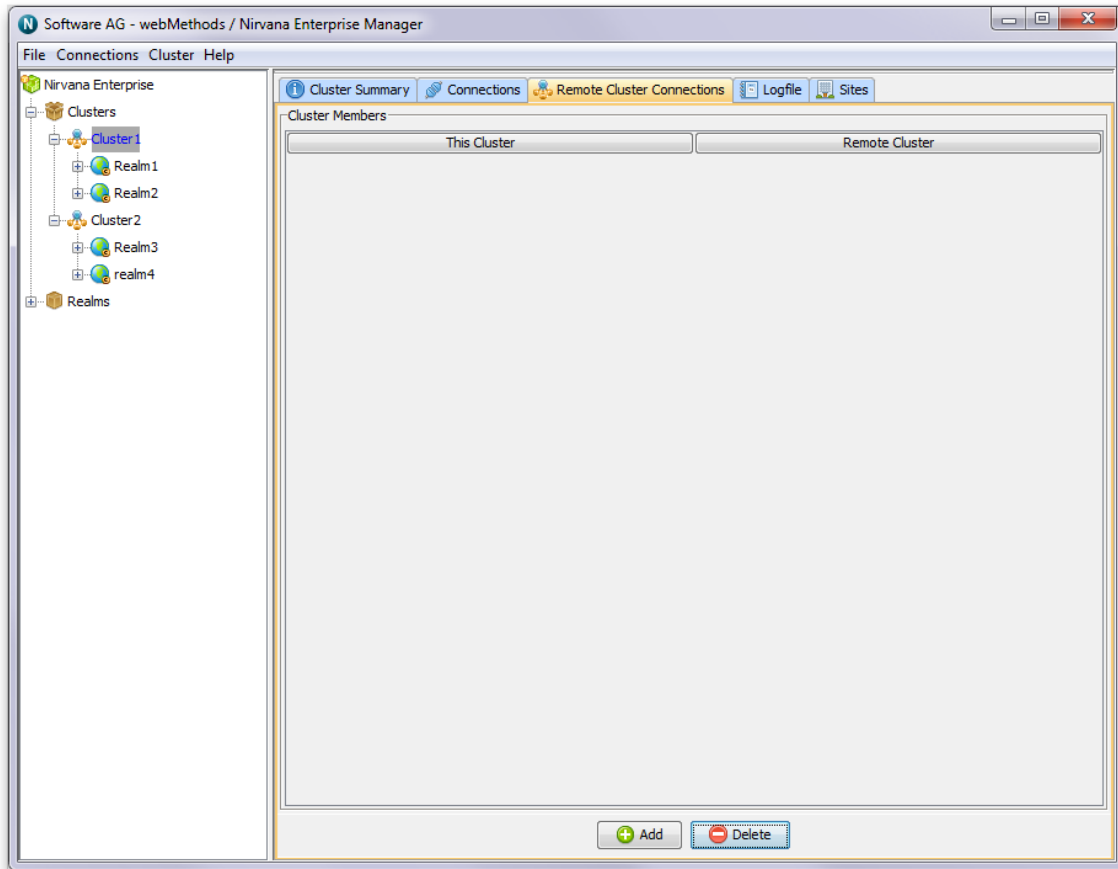


The Cluster Sites Tab.

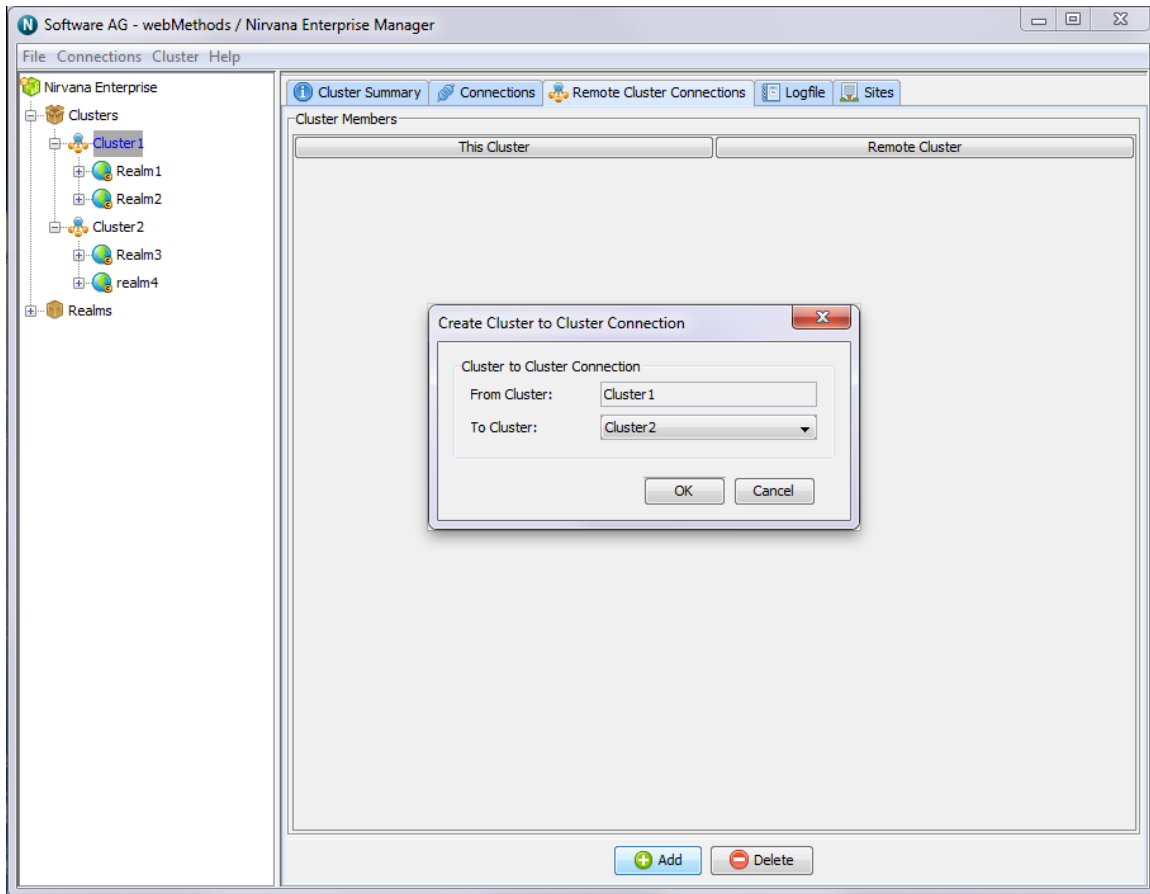
Manage Inter-Cluster Connections

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.



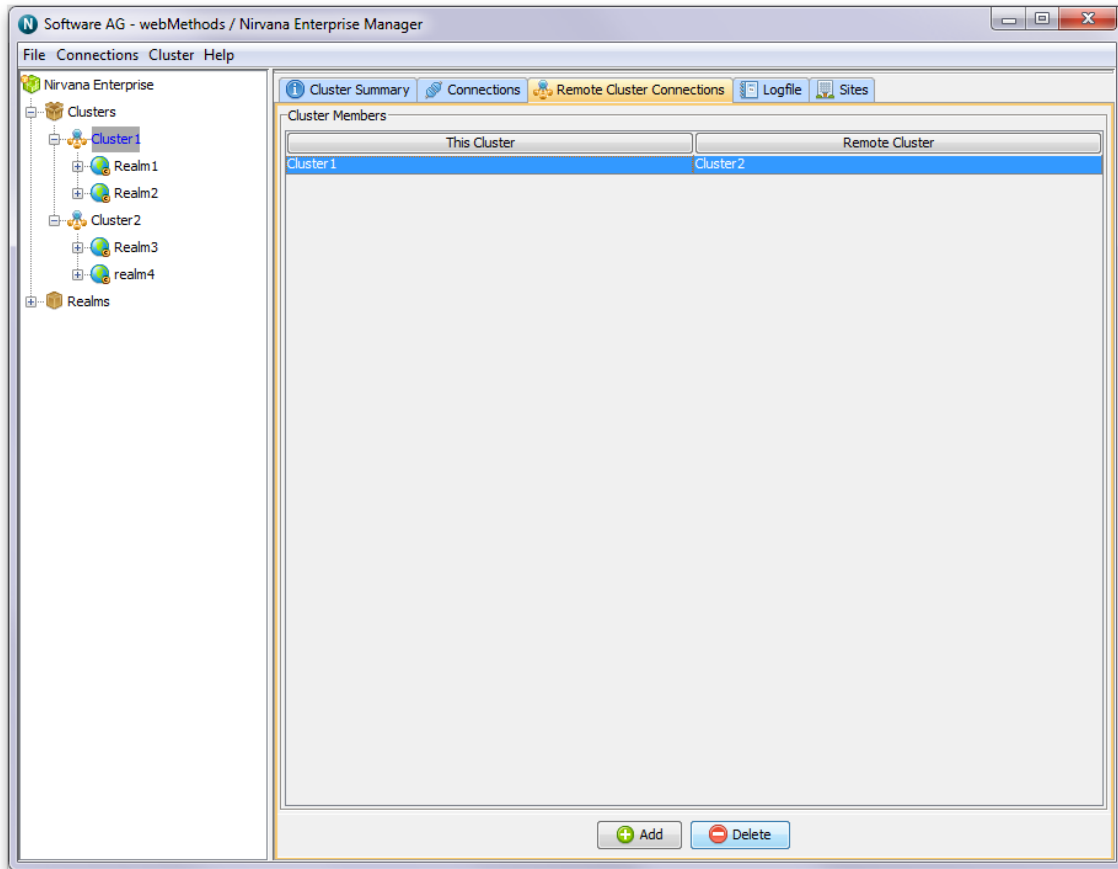
Next, select "Add" and choose the remote cluster from the dropdown list in the popup dialog which will now appear:



The inter-cluster connections should now be established, and inter-cluster joins can now be formed through the Enterprise Manager (see "[Channel Join](#)" on page 327) or programmatically.

Deleting Inter-Cluster Connections

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



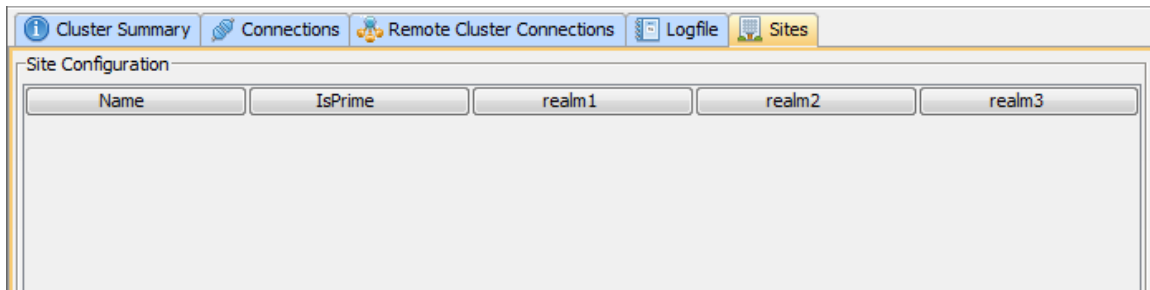
Creating and Managing Clusters with Sites

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, 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" (see ["Enterprise Summary" on page 259](#)). One level below this is a tree node labelled "*Clusters*", which contains any known clusters.

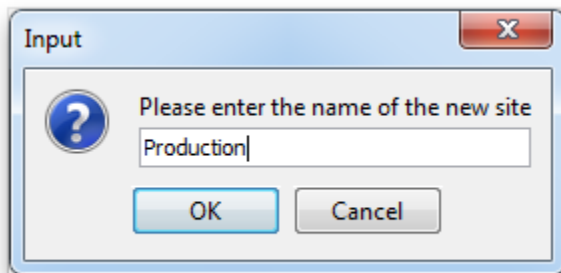
After creating your cluster (see ["Creating a Cluster" on page 57](#)) and selecting the cluster's icon in the Enterprise Manager, click the *Sites* tab:



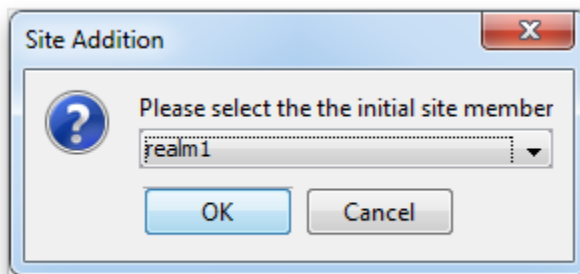
Sites Tab in Cluster View. No Sites have yet been created for the above cluster.

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:

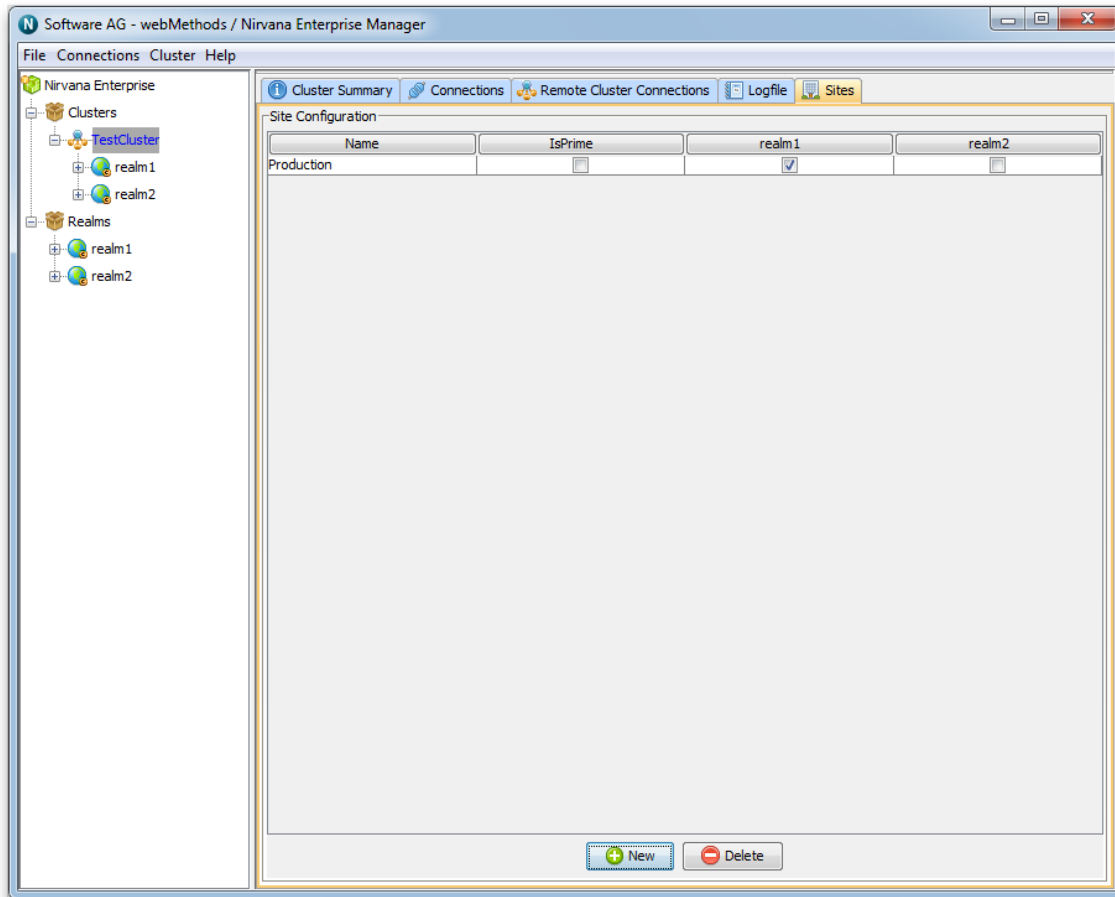


Creation of a "Production" Site.



Addition of an Initial Member to "Production" Site.

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

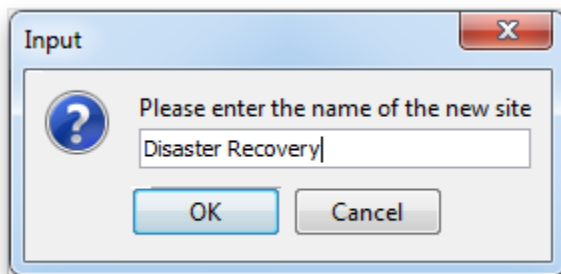


The "Production" Site with its initial member realm is shown.

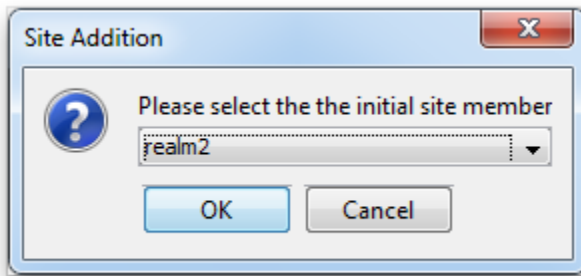
Note that the table will contain a column for all realms in the cluster. In this example we have only added one realm to the Production Site. Checking and unchecking the appropriate checkboxes will add or remove clustered realms from the corresponding Sites.

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**

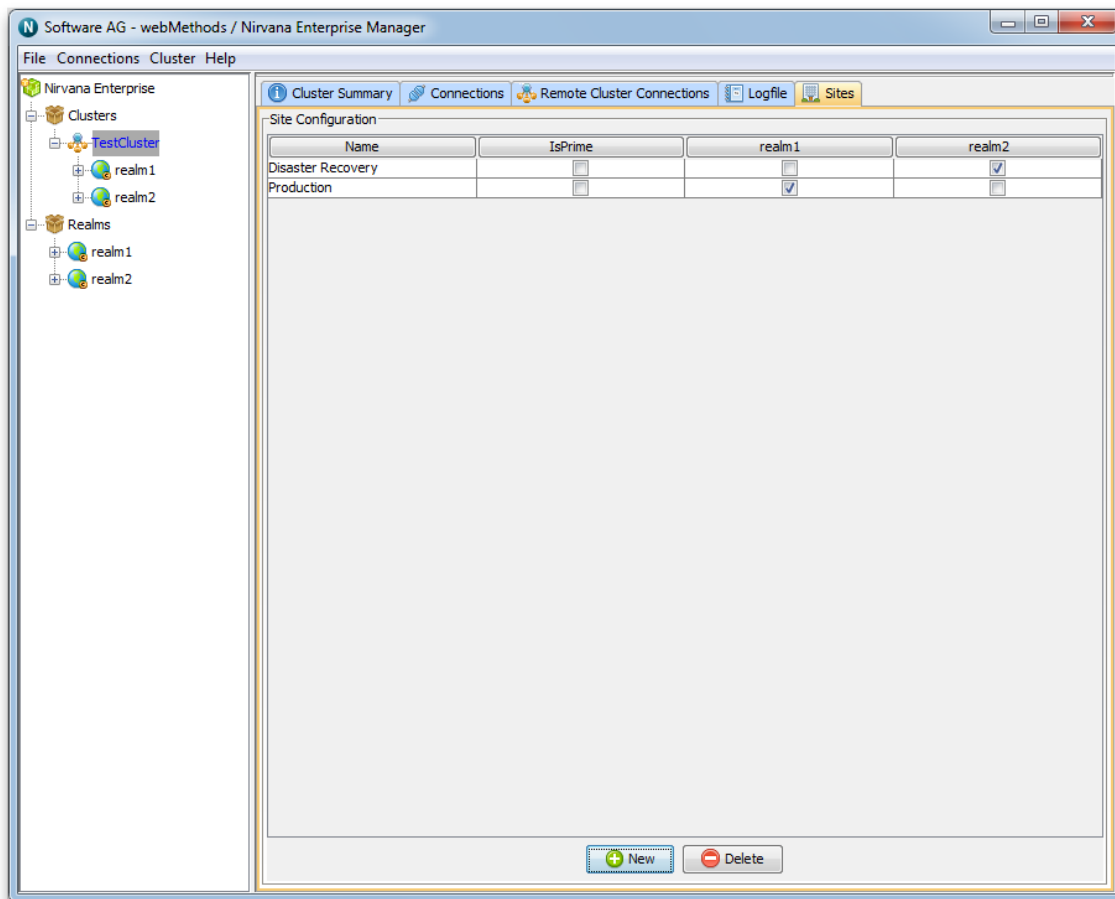


Creation of a "Disaster Recovery" Site and addition of an initial site member realm.



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

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

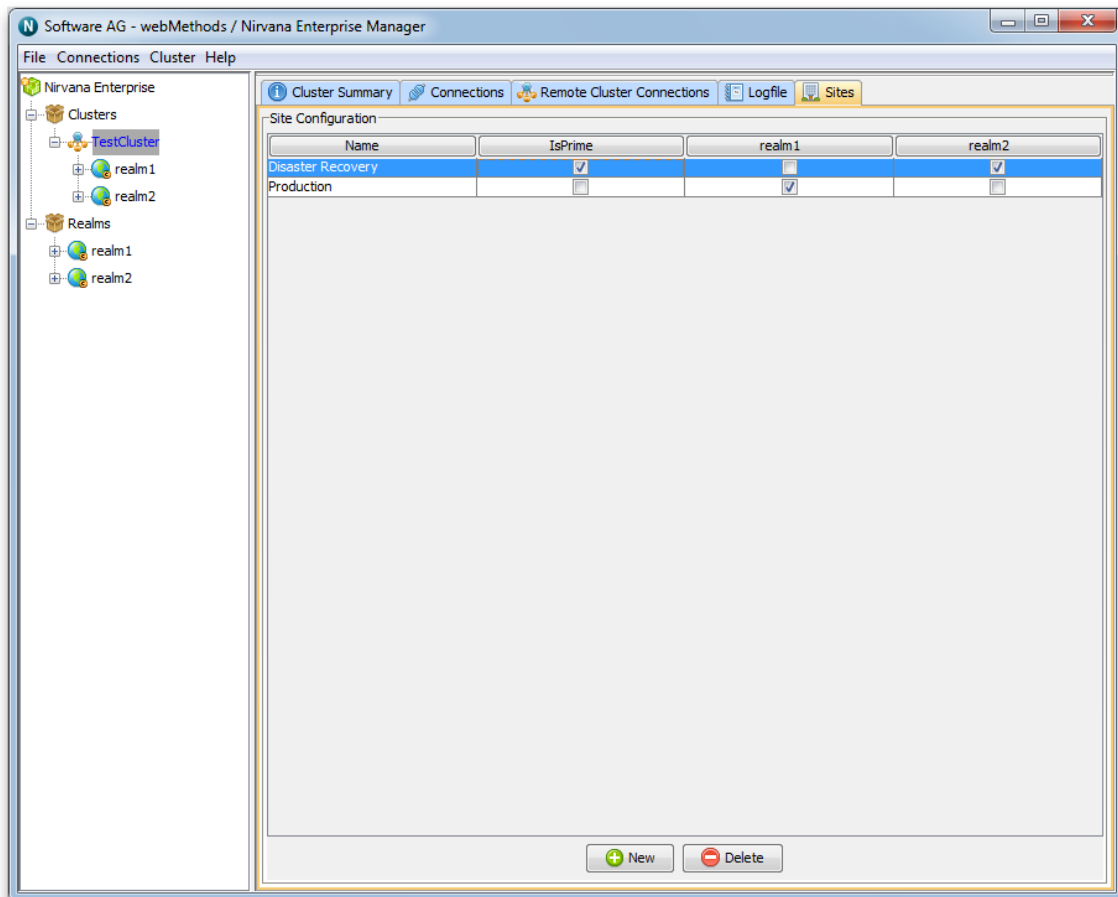


Sites Tab in Cluster View. The "Production" and "Disaster Recovery" Sites are both shown.

Setting a Site's IsPrime Flag

Administrators use the Sites *isPrime* flag to determine which site will contain a cluster's Master realm. Typically, the Disaster Recovery Site will be set to be Prime, allowing the Production site to fully fail and allowing the Disaster Recovery Site to vote a new master. Should the Disaster Recovery Site fail, the administrators would set the

Production Site to be Prime, which permits the election of a new Master without the usual 51% quorum requirement:



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

Channel Administration

The links below describe the Channel management features available within Universal Messaging's Enterprise Manager

- ["Creating Channels" on page 80](#)
- ["Editing Channels" on page 85](#)
- ["Copying Channels" on page 88](#)
- ["Channel Joins" on page 327](#)
- ["Channel Snoop" on page 93](#)
- ["Publishing Events Onto Channels" on page 95](#)
- ["Named Object Administration" on page 98](#)

Channel Creation

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.

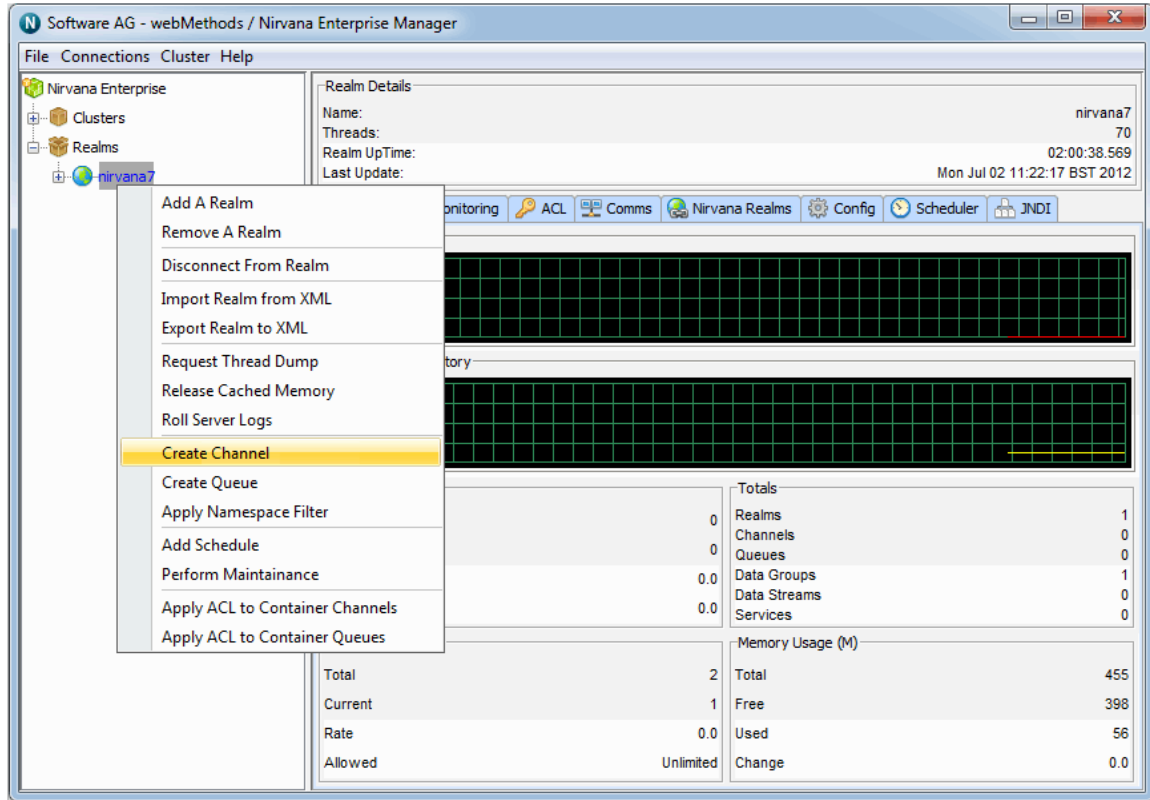
Each channel that is created consists of a physical channel within the Universal Messaging realm as well as its logical reference within a namespace that may be made up of resources that exist across multiple Universal Messaging realm servers. Creating channels using the Enterprise Manager creates the 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 'nirvana' simply right-click on the realm node to display a pop-up menu which contains a 'Create Channel' option.



After selecting 'Create Channel', an Add channel dialog box appears. Channels have a set of attributes assigned to them when they are created. The create channel dialog allows you to input values for each of these attributes.

In order to create the channel '/eur/rates' please fill in the name and choose the channel type as well as any other attributes.

Universal Messaging channels can be of the following types:

- persistent
- mixed
- reliable
- simple
- transient
- off-heap
- paged

For information about these types, see the summary of channel attributes in the Commonly Used Features section of the Universal Messaging Developer's Guide.

In the example below, a channel type of Simple and a TTL of 7 seconds have been given to the channel.

Add channel to realm nirvana7

Channel Attributes

Channel Name:

Channel Type:

Channel TTL:

Channel Capacity:

Parent Realm:

Dead Event Store:

Use JMS Engine: Use Merge Engine:

Storage Properties:

Channel Keys

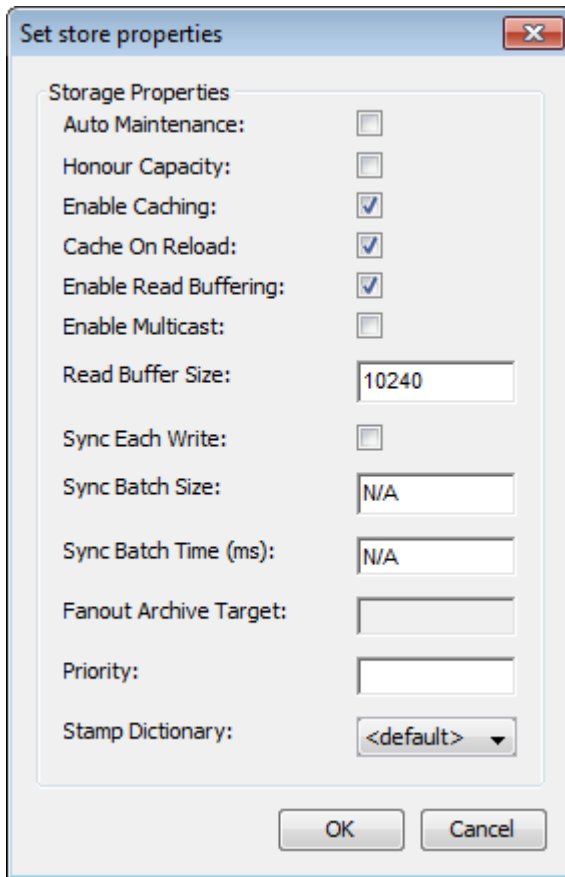
Select Key To Edit:

Key Properties

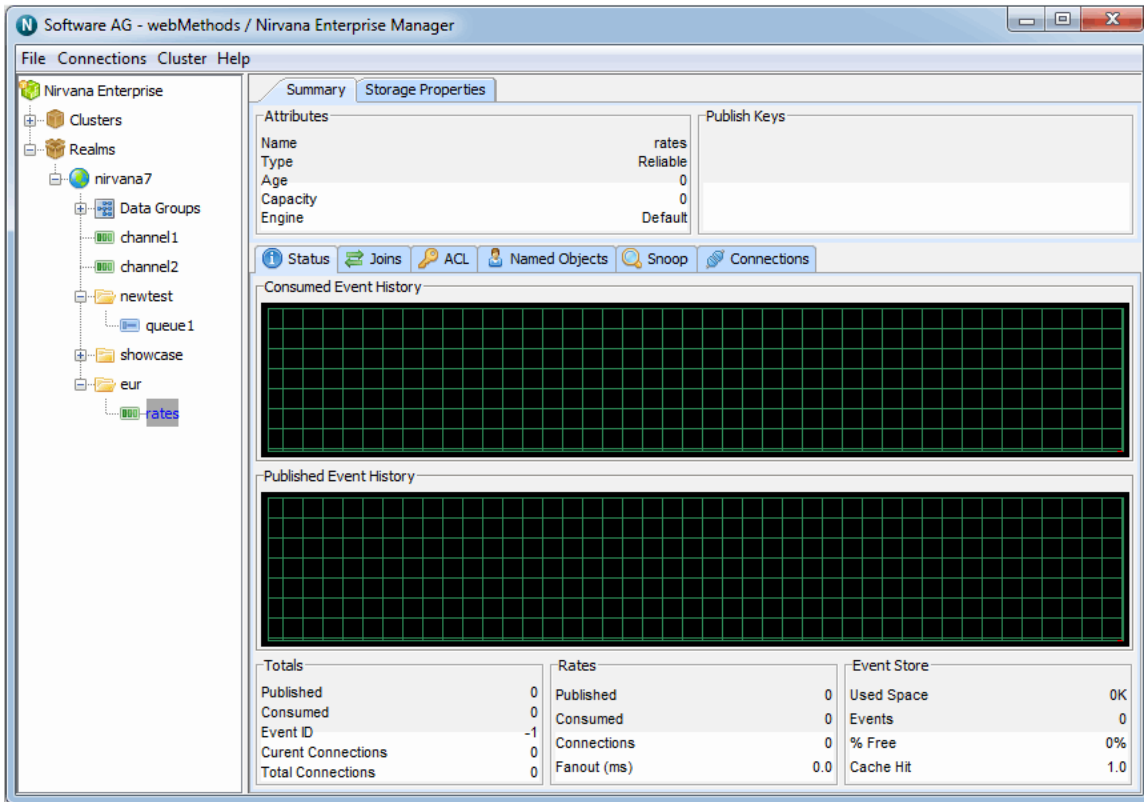
Key Name:

Depth:

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

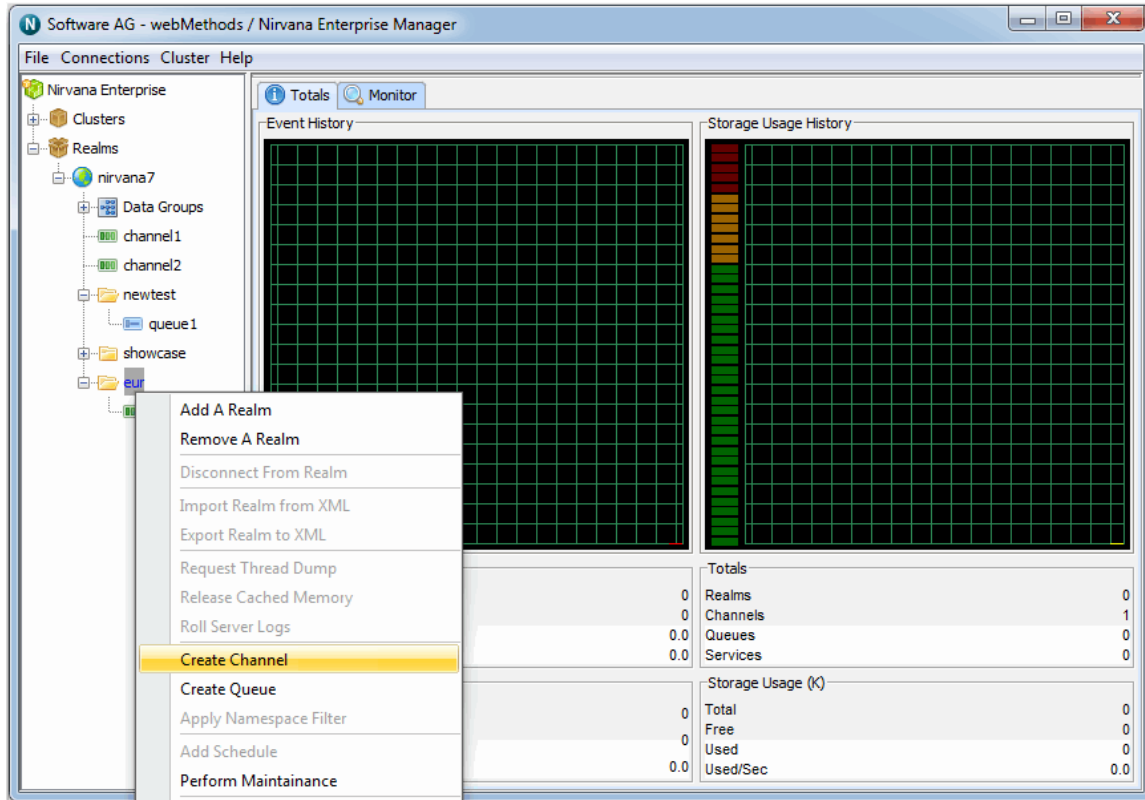


There are also a number of Storage Properties associated with the channel which can be configured by clicking the "Edit..." button to the right of "Storage Properties".



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 'nirvana'.

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.

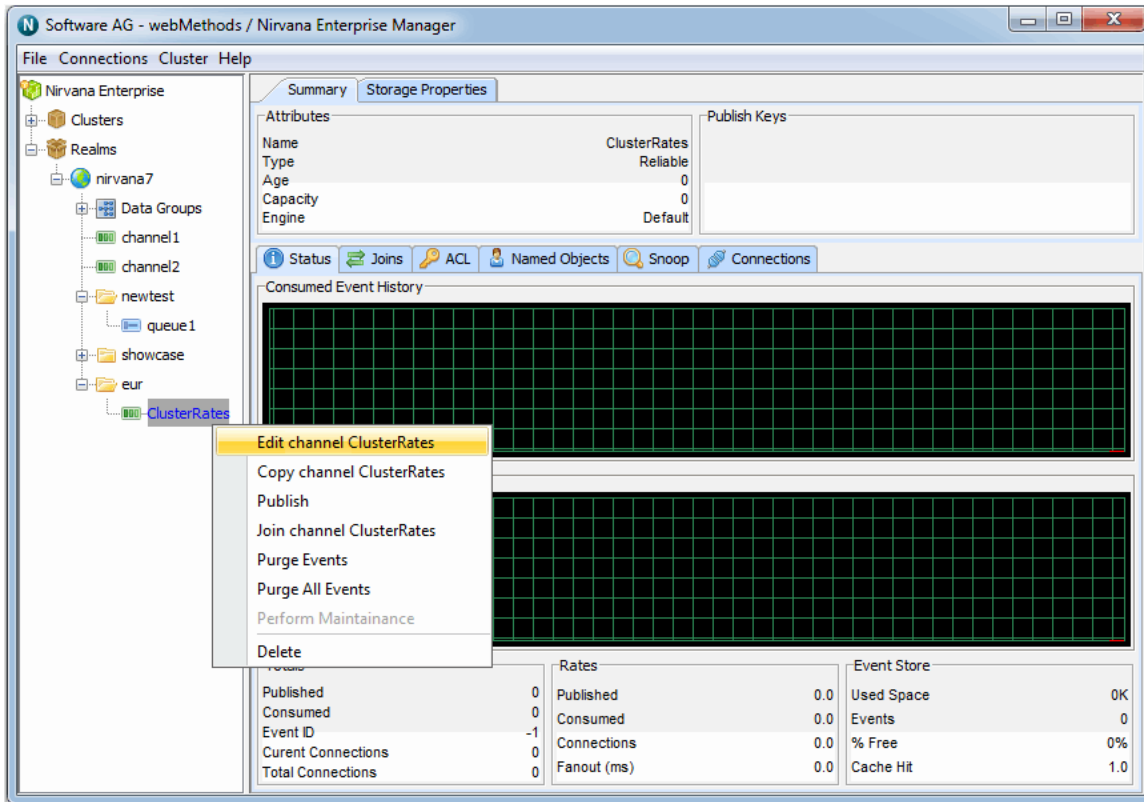
Channel Editing

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 for a channel, such as name, event time to live (TTL), capacity, channel keys or even the realm on which the channel exists.

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 removed and then the new channel is created. The original events are then copied from the temporary channel onto the new channel.

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 exists. The image below shows the edit channel dialog.

Modify channel ClusterRates

Channel Attributes

Channel Name: ClusterRates

Channel Type: Mixed

Channel TTL: 0

Channel Capacity: 0

Parent Realm: nirvana7

Dead Event Store:

Use JMS Engine: Use Merge Engine:

Storage Properties: Edit...

Channel Keys

Select Key To Edit: CCY:1 New

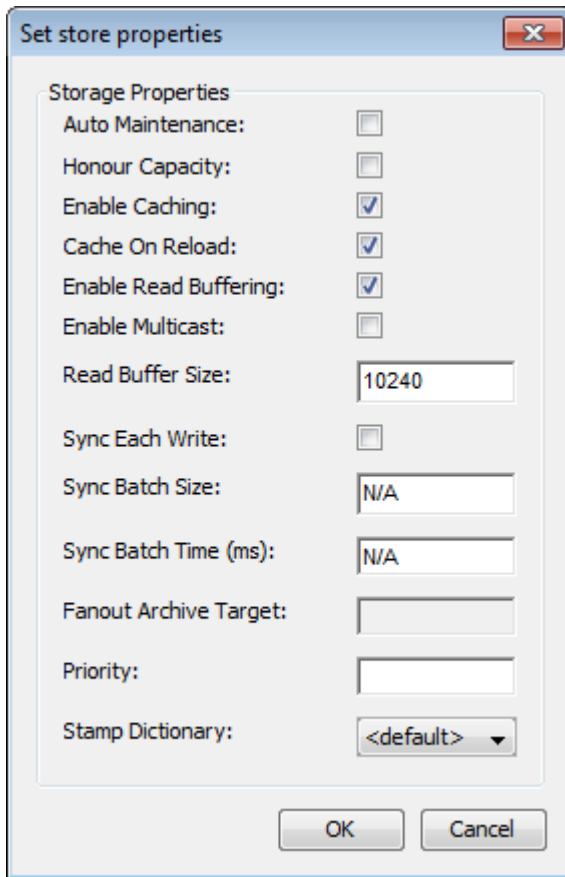
Key Properties

Key Name:

Depth: 1 Save Delete

OK Cancel

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 channel to any of the available realms. Clicking on the 'OK' button will perform the edit operation on the channel.



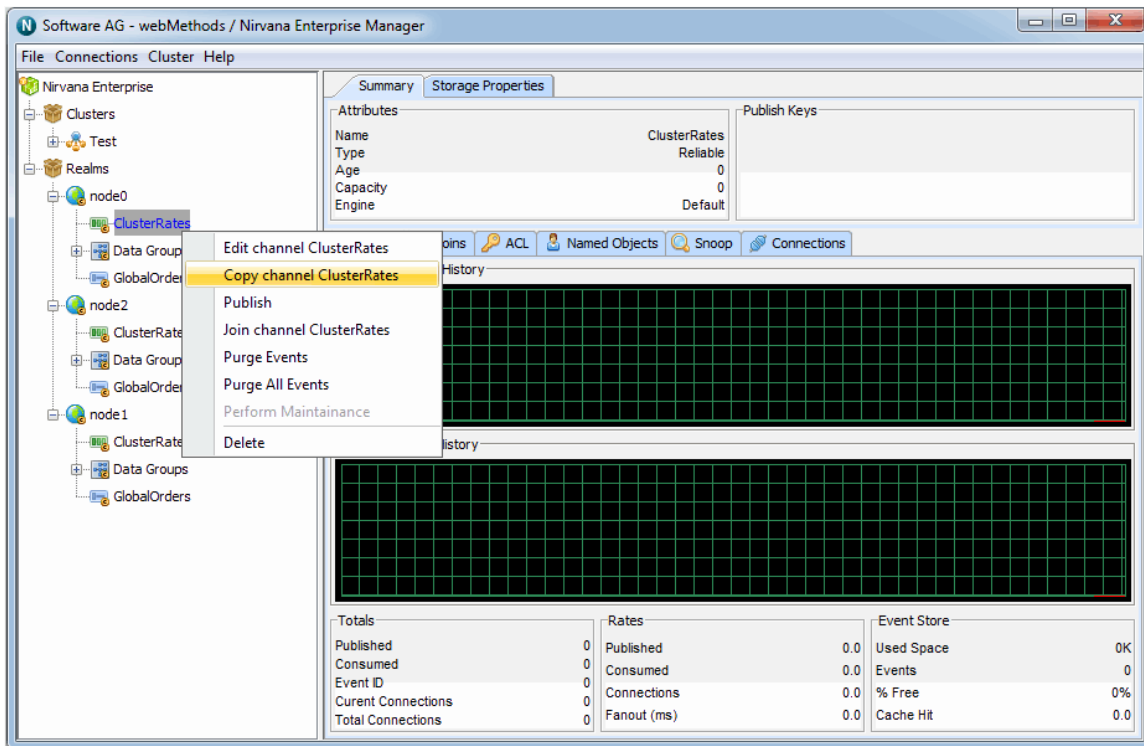
There are also a number of Storage Properties associated with the channel which can be configured by clicking the "Edit..." button to the right of "Storage Properties".

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 to. The image below shows the copy channel dialog.

The image shows a dialog box titled "Copy channel ClusterRates". It contains the following fields and controls:

- Channel Attributes:**
 - Channel Name: ClusterRates
 - Channel Type: Mixed
 - Channel TTL: 0
 - Channel Capacity: 0
 - Parent Realm: node0
 - Dead Event Store: (empty)
 - Use JMS Engine:
 - Use Merge Engine:
 - Storage Properties: Edit...
- Channel Keys:**
 - Select Key To Edit: CCY:1
 - New
- Key Properties:**
 - Key Name: (empty)
 - Depth: 1
 - Save
 - Delete

Buttons: OK, Cancel

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 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 are the logical rendezvous point for data that is published and subscribed .

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

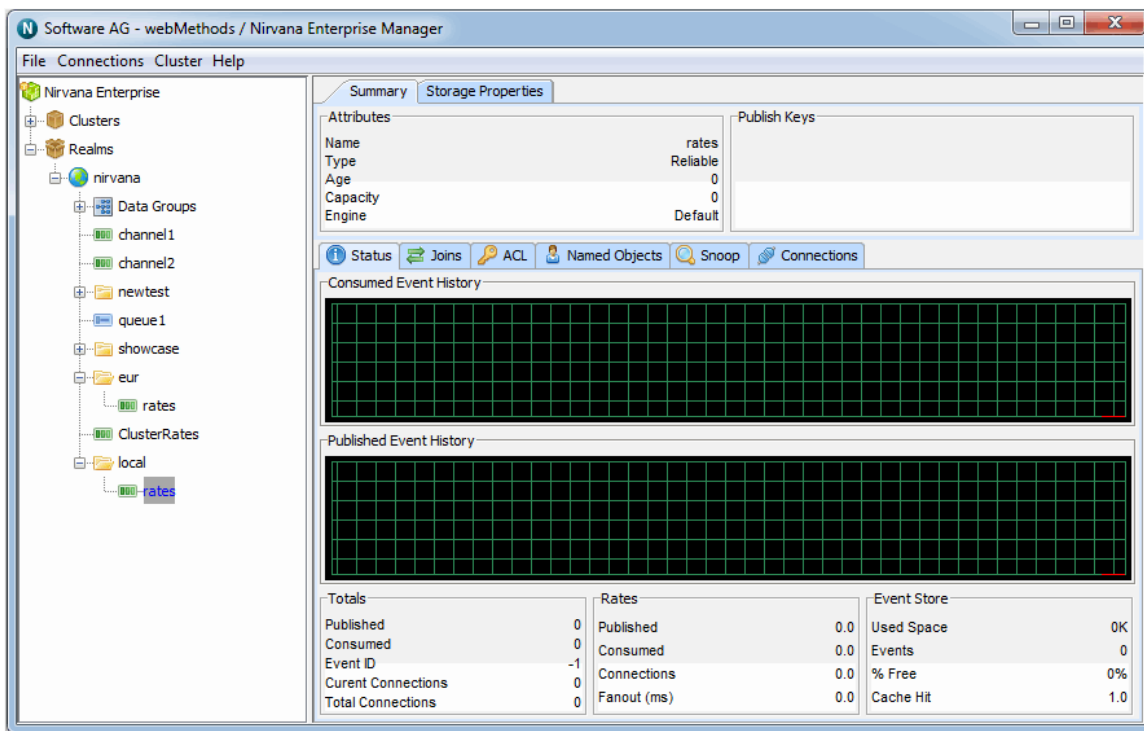
Joining channels using the Enterprise Manager creates a physical link between 2 channels, a source channel and a destination channel. Once created, any events published to the source channel will be republished onto the destination channel.

Joins can be created using filters , so that only specific events published to the source channel that match a criteria will be routed to the destination channel.

Joins can also be created in several configurations, firstly joins may be created between channels on a realm and a stores on another realm federated with the source realm. Alternatively joins can be created from channels on a clustered realm to stores within the same cluster. Non-clusterwide channels can be joined to cluster-wide stores, (but not vice versa). Additionally, channels can be joined from channels on one cluster to channels on another cluster by using inter-cluster joins (see ["Inter-Cluster Joins" on page 93](#)).

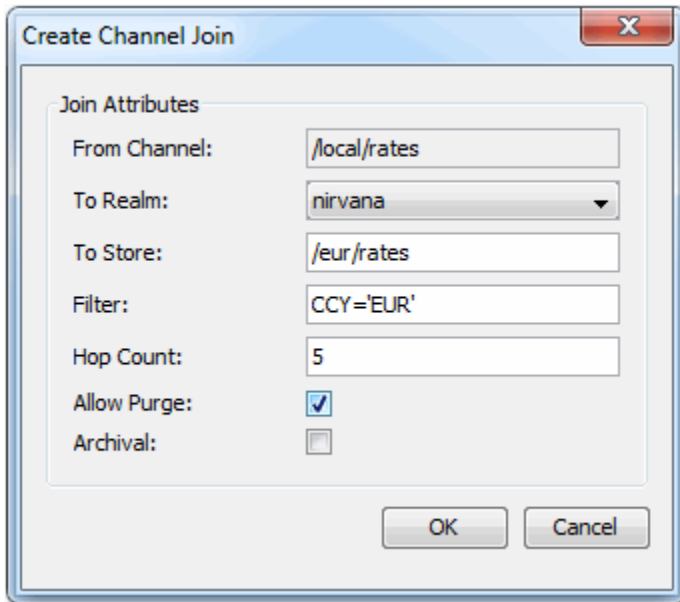
Universal Messaging also supports joins where the destination (incoming) is a queue. Universal Messaging does not support joins where the source of the join is a queue.

The image below shows a set of realms that is part of a cluster 'productioncluster'. There exists within this cluster, a cluster channel called '/eur/rates'.

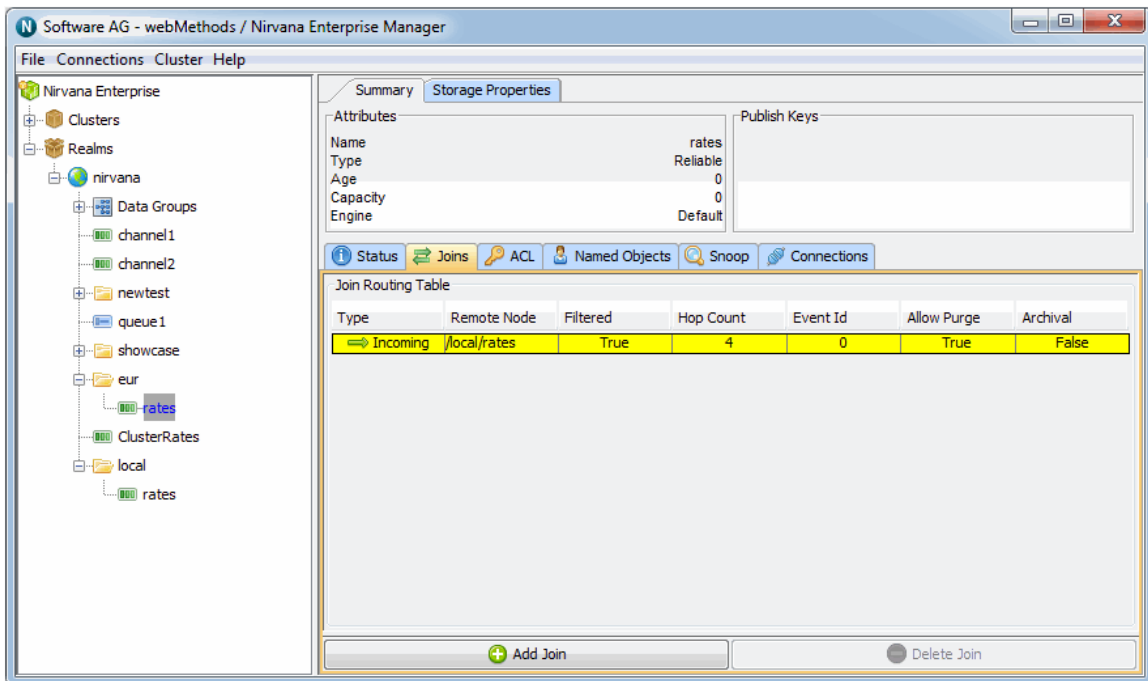


By selecting the cluster channel '/eur/rates' on the productionmaster realm, and right-clicking you can select the 'Join Channel' menu option, you will be presented with the join dialog. This allows you to create a join between the clustered channel and any other channel in any realm.

In this example, the realm 'productionmaster' contains a channel called /local/rates. If we select this channel as the channel we wish to join to 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 equals to 'EUR' will be published to the '/local/rates' channel.



Clicking on the 'OK' button will create the join. By selecting the 'Joins' tab panel for the '/eur/rates' channel you will be presented with a panel that shows the join just created, as shown in the image below. Selecting the newly created join will also show you any relevant filtering criteria that the join has been created with.

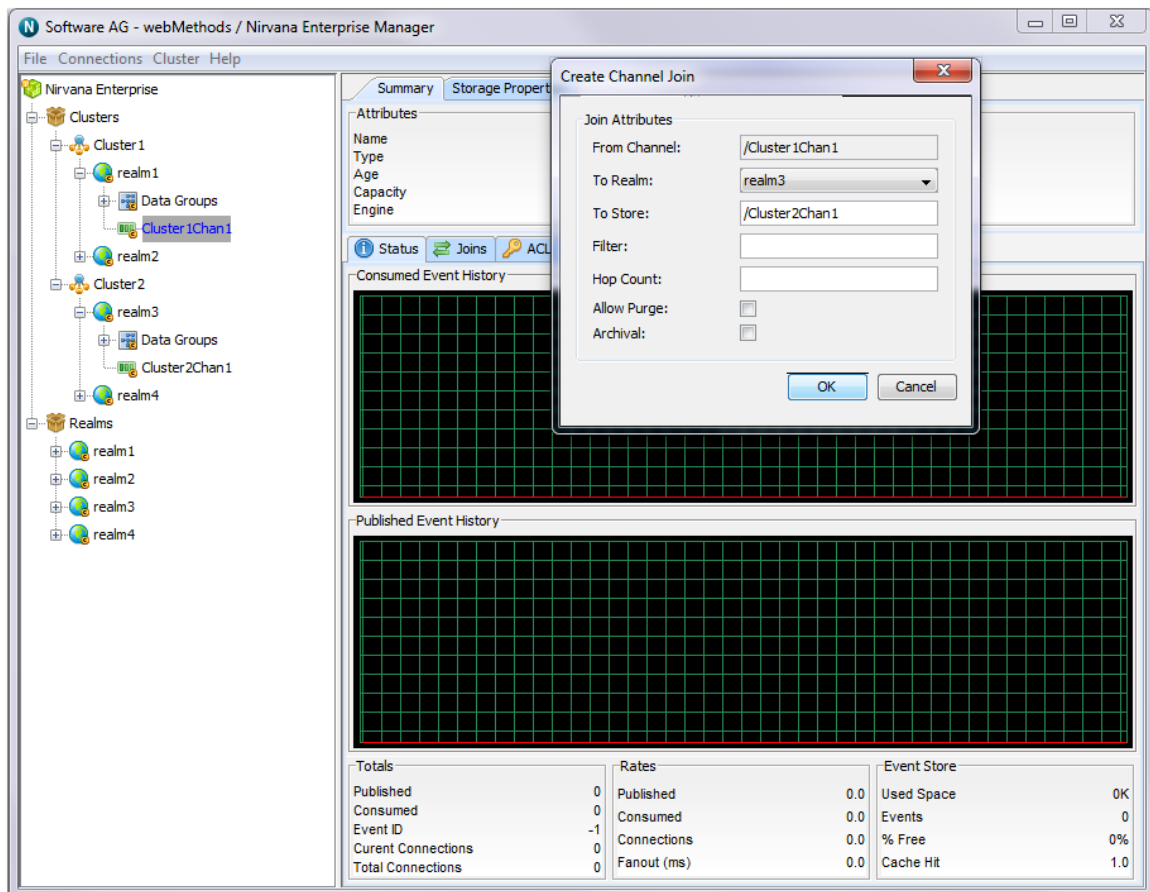


The table that shows the joins for a channel will indicate in the 'Type' column whether the join is 'Outgoing' or 'Incoming'. An outgoing join indicates that the selected channel is the source for the join, whereas 'Incoming' will indicate the selected channel is the destination channel.

It is also possible to delete the join by selecting a 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 joins 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 .

Channel Snoop

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

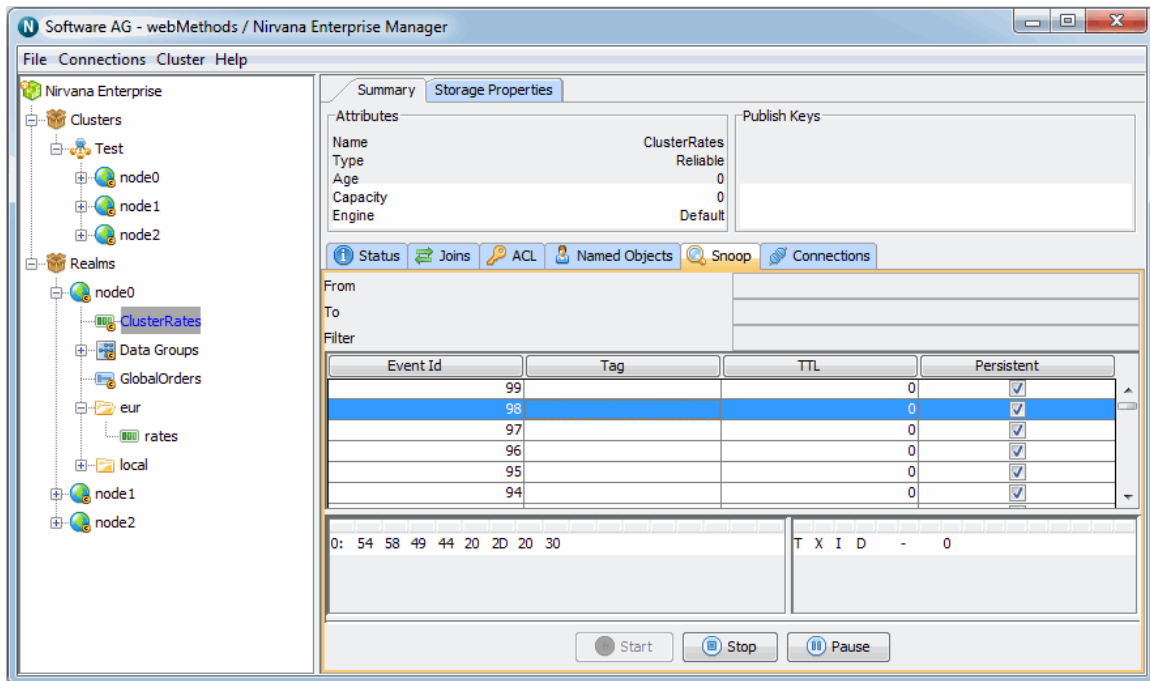
Snooping a channel using the Universal Messaging enterprise manager allows the display of the contents of events contained within that channel. Each channel node in the

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 a 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 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 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. As shown in the image below.

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 120](#)) please see that section of the enterprise .

Channel Publishing

This section will describe 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 93](#)) 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.

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.

Publish event to Channel rates

Publish Location
 Realm: nirvana
 Channel: ClusterRates

Event Data
 Corporate Sale
 File
 Clear

Event Details
 Event Tag: CSale
 Event TTL: 0
 Num Of Publishes: 1
 Is Persistent:
 Is Transient:

Property Input
 Key: discount
 Value: 0.05
 Type: Float
 Add

Property Display

Key	Value	Type
SalesPerson_ID	1024	Integer
quantity	2.0	Float
price	30.05	Float
discount	0.05	Float

Purge Original Event: OK Cancel

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.

To add an XML document as the event data, click on the 'Open' button in the top right hand corner of 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.

Publish event to Channel ClusterRates

Publish Location

Realm: nirvana

Channel: ClusterRates

Event Data

```
<?xml version="1.0"
encoding="UTF-8"?><Nirvan
aRealm comment="Realm
configuration from nirvana"
exportDate="2012-07-02+01
:00" name="nirvana">
<RealmConfiguration>
<ConfigGroup
```

File

Clear

Event Details

Event Tag: XML-0

Event TTL: 0

Num Of Publishes: 1

Is Persistent:

Is Transient:

Property Input

Key: discount

Value: 0.05

Type: Float

Add

Property Display

Key	Value	Type
price	30.05	Float
quantity	2.0	Float

Purge Original Event:

OK Cancel

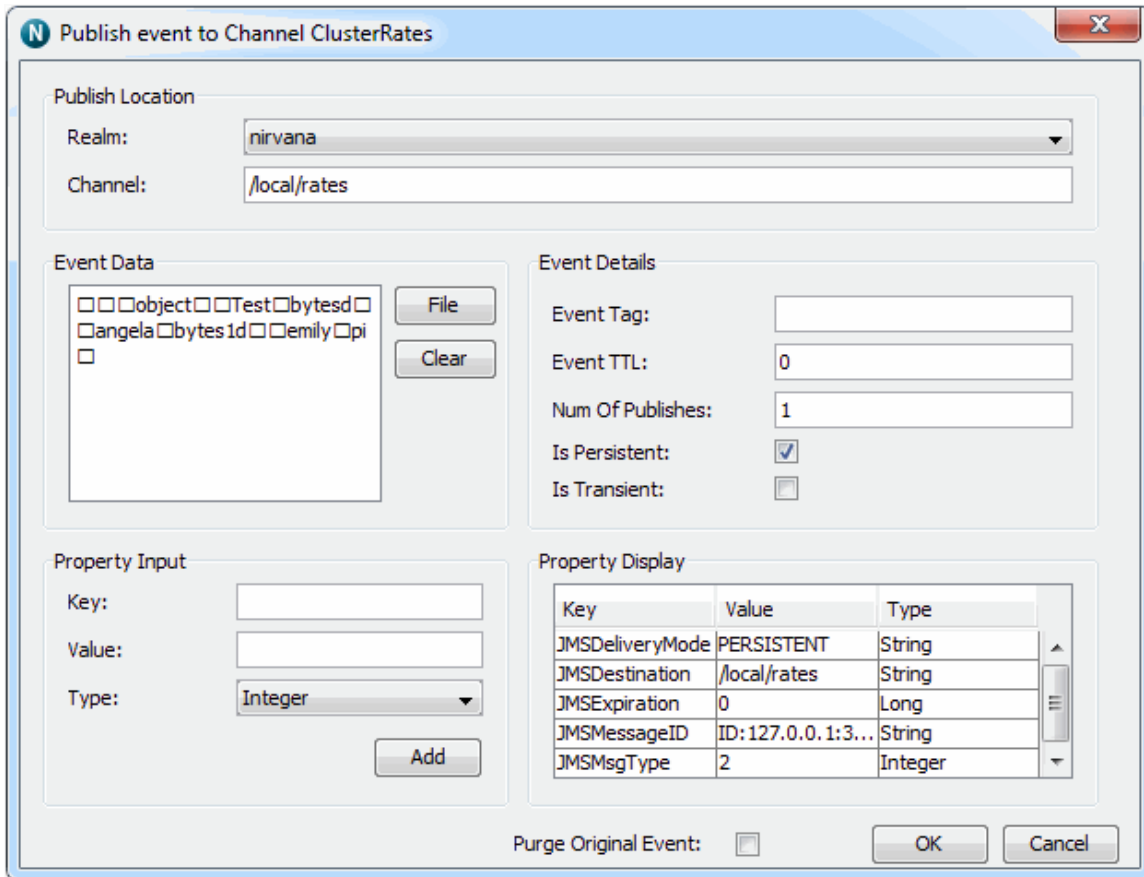
To publish the contents of any other file within the event data, repeat the above steps for XML and select a non-xml file. The contents of the file will not be displayed, however the file will be read in binary format when the 'Publish' button is clicked and published to the channel. Once again, when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

In order to duplicate or modify 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 93) 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 modify and republish from the table of events. Right-clicking on the event will display a menu with 2 options. The first option allows you to purge an individual event from a channel. The second menu option will open 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 as well as modified. To modify a property, double-click in the cell of the property you wish to change and then modifying its contents. With the republish option, you can also choose to purge the original event from the channel by checking the 'Purge Original Event' checkbox. Once the 'Publish' button is clicked, the duplicate event will be published onto the channel.

The number of publishes depends on the amount entered in the 'Num Of Publishes' field. 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.

Channel Named Objects

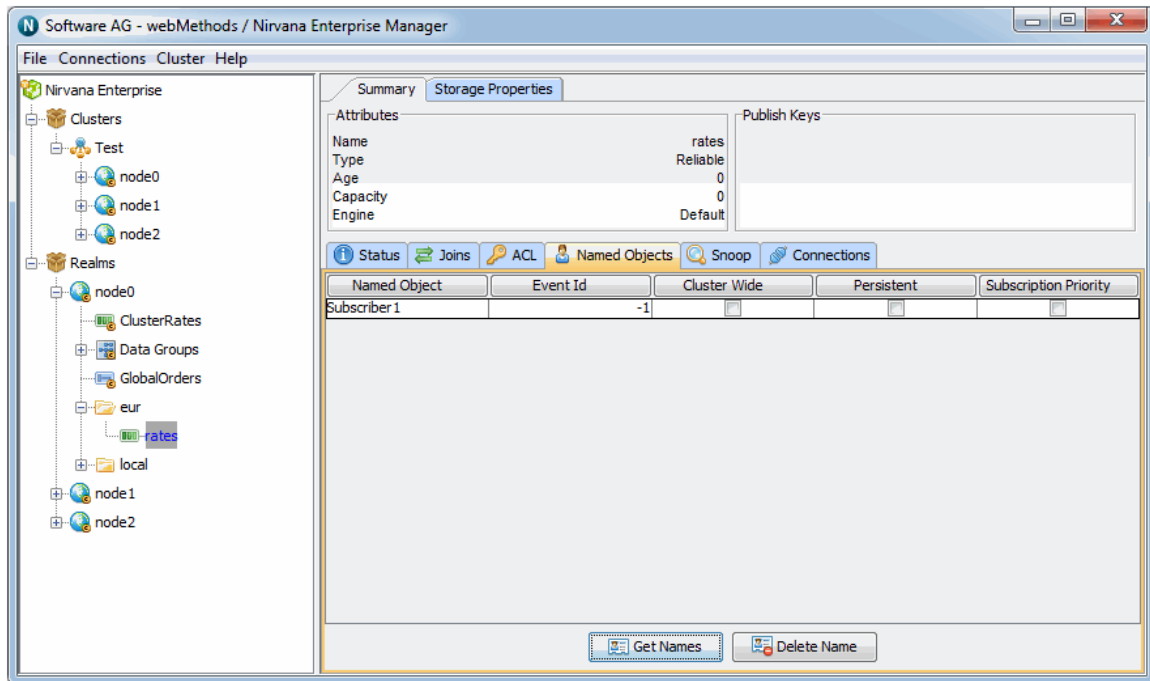
This section describes how to manage named objects stored by a realm server against a Universal Messaging channel. Named objects are stored by the server and provide state information for durable consumers mapped to specific names on channels. These named objects are stored by the server, and so every time a consumer uses the named object, it will always begin consuming from where the previous consumer finished.

The Universal Messaging apis provide methods for managing named objects programmatically, in Java and C#.

The Enterprise Manager also provides a method to remove named objects from channels.

In order to retrieve and remove any named objects present on a channel, firstly you need to select the channel. This will present a set of panels within a tabbed pane for the channel in question. One of these is labeled 'Named Objects'. When you first select

the 'Named Object' tab, this table will be empty. If you click on the button labeled 'Get Names', this will populate the table with any named objects found for that channel. The image below shows the named object table for the /eur/rates channel has a named object for the name 'administrator'.



The named object table shows each named object as a row in the table. The columns of the table show the name, current event id (as known by the server), whether the named object is cluster wide and whether it is persistent (when a realm is restarted the named object state is read from disk as opposed to being held in memory where it would be lost after a restart).

By highlighting a named object in the table, you can remove it by clicking on the 'Delete Name' button. This button will remove the named object from the server.

DataGroup Administration

The links below describe the DataGroup management features available within Universal Messaging's Enterprise Manager

- ["Creating DataGroups" on page 100](#)
- ["Adding Existing DataGroups to DataGroups" on page 104](#)
- ["Removing DataGroups from DataGroups" on page 106](#)
- ["Deleting DataGroups" on page 107](#)

Creating DataGroups

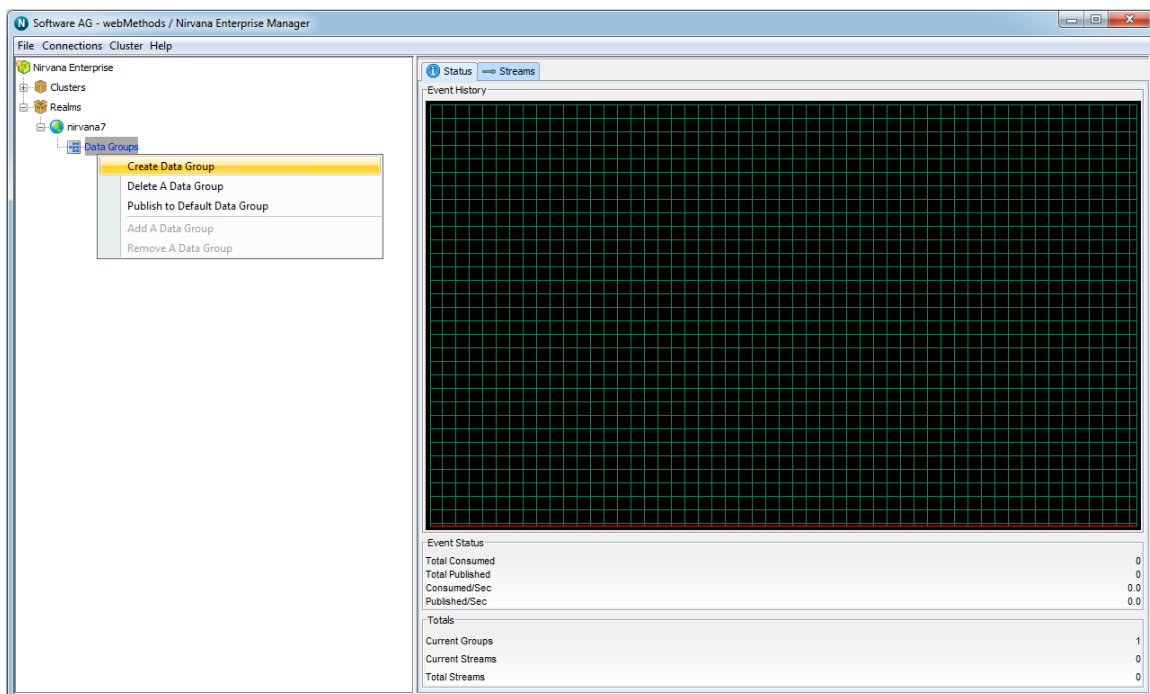
Universal Messaging DataGroups provide a very lightweight grouping structure that allows developers to manage user subscriptions remotely and transparently.

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

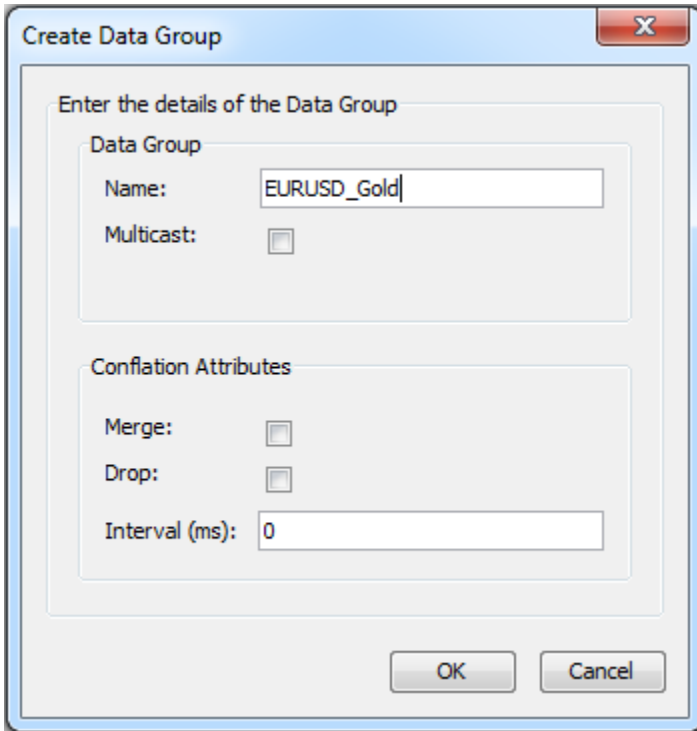
When you connect to a Universal Messaging realm in the Enterprise Manager, all resources and services found within the realm namespace are displayed in a tree structure under the realm node itself. It is possible to view multiple Universal Messaging realm servers from a single enterprise manager instance.

Creating a DataGroup

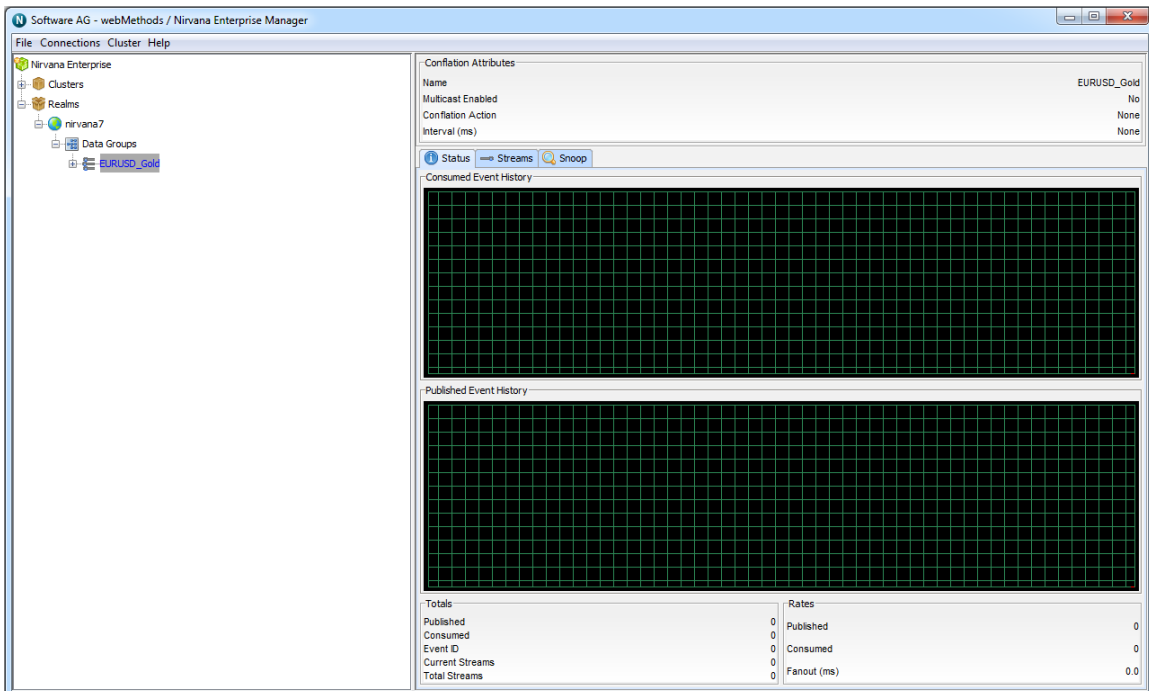
In Enterprise Manager, the **Data Groups** node exists within the realm node. Locate the Data Group Node, and right click on it to bring up a context menu:



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



The new DataGroup can now be seen in Enterprise Manager:

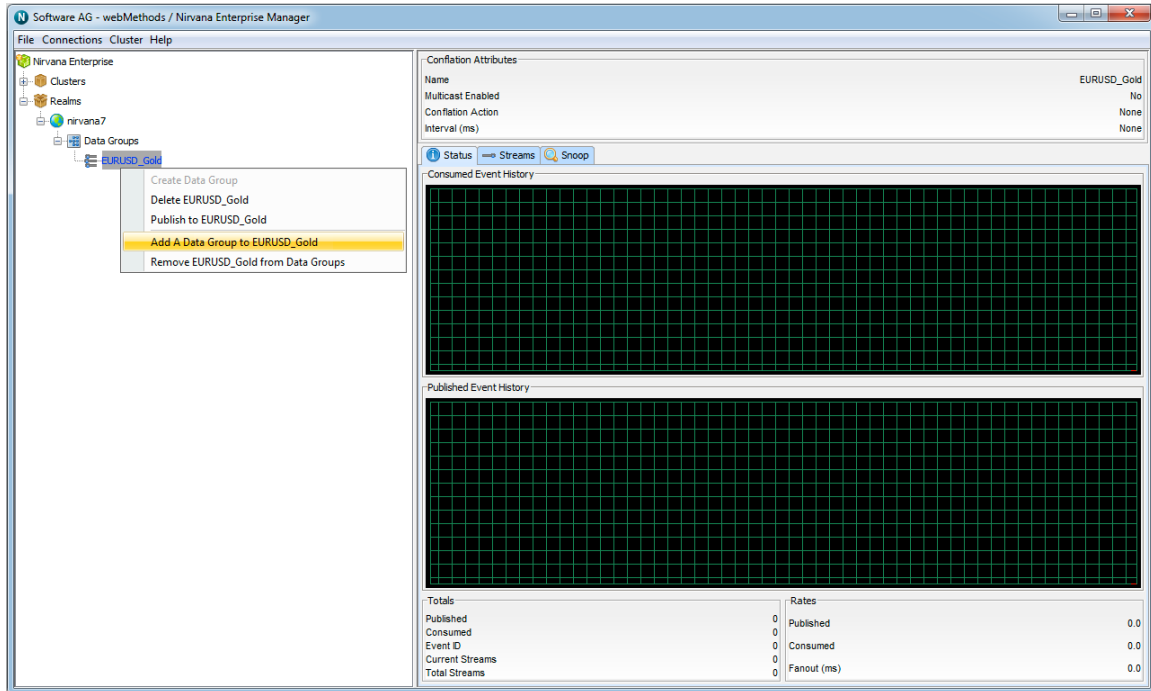


Publishers with the *Publish to DataGroups* ACL permission can now publish messages to the new DataGroup programmatically.

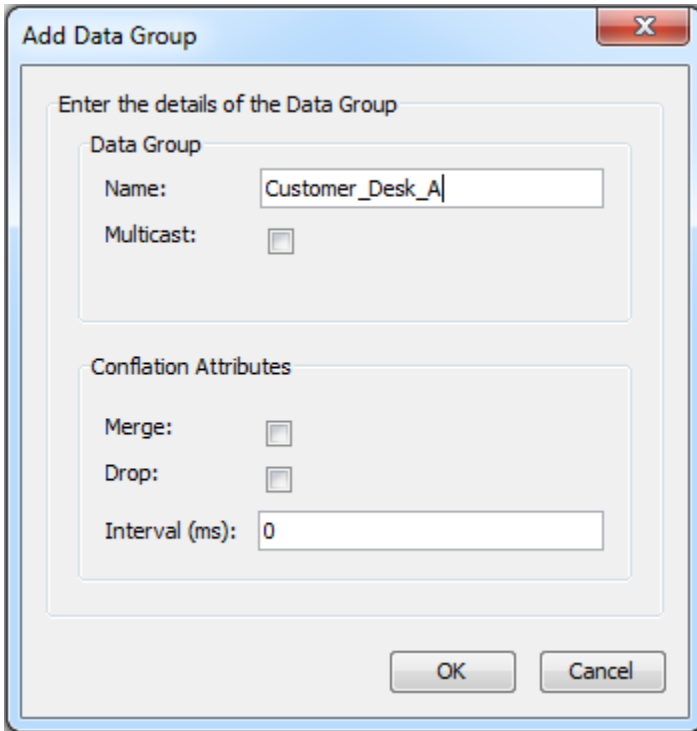
Creating a Nested DataGroup

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

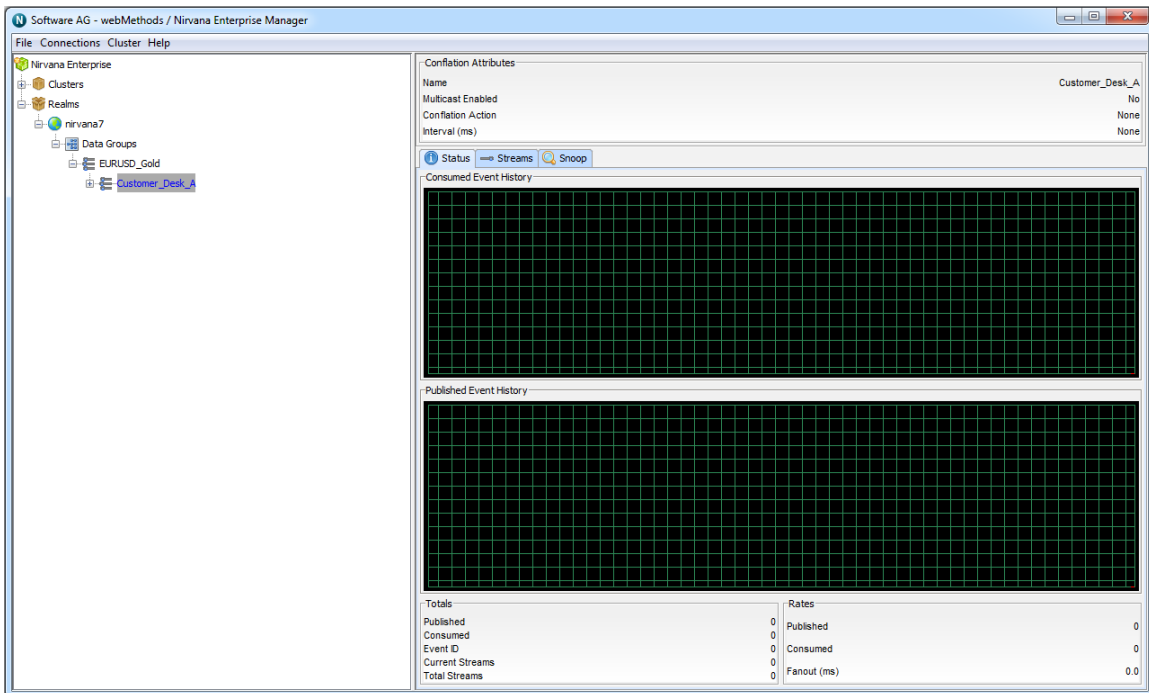
Right-click its icon, and the following context menu appears:



Add the nested DataGroup:



The nested DataGroup can now be seen in the tree:



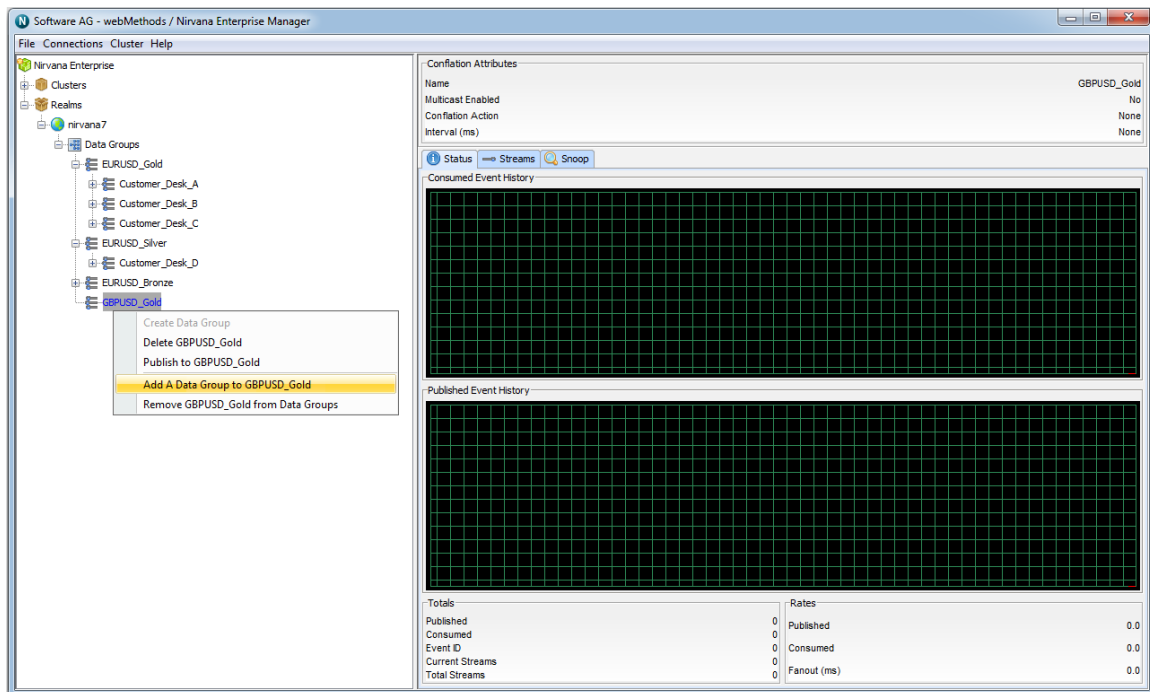
Adding Existing DataGroups to DataGroups

We've already seen how to use Enterprise Manager to create a new DataGroup and add it to an existing DataGroup (see "[Creating DataGroups](#)" on page 100). In this section, we will add an existing DataGroup to a DataGroup.

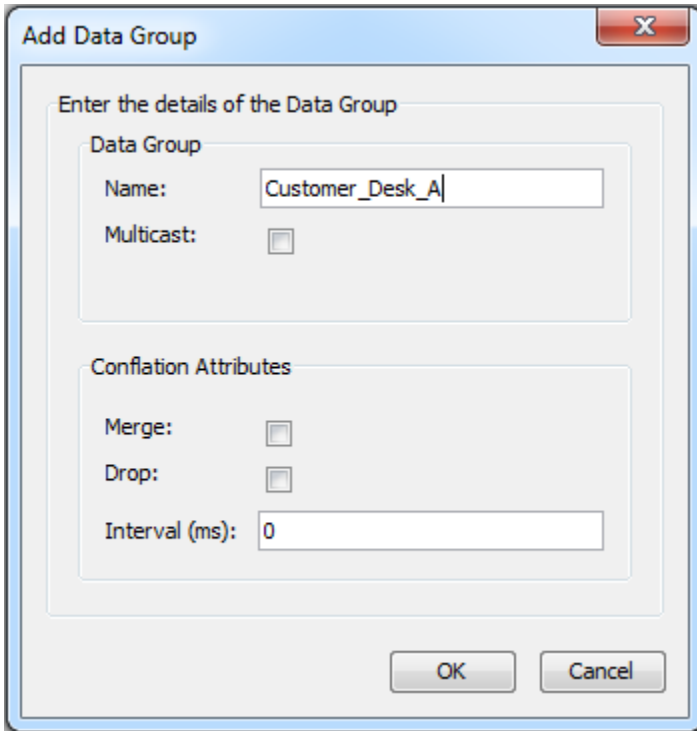
Assume that in the following example structure of DataGroups, we would like to add the existing **Customer_Desk_A** DataGroup, which itself already a member of the **EURUSD_Gold** DataGroup, to the **GBPUSD_Gold** DataGroup too.

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

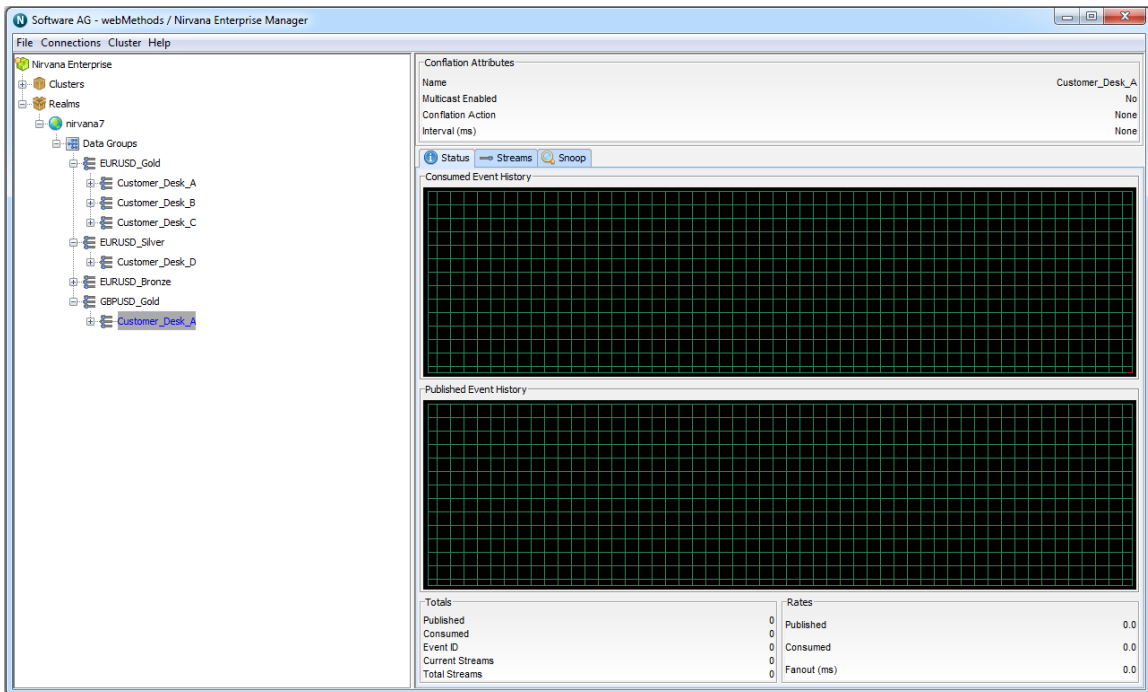
Right-click its icon, and the following context menu appears:



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



Because this DataGroup already exists (and was a member of **EURUSD_Gold**), it now appears as a member of two DataGroups (and thus appears twice in the tree of DataGroup nodes):



Now, any events published to the DataGroups **EURUSD_Gold** or **GBPUSD_Gold**, or directly to the DataGroup **Customer_Desk_A**, will be delivered to any DataStreams which are members of the **Customer_Desk_A** DataGroup.

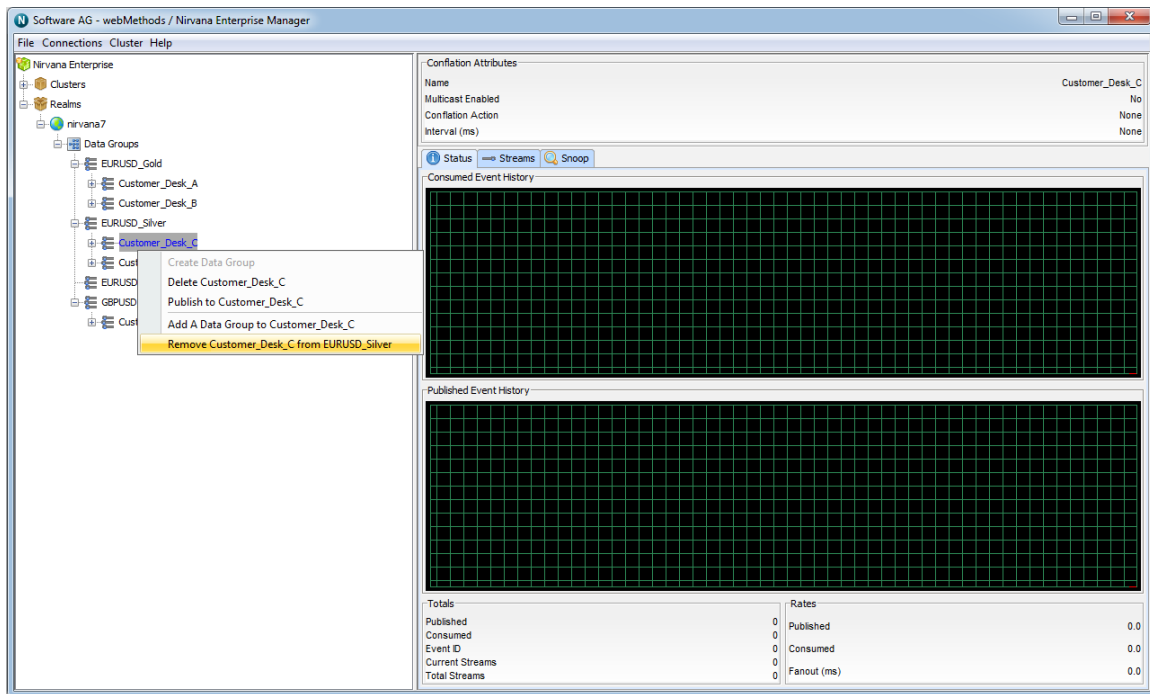
Removing DataGroups from DataGroups

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

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

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

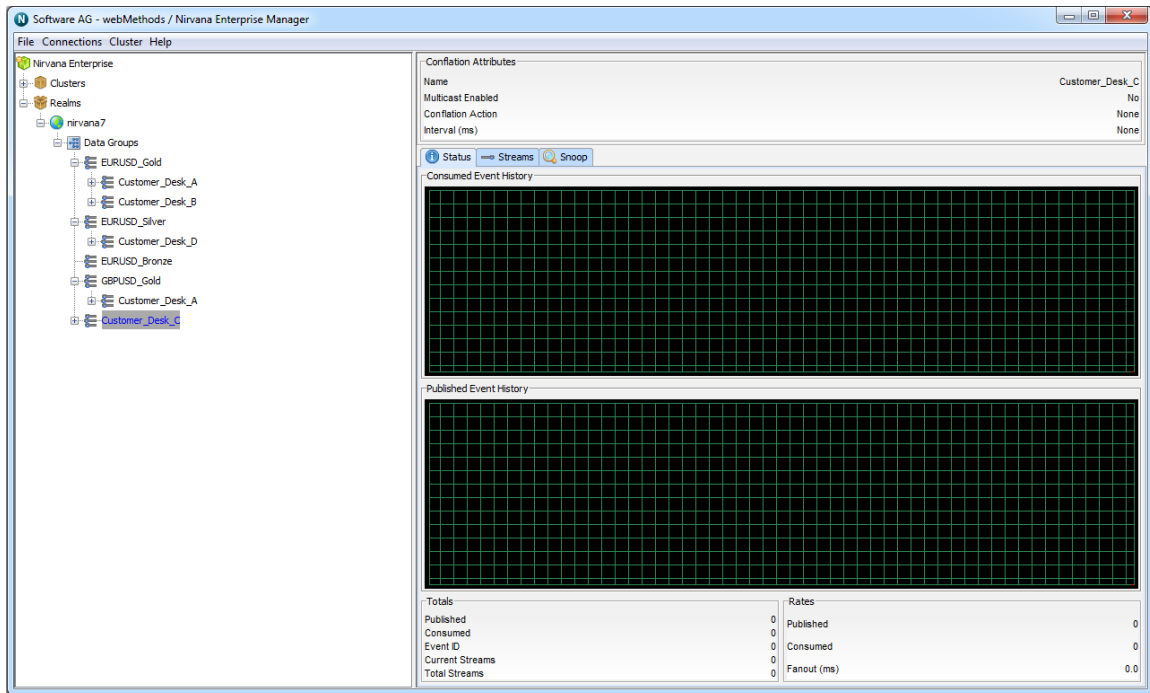
Right-click its icon, and the following context menu appears:



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

The DataGroup, having been removed, will either:

- Be moved to the top level DataGroups node if it has no other parent DataGroups, or
- Appear in other nodes in the tree if it has at least one other parent DataGroup.



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

Deleting DataGroups

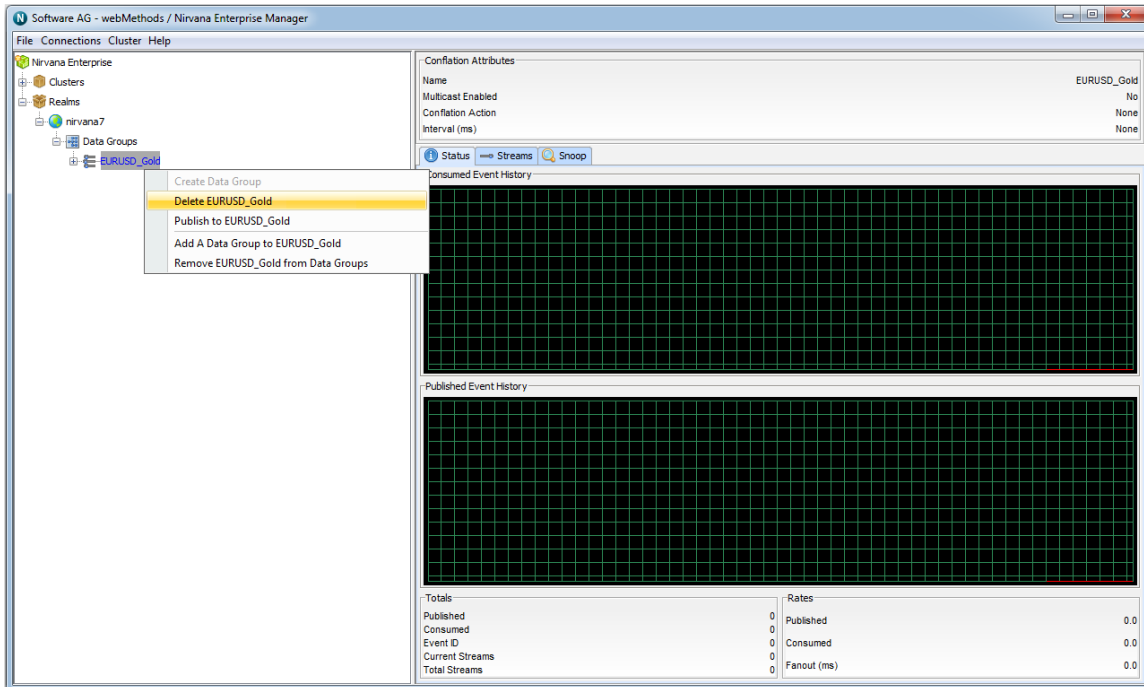
There are two ways of deleting a DataGroup using the Enterprise Manager:

1. by navigating the DataGroups tree
2. by typing in its name

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

Deleting by Navigating the DataGroups Tree

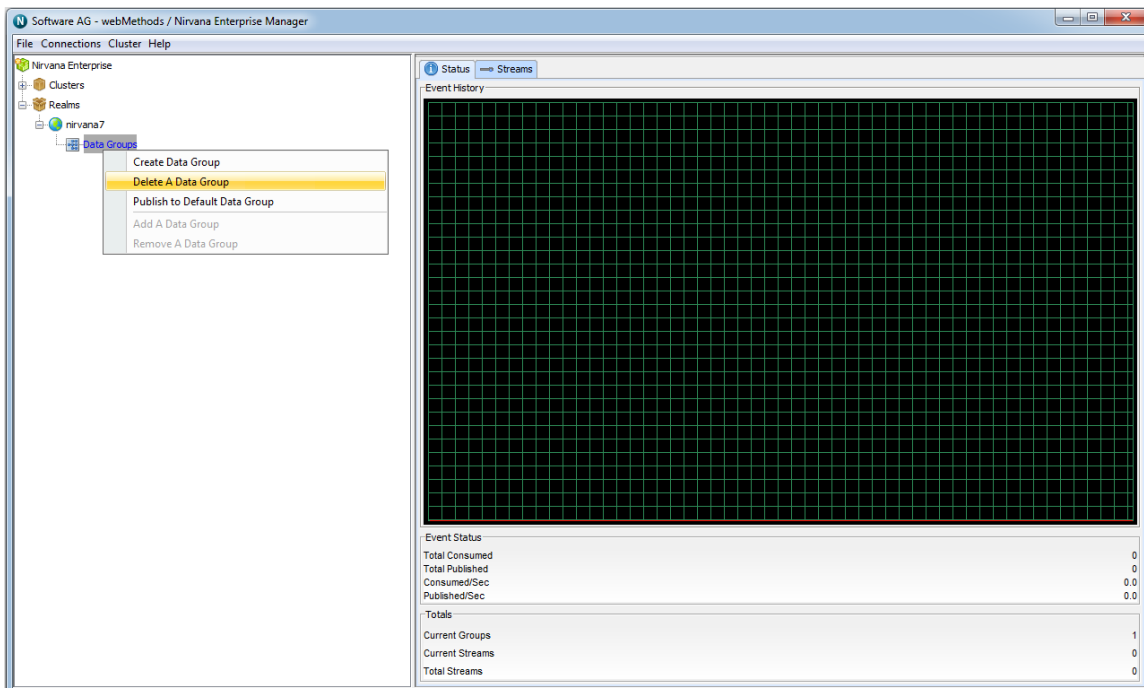
In Enterprise Manager, locate the Data Group Node (in this example, **EURUSD_Gold**), and right click on it to bring up a context menu:



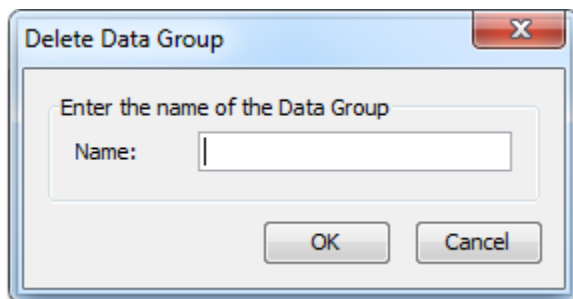
Click OK on the confirmation dialog, and the DataGroup will be deleted.

Deleting by Typing in the DataGroup Name

In Enterprise Manager, right-click the DataGroups node and select the "Delete A DataGroup" context menu option:



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



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

Queue Administration

The links below describe the queue management features available within Universal Messaging's Enterprise Manager

- ["Creating Queues" on page 109](#)
- ["Editing Queues" on page 115](#)
- ["Copying Queues" on page 118](#)
- ["Queue Snoop" on page 120](#)

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.

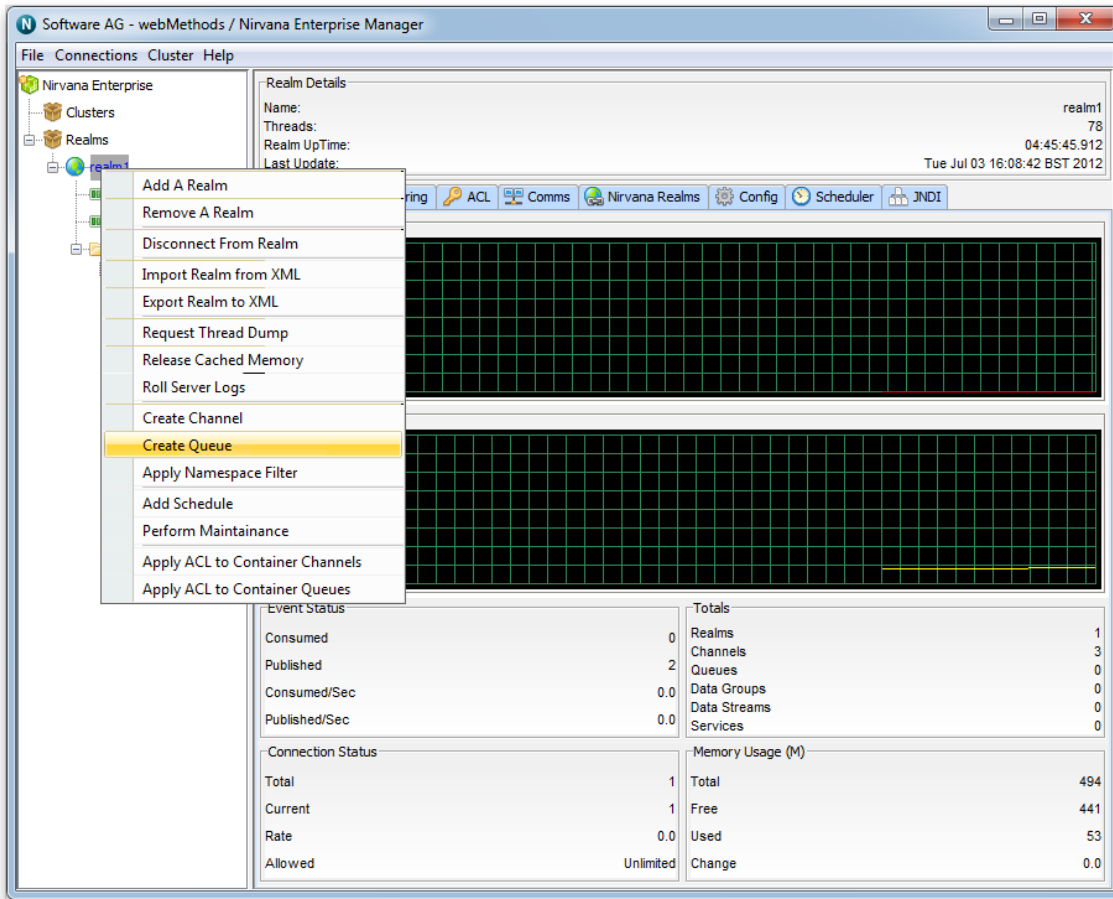
Creating queues using the Enterprise Manager creates the physical object within the realm. Once created, references to queues can be obtained using the Universal Messaging Client and Admin APIs. Queues can also be monitored and managed using the Enterprise Manager.

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.

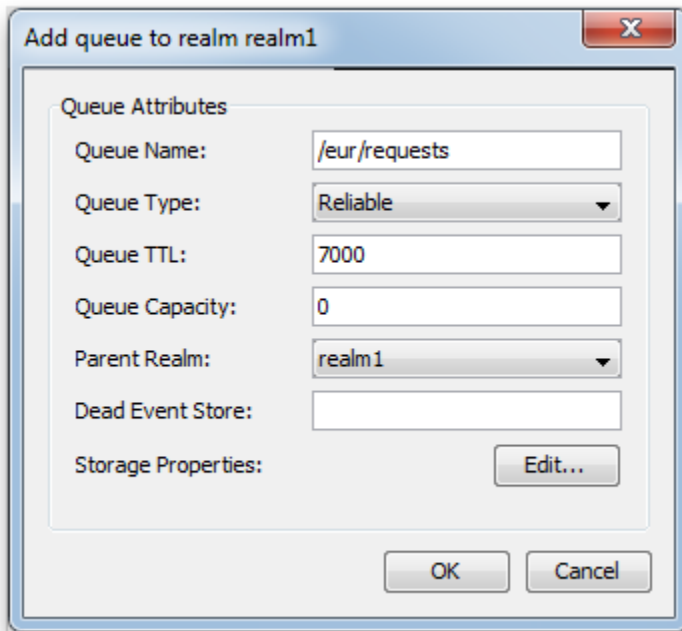
Creating Realm Queues

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

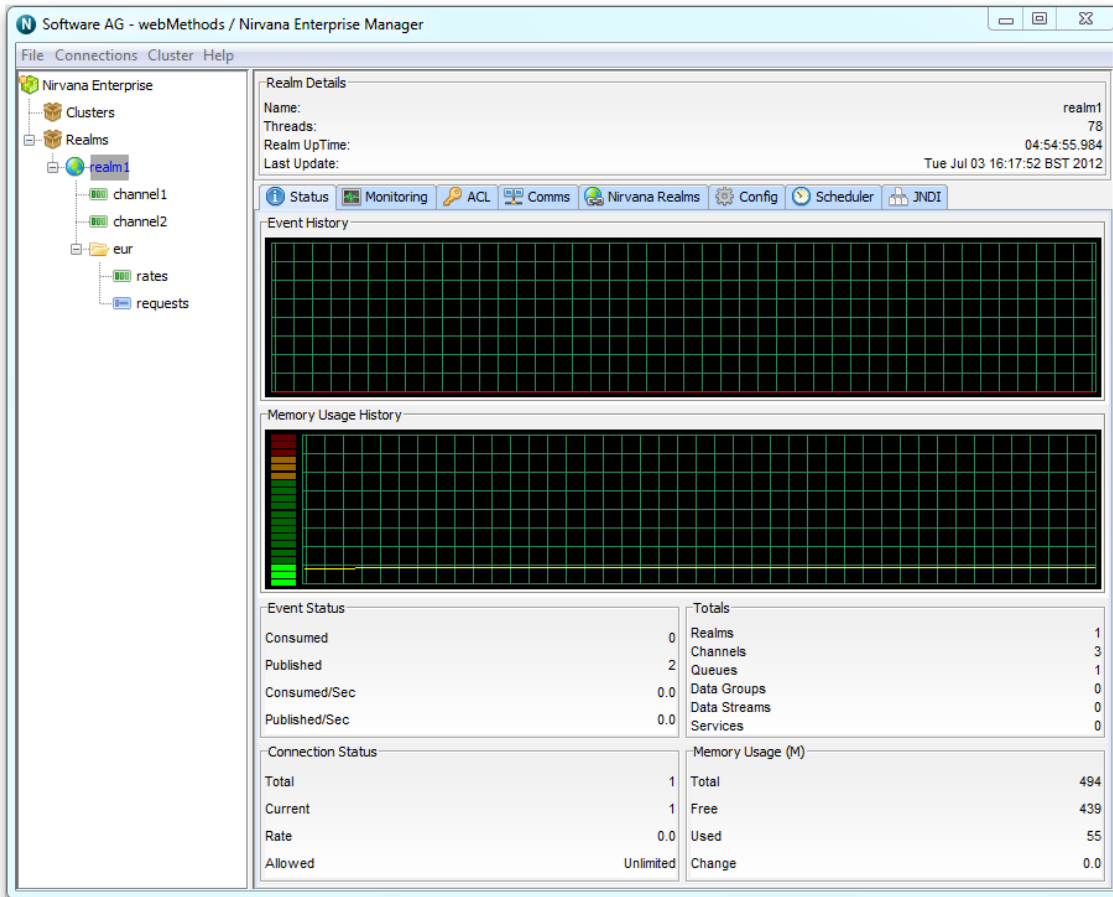
In order to create a queue called '/eur/requests' on a realm called 'nirvana' simply right-click on the realm node called 'nirvana' to display a pop-up menu containing an option called 'Create queue' (as shown in the image below).



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.

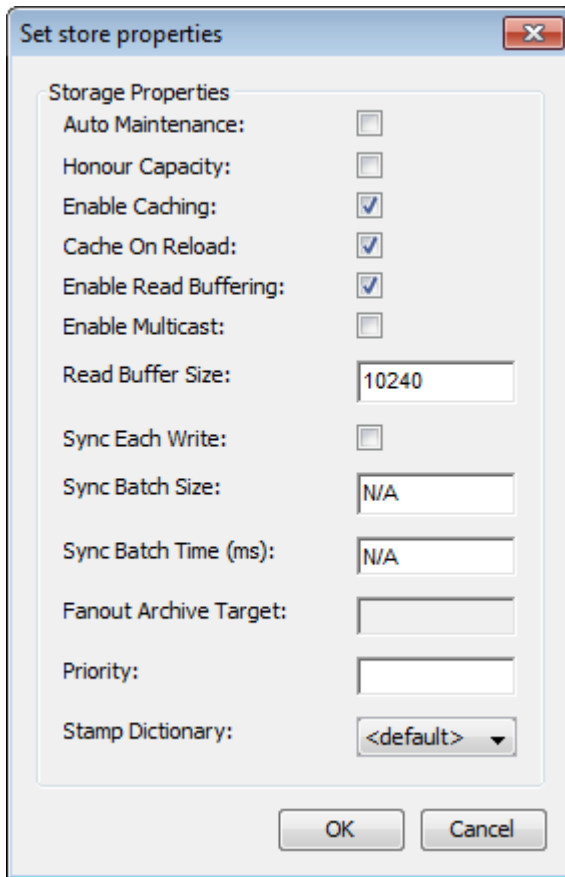


Clicking on the 'OK' button will create the queue '/eur/requests' on the Universal Messaging realm 'nirvana' and render the queue object in the namespace tree of the Enterprise Manager underneath the realm node. This is shown in the image below.



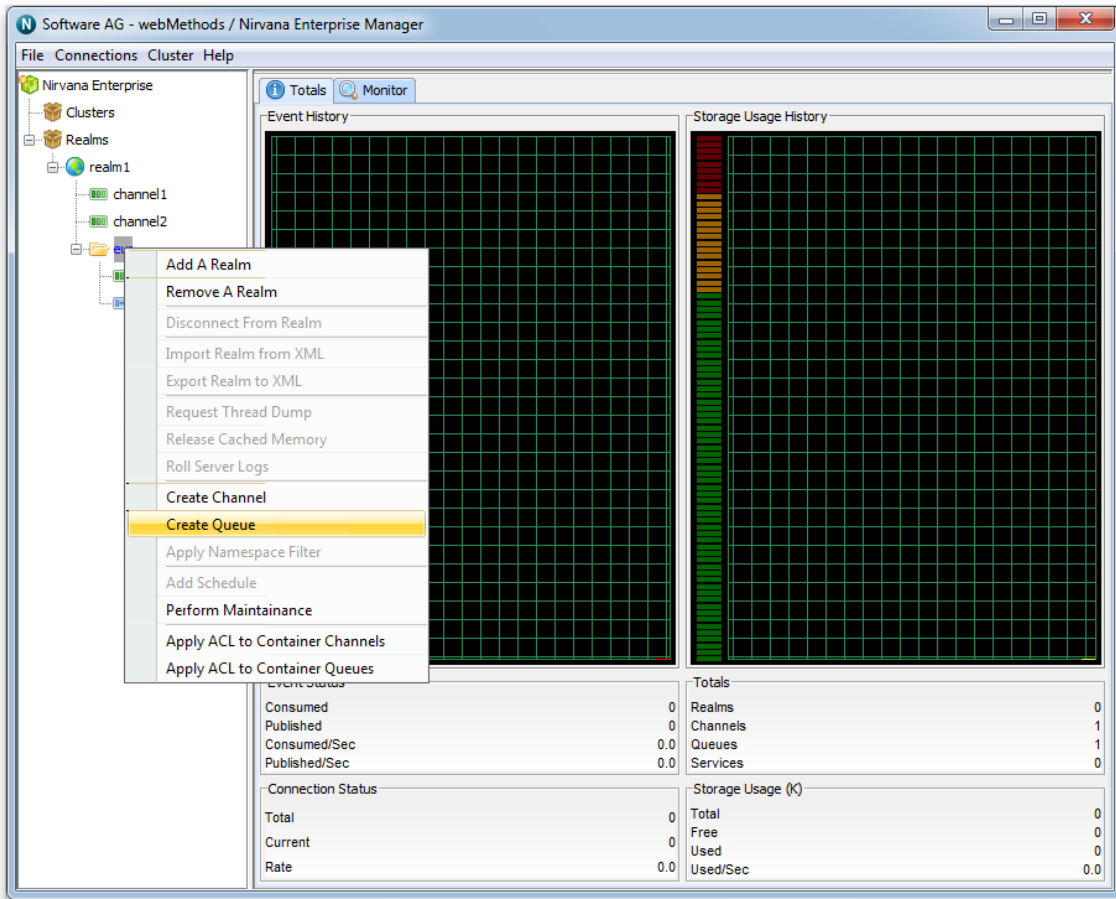
As you can see from the image above, the queue node in the tree has been created under a folder (container node) called '/eur' under the realm 'nirvana'.

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".

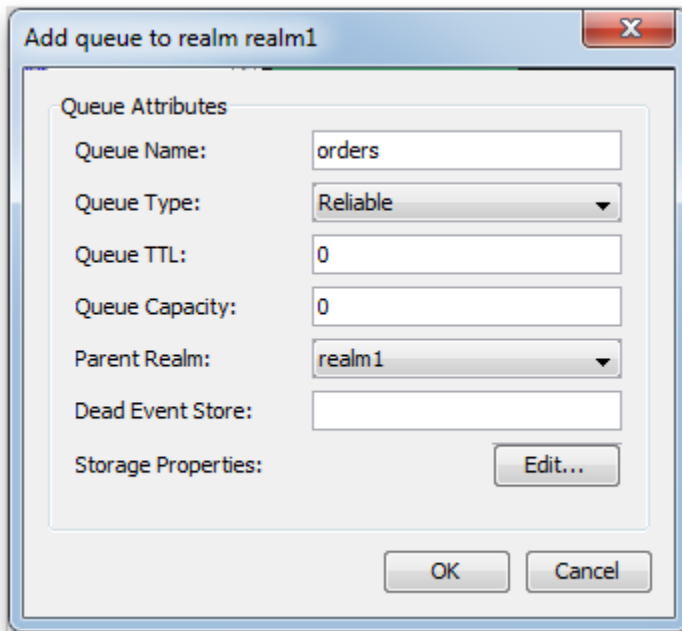


Queues can also be created from the context of a container nodes by specifying a relative queue name.

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.



The screenshot shows a dialog box titled "Add queue to realm realm1". It contains the following fields and controls:

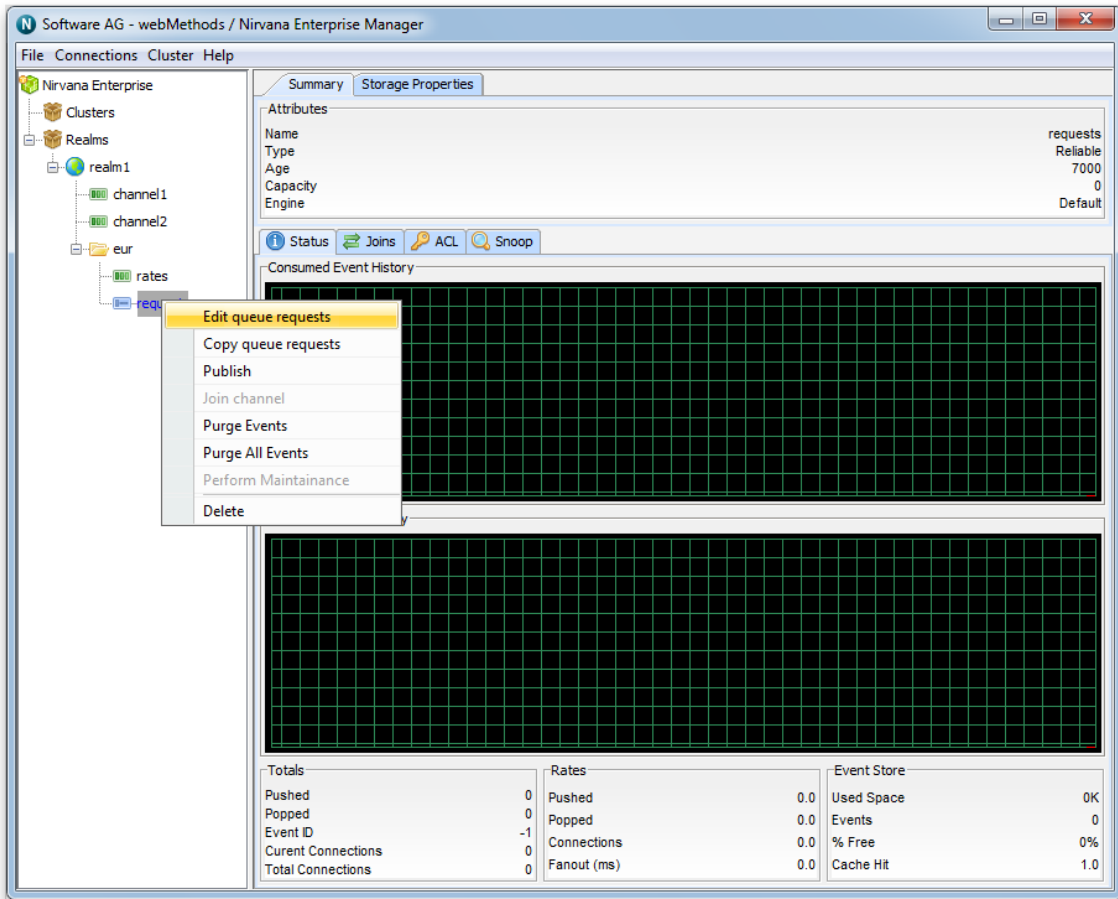
- Queue Name:
- Queue Type:
- Queue TTL:
- Queue Capacity:
- Parent Realm:
- Dead Event Store:
- Storage Properties:
- OK button
- Cancel button

Editing Queues

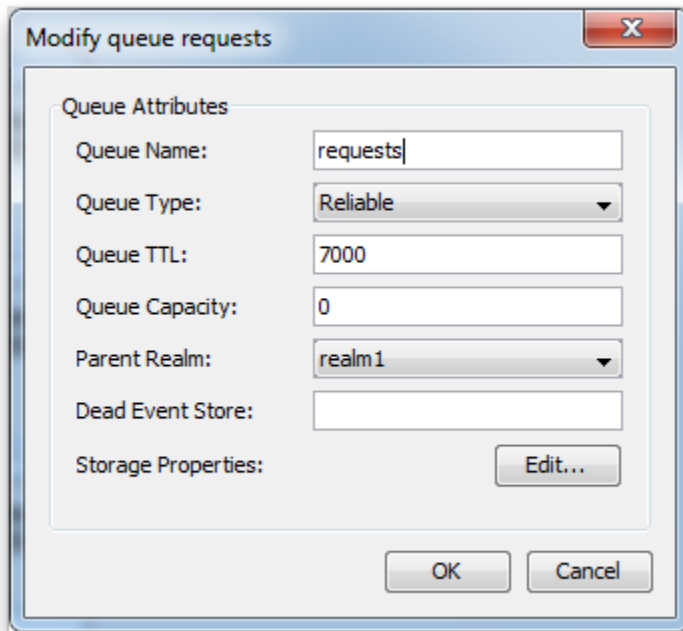
This section describes the process of editing queues in Universal Messaging realms. Each queue that is created consists of a physical object within the Universal Messaging realm as well as its logical reference within the namespace.

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. 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 removed and then the new queue is created and the events are then copied from the temporary queue onto the new queue.

Firstly, by selecting the queue in the namespace that you wish to edit 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.

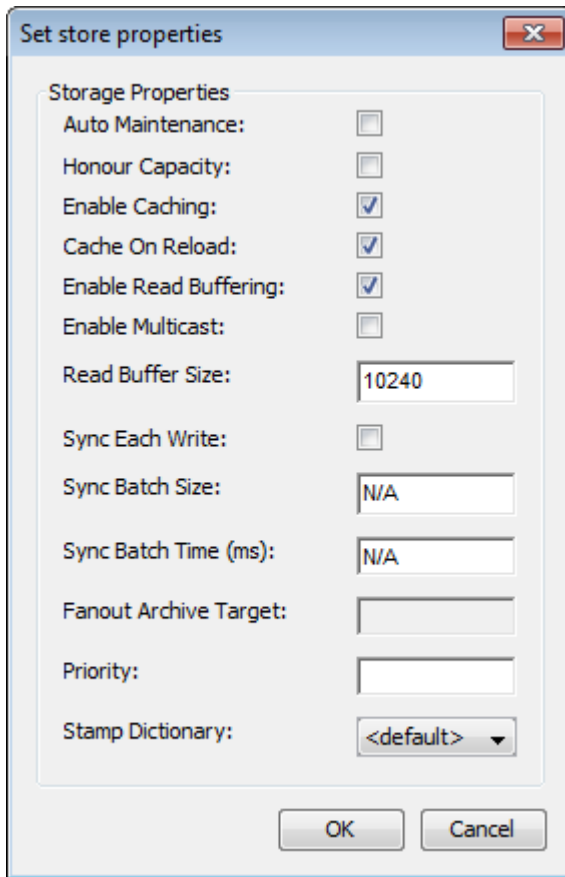


By selecting the 'Edit Queue' option, you will be presented with a dialog that allows you to modify the details of the queue. These details not only include the queue attributes, but also the realm to which the queue exists. The image below shows the edit queue dialog.



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".

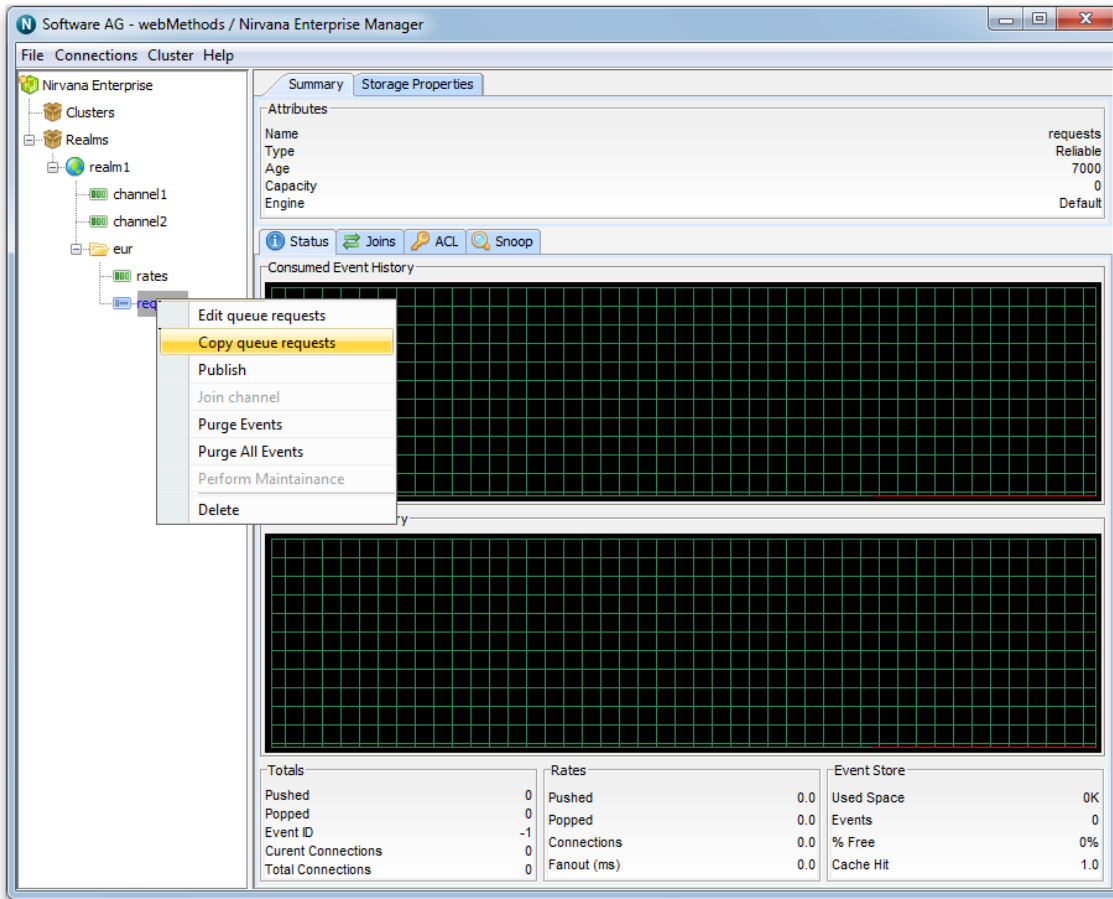


Copying Queues

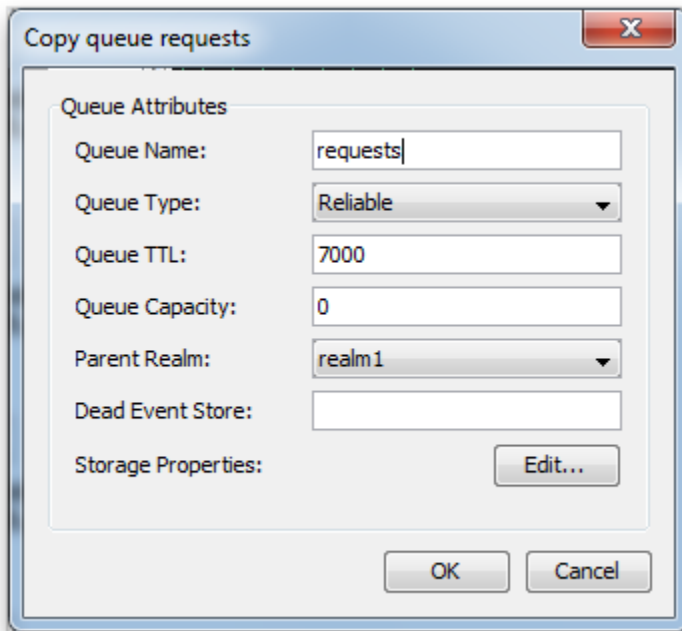
This section will describe the process of copying queues in Universal Messaging realms. Each queue that is created consists of a physical object within the Universal Messaging realm as well as its logical reference within the namespace.

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.

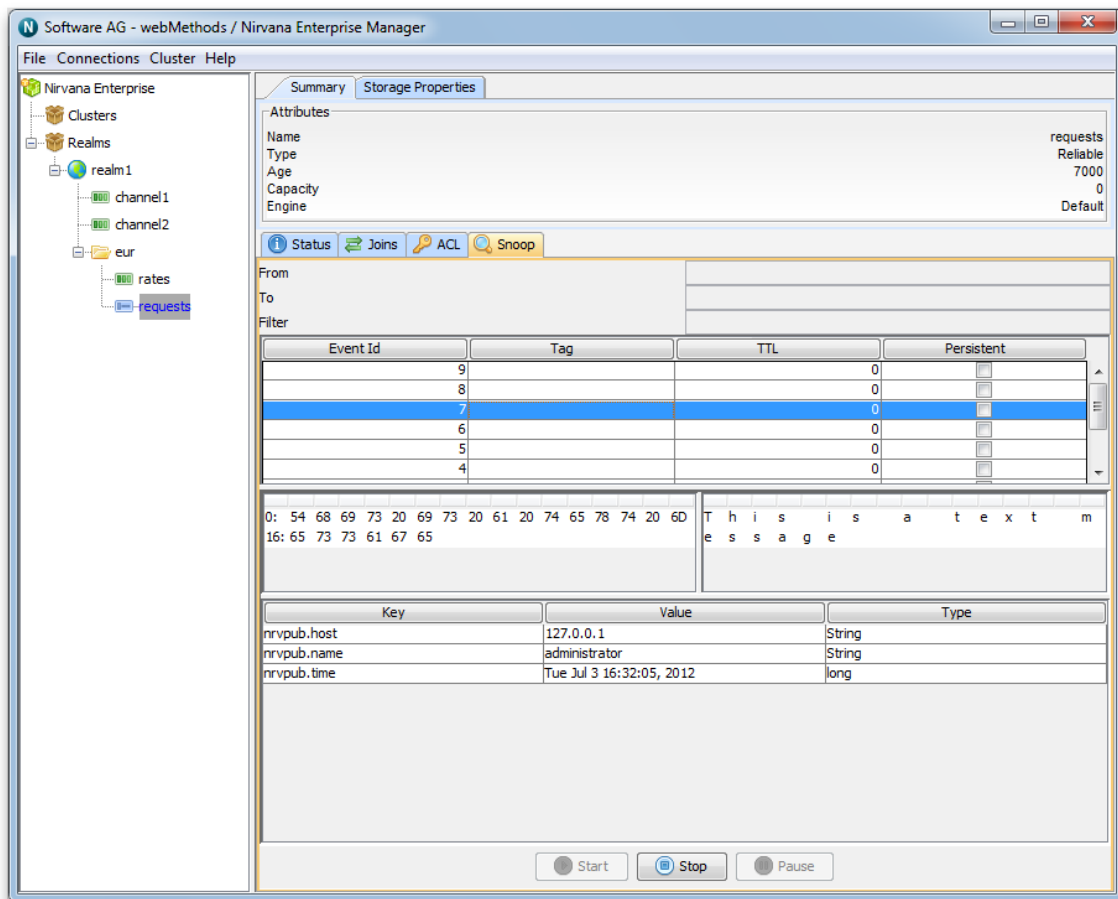
First of all, by selecting the queue you wish to snoop in the namespace tree, the Enterprise Manager will display a number of panels in a tabbed pane. One of these tabs is labelled 'Snoop'. Selecting the snoop tab will display a panel like the one shown in the image below.

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.



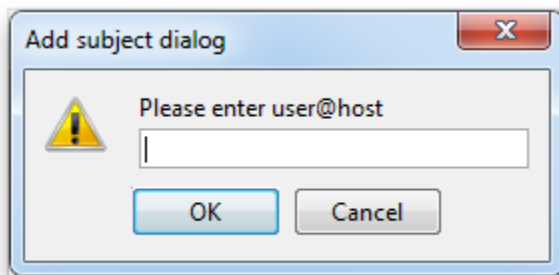
The button labelled 'Pause' will temporarily suspend receipt of any new message being received into the snoop panel for the selected queue. The 'Stop' button will stop snooping events and clear all the panels and tables.

Channels can also be snooped using the snoop panel (see "[Channel Snoop](#)" on page 93).

Security

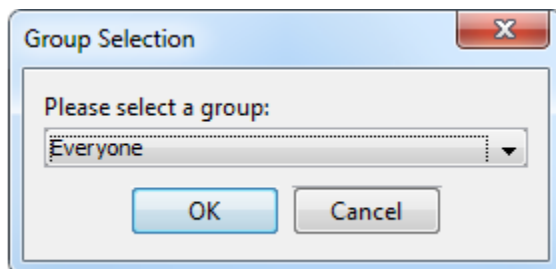
Adding ACLs

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 336) 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:

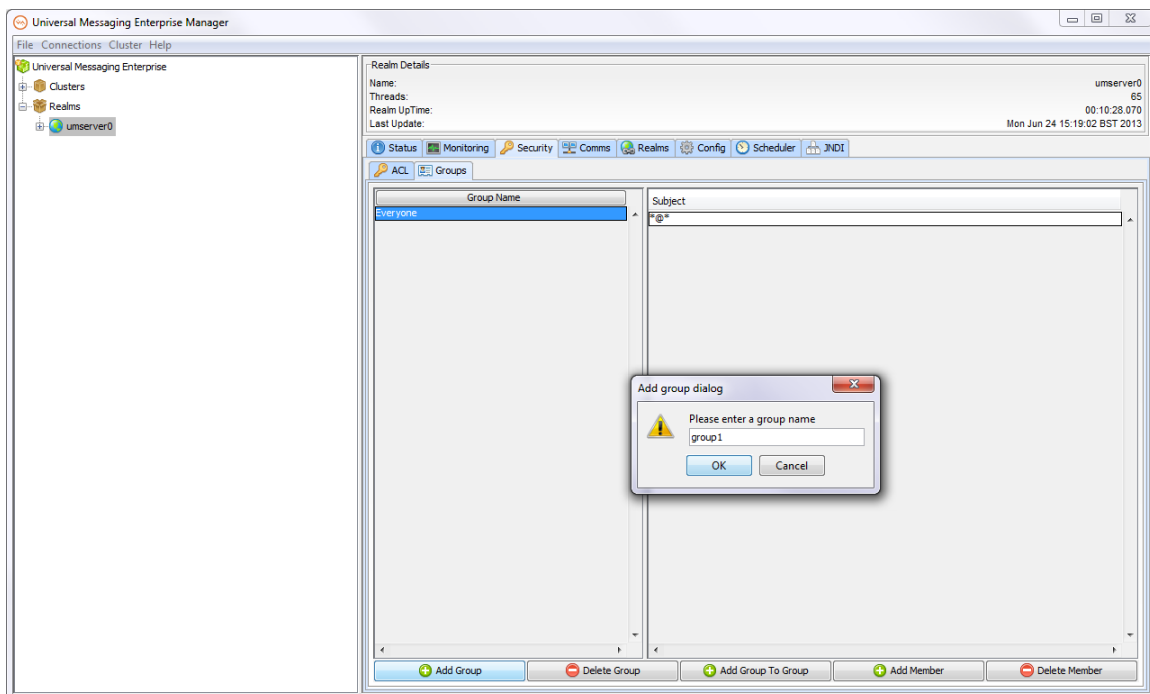
- ["Security Groups" on page 336](#)
- ["Realm ACL" on page 125](#)
- ["Channel ACL" on page 127](#)
- ["Queue ACL" on page 129](#)
- ["Service ACL" on page 131](#)

- ["Interface VIA ACL" on page 133](#)

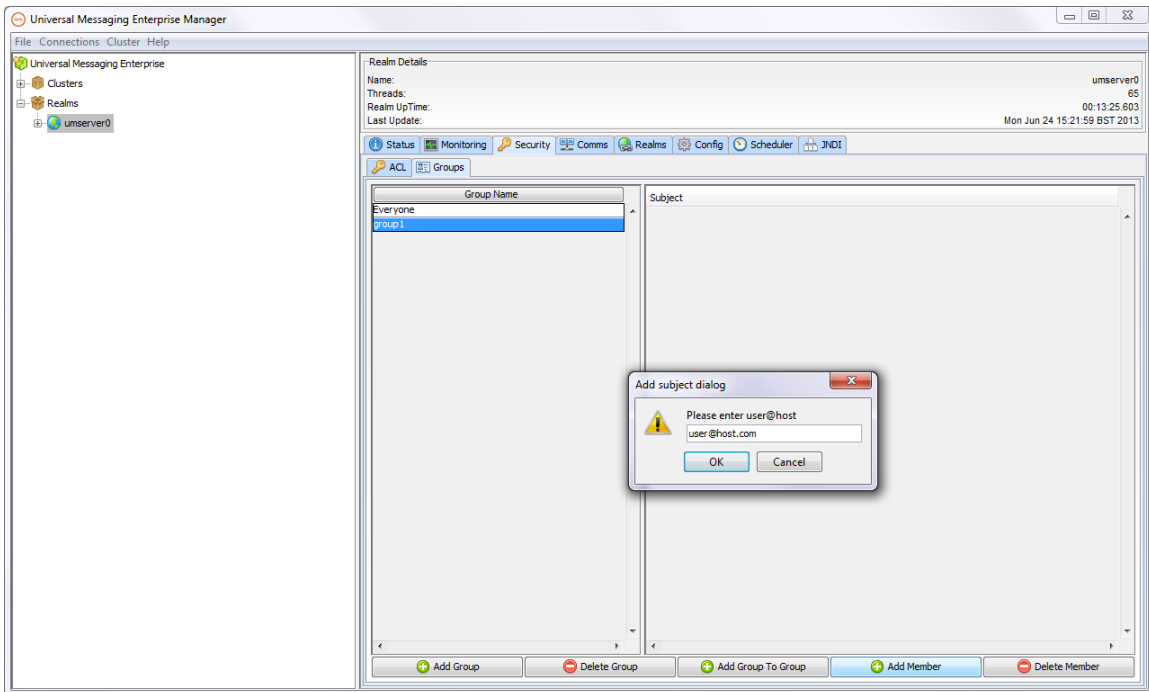
Nirvana Enterprise Manager - Security Groups

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.

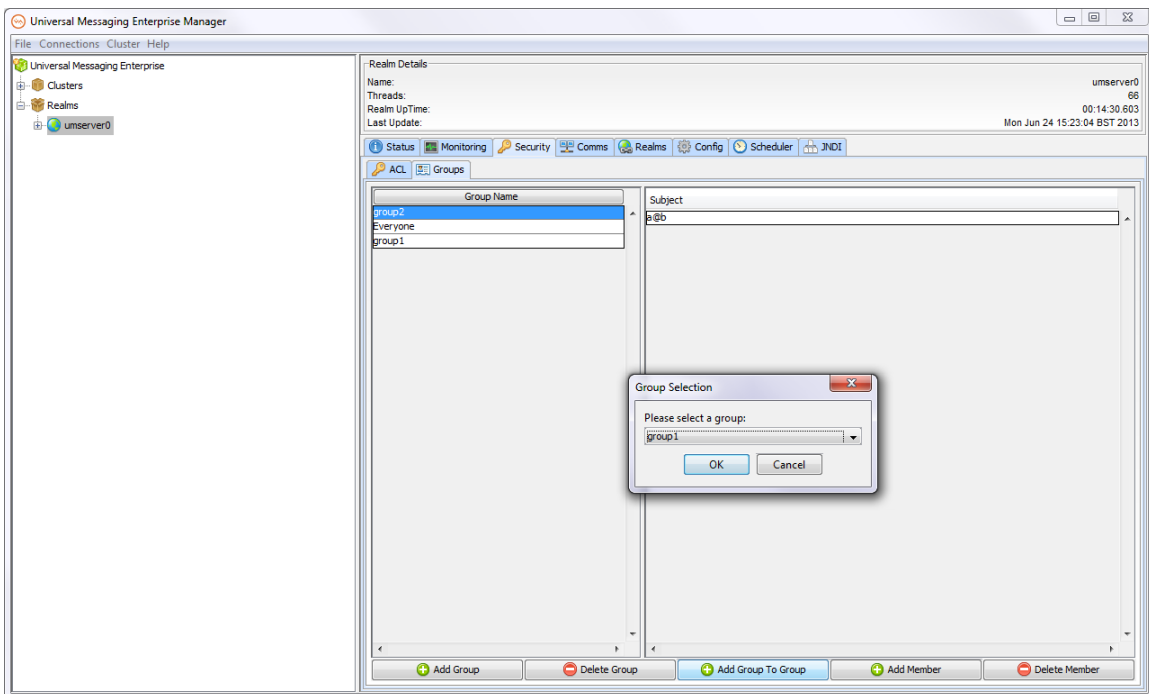
Before adding a Security Group to an ACL, it must first be created. This can be done programmatically or via the Enterprise Manager, as shown below.



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 Nirvana, 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 not 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

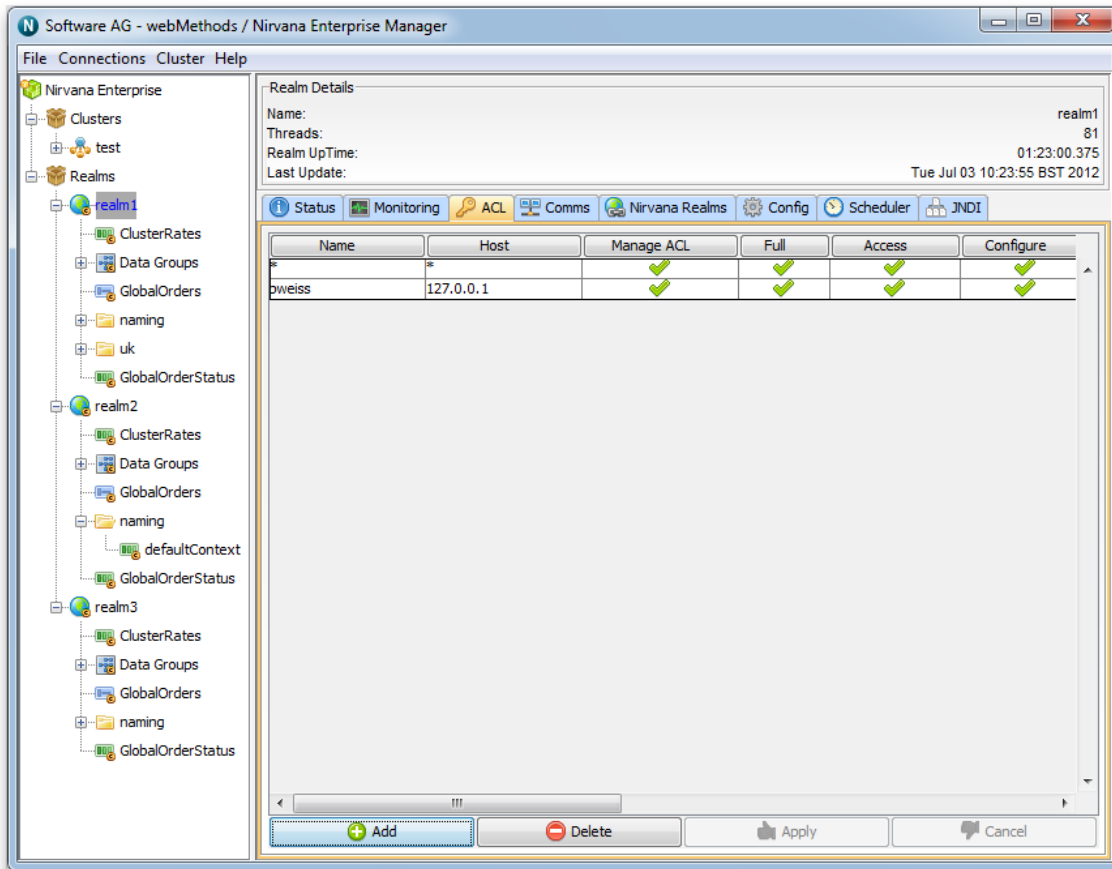
As mentioned in the security introduction (see "[Security](#)" on page 122), in order to perform operations within a Realm clients connecting to the realm must be given the correct entitlements.

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 an 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 manage the list of ACL entries
- 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
- P2P - Can create/destroy P2P services
- 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, who's 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 i wanted grant the `*@*` user the 'Access' realm privilege, i 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.

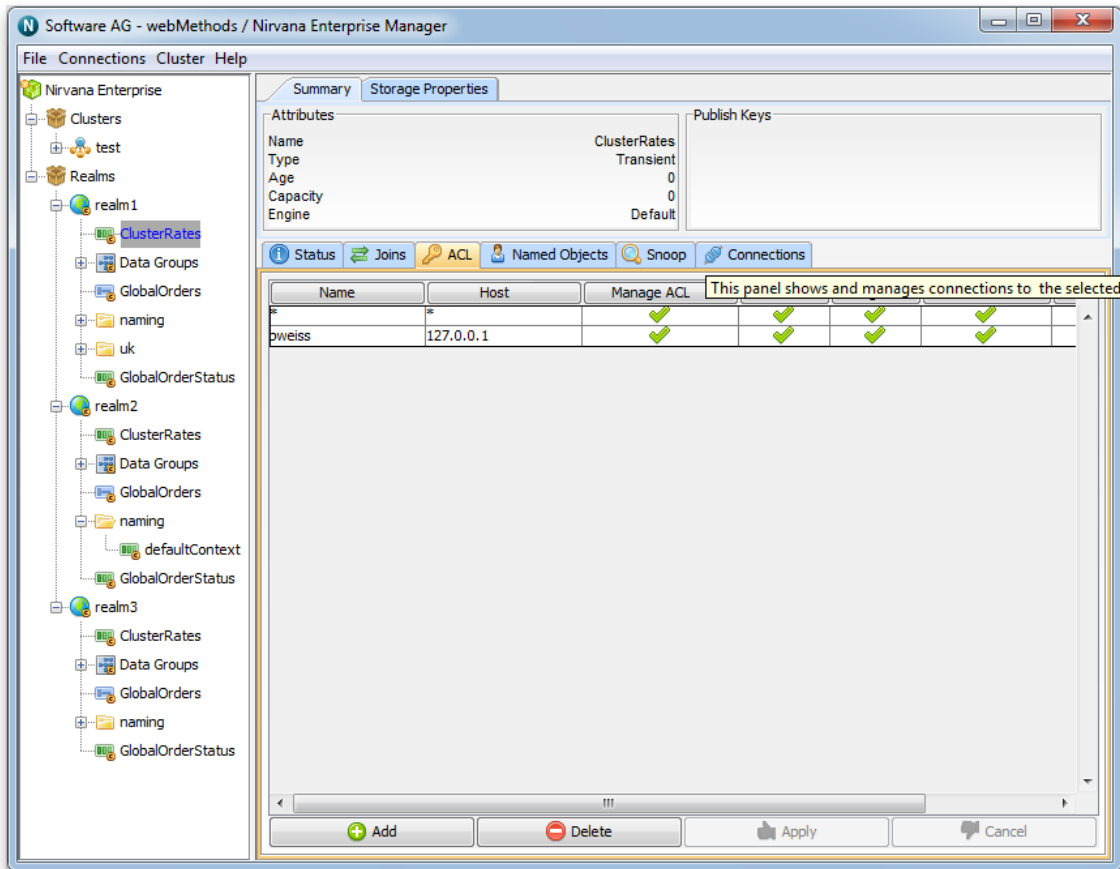
Channel Entitlements

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 manage the list of ACL entries
- 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, the subject `*@*` is shown as having 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 I wanted remove the subscribe permission for the *@* subject I 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.

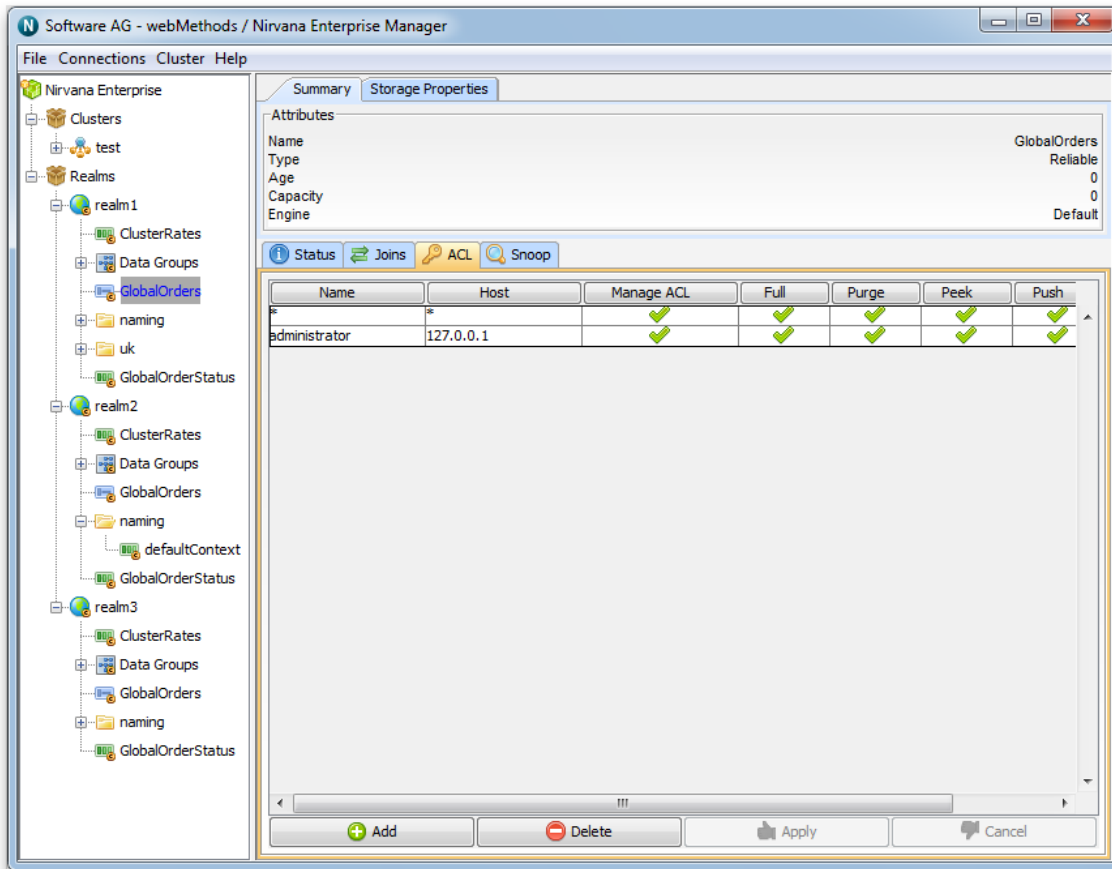
Queue Entitlements

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 an 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 manage the list of ACL entries
- 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.

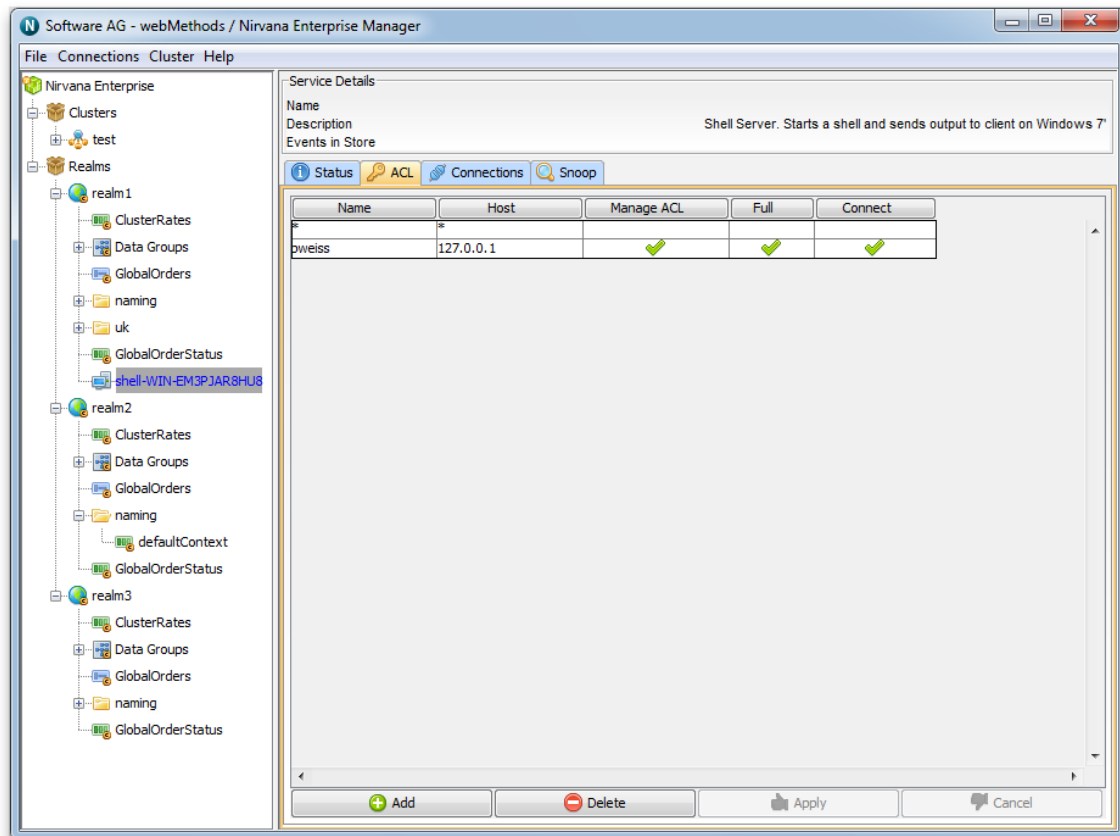
P2P Service

Service 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 service objects, the correct entitlements must be granted to the subject on the required services. Each service has an associated ACL that contains a list of subjects and a set of privileges the subject is given for operations on the service.

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

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



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

- Manage ACL - Allows the subject to get a list of ACL entries
- Full - Has complete access to the secured object
- Connect - Can access this service

The green check icon shows that a subject is permitted to perform the operation. For example, the subject '*@*' is shown as not having any permissions for this service. This means that any client who has successfully established a session will not be allowed to connect to the p2p service unless the subject exists in the ACL.

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 to allow any client to connect to this service I would simply click on the '*@*' row at the column labelled 'connect'. 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 connect to the service.

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 service.

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.

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.

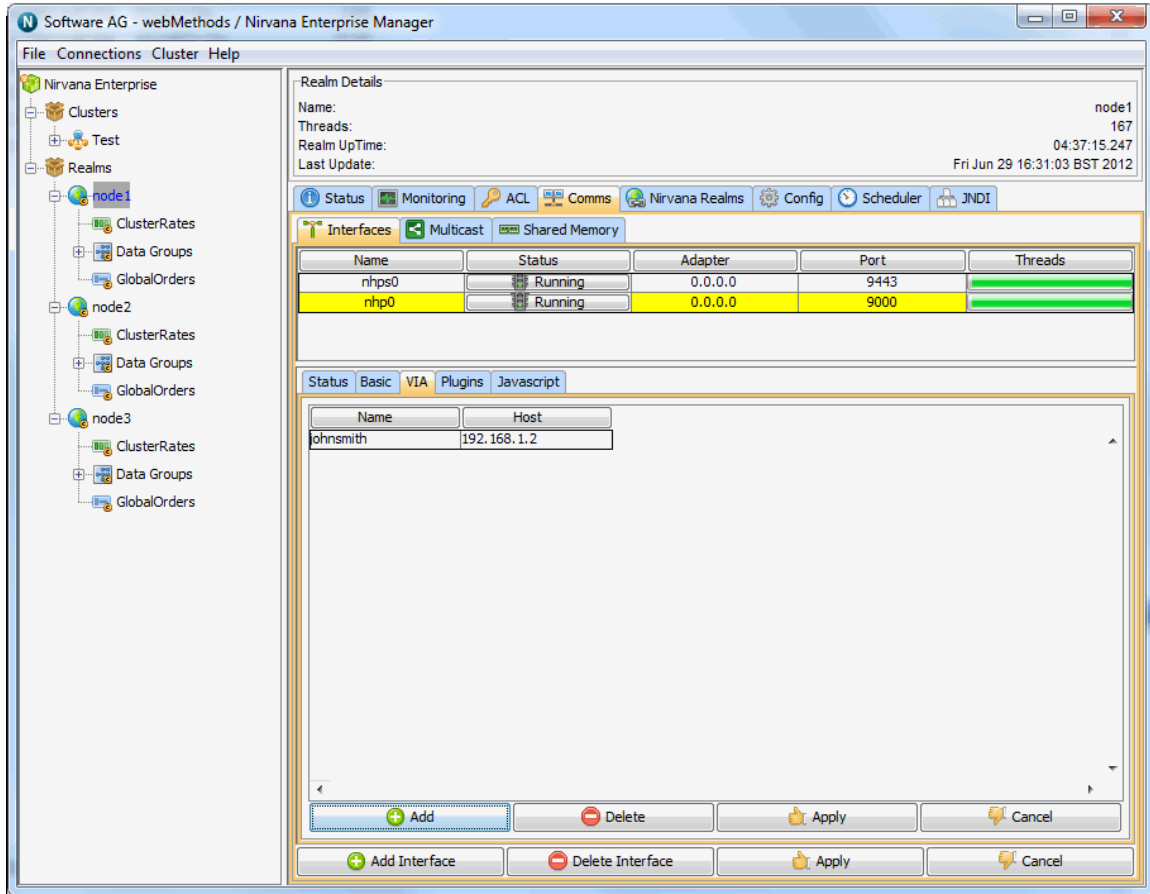
The default behaviour for all interfaces is that when no VIA lists exist on any defined interfaces, all users can connect on any interface (Realm ACLs permitting, see "[Realm Entitlements](#)" on page 125). When a user subject exists on an interface, that subject cannot use any other interface other 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

Universal Messaging provides a sophisticated scheduling engine that enables tasks to be executed 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.

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 will guide you through the basics of Universal Messaging Scheduling. From the links found below you can learn 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.

- ["Writing Scripts" on page 135](#)
- ["Time Based Scheduling" on page 139](#)
- ["Conditional Triggers" on page 141](#)
- ["Tasks" on page 154](#)
- ["The Universal Messaging Enterprise Manager Schedule Editor" on page 167](#)
- ["Example Scheduling Scripts" on page 173](#)

Universal Messaging Scheduling : Writing Schedule Scripts

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 135](#)
- ["Declarations" on page 137](#)
- ["Initial Tasks" on page 138](#)
- ["Every Clause" on page 138](#)
- ["When Clause" on page 139](#)
- ["Else Clause" on page 139](#)

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 "[Universal Messaging Scheduling : Conditional Triggers](#)" on page 141.
- TRIGGER_ARGUMENT_LIST ::= Valid comma separated list of arguments for the trigger
- TASK_DECLARATION ::= Valid task. Learn more about tasks at "[Universal Messaging Scheduling : Tasks](#)" on page 154.
- 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);
myGlobalConfig.StatusUpdateTime(86400000);
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 past 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 173](#).

Universal Messaging Scheduling : Calendar Triggers Schedules

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 ["Universal Messaging Scheduling : Writing Schedule Scripts" on page 135](#)). 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 139](#), ["Daily Triggers" on page 140](#), ["Weekly Triggers" on page 140](#), ["Monthly Triggers" on page 140](#), ["Yearly Triggers" on page 141](#).

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 {
}

```

Universal Messaging Scheduling : Conditional Triggers

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 "[Universal Messaging Scheduling : Writing Schedule Scripts](#)" on page 135). 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 will describe in detail the triggers that are available and how to use them within a trigger expression :

- "[Trigger Expressions](#)" on page 142
- "[Store Triggers](#)" on page 142
- "[Interface Triggers](#)" on page 143
- "[Memory Triggers](#)" on page 144
- "[Realm Triggers](#)" on page 144
- "[Cluster Triggers](#)" on page 145
- "[Counter Triggers](#)" on page 145
- "[Timer Triggers](#)" on page 146
- "[Config Triggers](#)" on page 146

To view examples of scheduling scripts, see "[Scheduler Examples](#)" on page 173.

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 ["Universal Messaging Scheduling : Writing Schedule Scripts" on page 135](#)). 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

Trigger Object	Parameters	Description
usedSpace		store (used space - size of all purged events) 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

Trigger Object	Parameters	Description
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)
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
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.isOnline("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 (below example refers to the 'GlobalValues' configuration group):

```
declare Config myGlobal ("GlobalValues");
```

The table below lists those 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
GlobalValues		
SchedulerPoolSize	None	The number of threads assigned to the scheduler
MaxNoOfConnections	None	Sets the maximum concurrent connections to the server, -1 indicates no restriction
StatusUpdateTime	None	The number of ms between status events being written to disk
StatusBroadcast	None	The number of ms between status events being published

Trigger Object	Parameters	Description
fLoggerLevel	None	The server logging level
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
HandshakeTimeout	None	The number of milliseconds that the server will wait for the session to be established
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)
ServerTime	None	Allow the server to send the current time to the clients (true/false)
SecureHandshake	None	Performs a security handshake when connecting into a cluster
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)

Trigger Object	Parameters	Description
AuditSettings		
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
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

Trigger Object	Parameters	Description
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
ClientTimeoutValues		
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

Trigger Object	Parameters	Description
		and to push events before notifying listeners
ClusterConfig		
HeartBeatInterval	None	Heart Beat interval in milliseconds
SeparateLog	None	Create a separate log file for cluster events
EventsOutStanding	None	Number of events outstanding
EventStorage		
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
FanoutValues		
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

Trigger Object	Parameters	Description
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
OutputBlockSize	None	The size of the output block size
PublishDelay	None	How long to delay the publisher when subscribers queue start to fill, in milliseconds
RoundRobinDelivery	None	Use a round robin approach to event delivery (true/false)
PublishExpiredEvents	None	Publish expired events at server startup (true/false)
JVMManagement		
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

Trigger Object	Parameters	Description
ExitOnMemoryError	None	If true, the server will exit if it gets an out of memory exception
ExitOnDiskIOError	None	If true, the server will exit if it gets a I/O Exception
JoinConfig		
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
RecoveryDaemon		
ThreadPool	None	Number of threads to use for client recovery
EventsPerBlock	None	The number of events to send in one block
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

Trigger Object	Parameters	Description
TTLThreshold	None	The minimum time in milliseconds, below which the server will not store the Transaction ID

Universal Messaging Scheduling : Tasks

Tasks are executed by either time based (calendar, see "[Universal Messaging Scheduling : Calendar Triggers Schedules](#)" on page 139) or conditional triggers (see "[Universal Messaging Scheduling : Conditional Triggers](#)" on page 141). 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 154](#)
- ["Store Tasks" on page 155](#)
- ["Interface Tasks" on page 156](#)
- ["Memory Tasks" on page 157](#)
- ["Counter Tasks" on page 157](#)
- ["Timer Tasks" on page 158](#)
- ["Config Tasks" on page 158](#)

To view examples of scheduling scripts, see "[Scheduler Examples](#)" on page 173.

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 writing scripts help file (see "[Universal Messaging Scheduling : Writing Schedule Scripts](#)" on page 135). 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	<pre>Store.maintain("*"); Store.maintain("/customer/sales"); myChannel.maintain();</pre>	Perform maintenance on a channel so that any purged events are removed from the channel or queue event store.
publish	<pre>myChannel.publish("Byte array data", "tag", "key1=value1:key2=value2");</pre>	Publish an event to the channel / queue, using the given byte array, event tag and event dictionary values.
purge	<pre>myChannel.purge(); myChannel.purge(0, 100000); myChannel.purge(0, 10000, "key1 = 'value1'");</pre>	Purge all events on a channel, or events between a start and end eid, or using a purge filter.
createChannel	<pre>myChannel.createChannel(0, 0, "P");</pre>	Create the channel using the name it was declared as, and the ttl, capacity and type specified in the parameters
createQueue	<pre>myChannel.createQueue(0, 0, "P");</pre>	Create the queue using the name it was declared as, and 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 clients using the admin API.
certificateValidation	myNHP.certificateValidation("true"); myNHP.certificateValidation("false");	Set whether an interface (SSL)

Task Object	Syntax	Description
threads	<pre>myNHP.threads(10); myNHP.threads("+10");</pre>	<p>requires clients to provide a certificate to authenticate.</p> <p>Set the interface accept threads to a value or increase / decrease it by a value.</p>

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	<pre>mem.flush(true); mem.flush(false);</pre>	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

Task Object	Syntax	Description
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 'GlobalValues' configuration group):

```
declare Config myGlobal ("GlobalValues");
declare Config myAudit ("AuditSettings");
declare Config myClientTimeout ("ClientTimeoutValues");
declare Config myCluster ("ClusterConfig");
declare Config myEventStorage ("EventStorage");
declare Config myFanout ("FanoutValues");
declare Config myJVM ("JVMMangement");
declare Config myJoinConfig ("JoinConfig");
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
GlobalValues		
SchedulerPoolSize	myGlobal.SchedulerPoolSize(2);	The number of threads assigned to the scheduler
MaxNoOfConnections	myGlobal.MaxNoOfConnections(-1);	Sets the maximum concurrent connections to the server, -1 indicates no restriction
StatusUpdateTime	myGlobal.StatusUpdateTime(60000);	The number of ms between status events being written to disk
StatusBroadcast	myGlobal.StatusBroadcast(2000);	The number of ms between status events being published
fLoggerLevel	myGlobal.fLoggerLevel(4);	The server logging level
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
HandshakeTimeout	myGlobal.HandshakeTimeout(12000);	The number of milliseconds that the server will wait for the session to be established
StampDictionary	myGlobal.StampDictionary(true);	Place Universal Messaging details

Trigger Object	Syntax	Description
		into the dictionary (true/false)
ExtendedMessageSelector	myGlobal.ExtendedMessageSelector (true);	If true, allows the server to use the extended message selector syntax (true/false)
ServerTime	myGlobal.ServerTime(true);	Allow the server to send the current time to the clients (true/false)
SecureHandshake	myGlobal.SecureHandshake(true);	Performs a security handshake when connecting into a cluster
ConnectionDelay	myGlobal.ConnectionDelay(2000);	When the server has exceeded the connection count, how long to hold on to the connection before disconnecting
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)
AuditSettings		
RealmMaintenance	myAudit.RealmMaintenance (false);	Log to the audit file any realm maintenance activity
InterfaceManagement	myAudit.InterfaceManagement (false);	Log to the audit file any interface

Trigger Object	Syntax	Description
		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
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

Trigger Object	Syntax	Description
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
ClientTimeoutValues		
EventTimeout	myClientTimeout.EventTimeout(10000);	The amount of ms the client will wait for a response from the server
DisconnectWait	myClientTimeout.DisconnectWait(30000);	The maximum amount of time to wait when

Trigger Object	Syntax	Description
TransactionLifeTime	myClientTimeout.TransactionLifeTime(10000);	<p>performing an operation when disconnected before throwing session not connected exception</p> <p>The default amount of time a transaction is valid before being removed from the tx store</p>
KaWait	myClientTimeout.KaWait(10000);	<p>The amount of time the client will wait for keep alive interactions between server before acknowledging disconnected state</p>
LowWaterMark	myClientTimeout.LowWaterMark(200);	<p>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</p>
HighWaterMark	myClientTimeout.HighWaterMark(500);	<p>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.</p>

Trigger Object	Syntax	Description
QueueBlockLimit	myClientTimeout.QueueBlockLimit (5000);	The maximum number of milliseconds a queue will have reached HWM before notifying listeners
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
ClusterConfig		
HeartBeatInterval	myCluster.HeartBeatInterval (60000);	Heart Beat interval in milliseconds
SeparateLog	myCluster.SeparateLog(true);	Create a separate log file for cluster events
EventsOutStanding	myCluster.EventsOutStanding (10);	Number of events outstanding
EventStorage		
CacheAge	myEventStorage.CacheAge(360000);	The time in ms that cached events will be kept in memory for

Trigger Object	Syntax	Description
ThreadPoolSize	<code>myEventStorage.ThreadPoolSize(2);</code>	The number of threads allocated to perform the management task on the channels
ActiveDelay	<code>myEventStorage.ActiveDelay(1000);</code>	The time in milliseconds that an active channel will delay between scans
IdleDelay	<code>myEventStorage.IdleDelay(60000);</code>	The time in milliseconds that an idle channel will delay between scans
FanoutValues		
ConcurrentUser	<code>myFanout.ConcurrentUser(5);</code>	The number of client threads allowed to execute concurrently in the server
KeepAlive	<code>myFanout.KeepAlive(60000);</code>	The number of milliseconds between the server will wait before sending a heartbeat
QueueHighWaterMark	<code>myFanout.QueueHighWaterMark(500);</code>	The number of events in a client output queue before the server stops sending events
QueueLowWaterMark	<code>myFanout.QueueLowWaterMark(200);</code>	The number of events in the clients queue before the server resumes sending events

Trigger Object	Syntax	Description
MaxBufferSize	myFanout.MaxBufferSize(1024000);	The maximum buffer size that the server will accept
OutputBlockSize	myFanout.OutputBlockSize(200);	The size of the output block size
PublishDelay	myFanout.PublishDelay(100);	How long to delay the publisher when subscribers queue start to fill, in milliseconds
RoundRobinDelivery	myFanout.RoundRobinDelivery(true);	Use a round robin approach to event delivery (true/false)
PublishExpiredEvents	myFanout.PublishExpiredEvents(true);	Publish expired events at server startup (true/false)

JoinConfig

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
IdleThreadPoolSize	myJoinConfig.IdleThreadPoolSize(2);	The number of threads to manage the idle and reconnection to remote servers

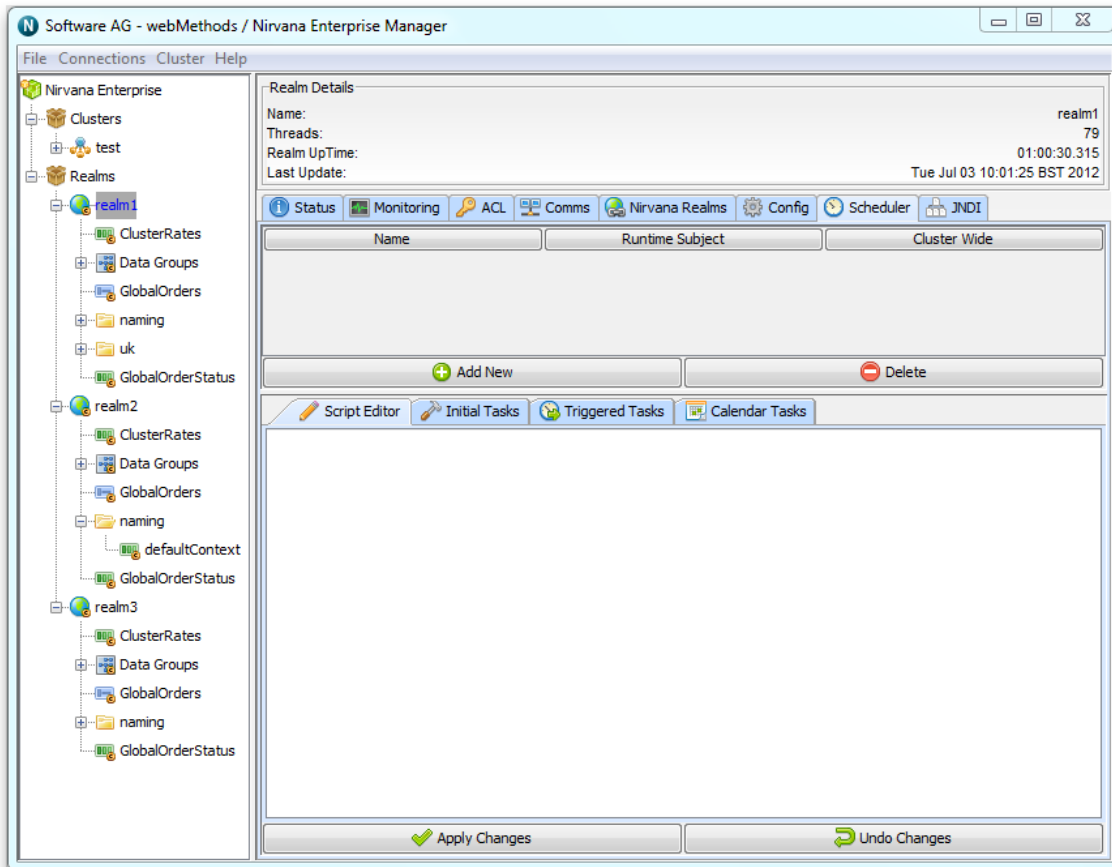
Trigger Object	Syntax	Description
RecoveryDaemon		
ThreadPool	myRecovery.ThreadPool(5);	Number of threads to use for client recovery
EventsPerBlock	myRecovery.EventsPerBlock(300);	The number of events to send in one block
TransactionManager		
MaxTransactionTime	myTXMgr.MaxTransactionTime(1000);	Time 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

Universal Messaging Scheduling: Editor

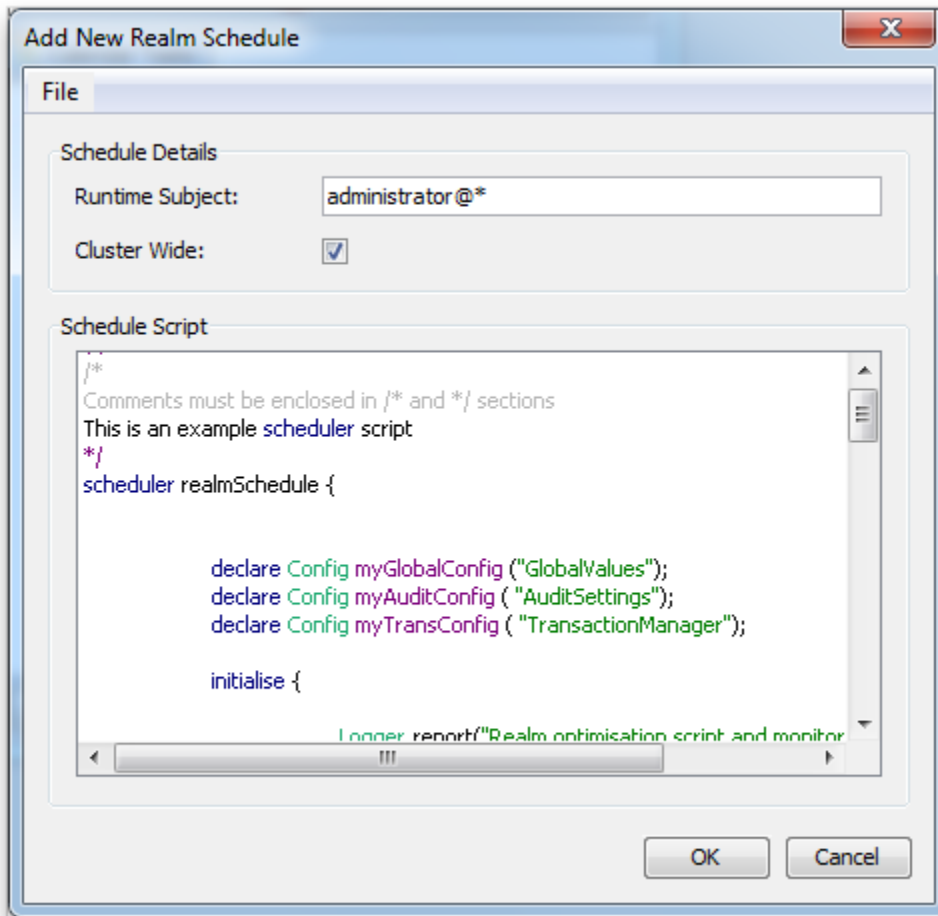
The Universal Messaging Enterprise Manager provides a scheduler panel that enables the user to view, add, delete and edit scheduler scripts. To view the scheduler panel, select the realm from the namespace and click on the 'Scheduler' tab.

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).

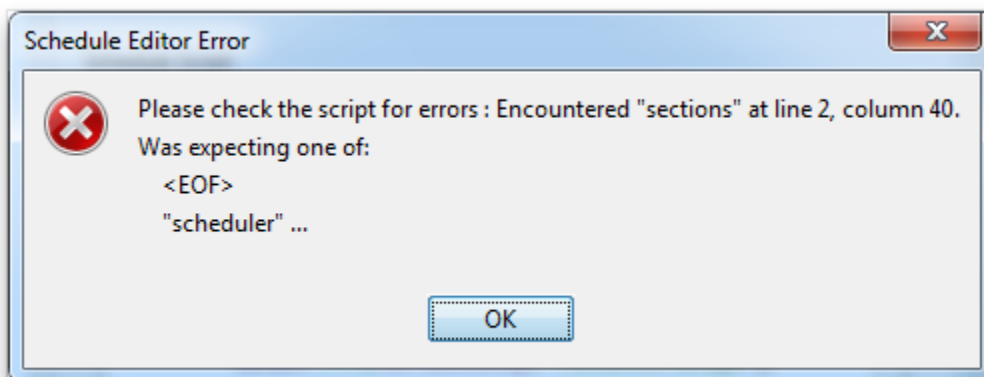
The image below shows the scheduler panel with no scheduling entries.



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 ["Universal Messaging Scheduling : Writing Schedule Scripts" on page 135](#)), as well as the calendar, triggers and tasks sections (see ["Universal Messaging Scheduling : Calendar Triggers Schedules" on page 139](#), ["Universal Messaging Scheduling : Conditional Triggers" on page 141](#) and ["Universal Messaging Scheduling : Tasks" on page 154](#)). 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.

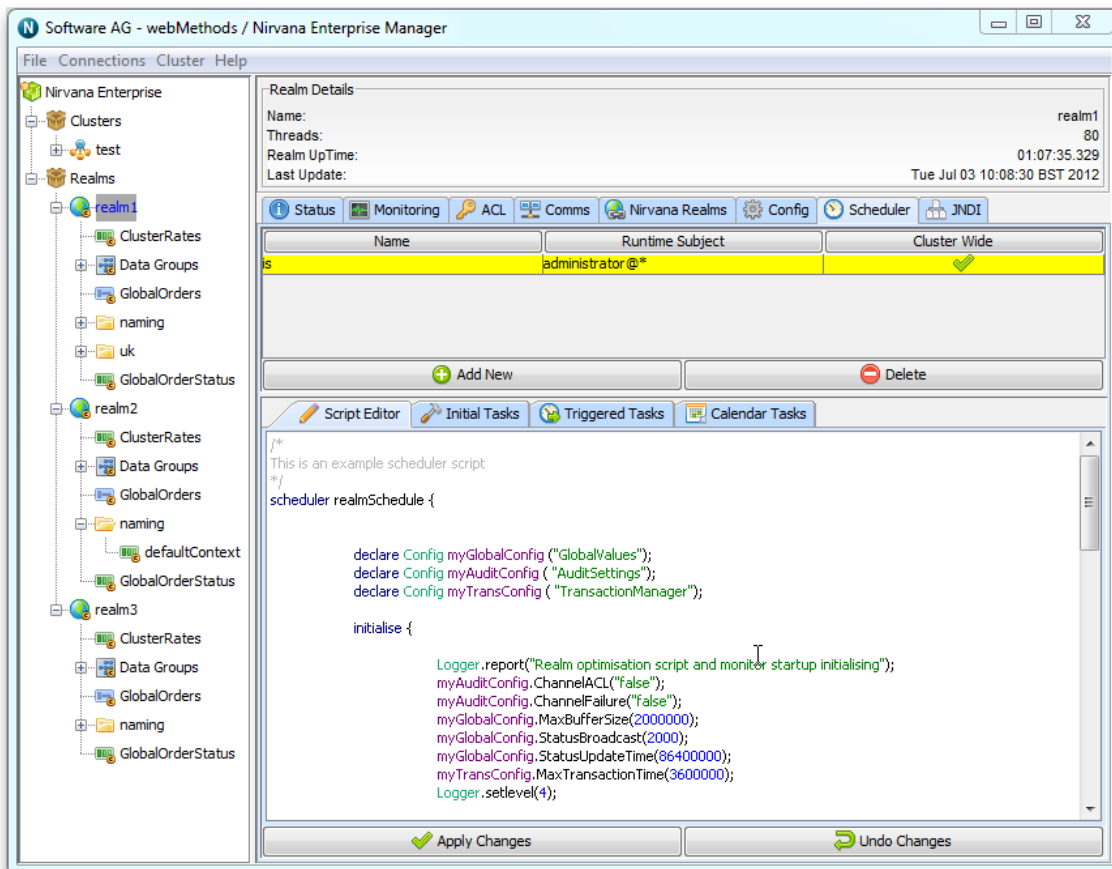


If the script does not contain any errors, the script editor panel will close and the new scheduler script will then appear within the scheduler table. Clicking on the newly created scheduler within the table will enable you to delete, view and edit the schedule.

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.

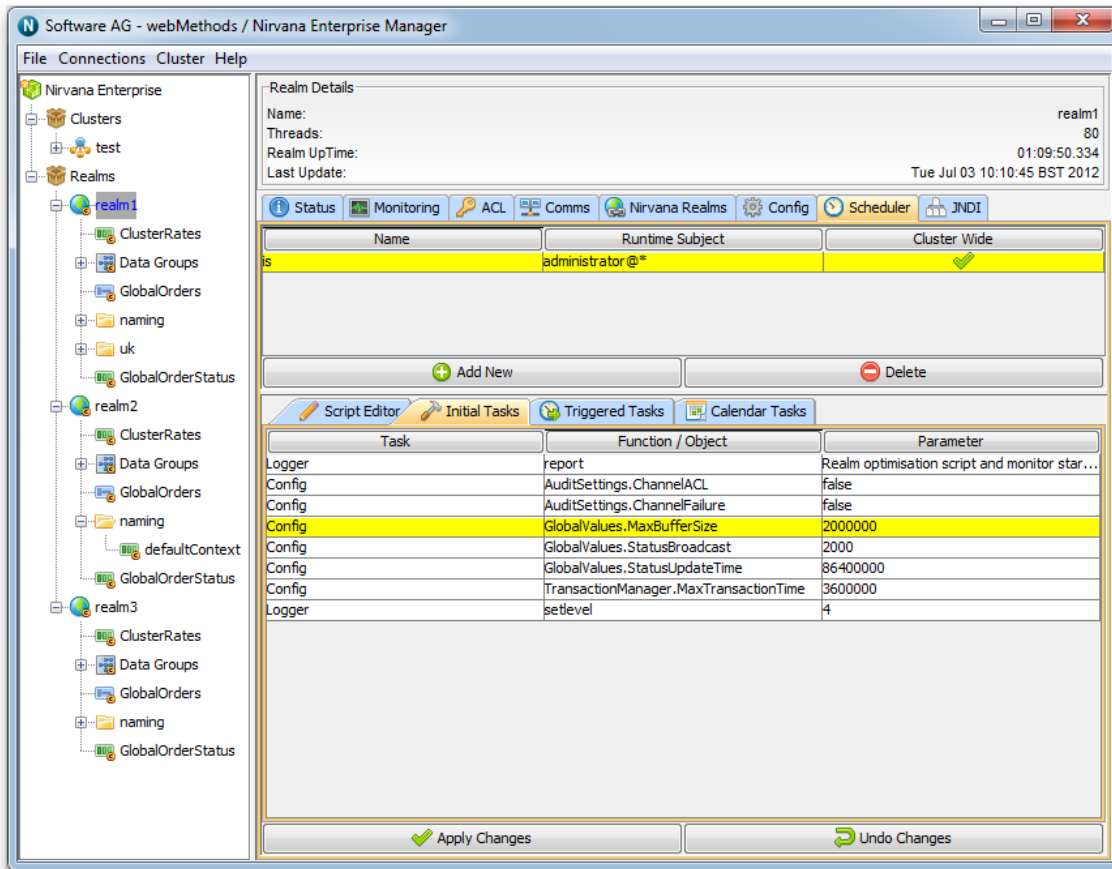
Script Editor Panel

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.



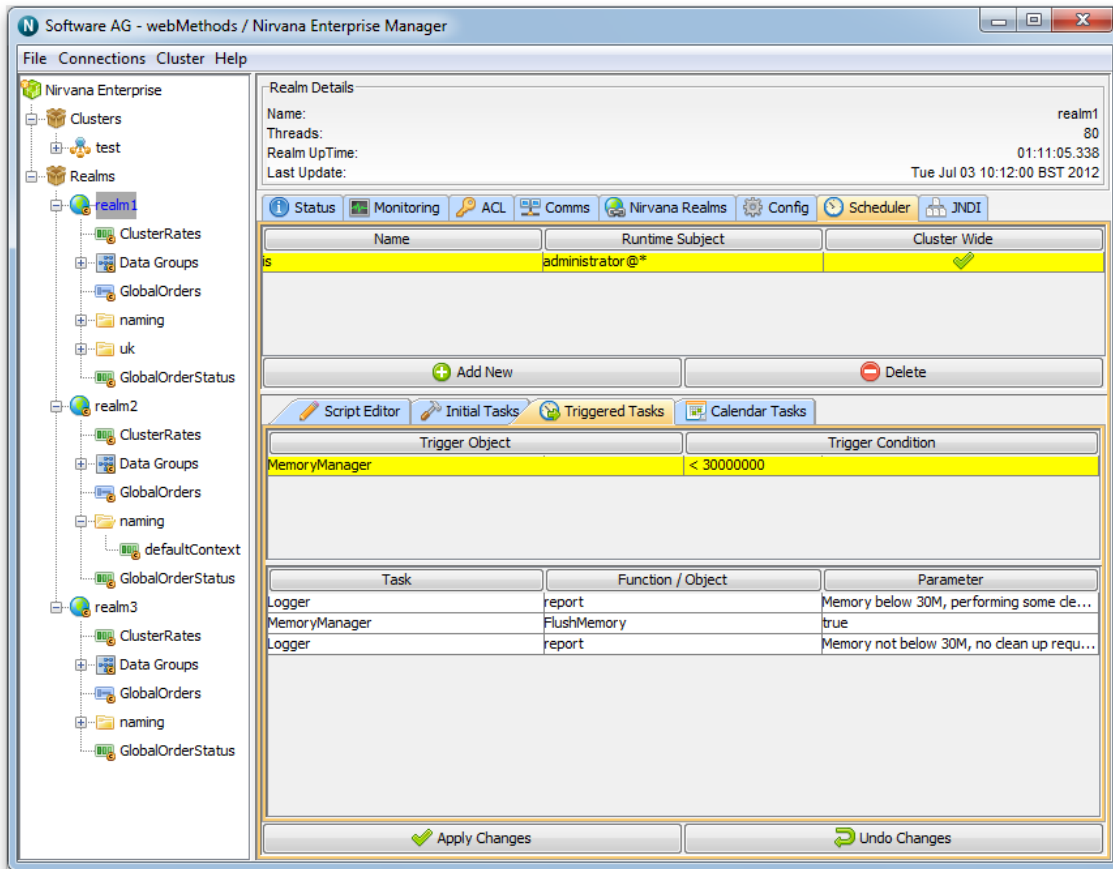
Initial Tasks Panel

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 ["Universal Messaging Scheduling : Tasks" on page 154](#)). Column 2 labelled 'Function / Object' represents the details of the task, for 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.

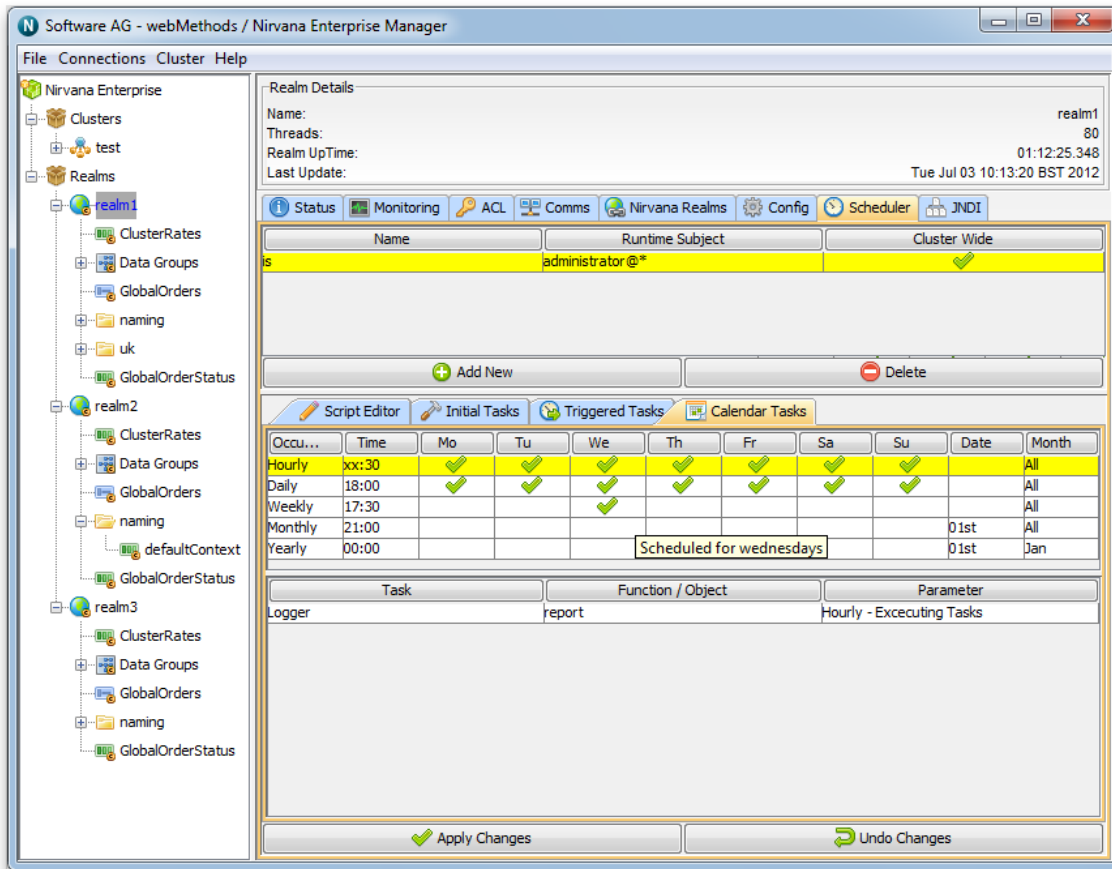


Calendar Tasks

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.

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 174](#)
- ["Store Triggers" on page 174](#)
- ["Interface Triggers" on page 176](#)
- ["Memory Triggers" on page 176](#)
- ["Realm Triggers" on page 176](#)
- ["Cluster Triggers" on page 177](#)
- ["Counter Triggers" on page 177](#)

- ["Timer Triggers" on page 178](#)
- ["Config Triggers" on page 178](#)

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);
    myGlobalConfig.StatusUpdateTime(86400000);
    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.isOnline == true ){
    Logger.report("Realm1 online");
  }
/*
This can also be written as
*/
  when ( Cluster.isOnline("realm1") == true ){
    Logger.report("Realm1 online");
  }
  when ( myNode2.isOnline == true ){
    Logger.report("Realm2 online");
  }
  when ( myNode3.isOnline ==true ){
    Logger.report("Realm3 online");
  }
  when ( myNode4.isOnline == true ){
    Logger.report("Realm4 online");
  }
  when ( Cluster.hasQuorum == true ){
    Logger.report("Cluster now has quorum and is running" );
  }
  when ( Cluster.isMaster == 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 than 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 delay fired, resetting timers");
    testTimer.reset();
    testTimer.stop();
    timer.reset();
}
}

```

Universal Messaging Scheduling : Configuration Example

```

scheduler myConfig {
/*
    Declare a local name for the Config(GlobalValues) config group and call it myGlobal.
    Can be used for both triggers and tasks
*/
    declare Config myGlobal ("GlobalValues");
/*
    When this scheduler task is initialised, set the Realms log level to 2
*/
    initialise{
        myGlobal.fLogLevel(2);
    }
}

```

```

    }
  /*
  Then if the log level is ever set to 0, automatically reset it to 2.
  */
  when(myGlobal.fLogLevel == 0){
    myGlobal.fLogLevel(2);
  }
  /*
  If the maximum number of connections on the realm is less than 0,
  implying no limit, then set it to 100.
  */
  when(myGlobal.MaxNoOfConnections <0){
    myGlobal.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.

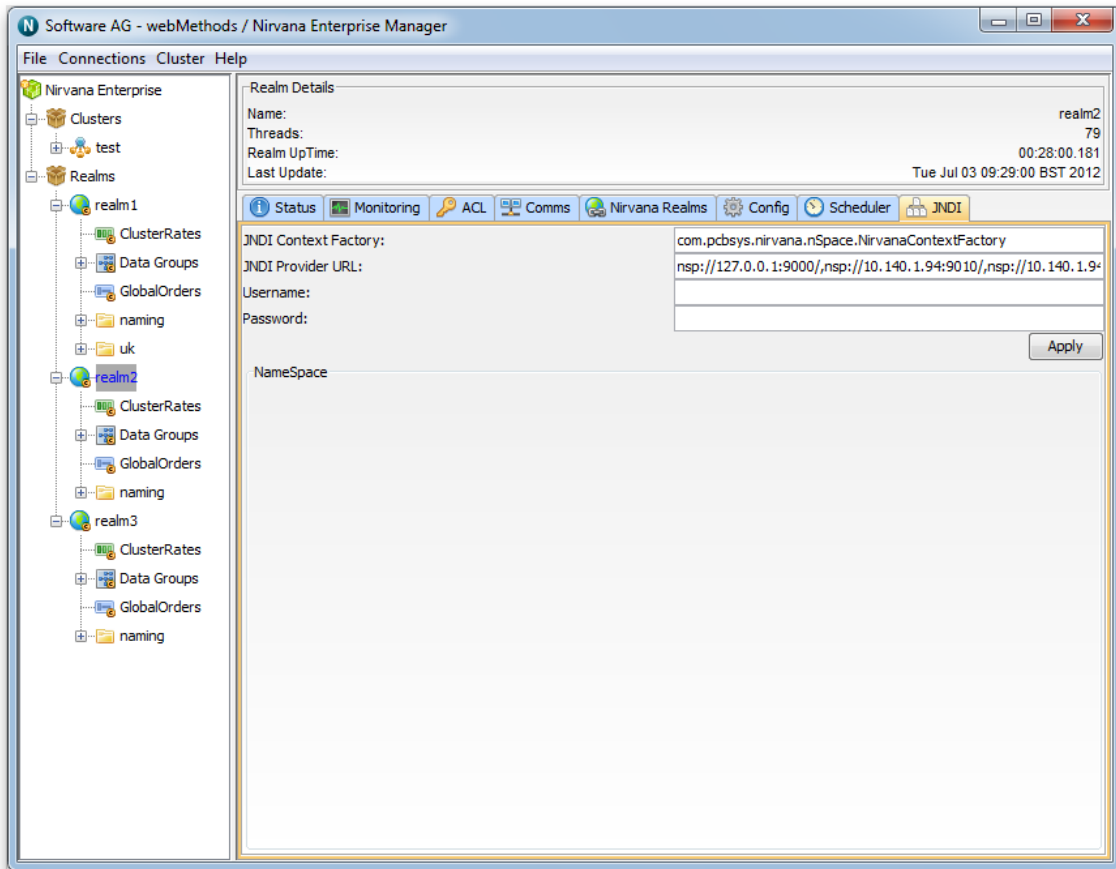
Setting Up the Context and Connection Factories for JNDI

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. 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. At this point there will be nothing drawn within the JNDI NameSpace tree.

The image below shows the default JNDI panel:



The JNDI panel at this point does not contain any JNDI context information (as can be seen by the empty **Namespace** section of the panel).

By default, the 2 text fields labelled **JNDI Context Factory** and **JNDI Provider URL** will already contain information:

- The JNDI Context Factory will default to `com.pcbSYS.nirvana.nSpace.NirvanaContextFactory`, which is the class that provides the Universal Messaging context factory functionality for JNDI.
- The JNDI provider URL is the RNAME used when creating each JNDI connection reference. Note that for a realm server that is part of a cluster, the Provider URL for JNDI will be a comma separated list of the RNAMEs for each realm server that is a member of the cluster. This ensures that not only will the JNDI context be the same within all cluster realms, but also that any JMS client using this Universal Messaging Context Factory will be able to use any of the realm servers specified in the Provider URL.

To actually create the Initial Context, you must click the **Apply** button.

The Initial Context uses the (potentially clustered) Universal Messaging context channel to store all JNDI references. This channel is called `/naming/defaultContext`. The channel is not created when the Initial Context is created; instead, the channel is created when the first client attempts to perform a JNDI bind using the Initial Context. Full

permissions are assigned to this first client 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).

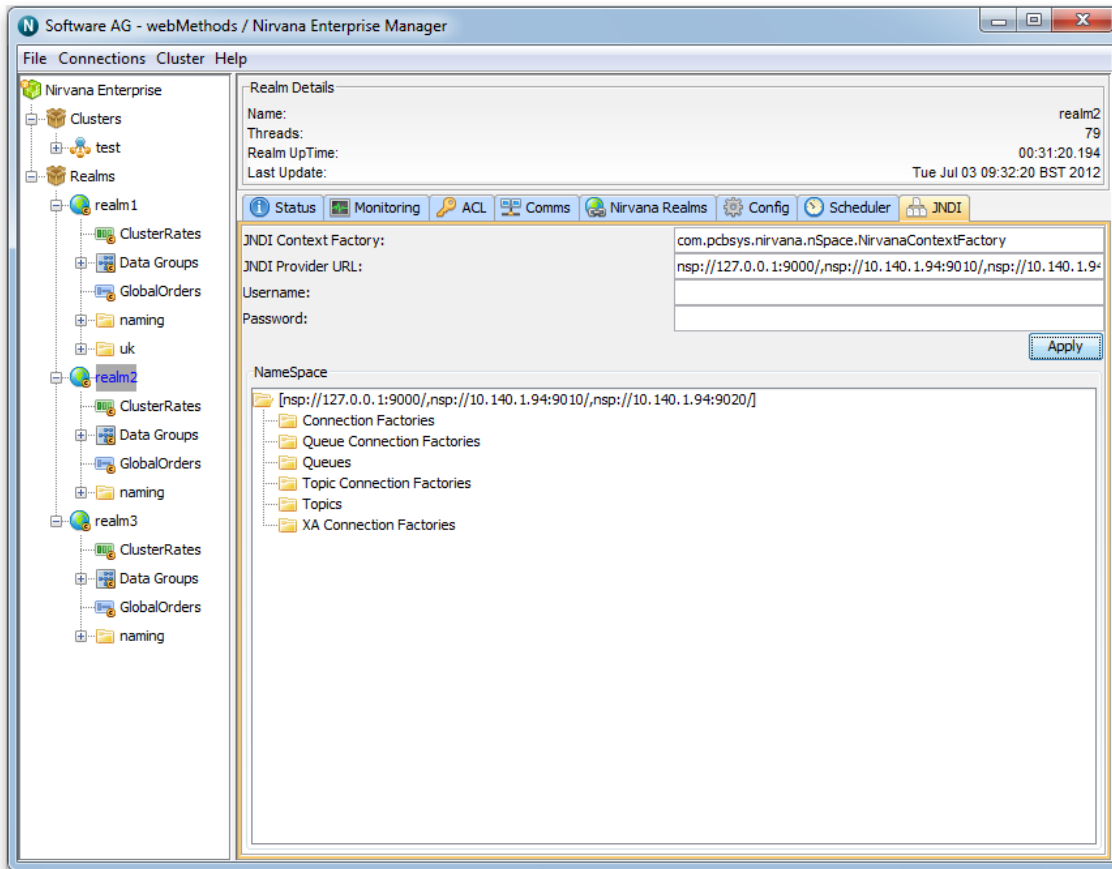
If the Initial Context no longer exists, then clicking **Apply** on this screen will recreate it (though it will not contain any of its previous JNDI entries/references). Subsequently, the **/naming/defaultContext** channel must also be recreated, as described above.

Viewing the JNDI Namespace

Whenever you click **Apply** on 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 comma-separated list of RNAME values for each server in the cluster). Double clicking on the root node in the JNDI namespace will render 6 "folders":

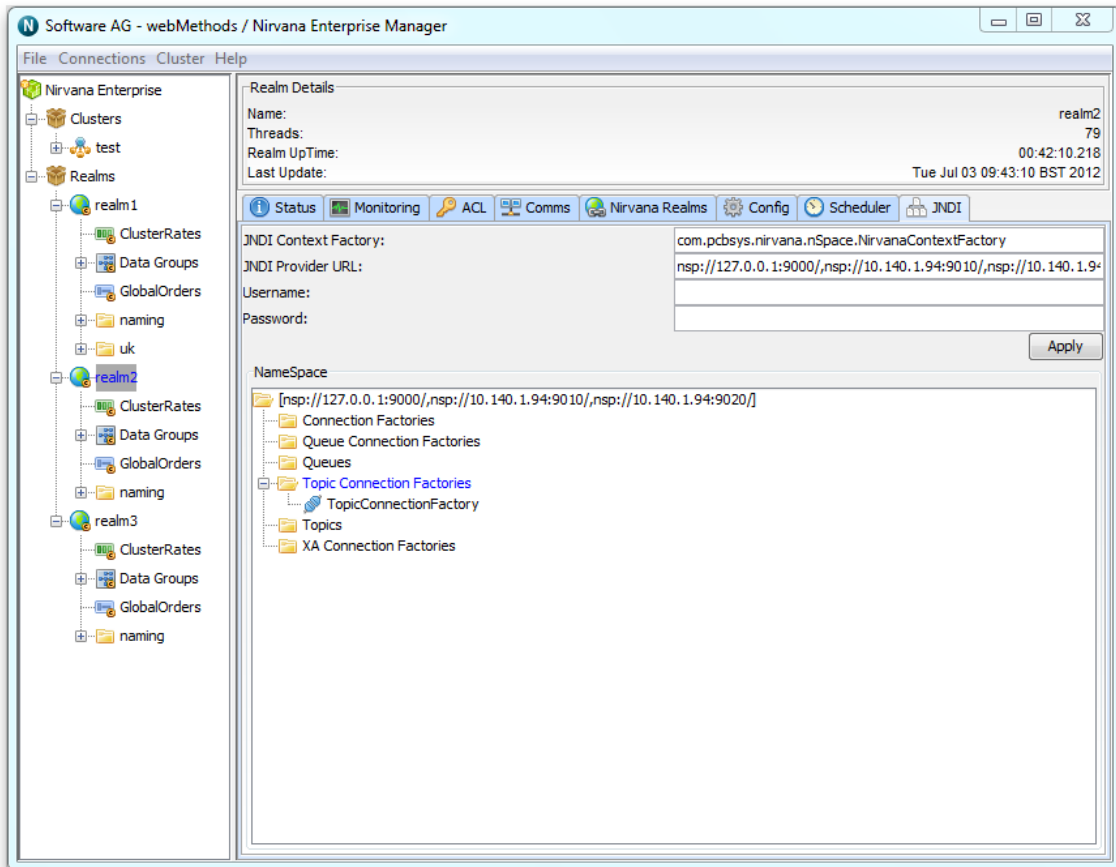
- Connection Factories
- Queue Connection Factories
- Queues
- Topic Connection Factories
- Topics
- XA Connection Factories

The image below shows this view after the **Apply** button has been clicked and the JNDI namespace tree has been double-clicked to expand it:



Creating Topic and Queue Connection Factories

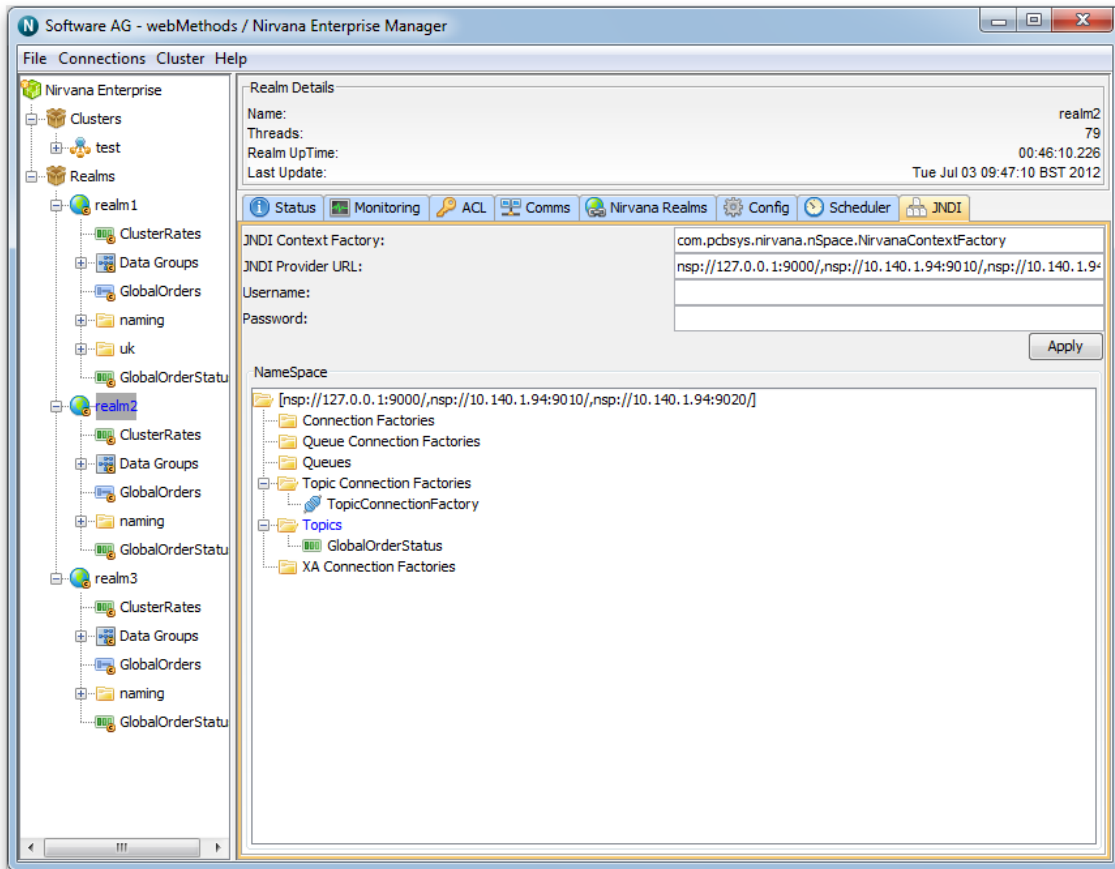
In order to allow JMS clients to use the Universal Messaging Context Factory to reference objects via JNDI, we first of all need to create Topic and Queue connection factories. To do this, right click on the tree node labelled **Topic Connection Factories** and select the menu option **New Topic Connection Factory**. This will display a dialog box allowing you to enter the name for the connection factory. Enter any name (in this example, we will use the name **TopicConnectionFactory**). Click on **OK** when you've entered the name, and you will see that a new node will have been created under the **Topic Connection Factories** folder with the same name as 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.

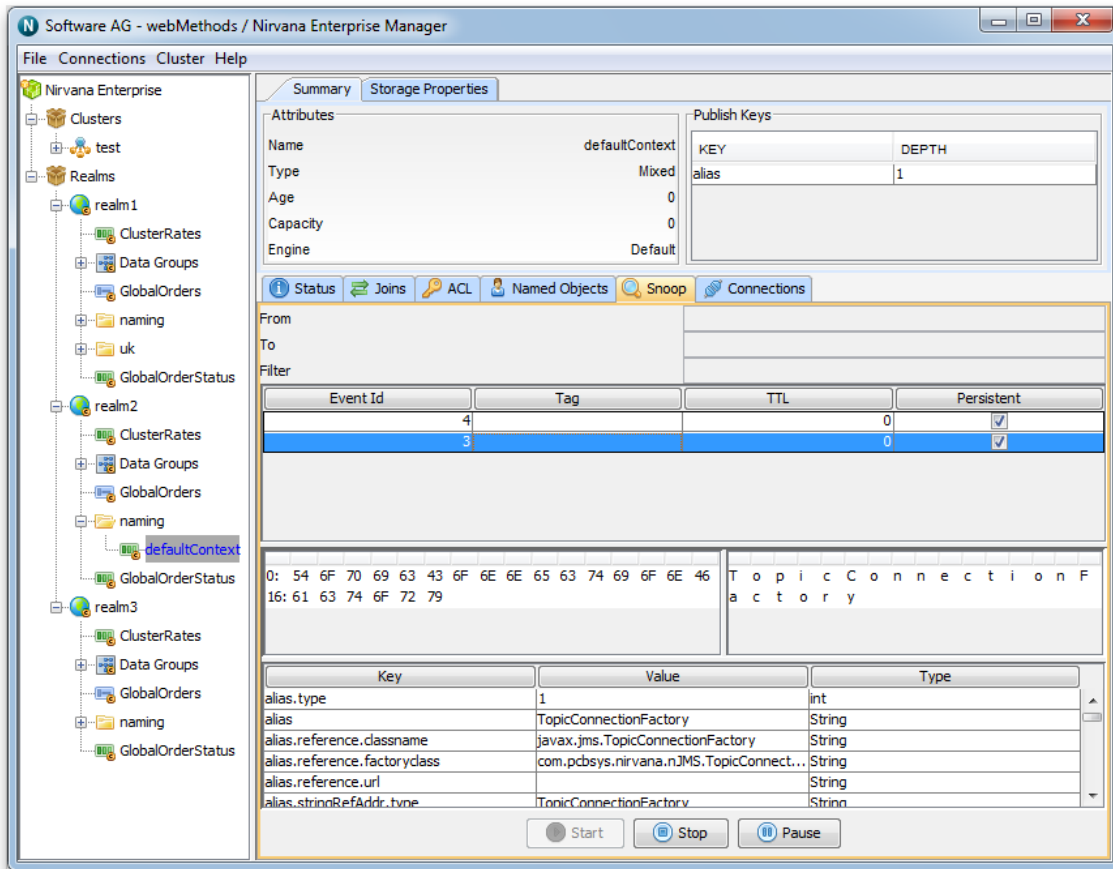
When JMS clients use the Universal Messaging Initial Context for JNDI, they also reference the topics and queues from the same Initial Context. In order for these clients to access these objects we need to create references to each topic and queue. Creating such references will also create the underlying 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 **/customer/sales**, then a new object will be created under the **Topics** folder called **/customer/sales**. 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 **/customer/sales**:



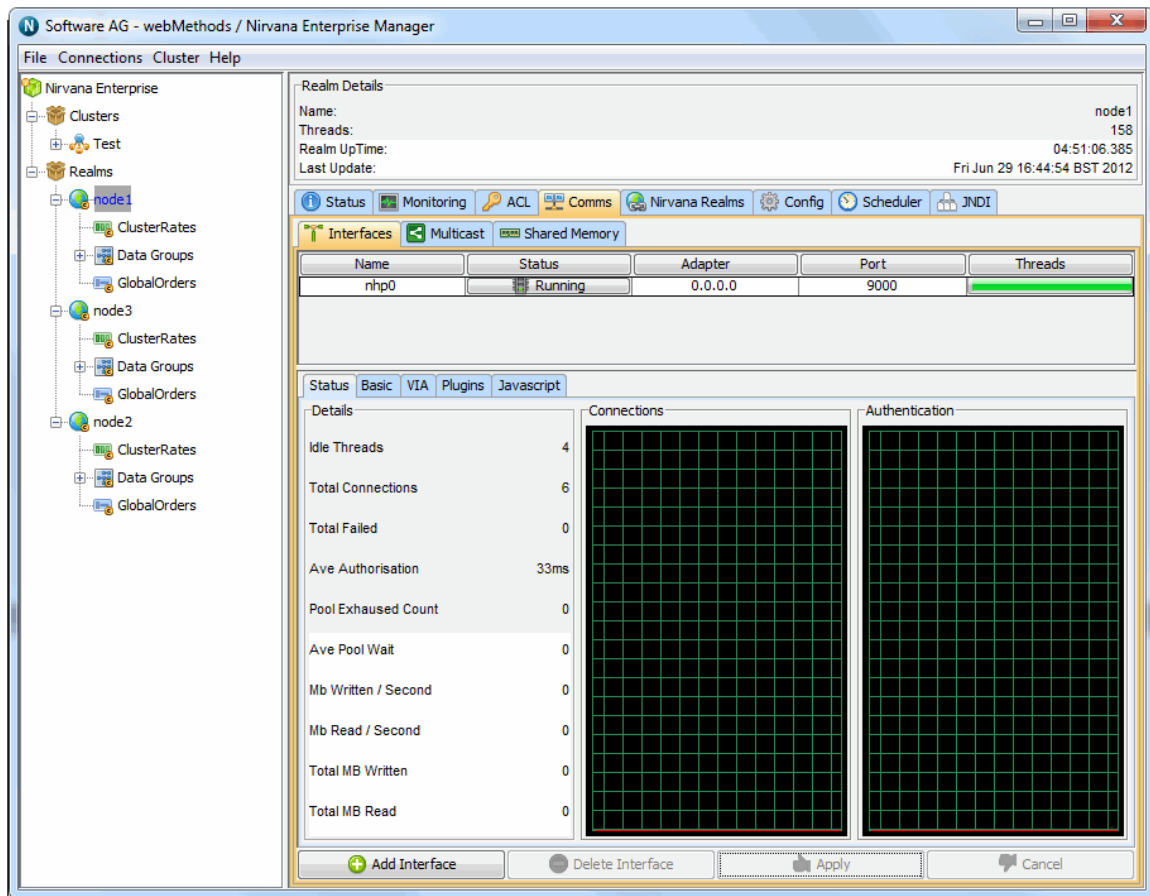
Once we have created both the topic connection factory and the topic, we can snoop (see "[Channel Snoop](#)" on page 93) the `/naming/defaultContext` channel to view the individual events that represent these references. If you click the `/naming/defaultContext` channel within the Enterprise Manager namespace, then the **Snoop** panel, and click **Start**, you will see the events representing the JNDI entries that have been created. By selecting any of the events 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.

The image below shows an example of the Topic Connection Factory created earlier using the JNDI panel:



Universal Messaging Enterprise Manager Comms: TCP Interfaces, IP Multicast and SHM

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 187](#)
- ["Deleting Interfaces" on page 190](#)
- ["Creating SSL Interfaces" on page 191](#)
- ["Stopping Interfaces" on page 191](#)
- ["Starting Interfaces" on page 192](#)
- ["Interface Configuration" on page 192](#)
- ["JavaScript Interface Panel" on page 195](#)
- ["Modify Interfaces" on page 198](#)
- ["Interface Plugins" on page 199](#)
- ["Interface VIA rules" on page 133](#)

IP Multicast

- ["IP Multicast Configuration" on page 334](#)

Shared Memory (SHM)

- ["Shared Memory Configuration" on page 206](#)

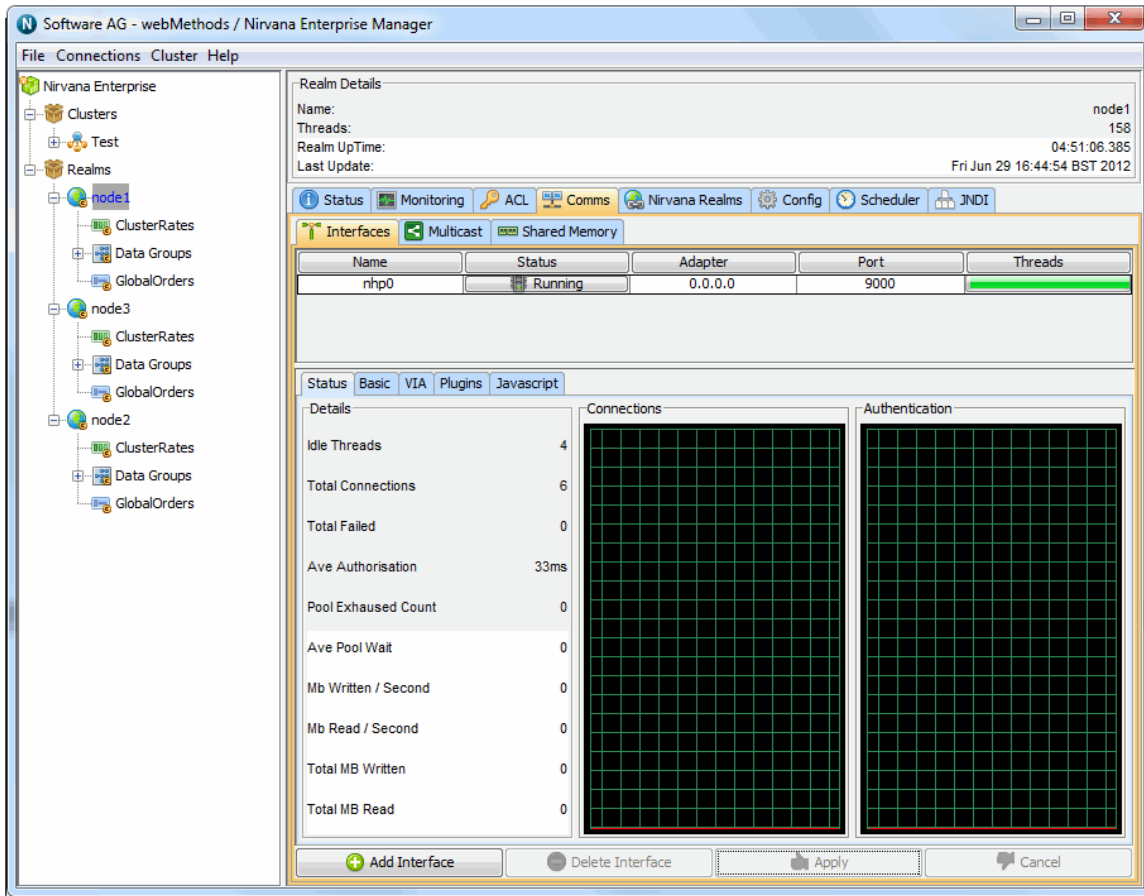
Creating Interfaces

In order to create an interface (see ["Universal Messaging Enterprise Manager Comms: TCP Interfaces, IP Multicast and SHM" on page 185](#)), you need to select the realm node from the namespace tree that you want to create an interface for. Once this node is selected, there will be a tab in the tabbed pane on the right hand side of the Enterprise Manager labelled 'Interfaces'. Selecting this tab will present the user with 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.

Please note that adding an SSL enabled interface (see ["Creating an SSL network interface to a Universal Messaging Realm server" on page 208](#)) for either SSL enabled sockets or HTTPS requires some additional steps.

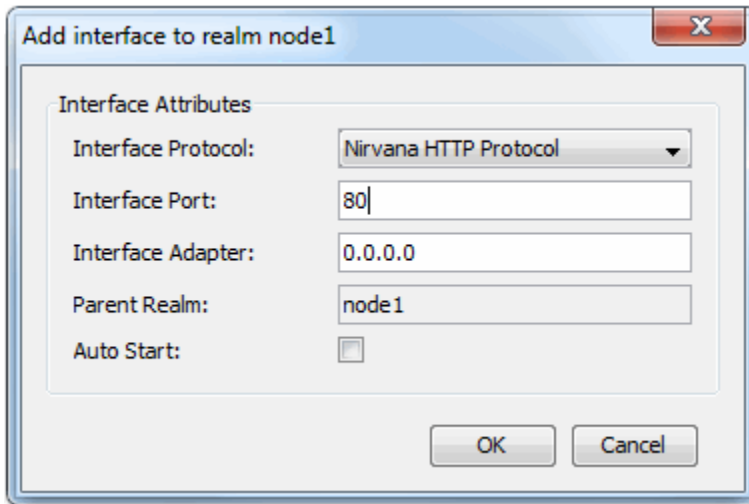
The image below shows the interfaces tab containing the default realm interface. When selected, an interface will be highlighted in yellow as shown below.



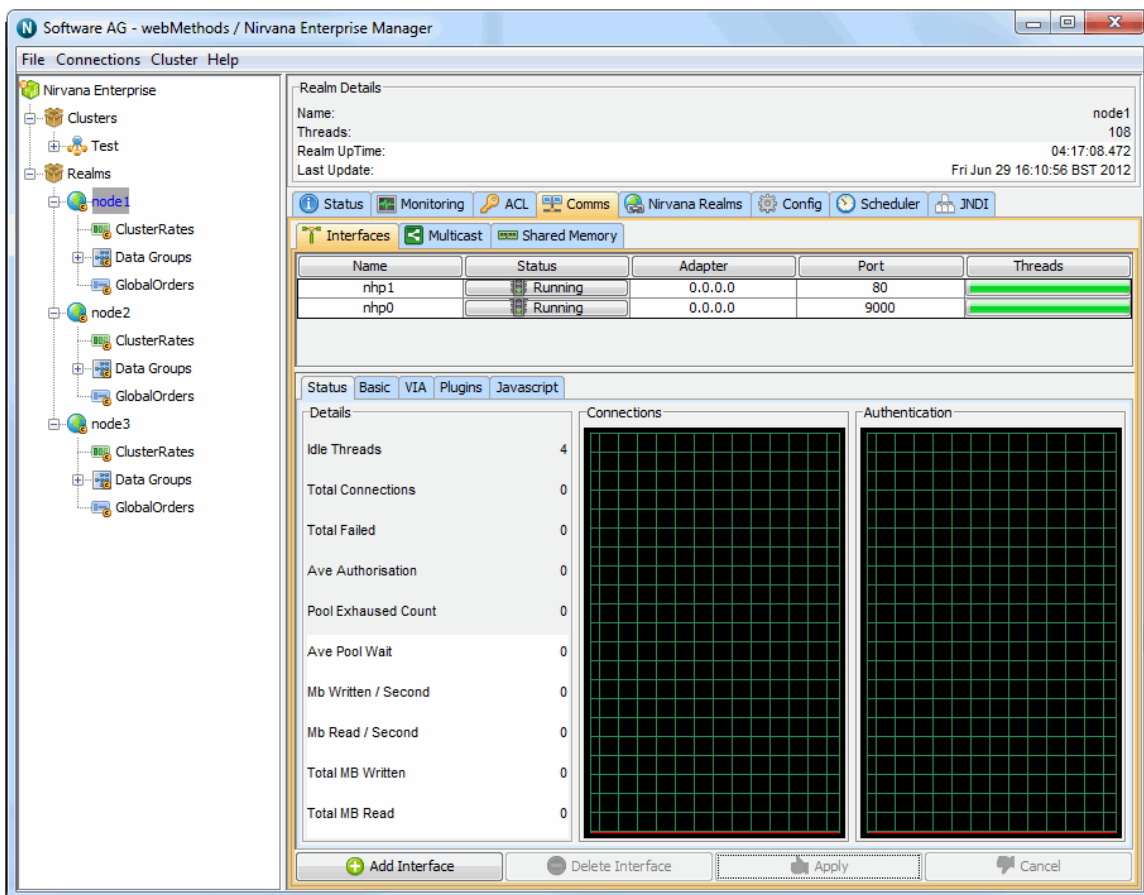
The interfaces table consists of 5 columns, each of which is described below:

- **Name** : defined as protocol + n, where n is a unique number of interfaces for that protocol
- **Status** : whether the interface is 'Running', 'Stopped' or 'Error' where the interface has not been started due to an error
- **Adapter** : the physical network interface to bind to, 0.0.0.0 defines all known interfaces
- **Port** : the port to bind to
- **Threads** : indicator for the number of accept threads the interface has free to accept connections, full green denotes all are free

To add a new interface, simply 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. With the 'Auto Start' checked as above, clicking on the 'OK' button 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 interfaces has been created it will appear in the interfaces table as shown in the image below.

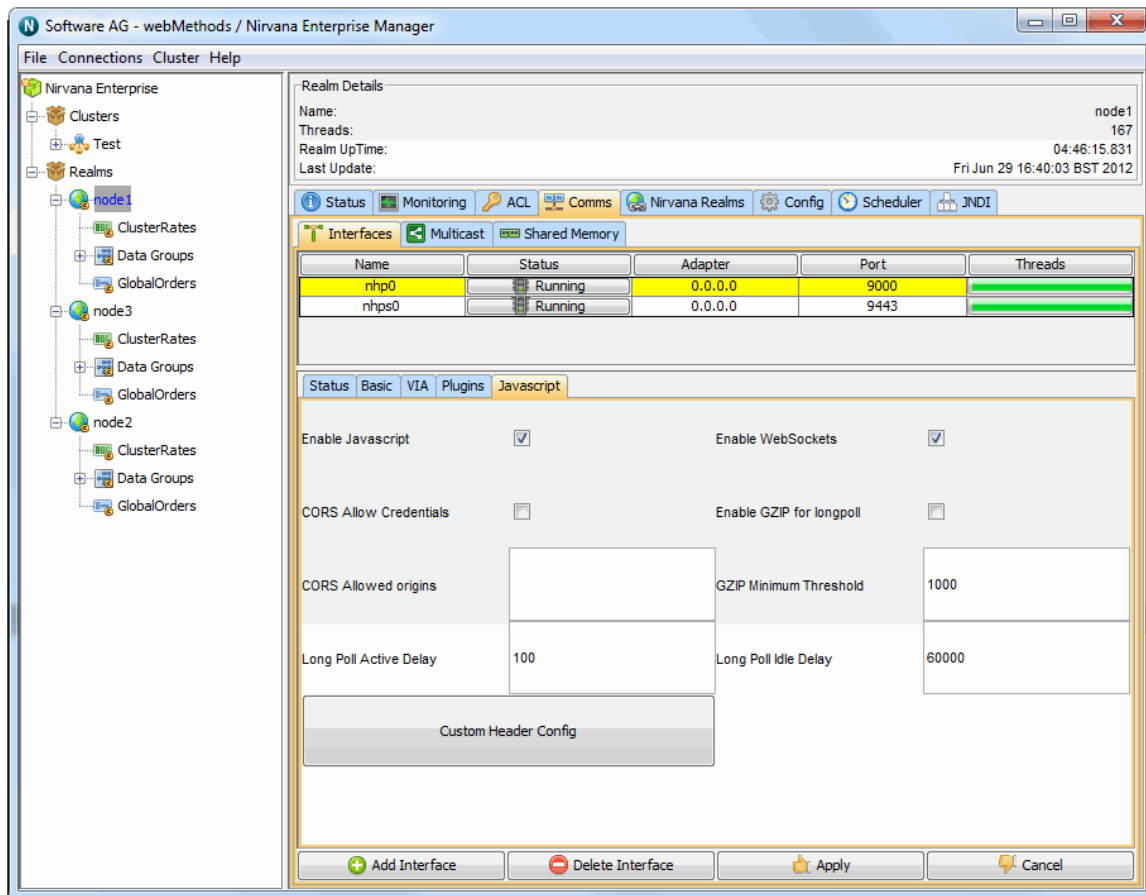


Further instructions on configuring Universal Messaging interfaces are also available in the enterprise manage guide (see "[Interface Configuration](#)" on page 192).

In addition a VIA rule (see "[Interface VIA Rules](#)" on page 133) can be added to interfaces as a security enhancement.

HTTP / HTTPS Interface

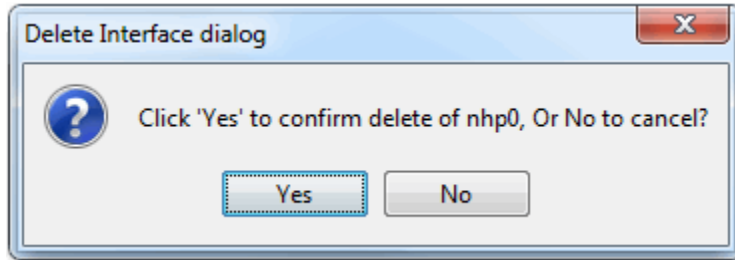
The Javascript tab allows configuration of Comet delivery and is available for HTTP / HTTPS (nhp / nhps) interfaces.



Deleting Interfaces

Interfaces can be deleted by simply selecting the realm node where the interface you want to delete is running, and select 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.

This will prompt you with a question to confirm the deletion of the selected interface. Clicking yes will stop the interface, closing all clients connected to the interface, then it will remove the interface from the realm. The image below shows the confirm dialog.



SSL Interfaces

Universal Messaging supports ssl encryption by providing 2 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 JVMs JSSE provider. This ensures that any unauthorised connections are SSL authenticated prior to any Universal Messaging specific operations can be performed.

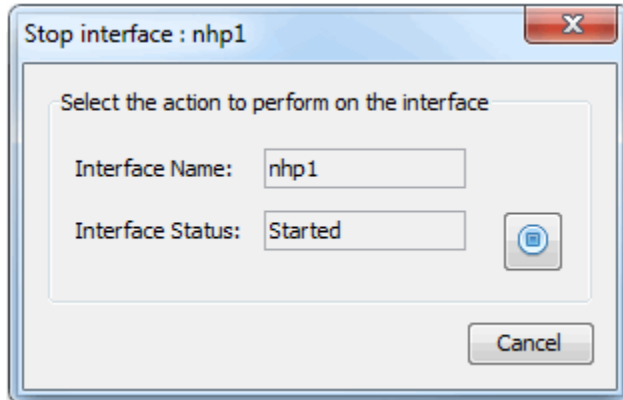
Creating an ssl enabled interface is the same as creating a non-ssl interface (see "[Creating Interfaces](#)" on page 187) except there are a number of ssl related attributes in addition to the basic attributes (see "[Interface Configuration](#)" on page 192).

For information on how to create an ssl interface (see "[Creating an SSL network interface to a Universal Messaging Realm server](#)" on page 208) using the Universal Messaging enterprise manager, please see the Universal Messaging FAQ.

Stopping Interfaces

Interfaces can be stopped by selecting the realm node where the interface you want to stopped is running, and select the 'Interfaces' tab. From the table of configured interfaces, select the interface you want to stop and double-click on the row.

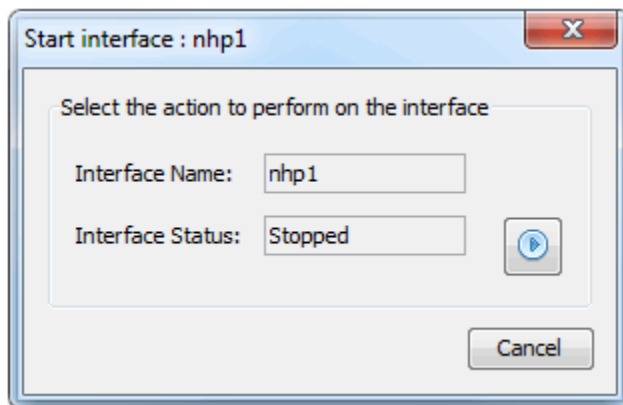
Alternatively, you can click on the 'Status' column for the interface. Both will have the same effect, and will present you with a dialog for stopping the selected interface. This dialog is shown in the image below.



Clicking on the 'Stop' button, will stop the interface on the realm server.

Starting Interfaces

Interfaces can be started by selecting the realm node where the interface you want to start is running, and select 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 have the same effect, and will present you with a dialog for starting the selected interface. This dialog is shown in the image below.



Clicking on the 'Start' button, will start the interface on the realm server.

Interface Configuration

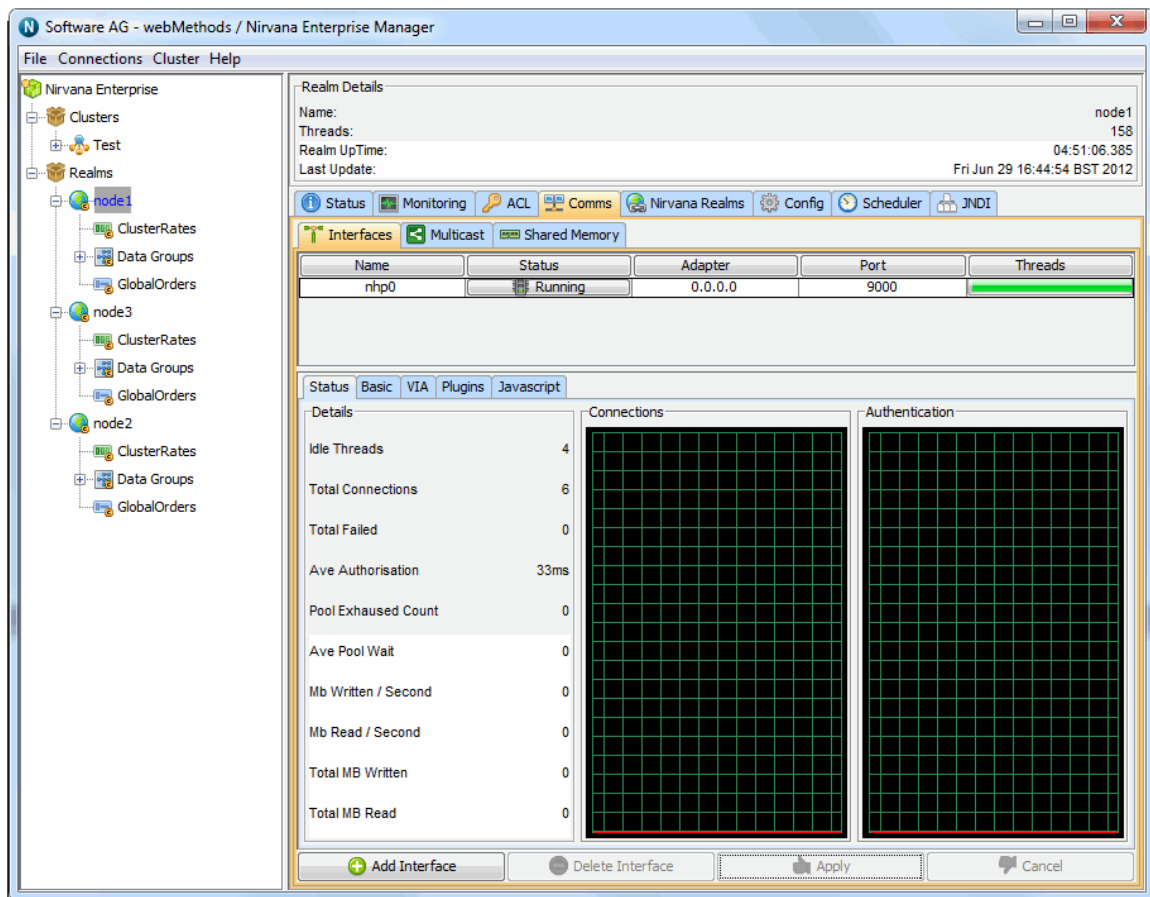
Each interface on a Universal Messaging Realm has a number of configurable attributes that determine the interface behaviour. Some of these attributes are standard across all types of interface protocols, and some are specific to the actual protocol.

This section will describe those attributes that are common for interfaces of all types.

For additional information on specific interfaces types, see "[Universal Messaging Enterprise Manager Comms: TCP Interfaces, IP Multicast and SHM](#)" on page 185.

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 are a set of 2 or more tabs, one of which is labelled 'Basic', as shown in the image below.



The basic interface configuration panel shows 6 configurable attributes. These are each explained in the following section:

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.

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.

Send Buffer size

This specifies the size of the send buffer on the socket.

Receive Buffer size

This specifies the size of the receive buffer on the socket

Select Threads

The select thread 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. While 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 passed on to the write thread pool. The number of select threads should not typically exceed the number of cores available.

Enable NIO

Specify whether NIO should be used for this interface.

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.

Backlog

The backlog attribute specifies the size of the IP socket queue. This value specifies the maximum size of the incoming socket request queue. The operating system 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.

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 199](#).

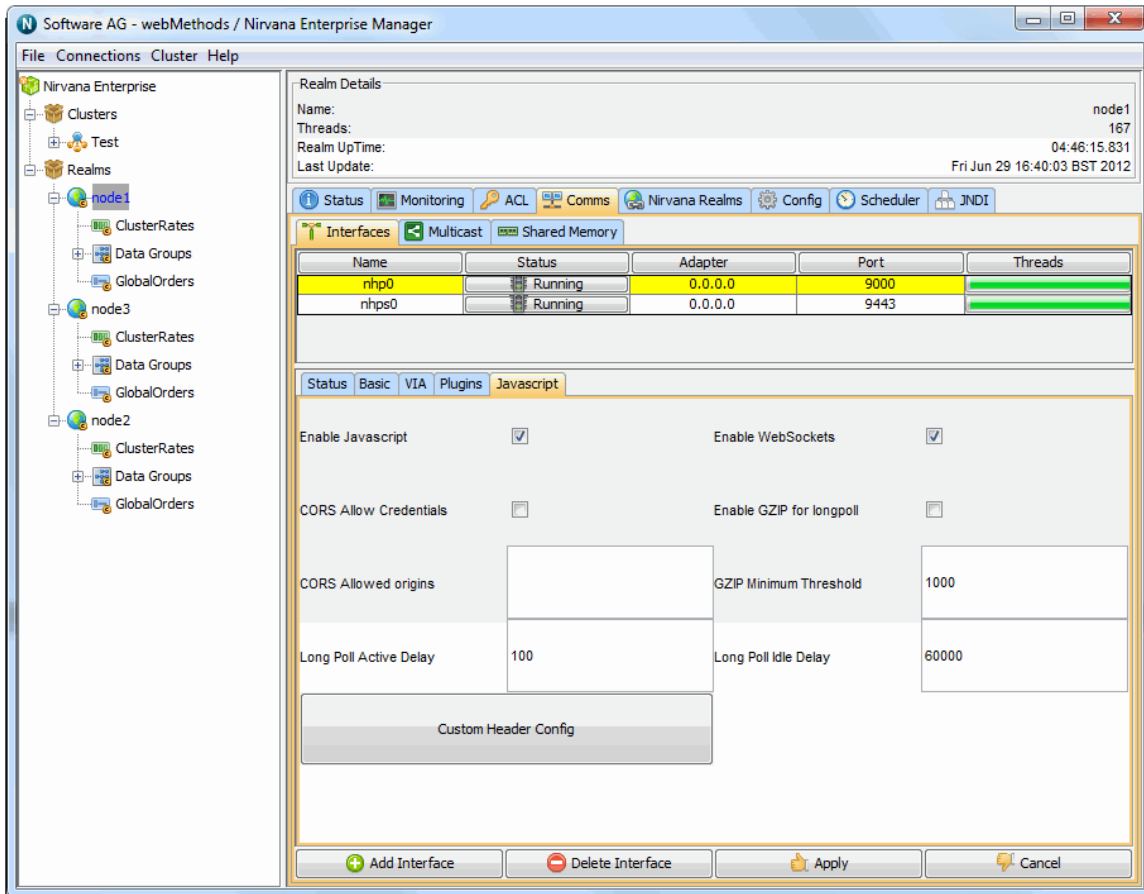
For information on adding VIA rules for an interface please see ["Interface VIA Rules" on page 133](#).

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 198](#)).

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 Toggles the server sending an "Access-Control-Allow-Credentials: true" header in response to XHR-with-

Option Name	Description
CORS Allowed Origins	<p>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.</p> <p>Leave this enabled unless recommended otherwise by support. Disabling this will in most environments prevent all CORS-based drivers from working correctly.</p> <p>Recommended Setting: *</p> <p>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.</p> <p>It is important that this is set appropriately, or approximately half of the communication drivers available to JavaScript clients will fail.</p>
Enable GZIP for LongPoll	<p>Recommended Setting: Enabled</p> <p>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.</p>
GZIP Minimum Threshold	<p>Recommended Setting: 1000</p> <p>The minimum message size is bytes required for the server to begin compressing data sent to LongPoll clients.</p>
Long Poll Active Delay	<p>Recommended Setting: 100</p>

Option Name	Description
Long Poll Idle Delay	<p>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.</p> <p>Recommended Setting: 25000</p> <p>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 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.</p>
Custom Header Config	Header Key/Value pairs which are sent in the HTTP packets to the client.

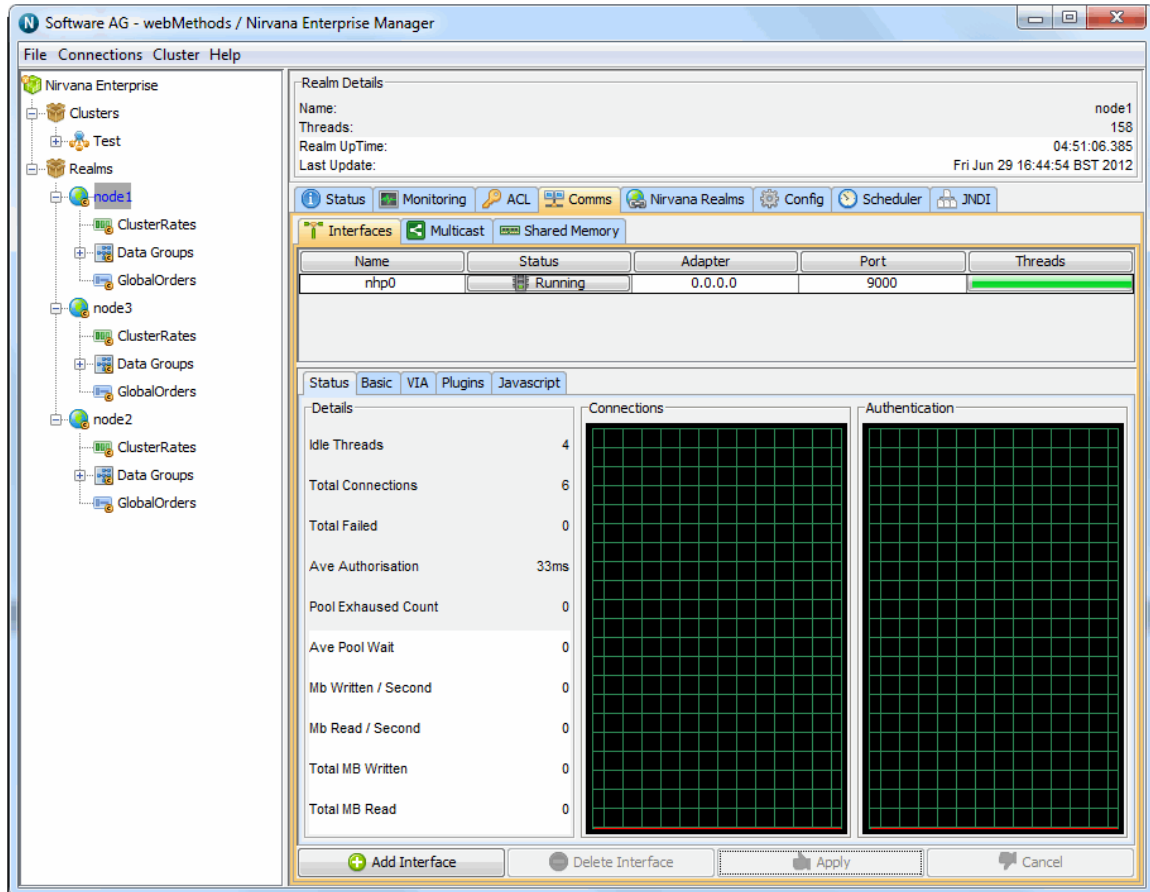
Modifying Interfaces

Each interface within a Universal Messaging realm has a number of configuration attributes (see "[Interface Configuration](#)" on page 192) 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.

The image below shows the Interfaces panel and the apply button.



Interface plugins

Universal Messaging supports the concept of plugins that actively process client requests made to nhp and nhps interfaces. There are currently 4 plugins, File (Web Server like behaviour), XML (Browse resources and events in XML), SOAP (Browse channels, queues and events using SOAP protocol) and 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 plugins section of this guide.

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.

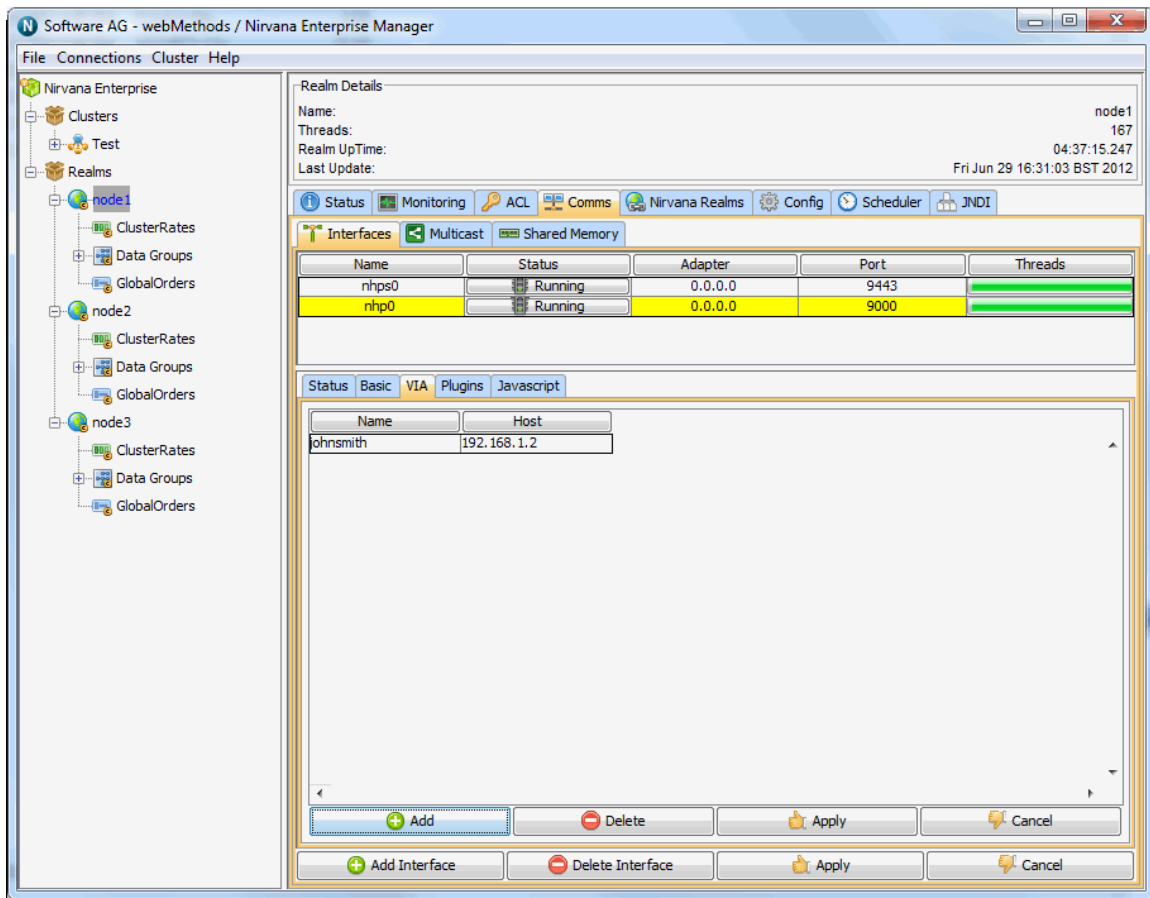
The default behaviour for all interfaces is that when no VIA lists exist on any defined interfaces, all users can connect on any interface (Realm ACLs permitting, see "[Realm Entitlements](#)" on page 125). When a user subject exists on an interface, that subject cannot use any other interface other 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.

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.

This section assumes the reader has some knowledge of IP Multicast.

Universal Messaging Multicast Architecture

Each Universal Messaging interface that you configure on a Universal Messaging Realm binds to one or all of the available physical Network adapters present on the host machine. In order to successfully configure Multicast on a Universal Messaging Realm you must ensure that you know the IP addresses of each of these network adapters

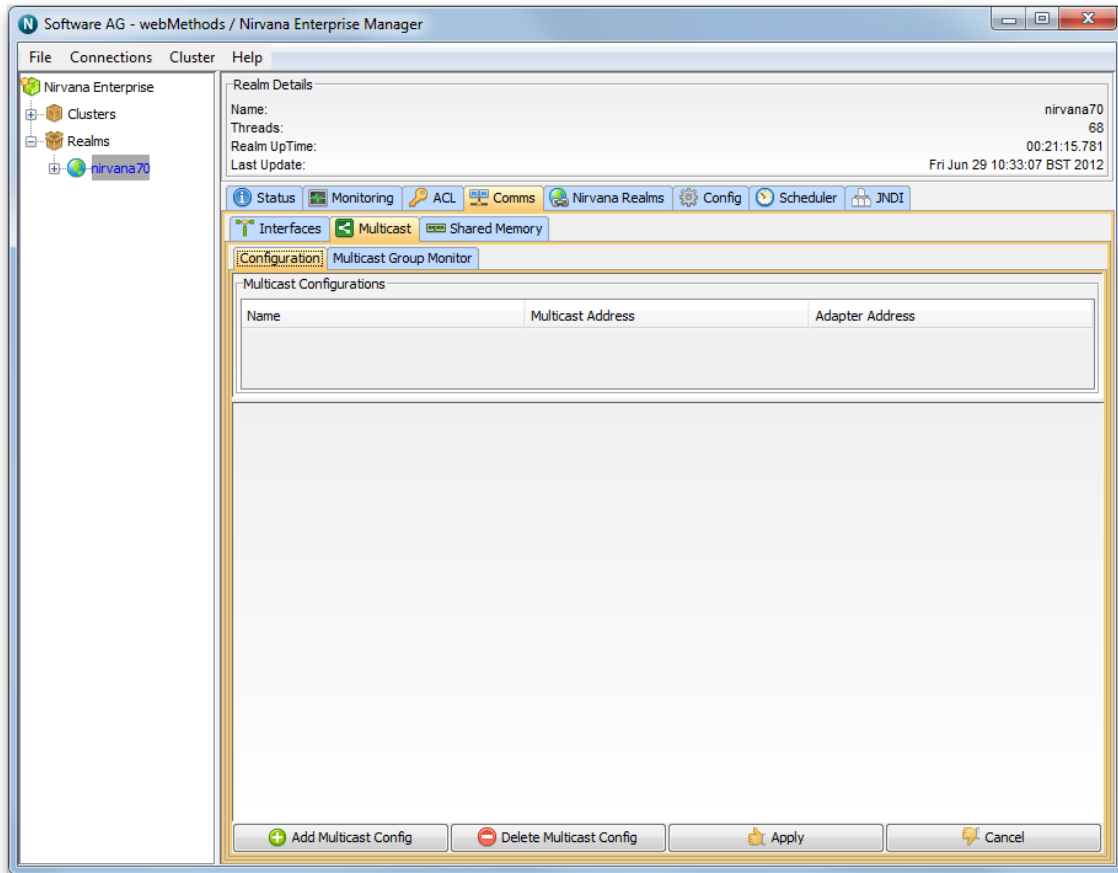
(including virtual addresses if running on a virtual host), and which physical network adapter and its address is capable of supporting IP Multicast. This information allows the correct network adapter to be selected and bound to by the Multicast configuration. Once this information is known, you then need to ensure that the physical network infrastructure including switches and routers can support Multicast. Once validated, the next step is to select an available Multicast address which can be used.

Universal Messaging servers can use IP Multicast either to deliver Data Group events to its consumers or between Universal Messaging realms within a cluster. If you wish to enable IP Multicast delivery to Data Group consumers, you can create a Multicast configuration and select it for use with Data Groups. Once you have configured a Multicast adapter, when you create a Data Group with the enable Multicast flag set, the Universal Messaging realm will automatically begin delivering the events via Multicast when published to that Data Group. The client application requires no extra setup to begin receiving Multicast. When a client Data Stream is added to a Multicast enabled Data Group, the client will transparently receive the information it needs to begin consuming Multicast for that Data Group. The client will at first both consume the Unicast Data Group events, and if Multicast is possible, also consume the Multicast events. When both Unicast and Multicast are in sync after a period of time, the Universal Messaging Server will stop sending Unicast events for that Data Stream to the client. The Universal Messaging server will track whether each client is in fact able to process the Multicast packets and if any client does not successfully acknowledge safe receipt of the Multicast events, it will simply continue to consume the Unicast events.

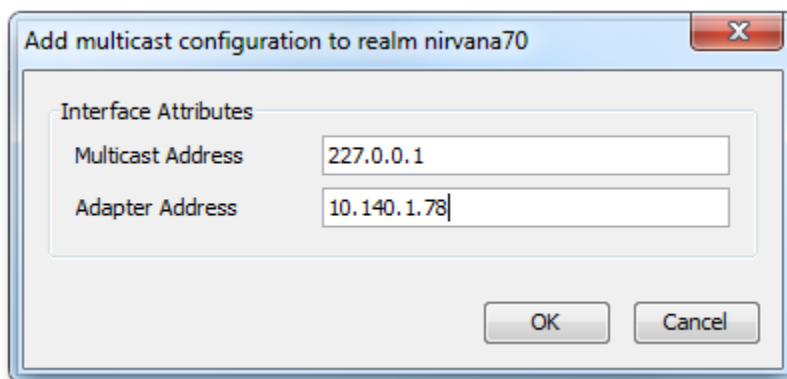
With this model, the client is able to seamlessly interact with the Universal Messaging server and begin consuming Multicast events with no changes to the Client application required.

Setting Up Multicast for Data Group Delivery

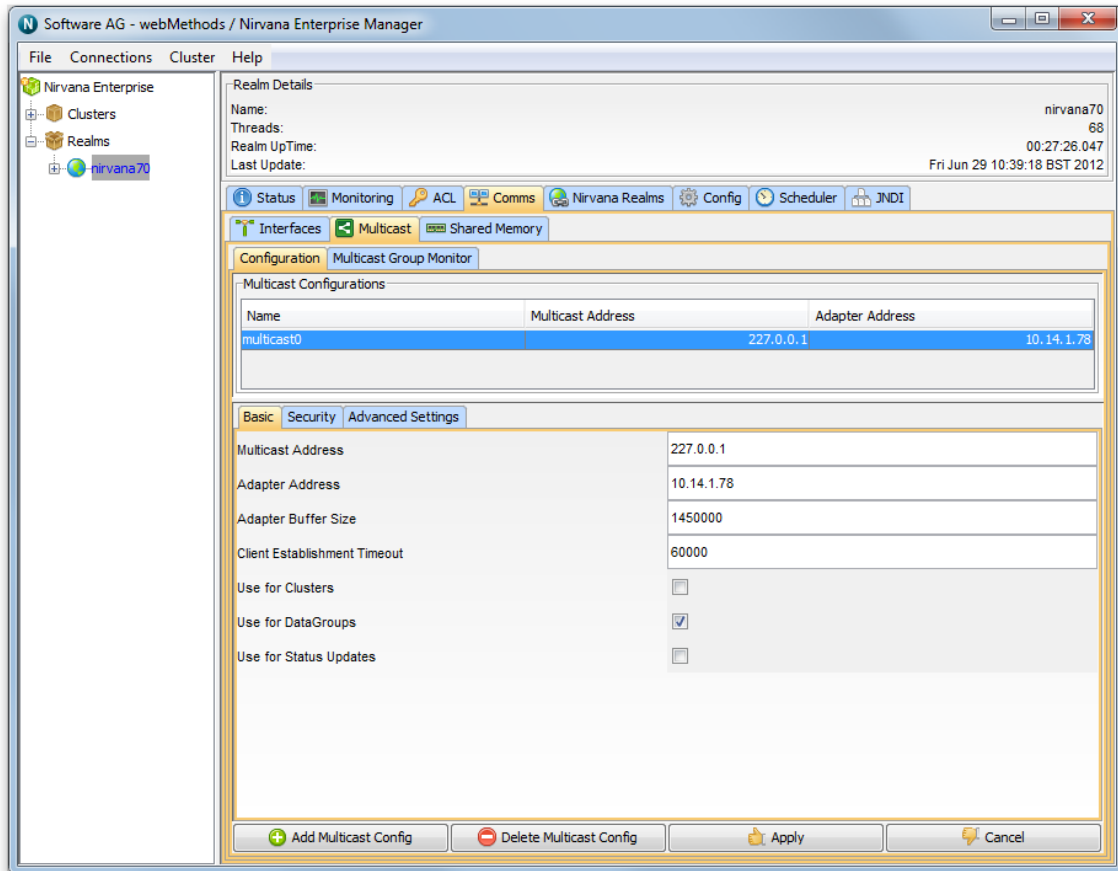
The first step in configuring a Universal Messaging Realm for Multicast Data Group delivery is to create the Multicast adapter configuration. Once you have the information described in the previous section, and your Universal Messaging realm is running, start the Enterprise Manager and connect to your 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.

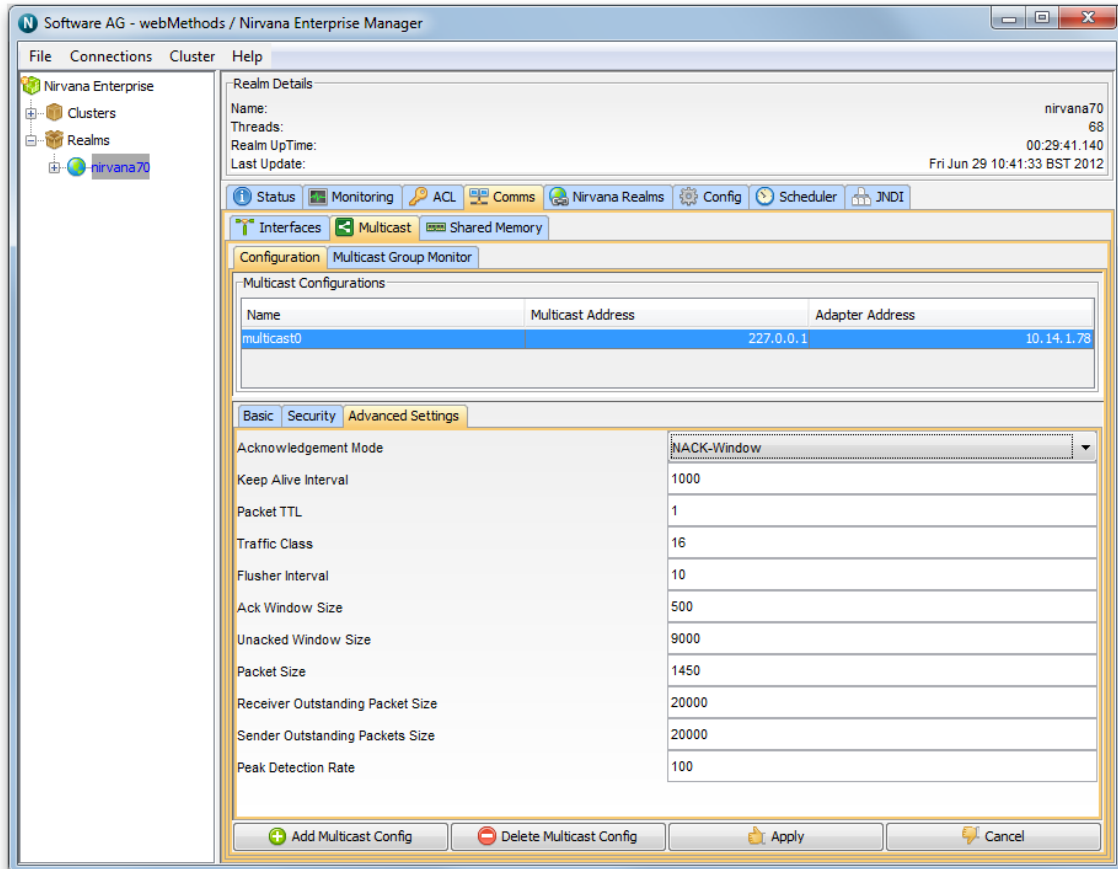
Setting Up Multicast for Cluster Inter Realm Communication

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

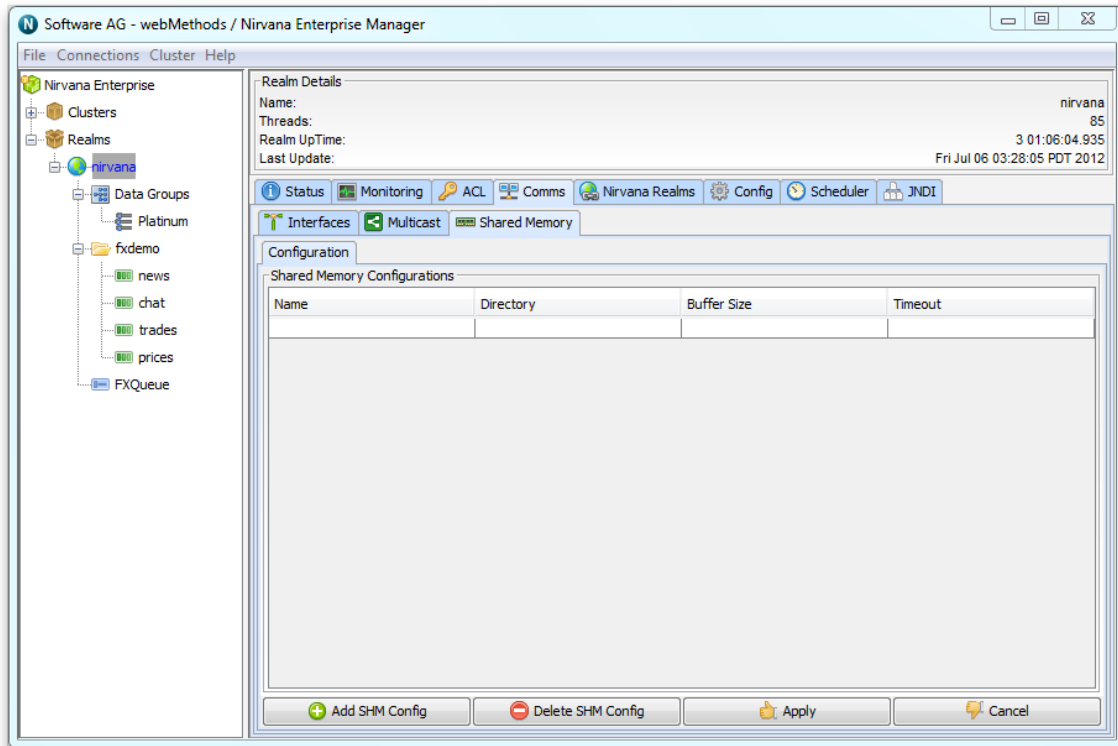
The default settings for the Multicast configurations you create are aimed at providing the lowest possible latency. With this in mind, the configuration is such that the multicast client will ack every 1 second, and the server will maintain a list of un-acked events (default 9000). Should the publish 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). In future releases of Universal Messaging 7.0 we will be changing this configuration so that there can be different modes of delivery where you have a choice of either no acks, therefore the publisher is free to move as fast as it can regardless of any loss, as well as a more guaranteed level of service where events are acknowledged.

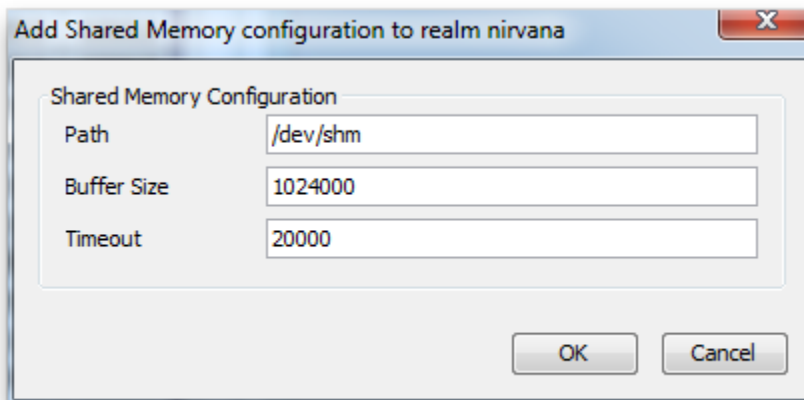


Shared Memory Configuration

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:



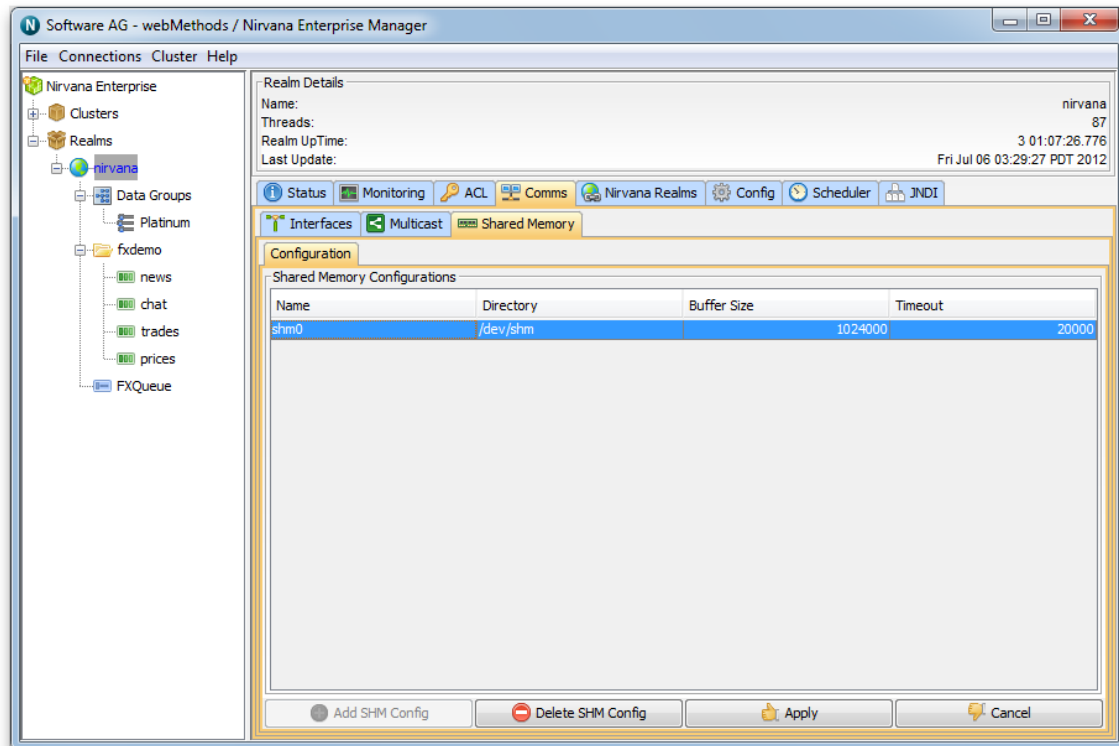
Now select "Add SHM Config" and you be presented with the below configuration box, it contains three parameters



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.

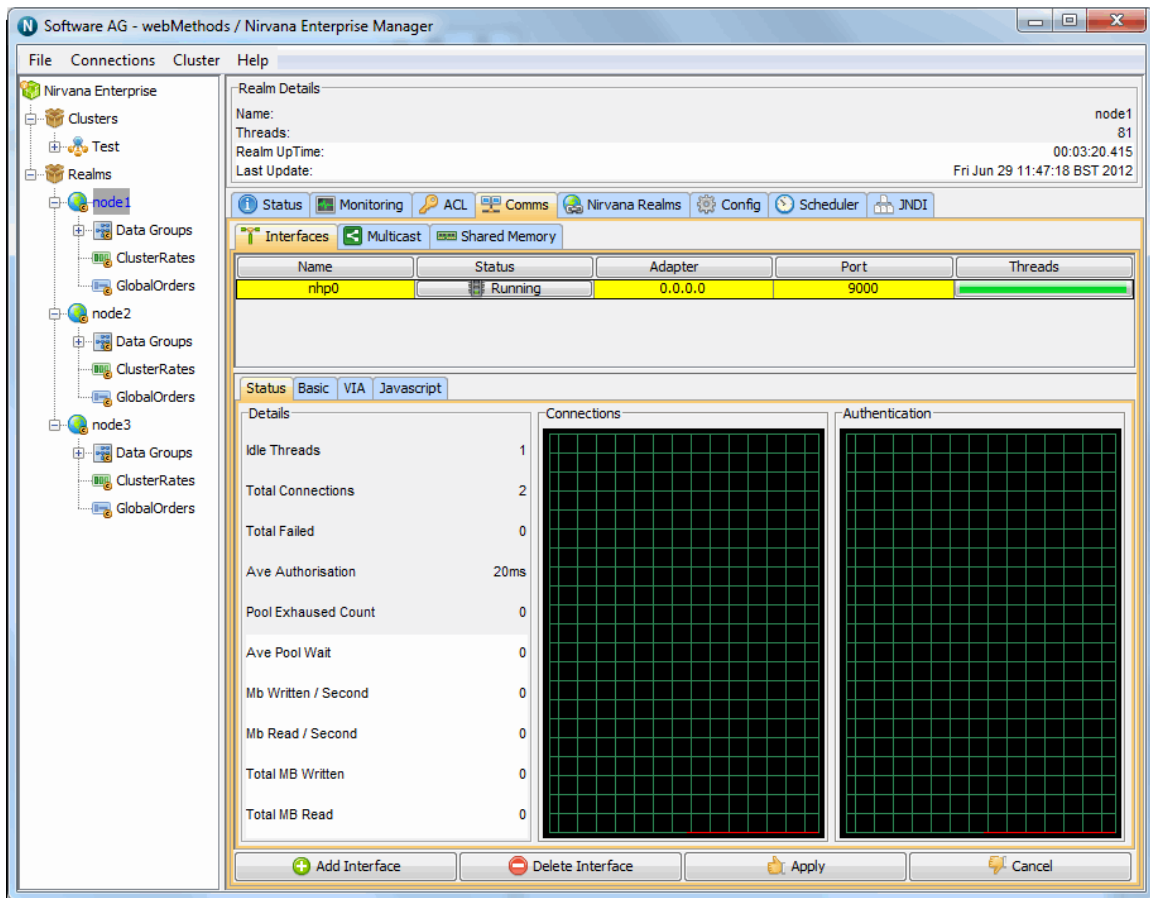


Creating an SSL network interface to a Universal Messaging Realm server

Network Interfaces can be added to a Universal Messaging Realm using the Universal Messaging Administration API or by using the Universal Messaging Enterprise Manager.

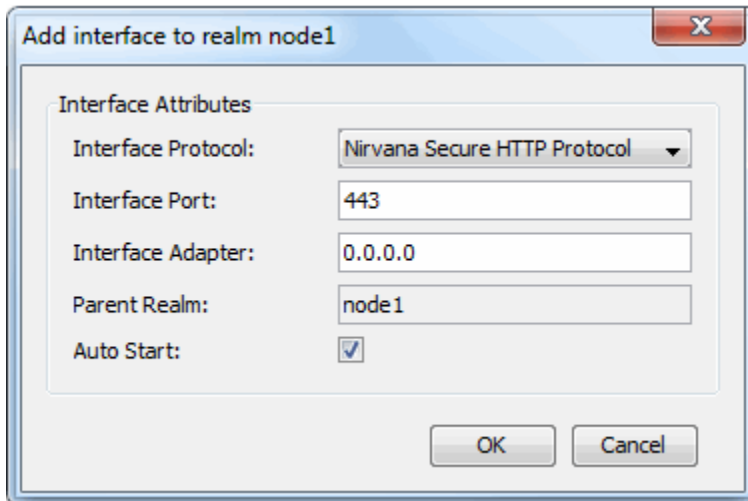
To add an ssl interface using the Enterprise Manager GUI follow the steps below:

Step 1: Click on the interfaces panel for a Realm. In the example below an interface is being added to the nirvana1 Realm. An interface could also be added however to any other realm shown in the enterprise manager. This ability makes centralised remote administration very easy using Universal Messaging.



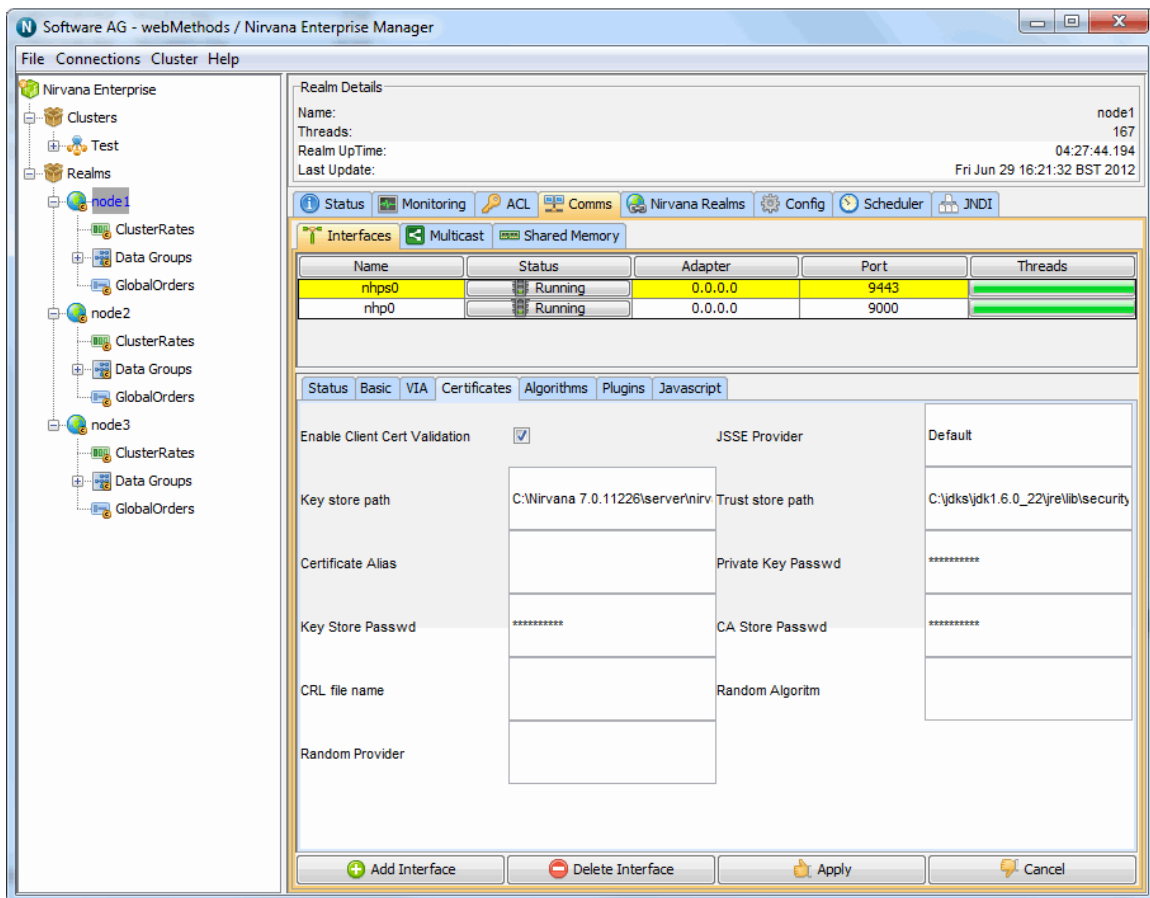
Step 2: Click on the Add button in order to bring up the Add Interface dialog box. In the dialog choose the network protocol you would like to use for this interface. The choices are Sockets, Secure Sockets, HTTP and HTTPS. Choose either Secure Sockets or Secure HTTPS to add an SSL interface.

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 clicked 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 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 212](#)) 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 212](#)) for use with Universal Messaging please see our FAQ.

The text field titled 'Key store path' should contain something similar to:

```
c:\Universal Messaging 6.0.XXXX\server\Universal Messaging\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 6.0.XXXX\server\Universal Messaging\bin\nirvacacerts.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.

Then click on apply and the Interface will be started. It will also start it now.

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 as 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. For related information on topics such as creating your own certificate chains and using the Java keytool, you can visit external links such as the following:

<http://docs.oracle.com/javase/7/docs/technotes/tools/windows/keytool.html>

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.

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 Java's keytool please see its documentation at <http://docs.oracle.com/javase/7/docs/technotes/tools/windows/keytool.html>.

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:

- Win32:
CertificateGenerator.exe
- Linux/Solaris/Generic Unix:
./CertificateGenerator
- OSX:
./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:

- Win32:
 - CertificateGenerator.exe <password> <host> <key size>
- Linux/Solaris/Generic Unix:
 - ./CertificateGenerator <password> <host> <key size>
- OSX:
 - ./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 something like this :

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBtTCCAR4CAQAwdTELMAkGA1UEBhMCVVMxZzANBgNVBAgTBmxxvbmRvbG9uZG9uMRQwEgYDVQKEwtteS1jaGFubmVsczEMMAoGA1UECzMjYml6MSAwHgYDVQQDEExdub2RlMjQ5Lm15LWNoYW5uZUwzLmNvbTCBnzANBeddiegkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAycg0MJ7PXkQM9sLj1vWa8+7Ce0FDU4tpcMX1L647dwok3uUGXuaz72DmFtb8OninjawingsjxrMBDK9fXG9hqfDvxWGyU0DEgbn+BgO3XqmUbyI6eMzGdf0vTyBFSeQIinigomontoyaU9Ahq1T7C6z1ryJ9n6XZTW79E5UcbSGjoNAPBOgVOCPKbs7/CRhZECAwEAAaAAMAOGCSqGSIb3DQEBAUAA4GBAB7TkFzQr+KvsZCV/pP5IT0c9tM58vMXkds2J7TYOp3AueMVixRo14ruLq1obbTudhc385pPgHLzO7QHEKI9gJnM5pR9yLL72zpVKPQ9XOImShvO05Tw0os69BjZeW8LTV60v4w3md47IeGE9typGGxBWscVbXzB4sgV1v0JtE7b
-----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 your 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-----
MIIFpAYJKoZIhvcNAQcCoIIF1TCCBZECAQEExADALBgkqhkiG9w0BBwGgggV5MIIC
2DCCAkGgAwIBAgICErYwDQYJKoZIhvcNAQEEBQAwwG9w0BBwGgggV5MIIC
IAYDVQQIEIx1G1T1IqVEVTVELORyBQVJVJQT1NFUyBPTkxZMR0wGwYDVQQKEXRUA
GF3dGUgQ2VydG1maWNhdG1vbjEXMBUGA1UECXMOMOVEVTVCBURVNUIFRFRU1QxHDAaBgNV
BAMTE1RoYXZ0ZSB1ZXB0IENBIFJvb3QwHhcNMDQwOTA2MTYwOTIwWhcNMDQwOTI3
MTYwOTIwWjB1MQswCQYDVQQGEwJVUzEPMA0GA1UECBMGbG9uZG9uMQ8wDQYDVQQH
EwZsb25kb24xZDASBgNVBAoTC215LWNoYW5uZWxzMQwwCgYDVQQLEwNiaXoxIDAe
BgNVBAMTF25vZGUyNDkubXktY2hhbm5lbHMUy29tMIGfMA0GCSqGSIb3DQEBAQUA
A4GNADCBiQKBgQDJyDQwns9eRAz2wuPW9Zrz7sJ7QUNTi2lwxeUvrjt3CiTe5QZe
5rPvYOYw1vw6PGswEMr19cb2Gp80/FYbJTMQMSBuf4GA7deqZRvIjp4zMZ1/S9PIE
VJ5AhT0CGrVPsLrOWvIn2fPd1Nbv0TlRxtIaOg0CkE6BU4I8oGzv8JGFkQIDAQAB
o2QwYjAMBgNVHRMBAf8EAjAAMDGA1UdHwQSMCOWKKAmoCSGIhm0dHA6Ly93d3cu
dGhhd3RlLmNvbS90ZXB0Y2VydC5jcmwwHQYDV01BBYwFAYIKwYBBQUHAWEGCCSG
AQUFBwMCMA0GCSqGSIb3DQEBAUAA4GBAHGPR6jxU/h1U4yZgt1BQoydQSaWW48e
r7s1od/2ff66LwC4d/fymiOTZpWvbiYFH1ZG98XjAvoF/V9iNpF5ALfIkeyJjNj4
ZryYjxGnbBa77GFIS4wvUkl1sngnoKpaxkQh24t3QwQJ8BRHwnwR3JraNMwDWHM1H
GaUdDBI7WYwQMIICmTCCAgKgAwIBAgIBADANBgkqhkiG9w0BAQQFADCBhzELMAkG
A1UEBHMwCkExIjAgBgNVBAGTGUZPUiBURVNUSU5HIFBVU1BPU0VTIE9OTFkxHTAB
BgNVBAoTFFRoYXZ0ZSB1ZXB0Y2F0aWZpY2F0aWZpY2F0aWZpY2F0aWZpY2F0aWZp
VEVTVDEcMBoGA1UEAxMTVGhhd3RlIFR1c3QgQ0EgUm9vdDAeFw05NjA4MDEwMDAw
MDBaFw0yMDEyMzE1MTU5NTlaIGHMqswCQYDVQQGEwJaQTEiMCAgA1UECBMZRk9S
IFRFRU1RJTtkcgUFVSUE9TRVMgT05MWTEdMBSGA1UEChMUUVGhhd3RlIEN1cnRpZmlj
YXRpb24xZmVzAVBvZSB1ZXB0IENBIFRFRU1QgVEVTVCBURVNUMRwwGgYDVQQDEwN1ZGUg
VGVzdCB0ZSB1ZXB0IENBIFRFRU1QgVEVTVCBURVNUMRwwGgYDVQQDEwN1ZGUg
fwzoZvrS1EH81TFhorPebBZhLZDDE19mYuJ+ougb86EXieZ487dSxXKruBFJPSYt
tHoCin5qkc5kBSz+/tZ4knXyRFB03CmONEKCPfdu9D06y4yXmjHApfgGJfPa/kS+
QbbiilNz7q2HLARk3umk74zHKqUyThnkjwIDAQABoxMwETAPBgNVHRMBAf8EBTAD
AQH/MA0GCSqGSIb3DQEBAUAA4GBAIAKM4+wZA/TvLIitldL/hGf7exH8/ywMupg+
yAVM4h8uf+d8phgBi7coVx71/lcB01Fmx66NyKlZK5mObgvd2dlnsAP+nnStyhVH
FIpKy3nsDO4JqrIgeHcSdpikSpbtDo18jUubV6z1kQ71CrRQtbi/WtdqxQEetgZC
JO21PoIWMQA=
-----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 208](#)).

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 recognised 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

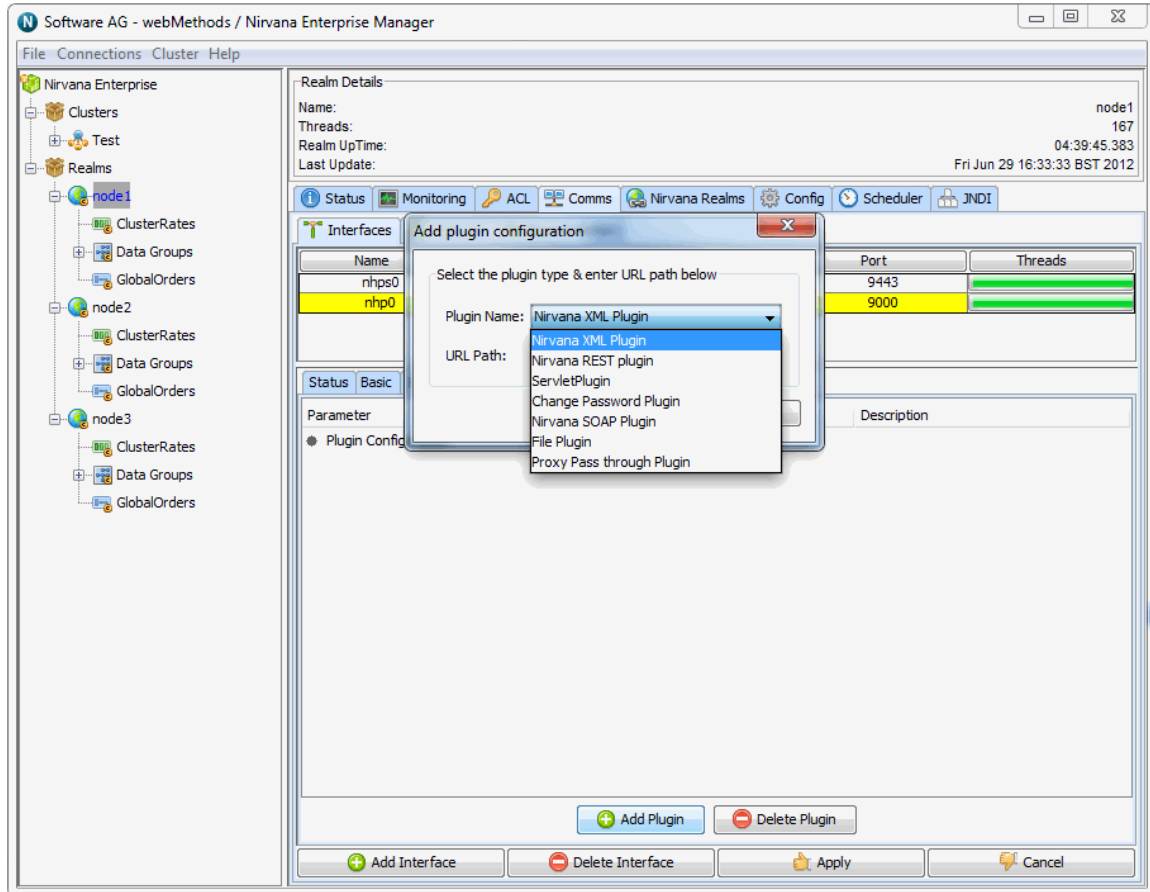
The Universal Messaging Realm Server supports the concept of Plugins within the context of the NHP or NHPS protocols. The plugins are initiated when the underlying Universal Messaging driver receives a n HTTP/S packet which is not part of the standard Universal Messaging protocol. At this point it passes the request over to the Plugin Manager to see if there is any registered plugins interested with the packets URL. If there is, then the request is forwarded to this Plugin for processing. There are several plugins supported by Universal Messaging 3.1. Please see below for available documentation. :

- ["File Plugin" on page 216](#)
- ["XML Plugin" on page 219](#)
- ["Proxy passthrough Plugin" on page 224](#)
- ["REST Plugin" on page 226](#)
- ["SOAP Plugin" on page 244](#)
- ["Servlet Plugin" on page 248](#)

Configuration

Configuration of a plugin can be done programmatically with the Administration API supplied with Universal Messaging or it can be done with the EnterpriseManager application. For the rest of this document the EnterpriseManager will be used.

In order to add a plugin, first of all you need to have created the nhp or nhps interface (see ["Creating Interfaces" on page 187](#)) that will use the plugin within the realm where you wish to run the plugin. Once the interface is created, select the interface from the table of configured interfaces for the chosen realm, and then select the tab labelled 'Plugins' from the interface configuration panels. Clicking on the 'Add Plugin' button will present you with a dialog that 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 used to determine if the request is destined for a plugin. If the server matches the URLPath portion of the URL supplied 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 we had a File Plugin configured with a URL Path as `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 vendor static Web pages. This can be used for example to have the server vendor applets and supported files without the need for a dedicated web server. For example, if I was running a File

plugin on my realm server host called 'webhost', on an nhp interface running on port 80, I could type in a url within a web browser `http://webhost/index.html` which will return the index page defined within the file plugin root file directory.

This enables the Realm Server to act as a webserver and can even be used to vend applets to client browsers that may directly communicate with the Realm Server and publish and consume events from channels.

Configuration

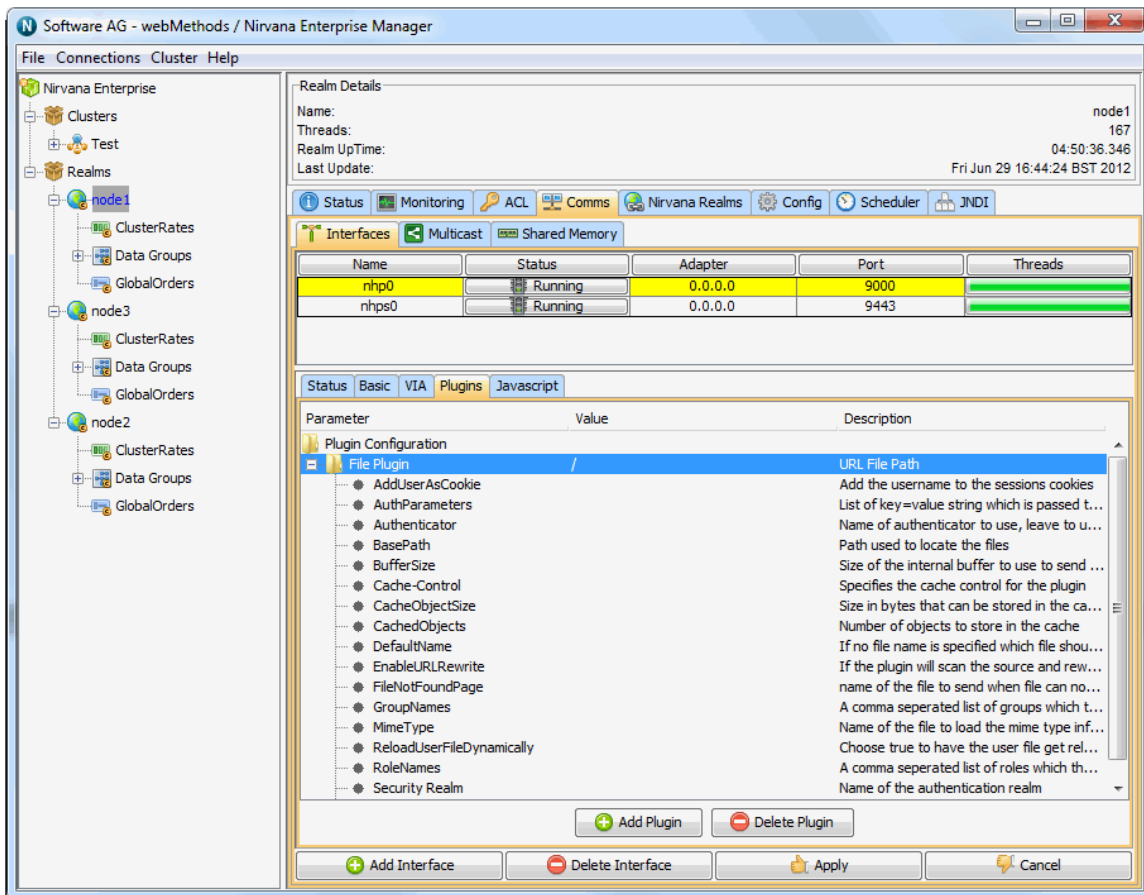
Once you have created the file plugin on the interface you require it on, you can then select it from the plugins panel for the selected interface and enter values as you wish for the configuration parameters.

The file plugin requires configuration information relating its behaviour as well as the location of the files it is required to vend 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.	<serverPath>/plugins/htdocs
DefaultName	If no file name is specified which file should be returned.	index.html
FileNotFoundPage	Name of the file to send when file can not 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.
CachedObjects	Number of objects to store in the cache	100

Parameter Name	Description	Default Value
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 interface panel with an nhp interface running on port 8080. This interface has a File Plugin configured with the default settings and its URL path is /docs. The default BasePath setting is <serverPath>/plugins/htdocs, which is your local install server directory/plugins/htdocs. The default installation places the Universal Messaging API docs within the htdocs directory. Once the plugin is created, you can hit 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:8080/docs/> which will then render the default index.html page 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 8080, and displaying the default API docs found in the htdocs directory.

The screenshot shows a web browser window with the URL `http://localhost/` and a tab titled 'Nirvana Java APIs (Version ...'. The page title is 'my-Channels'. Below the title, there are navigation tabs: 'Main Page', 'Related Pages', 'Packages', and 'Data Structures'. Under 'Data Structures', there are sub-tabs: 'Data Structures', 'Data Structure Index', 'Class Hierarchy', and 'Data Fields'. The main content area is titled 'Data Structures' and contains the text: 'Here are the data structures with brief descriptions:'. Below this text is a table with two columns: the class name and a brief description.

<code>com.pcbssystem.nirvana.client.nAbstractChannel</code>	This is the base class for all nirvana channel types, including queues and normal pub/sub channels and as such has no public methods
<code>com.pcbssystem.nirvana.client.nAsyncExceptionListener</code>	This interface should be implemented by classes wishing to register to receive asynchronous errors from the nirvana server
<code>com.pcbssystem.nirvana.client.nBaseClientException</code>	This exception forms the base for all <code>client</code> exceptions
<code>com.pcbssystem.nirvana.client.nCertificateSigner</code>	This class signs <code>nConsumeEvents</code> as they are published to the Realm Server
<code>com.pcbssystem.nirvana.client.nCertificateValidator</code>	This class validates signatures signed by the <code>nCertificateSigner</code> class
<code>com.pcbssystem.nirvana.client.nChannel</code>	This class is a Nirvana channel, offering the Nirvana channel related services
<code>com.pcbssystem.nirvana.client.nChannelAlreadyExistsException</code>	This exception is thrown if a user tries to make a channel that already exists
<code>com.pcbssystem.nirvana.client.nChannelAlreadySubscribedException</code>	The user has tried to subscribe to the channel more than once
<code>com.pcbssystem.nirvana.client.nChannelAttributes</code>	This class is a container for the attributes necessary to create or locate a Nirvana channel or queue

XML Plugin

The XML Plugin can be used to query the Realm Server, its queues and channels and 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 channel, then using a standard Web browser, get the server to transform the XML into HTML via a stylesheet thereby enabling 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.

Configuration

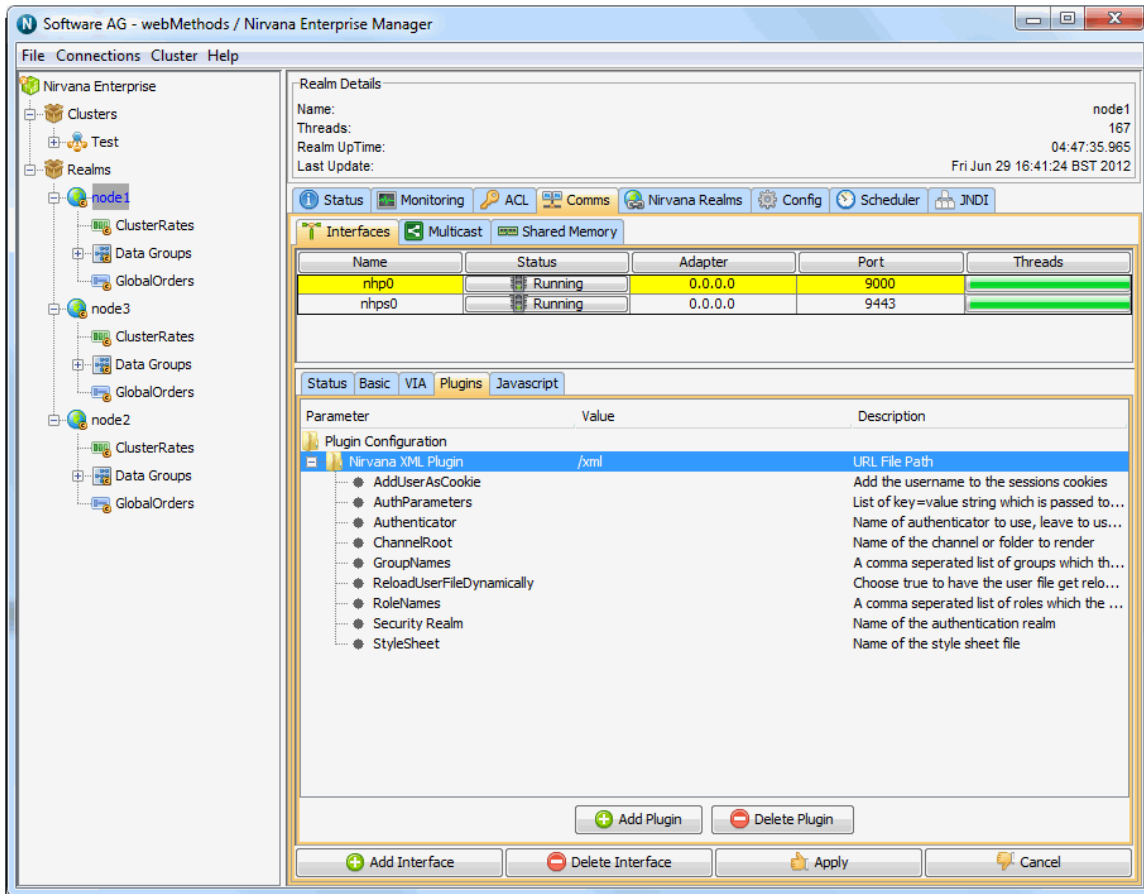
Once you have created the xml plugin on the interface you require it on, you can then select it from the plugins panel for the selected interface and enter values as you wish for the configuration parameters.

The xml plugin requires configuration information relating its behavior as well as the entry point in the namespace for the channels you wish to make available to vend 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.
UserFile	Name of the file containing the usernames and passwords	None.

The image below shows the enterprise manager interface panel with an nhp interface running on port 8080. This interface has a XML Plugin configured with the xml2html 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 hit the apply button which will restart the interface and enable the new xml plugin.



From a browser, it is now possible to enter the url 'http://localhost:8080/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 8080. The realm information is displayed at the top of the page, and the information on resources is shown beneath. The stylesheet 'xml2html.xsl' that renders this within the browser is supplied with the download under the <server>/plugins/htdocs/style directory and can be modified as you wish.

The screenshot shows a web browser window with the following content:

my-channels.com - Technologies Working Together - Microsoft Internet Explorer

Address: <http://localhost/xml/>

my-channels.com technologies working together

Realm Status for node1

Total Memory	266469376
Free Memory	258004264
Current Threads	83
Total Connections	4
Events Published	6
Events Consumed	0

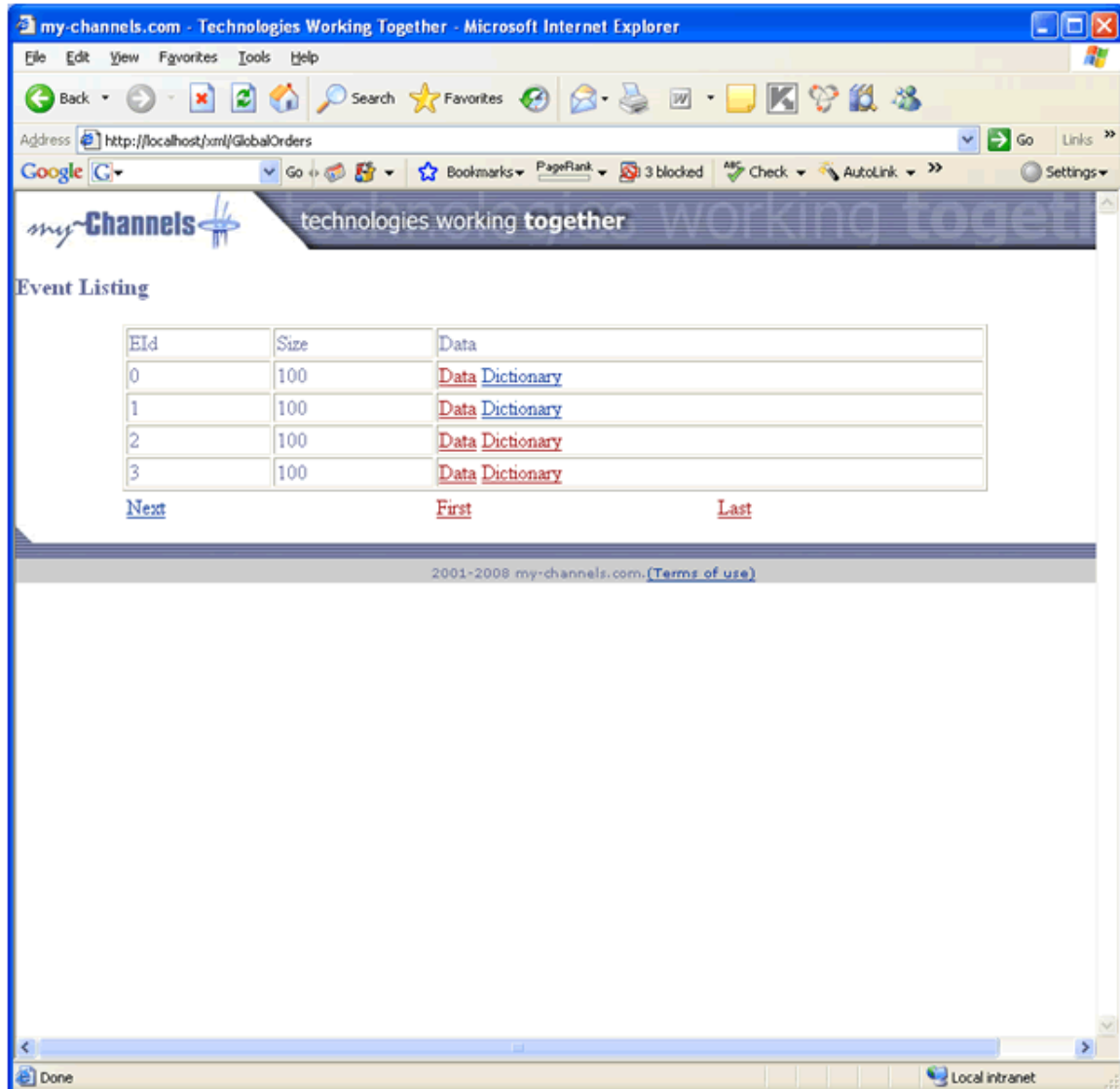
Channel Listing

Channel Name	Number of Events	Last Event Id
ClusterRates	2	1
GlobalOrders	4	3
-p2p/serviceinfo	0	-1

2001-2008 my-channels.com. [Terms of use](#)

Local intranet

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 either show 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 style sheet to render your own xml event documents.

Proxy Passthrough Plugin

The Proxy 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 pass through plugin.

This functionality enables realms to act as a proxy to forward URL requests to any host that accepts http(s) connections.

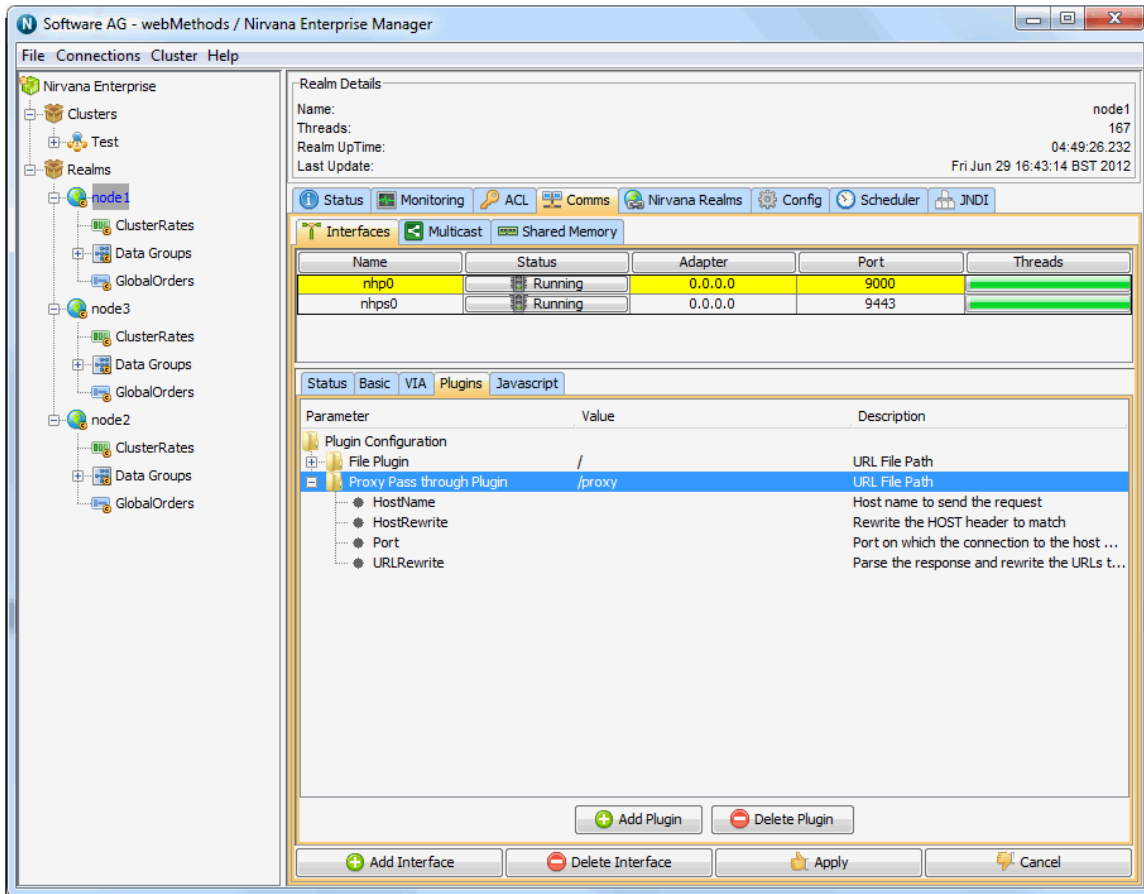
Configuration

Once you have created the proxy 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 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	www.Terracotta.com
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 9005 on the realm called 'node1'. This interface has a proxy Plugin configured to redirect requests from this interface using the URL path of "/" and will forward these requests to the File and XML plugins located on the 'productionmaster' realm nhp0 interface running on port 80.



From a browser, it is now possible to enter the url 'http://localhost:9005/xml/' which will redirect this request to the interface on the productionmaster realm interface running on port 8080. This will actually display the details of the productionmaster realm delivered through the XML plugin defined within the XML Plugin (see "XML Plugin" on page 219) help page. The image below demonstrates the browser view of the http request made and the results of the redirect to the productionmaster realm.



REST Plugin

The Universal Messaging REST plugin allows access to the 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

Parameter Name	Description	Default Value
SessionTimeout	Time in seconds to time-out inactive http sessions.	300

The Universal Messaging 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 230](#)
- ["application/xml" on page 230](#)

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 234](#)
- ["application/xml" on page 234](#)

Available response representations:

- ["text/xml" on page 238](#)
- ["application/xml" on page 238](#)

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 238](#)
- ["application/xml" on page 238](#)

Available response representations:

- ["text/xml" on page 238](#)
- ["application/xml" on page 238](#)

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 238](#)

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 241](#)

Available response representations:

- ["application/json" on page 243](#)

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 243](#)

Available response representations:

- ["application/json" on page 243](#)

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:

```
<Universal Messaging-RealmServer-ChannelList NumberOfChannels="2">
  <!--Constructed by my-channels Universal Messaging REST-Plugin :
    Wed Mar 02 16:07:28 EET 2011-->
  <Channel EventsConsumed="0" EventsPublished="0" LastEventID="-1"
    Name="testqueue" NumberEvents="0"
    fqfn="http://localhost:8080/rest/API/xml/testqueue"/>
  <Channel EventsConsumed="0" EventsPublished="2" LastEventID="223"
    Name="testchannel" NumberEvents="2"
    fqfn="http://shogun:8080/rest/API/xml/testchannel"/>
</Universal Messaging-RealmServer-ChannelList>
```

If the REST plugin is configured to include realm status, some additional information about the realm is presented:

```
<Universal Messaging-RealmServer-ChannelList NumberOfChannels="2">
  <!--Constructed by my-channels Universal Messaging REST-Plugin :
    Wed Mar 02 16:07:28 EET 2011-->
  <Channel EventsConsumed="0" EventsPublished="0" LastEventID="-1"
    Name="testqueue" NumberEvents="0"
    fqfn="http://localhost:8080/rest/API/xml/testqueue"/>
  <Channel EventsConsumed="0" EventsPublished="2" LastEventID="223"
    Name="testchannel" NumberEvents="2"
    fqfn="http://shogun:8080/rest/API/xml/testchannel"/>
  <RealmStatus FreeMemory="498101048" RealmName="nirvana6" Threads="87"
    TotalConnections="0" TotalConsumed="0"
    TotalMemory="530186240" TotalPublished="2"/>
</Universal Messaging-RealmServer-ChannelList>
```

Should the parameter point to an existing channel or queue, the response code is 200 and the response looks like this:

```
<Universal Messaging-RealmServer-EventList>
  <!--Constructed by my-channels Universal Messaging 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&EID=first"
      LastEID="223"
      LastEvent=
        "http://localhost:8080/rest/API/xml/testsrc?Data=Dictionary&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&EID=222"
      DataSize="9" EID="222" Tag="Test Tag" hasByte="true"/>
  <Event
    DictionaryLink=
      "http://localhost:8080/rest/API/xml/testsrc?Data=Dictionary&EID=223"
      EID="223" hasDictionary="true"/>
</Universal Messaging-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:

```

<Universal Messaging-RealmServer-RawData>
  <!--Constructed by my-channels Universal Messaging 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>
</Universal Messaging-RealmServer-RawData>

```

If you choose to look at an individual XML event, the response code is 200 and the response looks like this:

```

<Universal Messaging-RealmServer-XMLData>
  <!--Constructed by my-channels Universal Messaging 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>
</Universal Messaging-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>
  </Dictionary>

```

```

</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>
  </ArrayItem>
</DataArray>

```



```

    </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>

```

```

    </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 244](#)). 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>
  </Dictionary>
</DictionaryData>

```

```

    </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:

```
<Universal Messaging-RealmServer-PublishRequest>
  <response value="ok"/>
</Universal Messaging-RealmServer-PublishRequest>
```

Should the publish call fail for any reason, the response code is 400 and the response looks like this:

```
<Universal Messaging-RealmServer-Error>
  <response value="failInput"/>
  <errorcode value="ErrorCode"/>
  <errormessage value="Error Message"/>
</Universal Messaging-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:

```
<Universal Messaging-RealmServer-PurgeRequest startEID="10" endEID="20" purgeJoins="false">
  <selector>
    <![CDATA[]]>
  </selector>
</Universal Messaging-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:

```
<Universal Messaging-RealmServer-PurgeRequest>
  <response value="ok"/>
</Universal Messaging-RealmServer-PurgeRequest>
```

Should the purge call fail for any reason, the response code is 400 and the response looks like this:

```
<Universal Messaging-RealmServer-Error>
  <response value="failInput"/>
  <errorcode value="ErrorCode"/>
  <errormessage value="Error Message"/>
</Universal Messaging-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",
    "fqcn": "http://localhost:8080/rest/API/json/testqueue"
  }, {
    "EventsConsumed": "0",
    "EventsPublished": "0",
    "LastEventID": "212",
    "Name": "testchannel",
    "NumberEvents": "0",
    "fqcn": "http://localhost:8080/rest/API/json/testchannel"
  } ],
  "Comment": "Constructed by my-channels Universal Messaging REST-Plugin :
    Wed Mar 02 11:38:30 EET 2011",
  "Name":
    "Universal Messaging-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",
    "fqcn": "http://localhost:8080/rest/API/json/testqueue"
  }, {
    "EventsConsumed": "0",
    "EventsPublished": "0",
    "LastEventID": "212",
    "Name": "testchannel",
    "NumberEvents": "0",
    "fqcn": "http://localhost:8080/rest/API/json/testchannel"
  } ],
  "Comment": "Constructed by my-channels Universal Messaging REST-Plugin :
    Wed Mar 02 11:38:30 EET 2011",
  "Name": "Universal Messaging-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 Universal Messaging 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": "Universal Messaging-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 Universal Messaging REST-Plugin :
                Wed Mar 02 12:21:46 EET 2011",
    "Data": "VGVzdCBCb2R5",
    "EID": "213",
    "Name": "Universal Messaging-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 Universal Messaging REST-Plugin :
                Wed Mar 02 12:21:46 EET 2011",
    "Data": "VGVzdCBCb2R5",
    "EID": "213",
    "Name": "Universal Messaging-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": "VGZzdCBCb2R5",
  "isPersistent": true,
  "tag": "VGZzdCBUYWc="
}

```

Important: data and tag should always be submitted in base64 encoded form.

JSON DOM events can be represented as follows:

```

{
  "data": "VGZzdCBCb2R5",

```

```

    "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 244](#)). 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
Float	5
Character	6
Byte	7
Short	8
Dictionary	9
Array	100

SOAP Plugin

The Universal Messaging SOAP Plugin utilises the advanced HTTP/HTTPS stack capabilities of the Universal Messaging server to provide an implementation of a SOAP 1.2 based client API. Any SOAP toolkit can use the provided WSDL to generate client stubs for any SOAP compliant language. This way Universal Messaging functionality is offered to programming languages that were previously unsupported and all this without the need for any additional infrastructure component (SOAP server, web server etc).

The plugin can be configured to expose complete Universal Messaging namespaces or subsets, support username/password authentication, control web service listing and other options further explained in the section below.

Configuration

Once you have created the SOAP 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 SOAP plugin requires configuration details regarding the entry point in the namespace for the channels you wish to make available to vend to the clients, as well

as web-services specific configuration. Below is a table that shows each configuration parameter and describes what it is used for.

Parameter Name	Description	Default Value
URL File Path	The mount URL path for the SOAP plugin to be invoked.	None (/soap needed for samples)
ChannelRoot	Universal Messaging namespace node (channel or folder) to expose through soap	/
UserFile	Name of the file containing the usernames and passwords	None.
Security Realm	Name of the authentication realm	None.
AttachmentDir	Name of the directory to find the AXIS attachments	<ServerPath>/plugins/attachments/
EnableList	Enable or disable the listing of web services wsdl files. This is necessary if you want to generate client stubs.	false.
WebRootDir	Name of the file containing the usernames and passwords	<ServerPath>/plugins/
WSDLEncoding	Type of encoding to be used for WSDL, valid values are document, rpc, wrapped	rpc

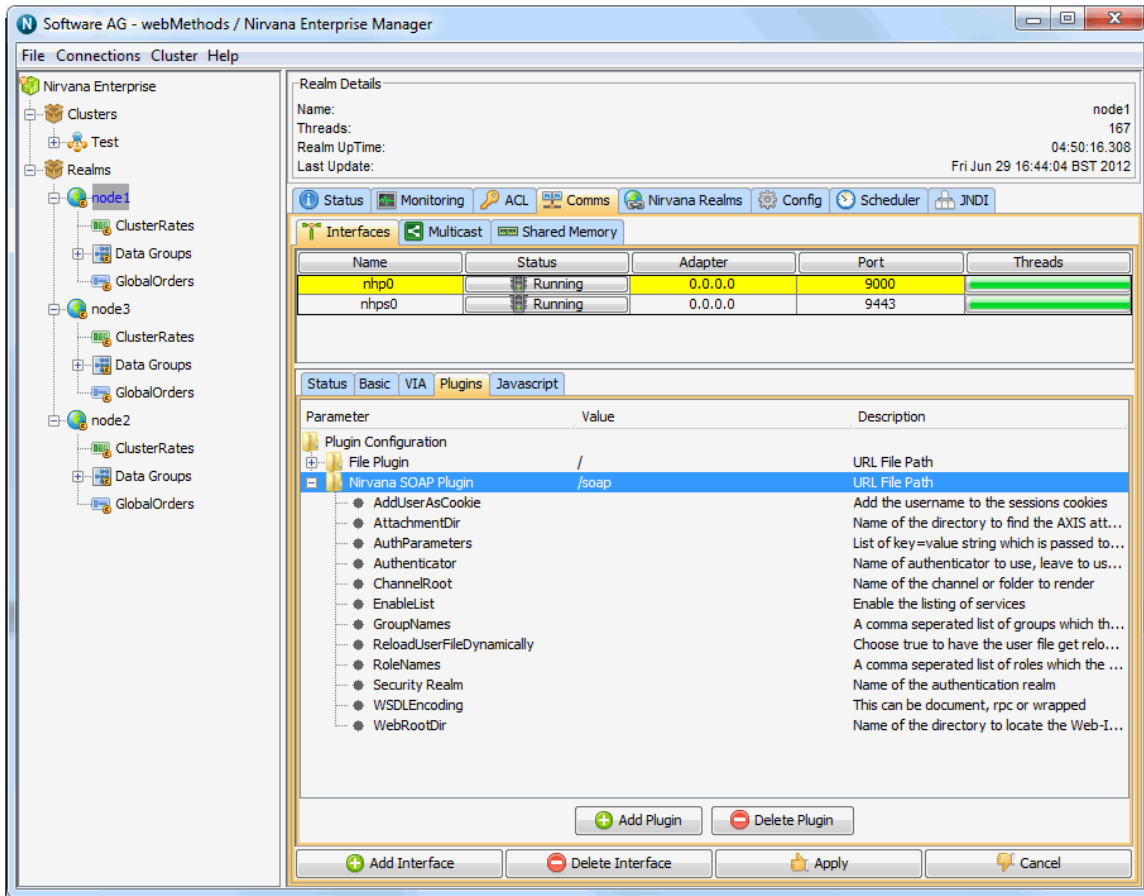
Once you have completed setting up your SOAP plugin, you can verify it works by opening a browser to the NHP interface in the mount URL path. For example for an NHP interface running on port 80 on localhost and having the plugin mounted on /soap, open a browser to <http://localhost/soap/> and you should see something like:

Universal Messaging Supported Services

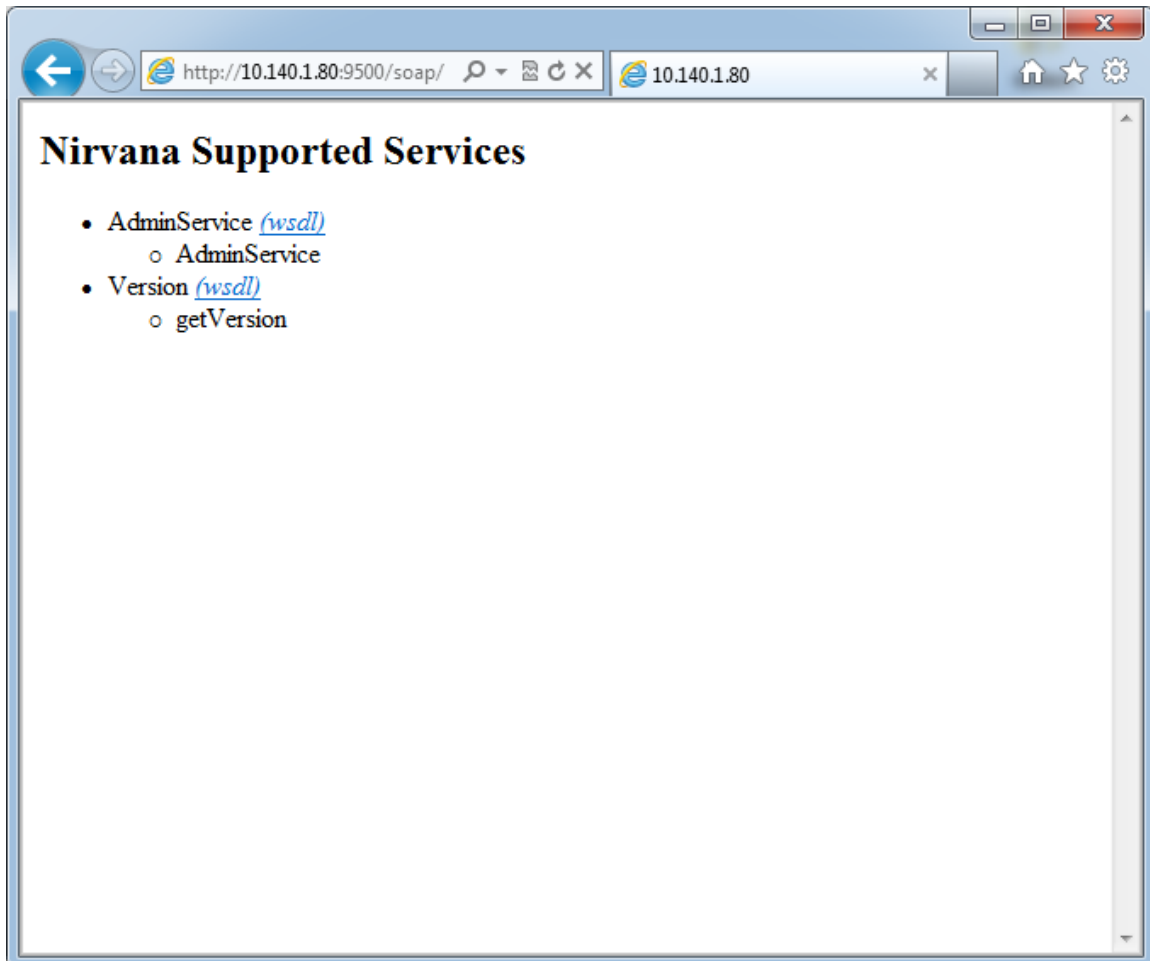
```
urn:nirvanasRealm (wsdl)
getRealmName
getStatus
getChannelDetails
getChannelCount
urn:nirvanasChannel (wsdl)
```

```
getLastEID
publish
getEvents
getEventCount
```

The image below shows the enterprise manager interface panel with an nhp interface running on port 80. This interface has a SOAP Plugin configured with its URL path as /soap. The default ChannelRoot setting is /, which is the root of the namespace, i.e. all channels. Once the plugin is created, you can hit the apply button which will restart the interface and enable the new SOAP plugin.



From a browser, it is now possible to enter the url 'http://localhost:8080/soap/' which shows the available services exposed via SOAP as well as links to the WSDL documents required to generate client stubs. The image below demonstrates the browser view from a Realm that has a SOAP plugin on an nhp interface on port 80.



Samples

The Universal Messaging installation includes a few samples for perl src/soap directory. The applications use the SOAP::Lite Perl module and were tested under cygwin for windows. Please note that all the samples included assume that you have an NHP interface running on port 80 on the local machine and having a SOAP plugin configured under /soap. Furthermore the `getEvent.pl` sample requires you to have a channel called `test` with at least 1 event inside.

For example to execute `getChannelDetails.pl`, open a cygwin (or other) shell and type:

```
$ perl getChannelDetails.pl
-----
firstEid=-1
ttl=0
name=Universal Messaging-p2p/serviceinfo
eventCount=0
type=Simple
capacity=0
lastEid=-1
-----
firstEid=0
ttl=0
name=test
```

```

eventCount=10
type=Reliable
capacity=0
lastEid=9

```

Servlet Plugin

The Servlet plugin enables the Universal Messaging Realm Server to vendor 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 its behavior, as well as the location of the Servlets it is required to vend 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 config 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
GroupNames	A comma separated list of groups to which a user must be a member of to be granted access.	

Parameter Name	Description	Default Value
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 <code>com.example.Servlet</code> defining a property called <code>RNAME</code> you should have a line as follows: <code>com.example.Servlet.RNAME=nsp://localhost:9000</code>	
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 <code>com.example.Servlet</code> 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.	

XML Configuration: Overview

The Universal Messaging Enterprise Manager allows you to export specific elements of the realm or the entire realm structure into an XML representation. The exported XML can contain all clusters, realm ACLs, channels, queues and their ACLs, configuration

parameters, JNDI assets, interfaces, plugins and scheduling information. Once exported, the XML can then be imported into any other realm, which is useful when you wish to clone realms and their internal structures. Importing the xml will automatically create and configure those objects selected for the import from the XML file.

The export and import marshals the realm objects from their Administration API representation into XML and back again. This provides a very powerful way of automatically configuring a realm based on a standard structure.

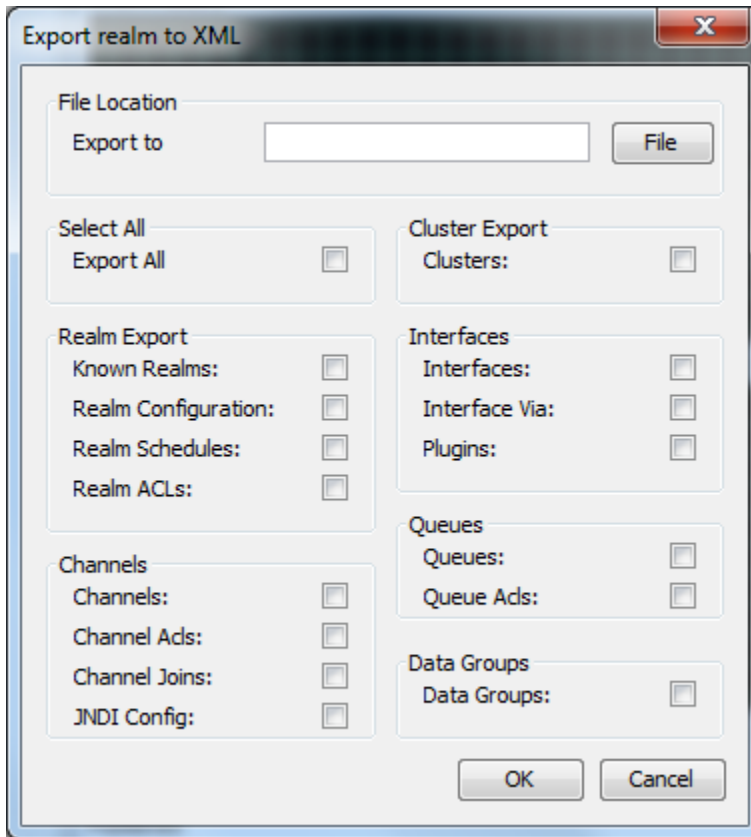
See the links below to view more about the import and export mechanism available in the Universal Messaging Enterprise Manager:

- ["XML Configuration: Exporting To XML" on page 250](#)
- ["XML Configuration: Importing From XML" on page 251](#)

XML Configuration: Exporting To XML

The ability to export an entire realm or specific elements of a realm's structure is a powerful enabler for managing the configuration of multiple realms within your enterprise. This section will discuss how to export realm elements to their xml representation.

Firstly, to export a realm to xml, you need to select the realm you wish to export from the Enterprise Manager namespace. Right-clicking on the realm node will present a menu for the options available on a realm node. One of those menu options is labelled 'Export Realm to XML'. Selecting this menu option will present a dialog as shown in the image below.



The dialog above shows the list of export options available for the realm, and the name and location of the file that will be exported. The 'Export to' field is the name and location of the file to export which can either be typed manually, or chosen by selecting the button with the folder icon that shows a file chooser dialog.

There are a large number of options for what can be exported from a realm. The check boxes indicated on the dialog can all be selected individually for specific elements of a realm to be exported, or by clicking on the 'Export All' button all options will be selected.

Clicking on the 'OK' button will export the realm to xml into the file and location specified.

To view an example of the XML produced from the export, see ["XML Configuration: Sample XML File for EXPORT" on page 252](#).

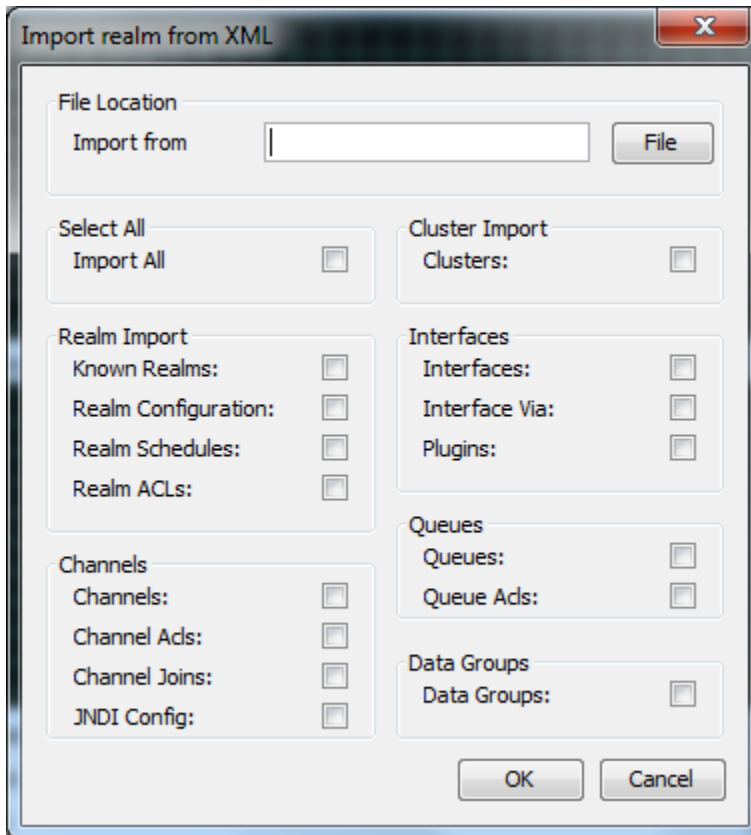
XML Configuration: Importing From XML

The ability to import realm elements from XML is a powerful enabler for managing the configuration of multiple realms within your enterprise. This section will discuss how to import realm elements from the xml representation.

Firstly, to import a realm to xml, you need to have first made sure you have the desired elements you wish to import within an XML file. For help on export, see ["XML Configuration: Exporting To XML" on page 250](#). To import XML, select the realm you wish to import the realm data to from the Enterprise Manager namespace. Right-clicking

on the realm node will present a menu for the options available on a realm node. One of those menu options is labelled 'Import Realm to XML'.

Selecting this menu option will present a dialog as shown in the image below.



The dialog above shows the list of import options available for the realm, and the name and location of the file that will be used for the import. The 'Import from' field is the name and location of the file to import from which can either be typed manually, or chosen by selecting the button with the folder icon that shows a file chooser dialog.

There are a large number of options for what can be imported from XML into a realm. The check boxes indicated on the dialog can all be selected individually for specific elements of a realm to be imported, or by clicking on the 'Import All' button all options will be selected.

Clicking on the 'OK' button will import the xml into the realm from file and location specified, and then attempt to create the objects and set the configuration elements defined within the XML.

To view an example of the XML produced from the export, see ["XML Configuration: Sample XML File for EXPORT"](#) on page 252.

XML Configuration: Sample XML File for EXPORT

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Universal MessagingRealm comment="Realm configuration from productionmaster"
  exportDate="2005-01-06+00:00" name="productionmaster">
```

```

<RealmConfiguration>
  <ConfigGroup name="AuditSettings">
    <ConfigItem name="ChannelACL" value="true"/>
    <ConfigItem name="ChannelFailure" value="true"/>
    <ConfigItem name="ChannelMaintenance" value="false"/>
    <ConfigItem name="ChannelSuccess" value="false"/>
    <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="ServiceACL" value="true"/>
    <ConfigItem name="ServiceFailure" value="true"/>
    <ConfigItem name="ServiceMaintenance" value="true"/>
    <ConfigItem name="ServiceSuccess" value="false"/>
  </ConfigGroup>
  <ConfigGroup name="ClientTimeoutValues">
    <ConfigItem name="DisconnectWait" value="120000"/>
    <ConfigItem name="EventTimeout" value="60000"/>
    <ConfigItem name="HighWaterMark" value="200"/>
    <ConfigItem name="KaWait" value="60000"/>
    <ConfigItem name="LowWaterMark" value="50"/>
    <ConfigItem name="QueueAccessWaitLimit" value="200"/>
    <ConfigItem name="QueueBlockLimit" value="500"/>
    <ConfigItem name="QueuePushWaitLimit" value="200"/>
    <ConfigItem name="TransactionLifeTime" value="20000"/>
  </ConfigGroup>
  <ConfigGroup name="ClusterConfig">
    <ConfigItem name="EventsOutStanding" value="10"/>
    <ConfigItem name="HeartBeatInterval" value="120000"/>
    <ConfigItem name="SeperateLog" value="false"/>
  </ConfigGroup>
  <ConfigGroup name="EnvironmentConfig">
    <ConfigItem name="JavaVendor" value="Sun Microsystems Inc."/>
    <ConfigItem name="JavaVersion" value="1.4.1_02"/>
    <ConfigItem name="OSArchitecture" value="x86"/>
    <ConfigItem name="OSName" value="Windows XP"/>
    <ConfigItem name="OSVersion" value="5.1"/>
    <ConfigItem name="ServerBuildDate" value="01-Jan-2005"/>
    <ConfigItem name="ServerBuildNumber" value="4000"/>
    <ConfigItem name="ServerVersion" value="$Name: $ - $Revision: 1.1 $"/>
  </ConfigGroup>
  <ConfigGroup name="EventStorage">
    <ConfigItem name="ActiveDelay" value="1000"/>
    <ConfigItem name="CacheAge" value="86400000"/>
    <ConfigItem name="IdleDelay" value="60000"/>
    <ConfigItem name="ThreadPoolSize" value="1"/>
  </ConfigGroup>
  <ConfigGroup name="FanoutValues">
    <ConfigItem name="ConcurrentUser" value="5"/>
    <ConfigItem name="KeepAlive" value="60000"/>
    <ConfigItem name="MaxBufferSize" value="1048576"/>
    <ConfigItem name="OutputBlockSize" value="1400"/>
    <ConfigItem name="PublishDelay" value="10"/>
    <ConfigItem name="PublishExpiredEvents" value="true"/>
    <ConfigItem name="QueueHighWaterMark" value="100"/>
  </ConfigGroup>

```

```

<ConfigItem name="QueueLowWaterMark" value="50"/>
<ConfigItem name="RoundRobinDelivery" value="false"/>
</ConfigGroup>
<ConfigGroup name="GlobalValues">
<ConfigItem name="ConnectionDelay" value="60000"/>
<ConfigItem name="ExtendedMessageSelector" value="false"/>
<ConfigItem name="HandshakeTimeout" value="5000"/>
<ConfigItem name="MaxNoOfConnections" value="-1"/>
<ConfigItem name="NHPScanTime" value="5000"/>
<ConfigItem name="NHPTimeout" value="120000"/>
<ConfigItem name="SchedulerPoolSize" value="2"/>
<ConfigItem name="SecureHandshake" value="true"/>
<ConfigItem name="SendRealmSummaryStats" value="false"/>
<ConfigItem name="ServerTime" value="true"/>
<ConfigItem name="StampDictionary" value="false"/>
<ConfigItem name="StatusBroadcast" value="5000"/>
<ConfigItem name="StatusUpdateTime" value="9223372036854775807"/>
<ConfigItem name="SupportVersion2Clients" value="true"/>
<ConfigItem name="fLoggerLevel" value="1"/>
</ConfigGroup>
<ConfigGroup name="JVMMManagement">
<ConfigItem name="EmergencyThreshold" value="94"/>
<ConfigItem name="ExitOnDiskIOError" value="true"/>
<ConfigItem name="ExitOnMemoryError" value="true"/>
<ConfigItem name="MemoryMonitoring" value="100"/>
<ConfigItem name="WarningThreshold" value="85"/>
</ConfigGroup>
<ConfigGroup name="JoinConfig">
<ConfigItem name="ActiveThreadPoolSize" value="2"/>
<ConfigItem name="IdleThreadPoolSize" value="1"/>
<ConfigItem name="MaxEventsPerSchedule" value="50"/>
<ConfigItem name="MaxQueueSizeToUse" value="100"/>
</ConfigGroup>
<ConfigGroup name="RecoveryDaemon">
<ConfigItem name="EventsPerBlock" value="500"/>
<ConfigItem name="ThreadPool" value="4"/>
</ConfigGroup>
<ConfigGroup name="TransactionManager">
<ConfigItem name="MaxEventsPerTransaction" value="0"/>
<ConfigItem name="MaxTransactionTime" value="86400000"/>
<ConfigItem name="TTLThreshold" value="1000"/>
</ConfigGroup>
</RealmConfiguration>
<RealmPermissionSet>
<RealmACLEntry addremoveChannels="false" addremoveJoins="false"
  addremoveRealms="false" changeRealmConfig="false" connectToRealm="true"
  createP2PService="false" fullControl="false" host="*"
  listACLEntries="false" modifyACLEntries="false" name="*"
  overrideConnectionCount="false" useAdminAPI="false"/>
<RealmACLEntry addremoveChannels="true" addremoveJoins="true"
  addremoveRealms="true" changeRealmConfig="true" connectToRealm="true"
  createP2PService="true" fullControl="true" host="192.168.1.2"
  listACLEntries="true" modifyACLEntries="true" name="johnsmith"
  overrideConnectionCount="true" useAdminAPI="true"/>
<RealmACLEntry addremoveChannels="false" addremoveJoins="false"
  addremoveRealms="false" changeRealmConfig="false" connectToRealm="false"
  createP2PService="false" fullControl="true" host="localhost"
  listACLEntries="false" modifyACLEntries="false" name="johnsmith"
  overrideConnectionCount="false" useAdminAPI="false"/>
<RealmACLEntry addremoveChannels="false" addremoveJoins="false"
  addremoveRealms="false" changeRealmConfig="false" connectToRealm="true"
  createP2PService="false" fullControl="false" host="192.168.1.2"
  listACLEntries="false" modifyACLEntries="false"

```

```

name="realm-productionmaster" overrideConnectionCount="false"
useAdminAPI="false"/>
<RealmACLEntry addremoveChannels="false" addremoveJoins="false"
addremoveRealms="false" changeRealmConfig="false" connectToRealm="true"
createP2PService="false" fullControl="false" host="192.168.1.2"
listACLEntries="false" modifyACLEntries="false"
name="realm-productionslave1" overrideConnectionCount="false"
useAdminAPI="false"/>
<RealmACLEntry addremoveChannels="false" addremoveJoins="false"
addremoveRealms="false" changeRealmConfig="false" connectToRealm="true"
createP2PService="false" fullControl="false" host="192.168.1.2"
listACLEntries="false" modifyACLEntries="false"
name="realm-productionslave2" overrideConnectionCount="false"
useAdminAPI="false"/>
</RealmPermissionSet>
<ClusterSet>
<ClusterEntry name="productioncluster">
<ClusterMember name="productionmaster" rname="nsp://192.168.1.1:9000"/>
<ClusterMember name="productionslave1" rname="nsp://192.168.1.2:9000"/>
<ClusterMember name="productionslave2" rname="nsp://192.168.1.3:9000"/>
</ClusterEntry>
</ClusterSet>
<ChannelSet>
<ChannelEntry>
<ChannelAttributesEntry EID="0" TTL="0" capacity="0" clusterWide="true"
name="/customer/sales" type="MIXED_TYPE"/>
<ChannelPermissionSet>
<ChannelACLEntry fullControl="false" getLastEID="false" host="*"
listACLEntries="false" modifyACLEntries="false" name="*" publish="false"
purgeEvents="false" subscribe="true" useNamedSubscription="false"/>
<ChannelACLEntry fullControl="true" getLastEID="true" host="192.168.1.2"
listACLEntries="true" modifyACLEntries="true" name="johnsmith"
publish="true" purgeEvents="true" subscribe="true"
useNamedSubscription="false"/>
</ChannelPermissionSet>
</ChannelEntry>
<ChannelEntry>
<ChannelAttributesEntry EID="0" TTL="0" capacity="0" clusterWide="true"
name="/naming/defaultContext" type="MIXED_TYPE"/>
<ChannelPermissionSet>
<ChannelACLEntry fullControl="false" getLastEID="true" host="*"
listACLEntries="false" modifyACLEntries="false" name="*" publish="false"
purgeEvents="false" subscribe="true" useNamedSubscription="true"/>
<ChannelACLEntry fullControl="true" getLastEID="true" host="192.168.1.2"
listACLEntries="true" modifyACLEntries="true" name="johnsmith"
publish="true" purgeEvents="true" subscribe="true"
useNamedSubscription="false"/>
</ChannelPermissionSet>
<ChannelKeySet>
<ChannelKeyEntry keyDepth="1" keyName="alias"/>
</ChannelKeySet>
</ChannelEntry>
<ChannelEntry>
<ChannelAttributesEntry EID="0" TTL="0" capacity="0" clusterWide="true"
name="/partner/sales"
type="MIXED_TYPE"/>
<ChannelPermissionSet>
<ChannelACLEntry fullControl="false" getLastEID="true" host="*"
listACLEntries="false" modifyACLEntries="false" name="*" publish="false"
purgeEvents="false" subscribe="true" useNamedSubscription="true"/>
<ChannelACLEntry fullControl="true" getLastEID="true" host="192.168.1.2"
listACLEntries="true" modifyACLEntries="true" name="johnsmith"
publish="true" purgeEvents="true" subscribe="true"

```

```

    useNamedSubscription="false"/>
  </ChannelPermissionSet>
</ChannelEntry>
</ChannelSet>
<QueueSet>
<QueueEntry>
<ChannelAttributesEntry EID="0" TTL="0" capacity="0" clusterWide="true"
  name="/customer/queries" type="MIXED_TYPE"/>
<QueuePermissionSet>
<QueueACLEntry fullControl="false" host="*" listACLEntries="false"
  modifyACLEntries="false" name="*" peek="true" pop="false" purge="false"
  push="false"/>
<QueueACLEntry fullControl="true" host="192.168.1.2" listACLEntries="true"
  modifyACLEntries="true" name="johnsmith" peek="true" pop="true"
  purge="true" push="true"/>
</QueuePermissionSet>
</QueueEntry>
<QueueEntry>
<ChannelAttributesEntry EID="0" TTL="0" capacity="0" clusterWide="true"
  name="/partner/queries" type="MIXED_TYPE"/>
<QueuePermissionSet>
<QueueACLEntry fullControl="false" host="*" listACLEntries="false"
  modifyACLEntries="false" name="*" peek="true" pop="false" purge="false"
  push="false"/>
<QueueACLEntry fullControl="true" host="192.168.1.2" listACLEntries="true"
  modifyACLEntries="true" name="johnsmith" peek="true" pop="true" purge="true" push="true"/>
</QueuePermissionSet>
</QueueEntry>
</QueueSet>
<RealmInterfaces>
<RealmNSPInterface>
<RealmInterface acceptThreads="2" adapter="0.0.0.0" advertise="true"
  authtime="10000" autostart="true" backlog="100" name="nsp0" port="9000"/>
<InterfacePermissionSet>
<InterfaceACLEntry host="192.168.1.2" name="johnsmith"/>
</InterfacePermissionSet>
</RealmNSPInterface>
<RealmNHPInterface>
<RealmInterface acceptThreads="2" adapter="0.0.0.0" advertise="true"
  authtime="10000" autostart="true" backlog="100" name="nhp0" port="80"/>
</RealmNHPInterface>
</RealmInterfaces>
</Universal MessagingRealm>

```

Management and Monitoring Sections

Enterprise view

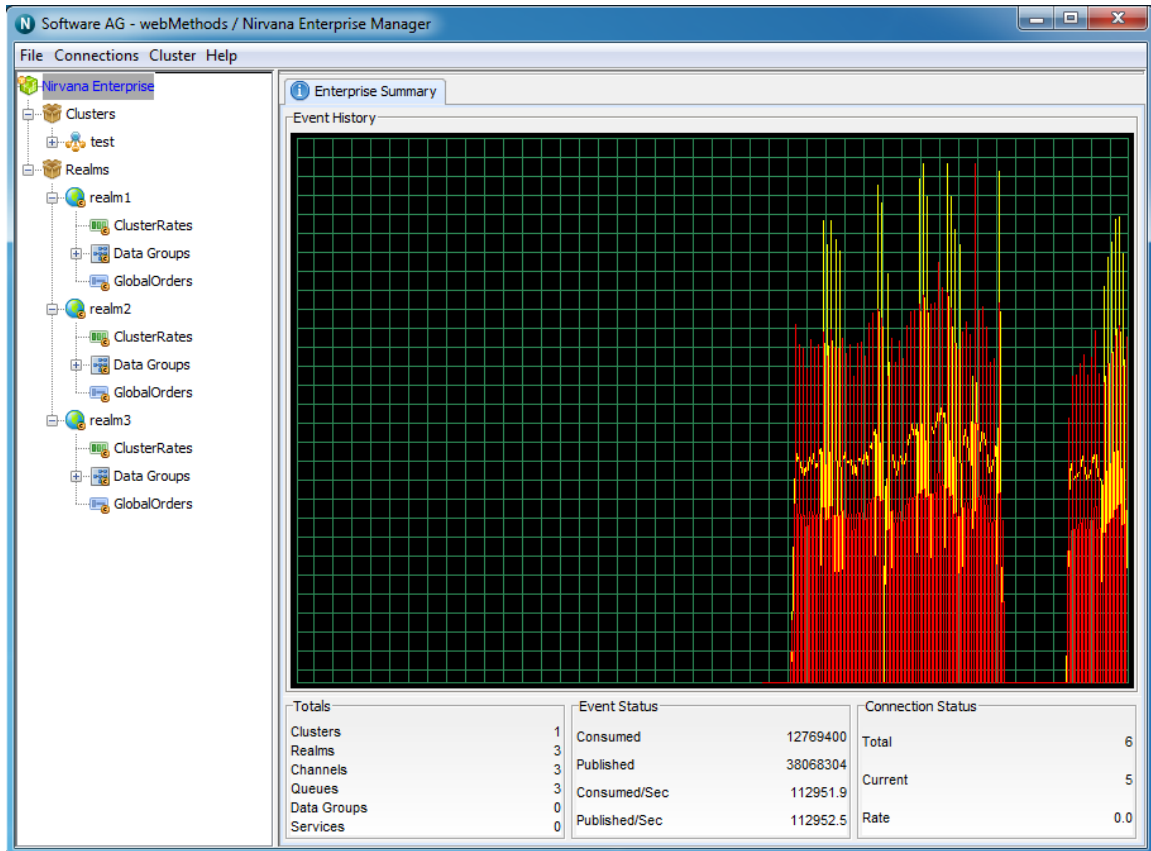
Universal Messaging Enterprise Manager facilitates centralised summary view for a set of Universal Messaging realm servers whether standalone or defined in a cluster setup.

- ["Enterprise Summary" on page 257](#)
- ["Cluster Summary" on page 257](#)

Enterprise Summary

The Universal Messaging Enterprise Manager provides a summary view of all Universal Messaging realms whether clustered or standalone, publish and consume event / connection totals and rates across the entire set of Realms that the instance of the Enterprise Manager is connected to.

For more information on these screens please see the Management Information section.



Cluster Summary

The Universal Messaging Enterprise Manager tool provides a summary view for each cluster defined, showing a real-time cluster status as well as publish and consume event / connections totals and rates.

The screenshot displays the Nirvana Enterprise Manager interface. On the left is a tree view showing the hierarchy: Nirvana Enterprise > Clusters > test > Realms > realm1, realm2, and realm3. Each realm contains sub-objects: ClusterRates, Data Groups, and GlobalOrders. The main window is titled 'Cluster Summary' and contains a table of cluster members and an event history graph.

Name	State	Master	realm1	realm2	realm3
realm1	Master	realm1	Local	OnLine	OnLine
realm2	Slave	realm1	OnLine	Local	OnLine
realm3	Slave	realm1	OnLine	OnLine	Local

Below the table is an 'Event History' graph showing a fluctuating line chart with red and yellow bars on a green grid background.

Totals		Event Status		Connection Status	
Realms	3	Consumed	15980469	Total	6
Channels	3	Published	47864694	Current	5
Queues	3	Consumed/Sec	111474.1	Rate	0.0
Services	0	Published/Sec	225065.8		

Management Information

Each Universal Messaging Administration API client, (the Enterprise Manager is an admin API client) connects to one or more realms and asynchronously consumes status events on all of the objects within each realm.

Status events are sent periodically (between configurable intervals, see "[Realm Configuration](#)" on page 25) and contain information pertaining to those objects where activity has occurred.

As each status event is received, the Enterprise Manager is updated with the relevant values and these are displayed on the status panels for each object within the namespace.

Selecting an object from the namespace automatically renders a set of panels to the right of the selected node, one of which is the 'Status' panel. The status panels and their contents are described in the links below.

The Universal Messaging enterprise manager has been written entirely with the Universal Messaging admin API and so the functionality seen in these screens can easily be added to any bespoke administration or monitoring tools.

- "[Enterprise Summary](#)" on page 259

- "Clusters Summary" on page 261
- "Cluster Status" on page 262
- "Realms Summary" on page 264
- "Realm Status" on page 266
- "Realm Monitoring" on page 268
- "Container Status" on page 283
- "Container Monitor Panel" on page 285
- "Channel Status Information" on page 288
- "Data Group Status Information" on page 290
- "Channel Connections" on page 292
- "Queue Status Information" on page 295
- "Interface Status Information" on page 297

For more information on other functionality provided in the enterprise manager please refer to the Enterprise Manager guide.

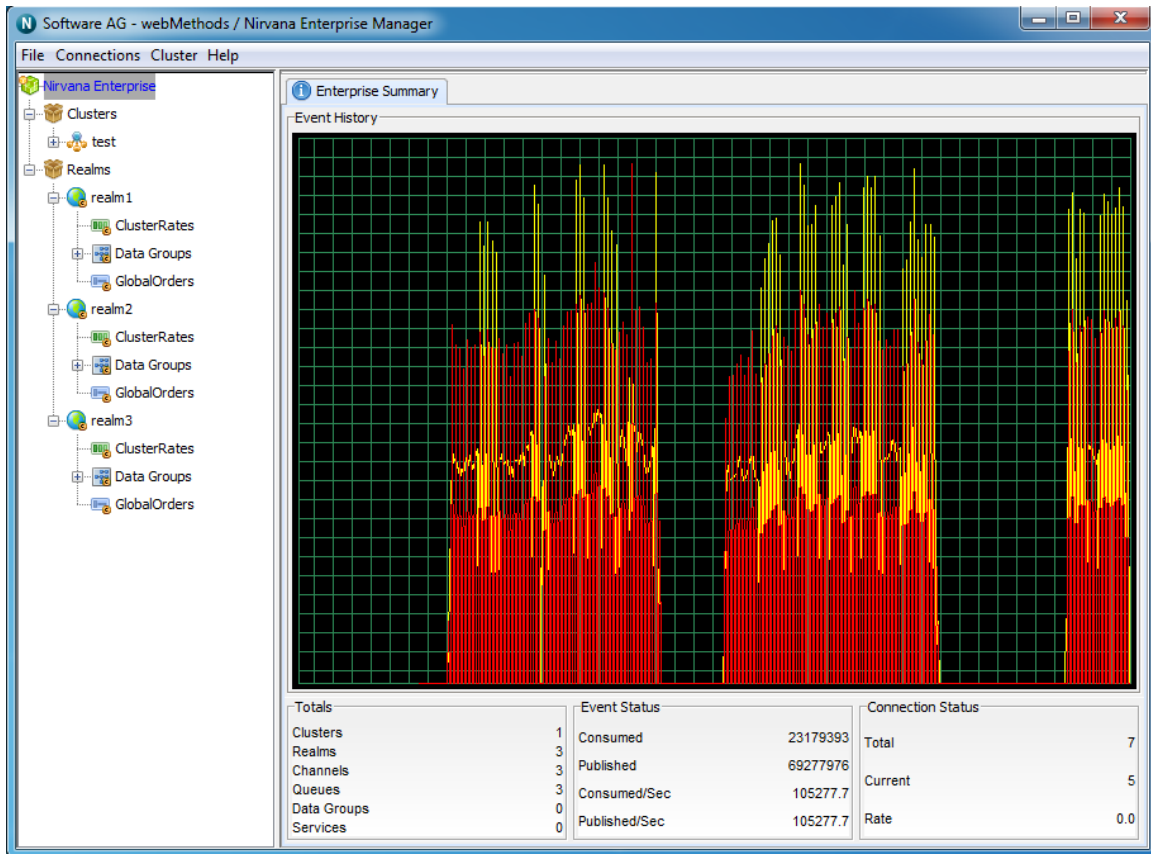
Enterprise Summary

The enterprise view is the first screen you see whenever the Universal Messaging 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

This section describes the type of status information that can be observed from the Enterprise level 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 realms.

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 5 values :

- **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 4 values:

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

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

Connection Status

The Connection Status section describes 3 values :

- **Total** - The total number of connections made to all realms
- **Current** - The current number of events across all realms
- **Rate** - The number of connections being made per second across all realms at this point in time

Clusters Summary

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 :

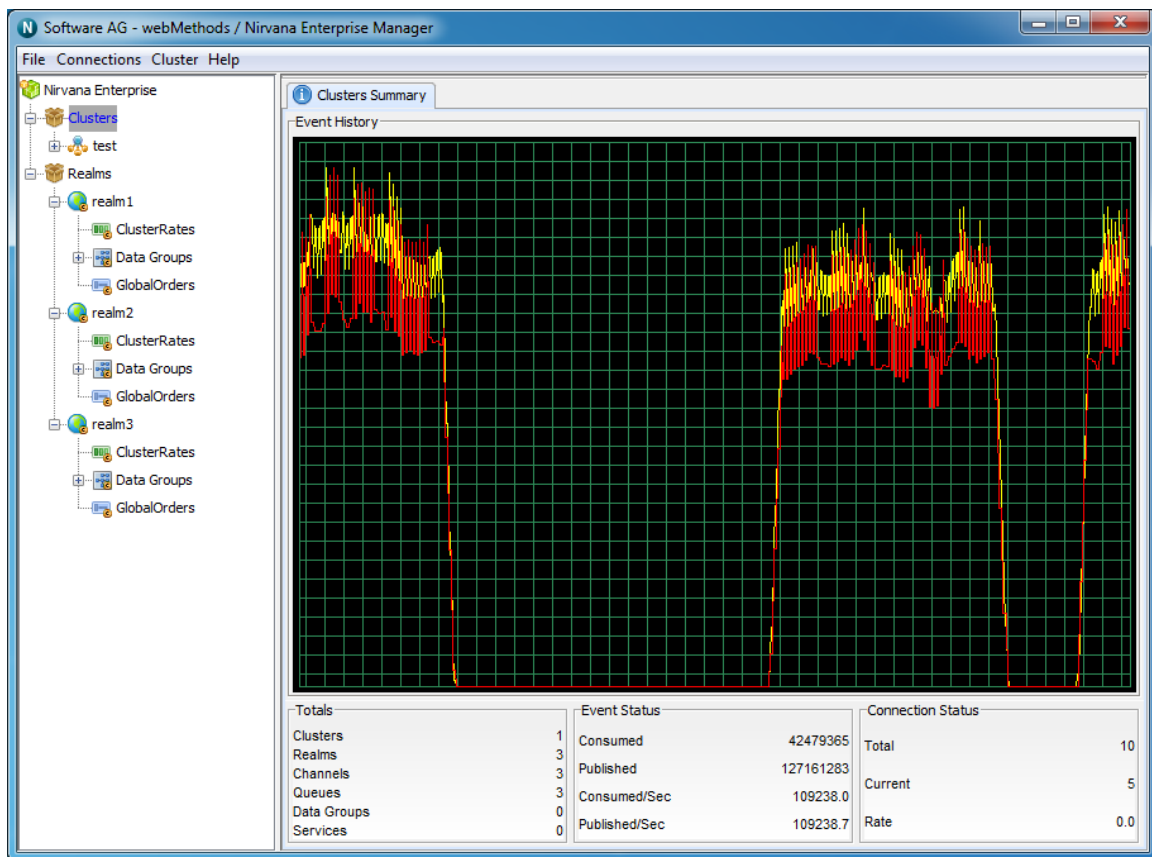
- **Published** - The total number of events published to all channels, queues and services across all realms within known clusters

- **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



Clusters Status

The cluster status view provides an overview of the characteristics as well as current status of a selected Universal Messaging cluster.

This section will describe the type of status information that you can observe from the Cluster Status 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 :

- **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

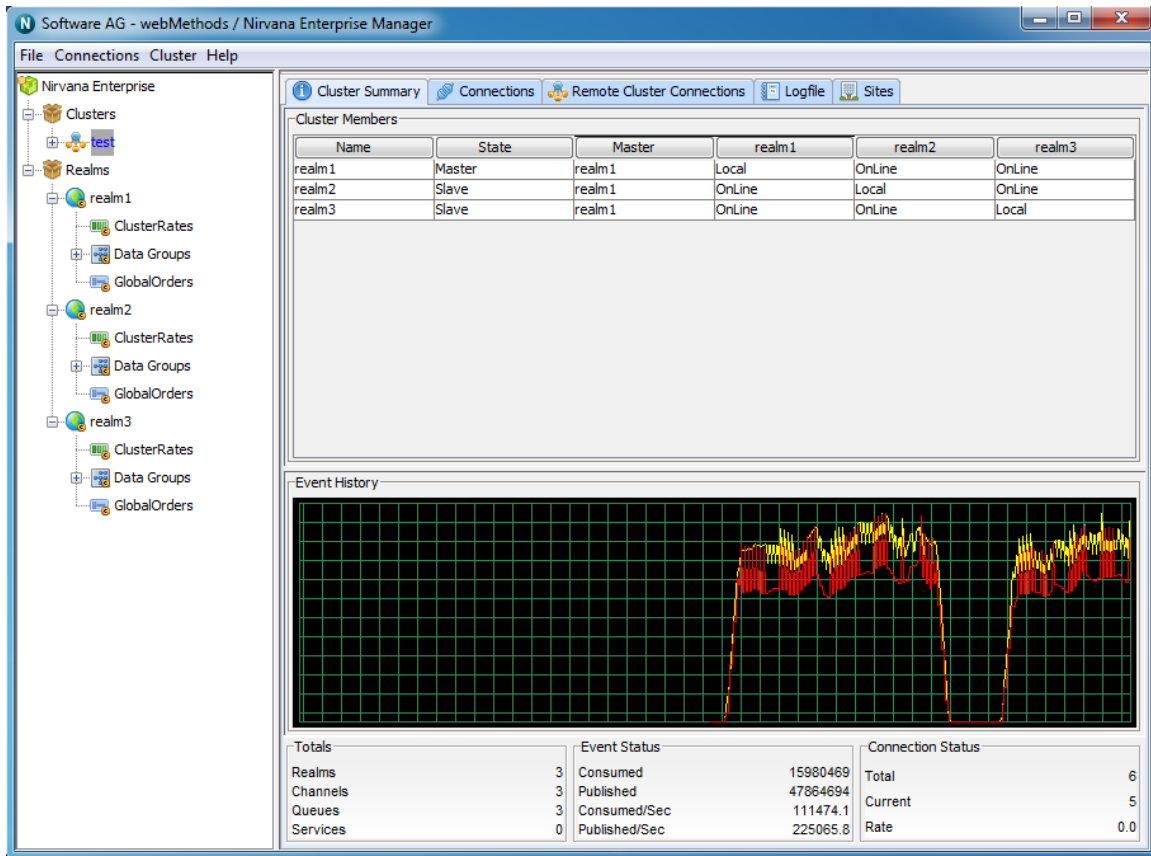
The Event Status section describes the following :

- **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



Realms Summary

The realms view 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 aware of.

This section will describe the type of status information that you can observe from the Realms 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 realms.

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

Totals

The Totals section contains the following values :

- **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

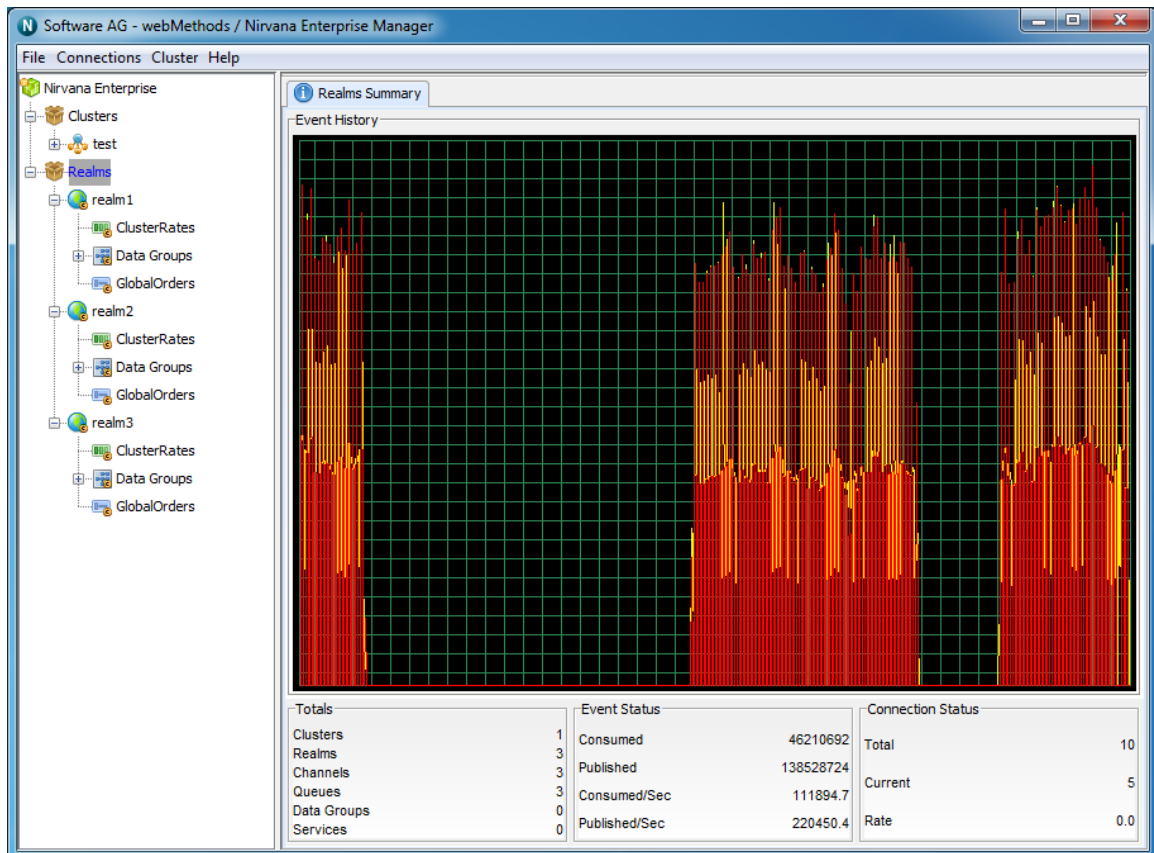
The Event Status section contains the following :

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

Connection Status

The Connection Status section contains the following :

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



Realm Status

The realm status view provides an overview of the characteristics as well as current status of a selected Universal Messaging realm. When you select a Realm node from the namespace, the status panel is displayed by default for the realm.

This section will describe the type of status information that you can observe from the Realm Status view.

The top of the screen displays a panel containing 4 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 top of the Status panel contains 2 large real time graphs illustrating the total number of events published (yellow) and consumed (red) across the Universal Messaging Realm, as well as the JVM memory status 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 4 values :

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

Connection Status

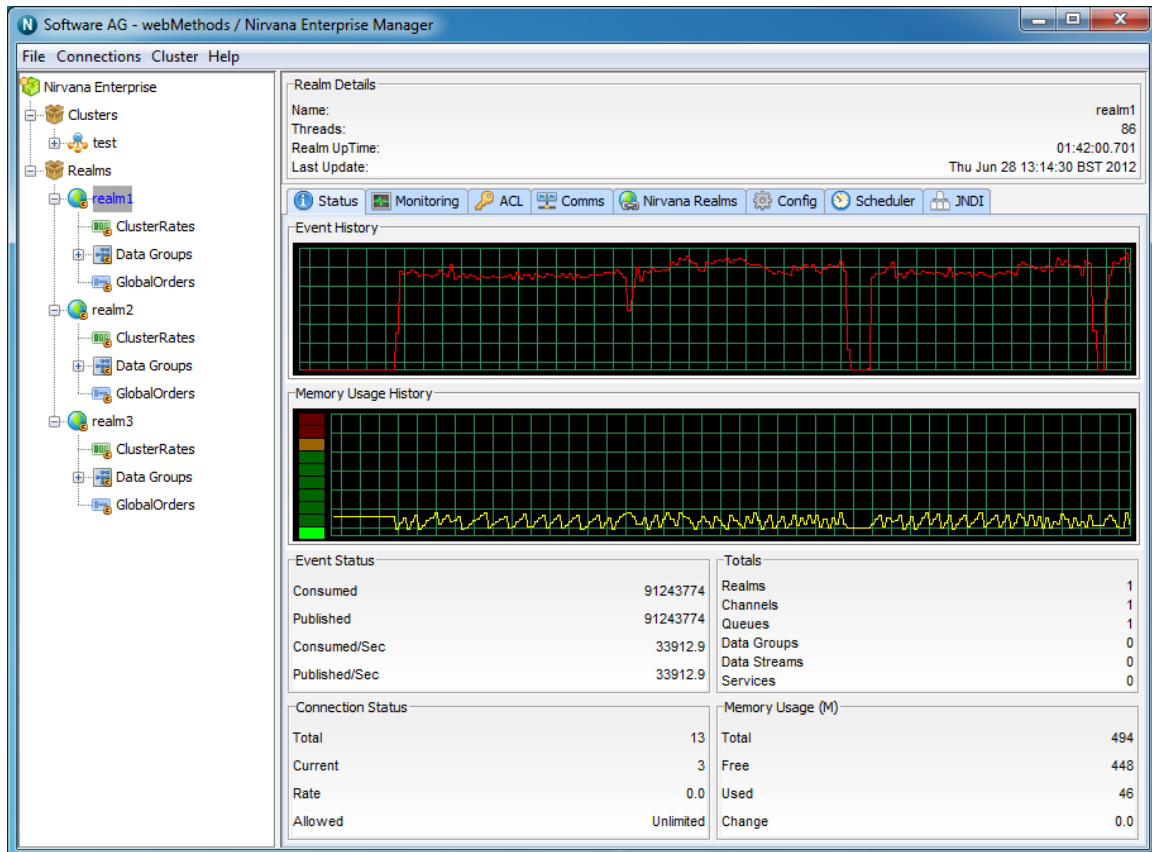
The Connection Status section contains the following values :

- **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

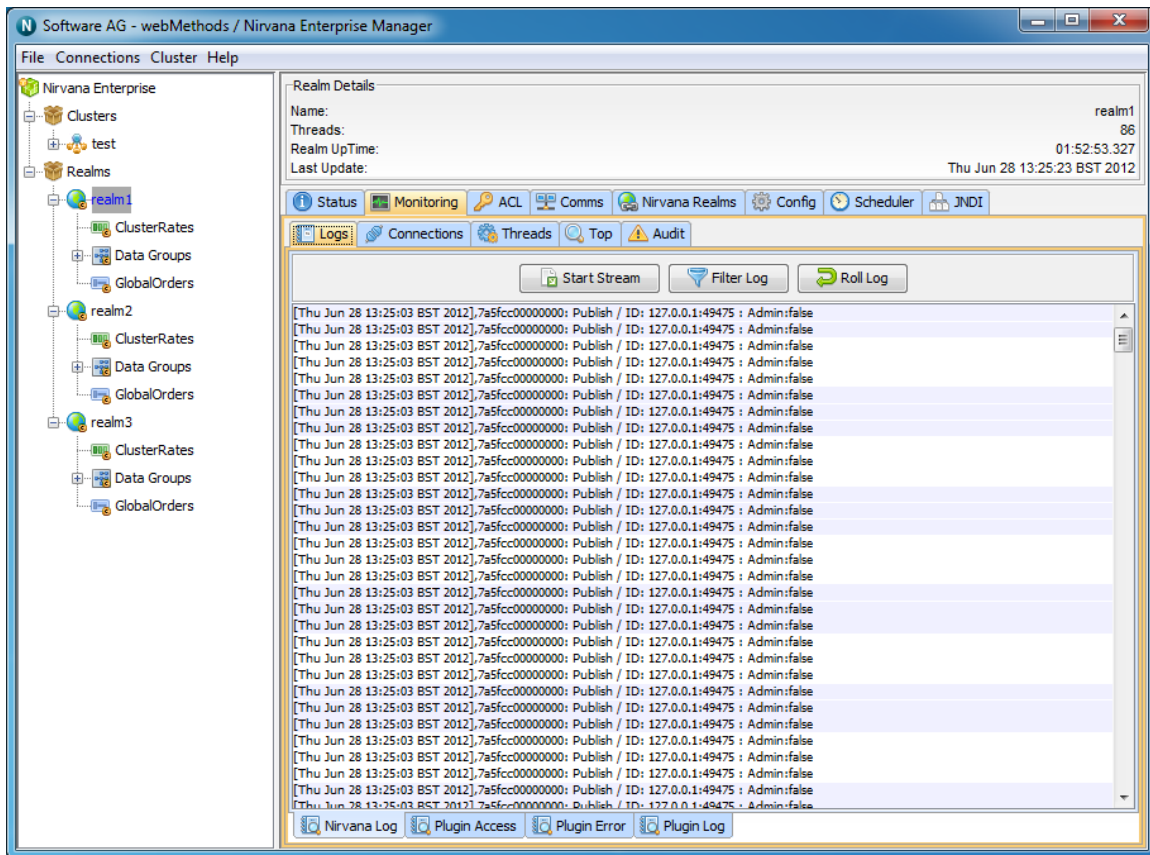


Realm Monitoring

In Universal Messaging 7 we have reorganized the panels associated with Realm nodes. 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 268
- "Realm Connections" on page 272
- "Threads Panel" on page 275
- "Top" on page 277
- "Audit" on page 280



Universal Messaging Enterprise Manager : Logs Panel

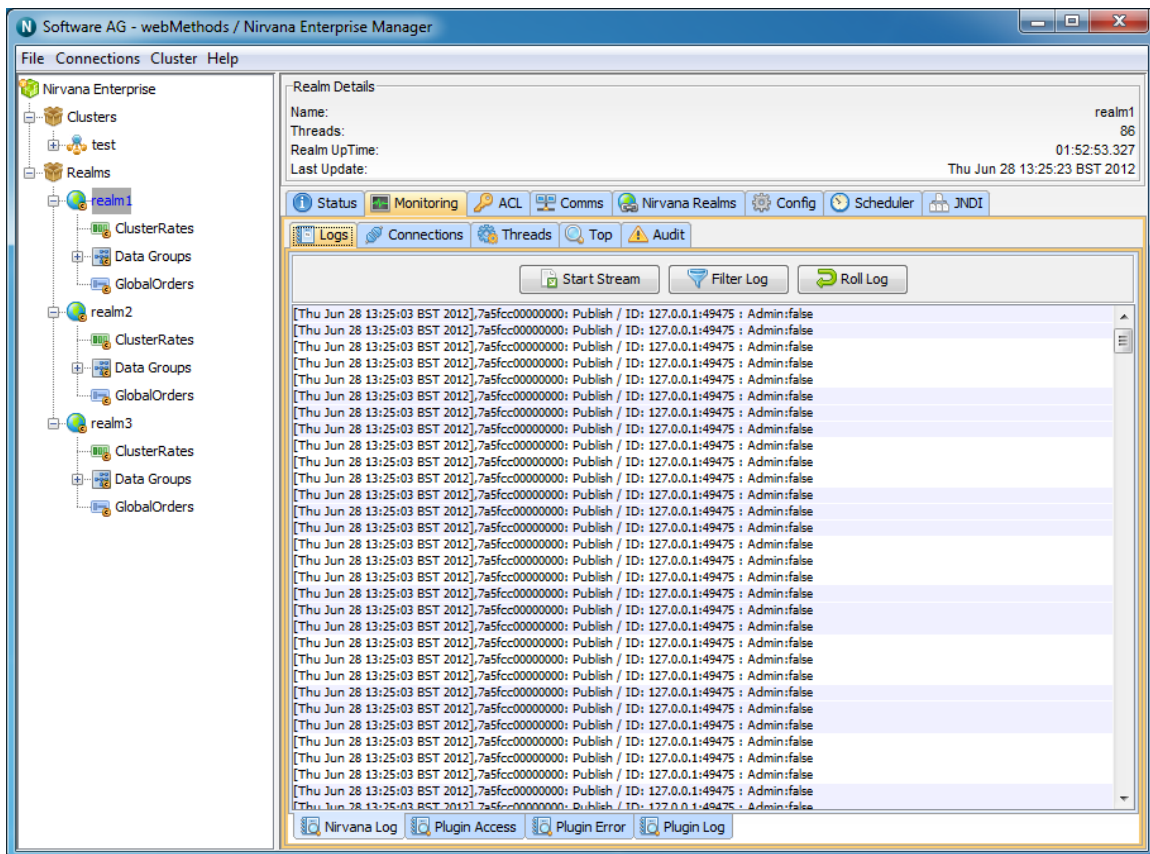
Each Universal Messaging Realm Server has a log file within the data directory called nirvana.log. The Enterprise Manager provides a panel that displays real time log messages as they are written to the log file. This enables you to remotely view the activity on a realm as it is happening. The Universal Messaging Administration API also

provides the ability to consume the log file entries from an nRealmNode. See the code example "Realm Log and Audit Listener" for an illustration of usage.

The Universal Messaging log file will contain 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 log files can be viewed. To view the log files, click on the realm node from the namespace and select the panel labeled 'Monitor' 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

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:

```
[Wed Jun 20 09:37:39 BST 2012],=====
[Wed Jun 20 09:37:39 BST 2012], Copyright . All rights reserved
[Wed Jun 20 09:37:39 BST 2012], Start date = Wed Jun 20 09:37:39 BST 2012
[Wed Jun 20 09:37:39 BST 2012], Process ID = 3216
[Wed Jun 20 09:37:39 BST 2012],
[Wed Jun 20 09:37:39 BST 2012], Realm Server Details :
[Wed Jun 20 09:37:39 BST 2012], Realm Server name = realm1
[Wed Jun 20 09:37:39 BST 2012], Realm Server version = $Name: $ - $Revision: 1.7 $
[Wed Jun 20 09:37:39 BST 2012], Build Number = Build 11248
[Wed Jun 20 09:37:39 BST 2012], Build Date = June 19 2012
[Wed Jun 20 09:37:39 BST 2012], Data Directory =
    C:\\Universal Messaging 7.0.11248\\server\\realm1\\data
[Wed Jun 20 09:37:39 BST 2012], Extension Directory =
    C:\\Universal Messaging 7.0.11248\\server\\realm1\\plugins\\ext
[Wed Jun 20 09:37:39 BST 2012], Low Latency Executor = false
[Wed Jun 20 09:37:39 BST 2012], Realm(s) Reloaded = 1
[Wed Jun 20 09:37:39 BST 2012], Channels Reloaded = 8
[Wed Jun 20 09:37:39 BST 2012], Queues Reloaded = 0
[Wed Jun 20 09:37:39 BST 2012], Interfaces Reloaded = 1
[Wed Jun 20 09:37:39 BST 2012],
[Wed Jun 20 09:37:39 BST 2012], Operating System Environment :
[Wed Jun 20 09:37:39 BST 2012], OS Name = Windows 7
[Wed Jun 20 09:37:39 BST 2012], OS Version = 6.1
[Wed Jun 20 09:37:39 BST 2012], OS Architecture = x86
[Wed Jun 20 09:37:39 BST 2012], Available Processors = 4
[Wed Jun 20 09:37:39 BST 2012],
[Wed Jun 20 09:37:39 BST 2012], Java Environment :
[Wed Jun 20 09:37:39 BST 2012], Java Vendor = Sun Microsystems Inc.
[Wed Jun 20 09:37:39 BST 2012], Java Vendor URL = http://java.sun.com/
[Wed Jun 20 09:37:39 BST 2012], Java Version = 1.6.0_30
[Wed Jun 20 09:37:39 BST 2012], Memory Allocation = 494 MB
[Wed Jun 20 09:37:39 BST 2012], Memory Warning = 420 MB
[Wed Jun 20 09:37:39 BST 2012], Memory Emergency = 465 MB
[Wed Jun 20 09:37:39 BST 2012], Clock Adjustment = 0ms
[Wed Jun 20 09:37:39 BST 2012], Startup: Starting Realm status monitoring
[Wed Jun 20 09:37:39 BST 2012], Nanosecond delay = Supported
[Wed Jun 20 09:37:39 BST 2012], Time Zone = Greenwich Mean Time
[Wed Jun 20 09:37:39 BST 2012], Startup: Stored Certificate and private key
    in servers keystore
[Wed Jun 20 09:37:39 BST 2012], Startup: Completed Realm Public and Private RSA Key
[Wed Jun 20 09:37:39 BST 2012], Security Provider 0 = SUN version 1.6
[Wed Jun 20 09:37:39 BST 2012], Startup: Reloading Realm Public for realm1
[Wed Jun 20 09:37:39 BST 2012], Startup: Cluster cryptographic initialisation, complete
[Wed Jun 20 09:37:39 BST 2012], Security Provider 1 = SunRsaSign version 1.5
[Wed Jun 20 09:37:39 BST 2012], Security Provider 2 = SunJSSE version 1.6
[Wed Jun 20 09:37:39 BST 2012], Security Provider 3 = SunJCE version 1.6
[Wed Jun 20 09:37:39 BST 2012], Security Provider 4 = SunJGSS version 1.0
[Wed Jun 20 09:37:39 BST 2012], Security Provider 5 = SunSASL version 1.5
[Wed Jun 20 09:37:39 BST 2012], Security Provider 6 = XMLDSig version 1.0
[Wed Jun 20 09:37:39 BST 2012], Security Provider 7 = SunPCSC version 1.6
[Wed Jun 20 09:37:39 BST 2012], Security Provider 8 = SunMSCAPI version 1.6
[Wed Jun 20 09:37:39 BST 2012],=====
```

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 Server, as well as environmental information like, OS, Java version, timezone.

Each log entry contains a date, the log level as well as the log message itself, in the format:

```
[DATE_TIME], LOG_LEVEL, Message
```

The Universal Messaging log level is a level from 0 to 7 that determines what information is written to the log. Log level 0 is the most verbose level of logging and on a heavily utilised server will produce a lot of log output. Log level 7 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 7. The following list explains the log file messages levels and how they correspond to the values:

- 0 - Success (Log level 0 will output any log entries with a level of 0 or above)
- 1 - Informative (Log level 1 will output any log entries with a level of 1 or above)
- 2 - Warning (Log level 2 will output any log entries with a level of 2 or above)
- 3 - Failure (Log level 3 will output any log entries with a level of 3 or above)
- 4 - Fatal (Log level 4 will output any log entries with a level of 4 or above)
- 5 - Security (Log level 5 will output any log entries with a level of 5 or above)
- 6 - Audit (Log level 6 will output any log entries with a level of 6 or above)
- 7 - Log (Log level 7 will output only log entries with a level of 7)

Log levels can be changed dynamically on the server by using the Config Panel (see ["Realm Configuration" on page 25](#)). 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 `nserver.lax` file in the `server/bin` directory of your install. You need to modify the `DLOGSIZE` property within this file to change the size.

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 25](#)). 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`

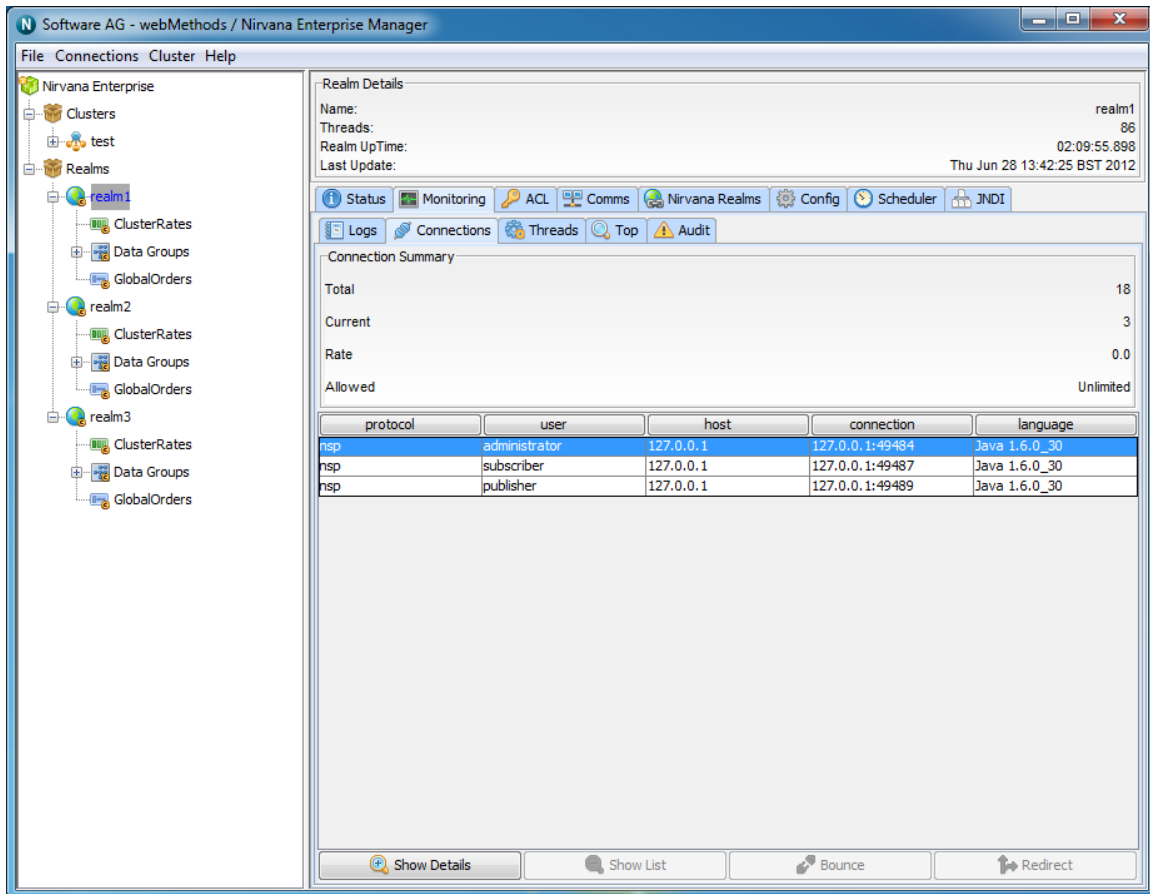
- **ROLLING_DATE** : The rolling date manager stores a configurable number of log files (RolledLogFileDepth). Rolled log files are stored with the date they were rolled appended to the active log file name. e.g. nirvana.logWed-Sep-14-02-31-40-117-BST-2011.
- **ROLLING_NUMBER** : The numbered log manager 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 347) 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 has 4 columns:

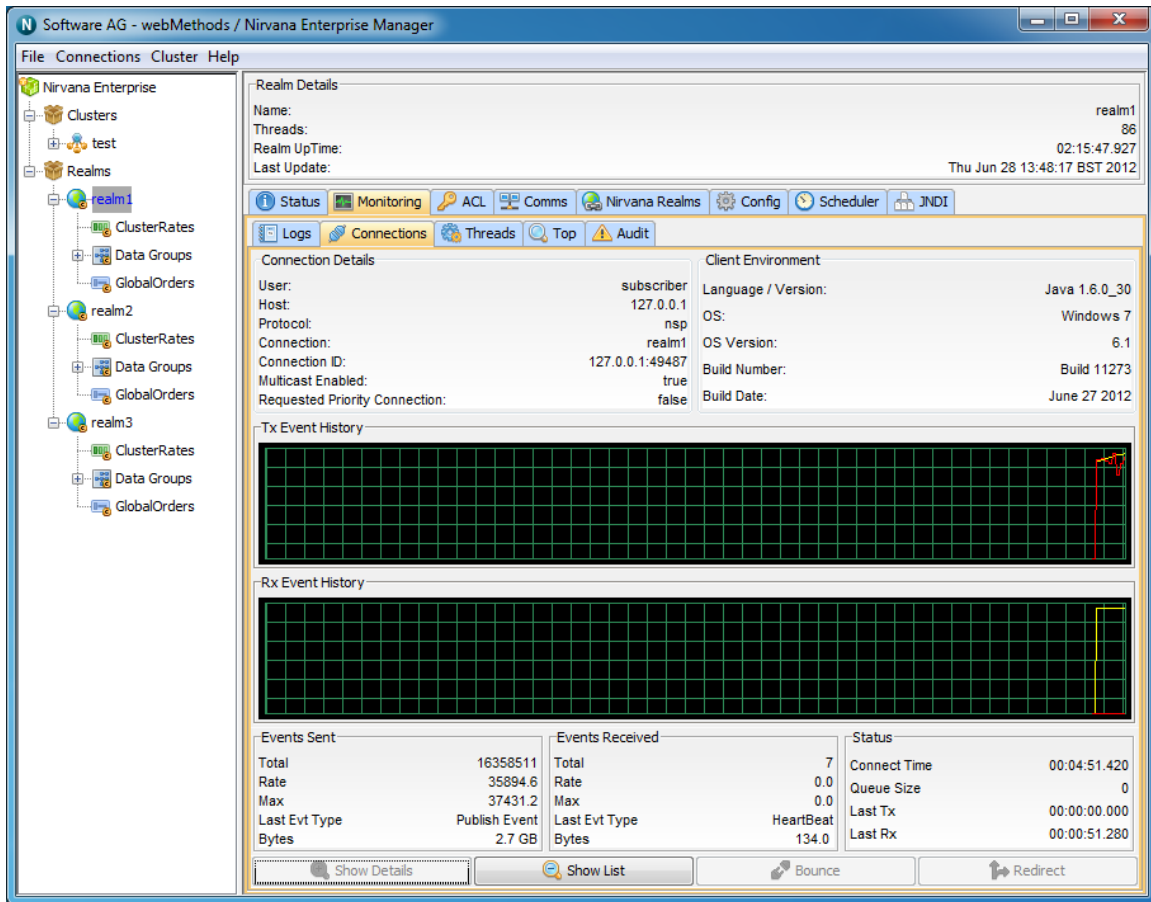
- **Protocol** - The protocol used in the connection
- **User** - The name of the user connected
- **Host** - The host machine that the user is connecting from
- **Connection** - The local connection id, defined as hostname:local_port

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 on the 'Disconnect' button.

Secondly, 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 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.

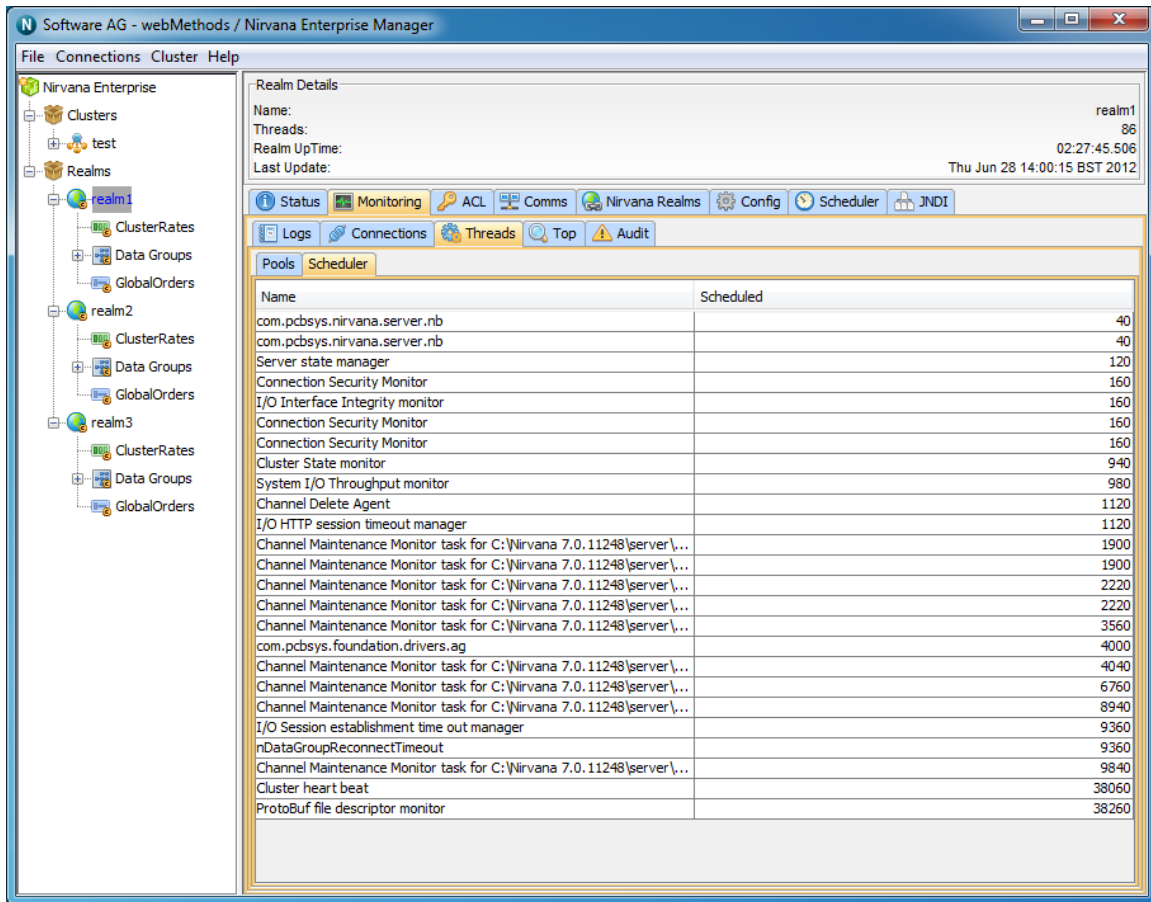
The screenshot displays the 'Scheduler' tab within the Nirvana Enterprise Manager. The interface includes a navigation tree on the left, a top menu bar, and a main content area. The main content area shows 'Thread Monitor' statistics and a 'Thread Pools' table.

Thread Monitor Summary:

- Scheduled Tasks: 30
- Reused Threads: 4
- Vended Threads: 74
- Queued Threads: 0
- Buffers Created: 1600
- Buffers Reused: 6552987
- Fanout backlog: 1

Thread Pools Table:

Name	Idle Threads	Allocated Threads	Queued Tasks	Tasks Executed
Scheduler Worker Pool	6	6	0	42789
CommonPool	3	3	0	29
Client Setup	4	4	0	21
PersistentStore	4	4	0	5094
Client session closer	1	1	0	0
Orphaned session closer	1	1	0	0
Cluster KeepAlive	4	4	0	1110
Inter Realm Connection ...	3	3	0	2
Flush-Monitor	0	4	5	207556
WritePool	5	5	0	397030
Recovery Pool	4	4	0	0
ClusterRecovery	5	5	0	0
KeepAlive	2	2	0	411
nhp0	0	1	0	23
HTTP-Plugins	2	2	0	0
Join Daemon	2	2	0	0
ReadPool	5	6	0	13395988
ClosePool	2	2	0	18



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 ["Realm Status" on page 266](#)) 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

The screenshot shows the Nirvana Enterprise Manager interface. The left sidebar displays a tree view of the system structure, including Clusters, Realms (realm1, realm2, realm3), ClusterRates, Data Groups, and GlobalOrders. The main panel is titled 'Nirvana Enterprise' and contains several sections:

- Realm Details:** Name: realm1, Threads: 86, Realm UpTime: 02:37:39.641, Last Update: Thu Jun 28 14:10:09 BST 2012.
- Monitoring Tools:** Status, Monitoring, ACL, Comms, Nirvana Realms, Config, Scheduler, JNDI.
- Navigation:** Logs, Connections, Threads, Top, Audit.
- Memory Usage History:** A line graph showing memory usage over time.
- JVM GC Stats:** A line graph showing JVM garbage collection statistics.
- Totals:**

Realms	1
Channels	1
Queues	1
Data Groups	0
Data Streams	0
Services	0
- Top Channel Usage Table:**

Name	Connections	Published	Consumed	Memory (...)	%Memory	Disk (KB)
/ClusterRates	1	36,413.316	36,413.316	0	0	0
/GlobalOrders	0	0	0	0	0	0

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.

The screenshot shows the Nirvana Enterprise Manager interface. On the left is a navigation tree with 'Nirvana Enterprise' expanded to show 'Clusters' (test) and 'Realms' (realm1, realm2, realm3). The main area displays 'Realm Details' for 'realm1' with the following information:

- Name: realm1
- Threads: 86
- Realm UpTime: 02:38:29.980
- Last Update: Thu Jun 28 14:10:59 BST 2012

Below the details are several monitoring panels: 'Memory Usage History' (a line graph), 'JVM GC Stats' (a line graph), and a 'Totals' table:

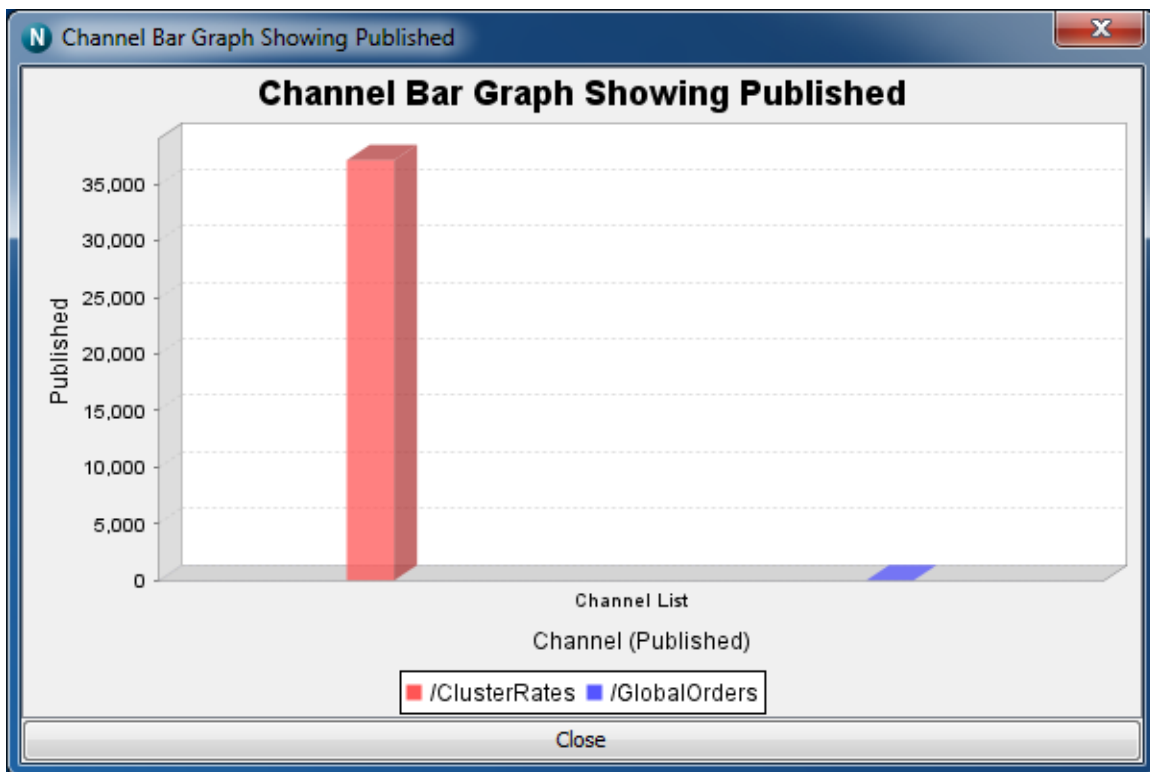
Totals	
Realms	1
Channels	1
Queues	1
Data Groups	0
Data Streams	0
Services	0

The 'Top Connection Usage' table is sorted by 'Events In' and contains the following data:

ID	Queued	Events In	Bytes In	Events Out	Bytes Out	Latency
publisher@127.0.0.1:49498	0	19,887,534	3,062,679,880	9	226	0
administrator@127.0.0.1:49493	0	161	1,282	1,579	775,466	0
subscriber@127.0.0.1:49487	0	23	242	49,987,639	9,173,506,597	0

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/) (<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 Universal MessagingAudit.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 "Realm Log and Audit Listener" 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 25) of the Universal Messaging Enterprise Manager.

Audit Events

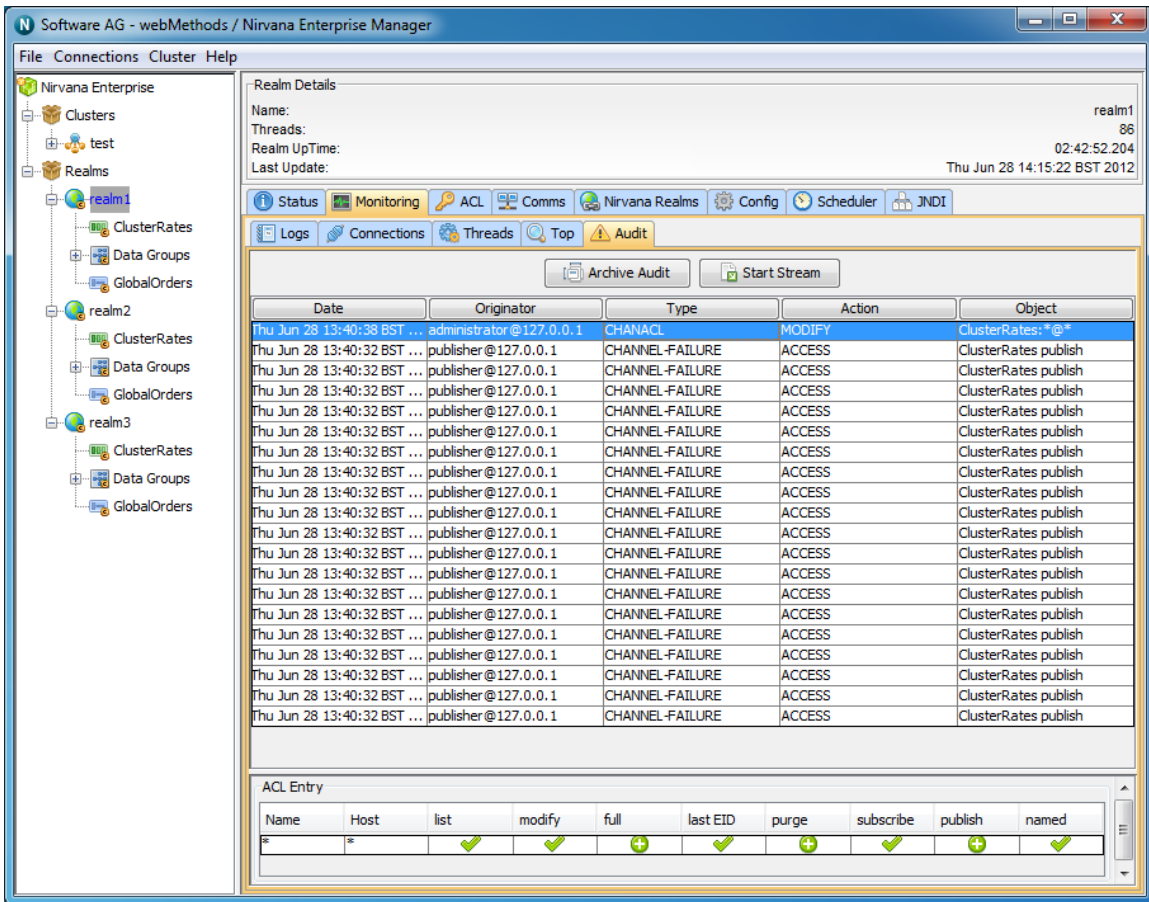
Each audit event corresponds to an operation performed on an object within a realm. The audit event contains the date on which it occurred, the object and the operation that was performed on the object.

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:

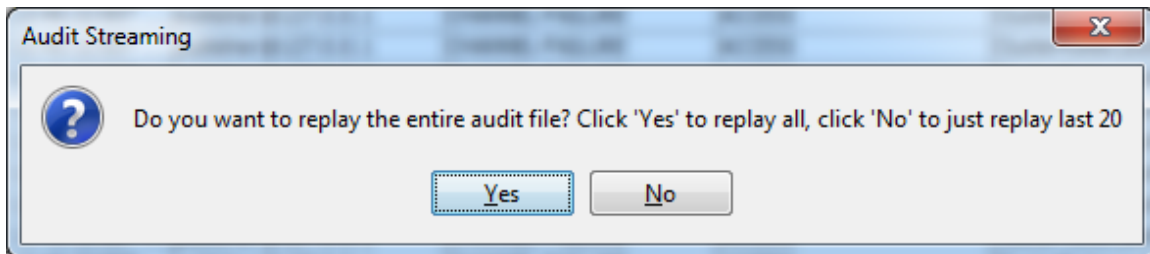
- **Date** - The time at which the audit event occurred on the server
- **Originator** - Who performed the operation
- **Type** - What type of object was the action performed on
- **Action** - What action was performed
- **Object** - The name of the object

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 excerpt from a sample audit file that 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.

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.

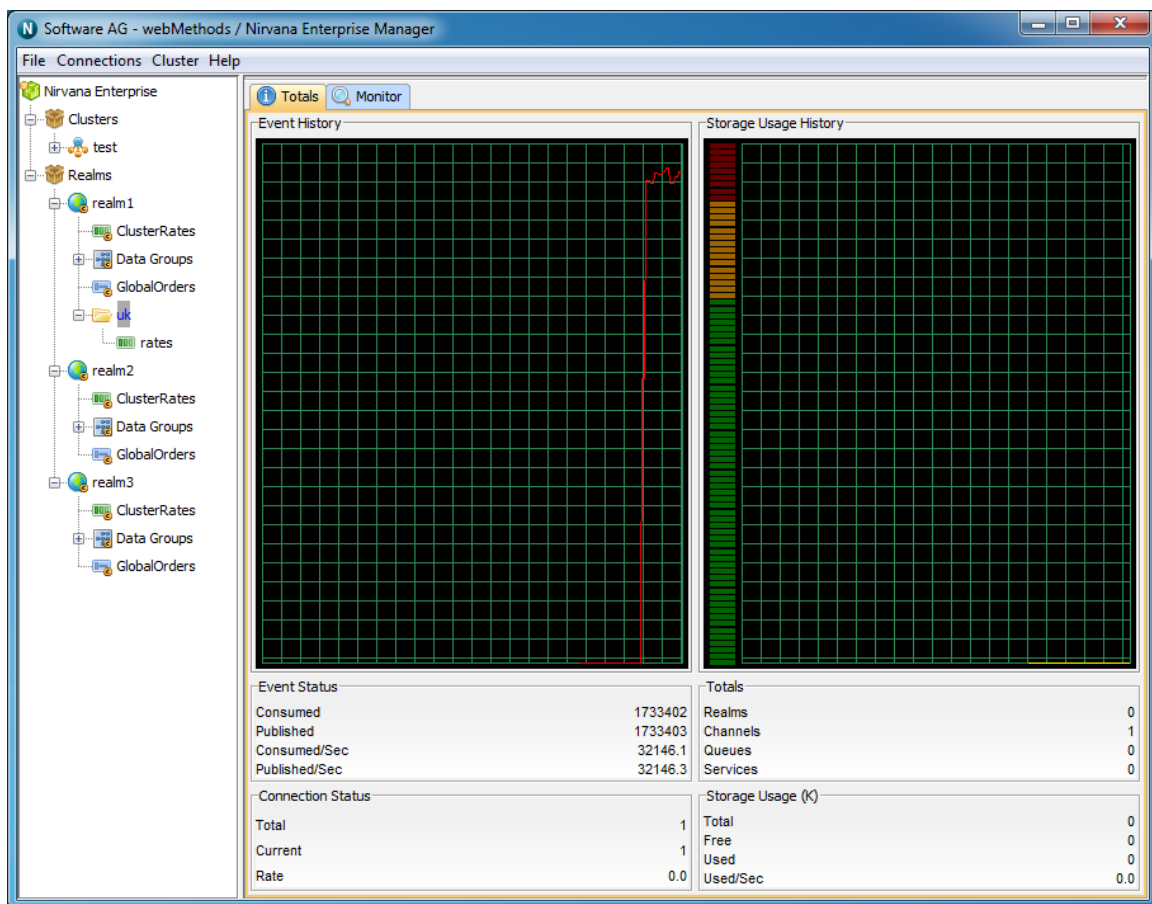
The status information shown within this panel is explained below.

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

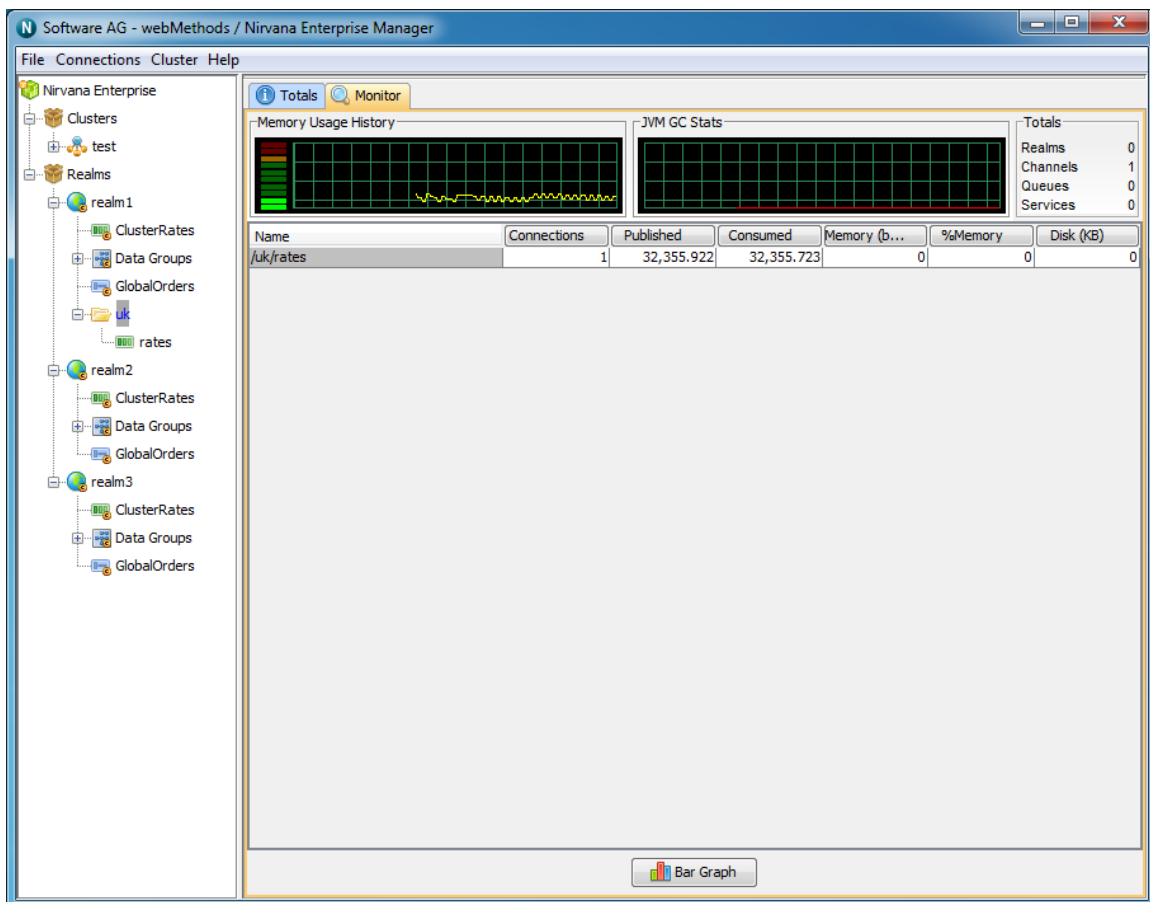
Container Monitor Panel

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 based 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 ["Realm Status" on page 266](#)) 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

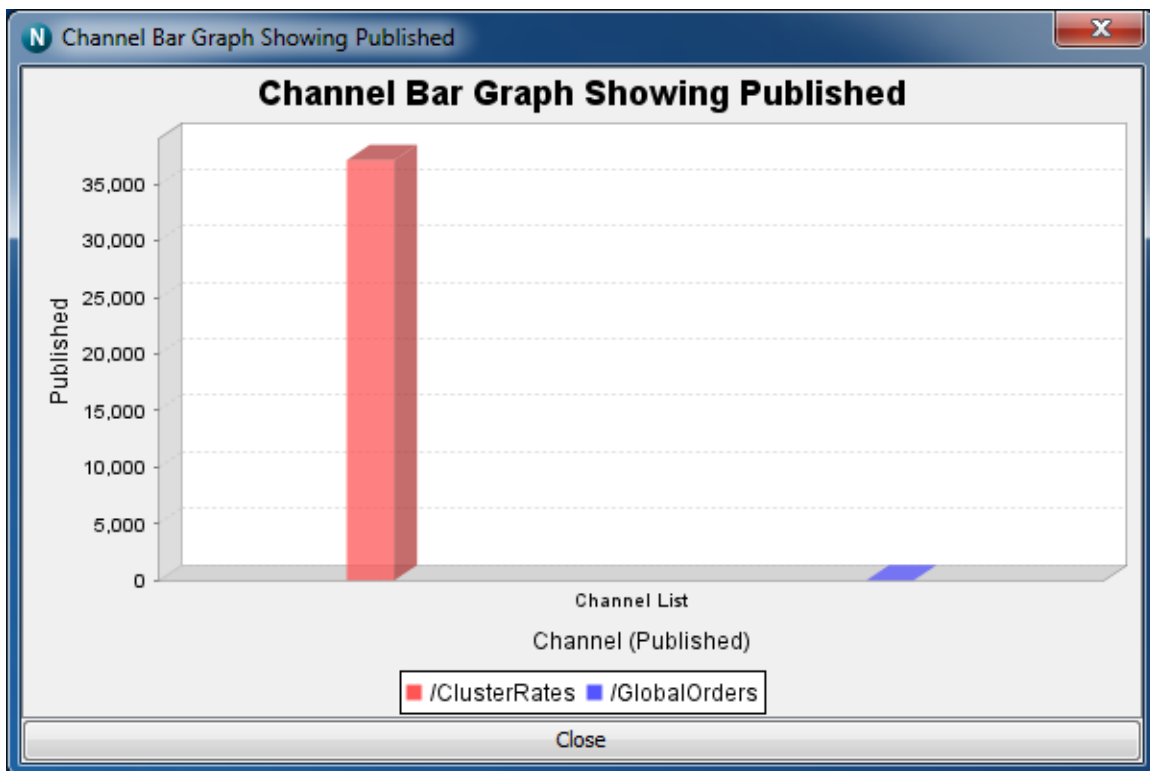
each channel has. The channel with the most number of consumers will appear at the top of the table.

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.

Channel Status

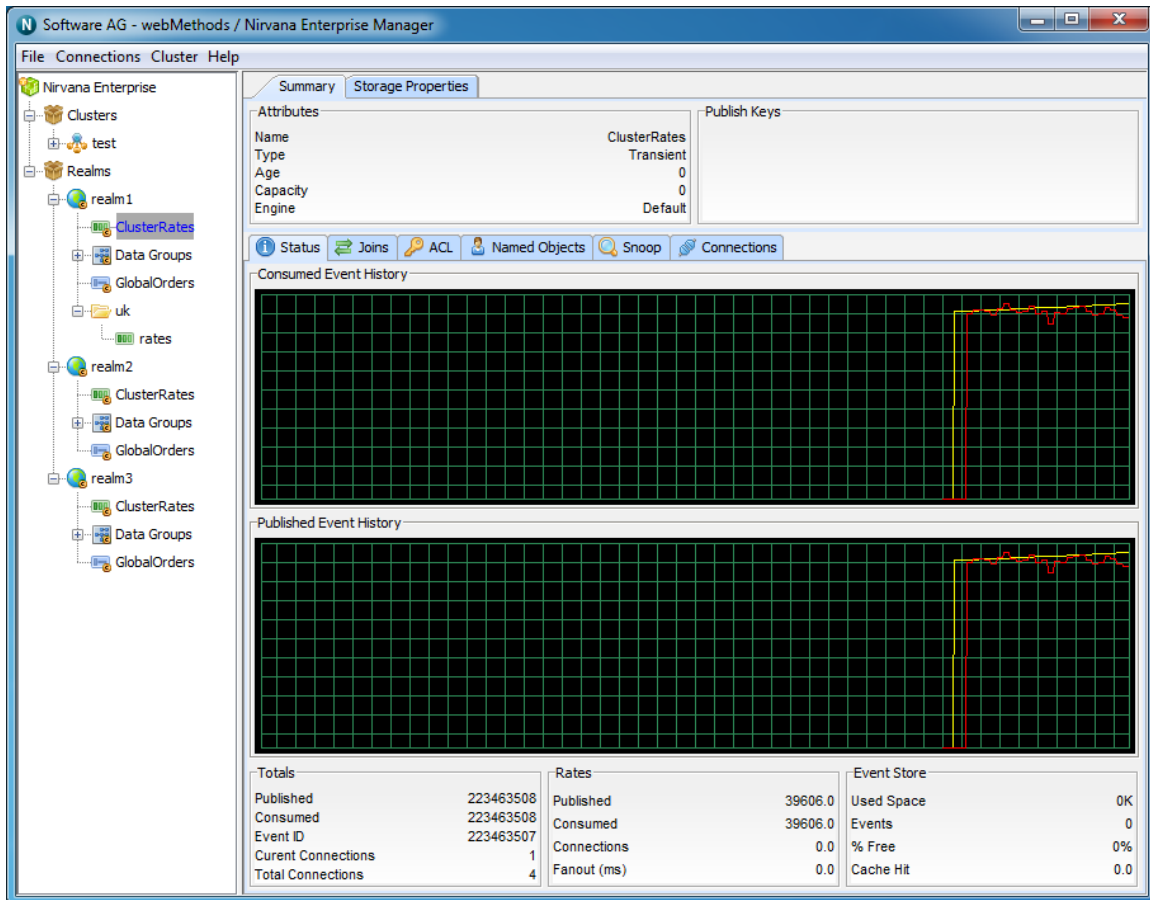
Introduction

Every time a channel object is selected 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 $(\text{total} - \text{previous total}) / (\text{interval } 1000 \text{ milliseconds})$
- **Consumed** - The current rate of events consumed from the channel, calculated as $(\text{total} - \text{previous total}) / (\text{interval } 1000 \text{ milliseconds})$
- **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 $(\text{used space} - \text{total space used by all purged or aged events})$)
- **Cache Hit** - The %age of events consumed from the channel event cache as opposed from the actual physical store if persistent or mixed

Data Group Status

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

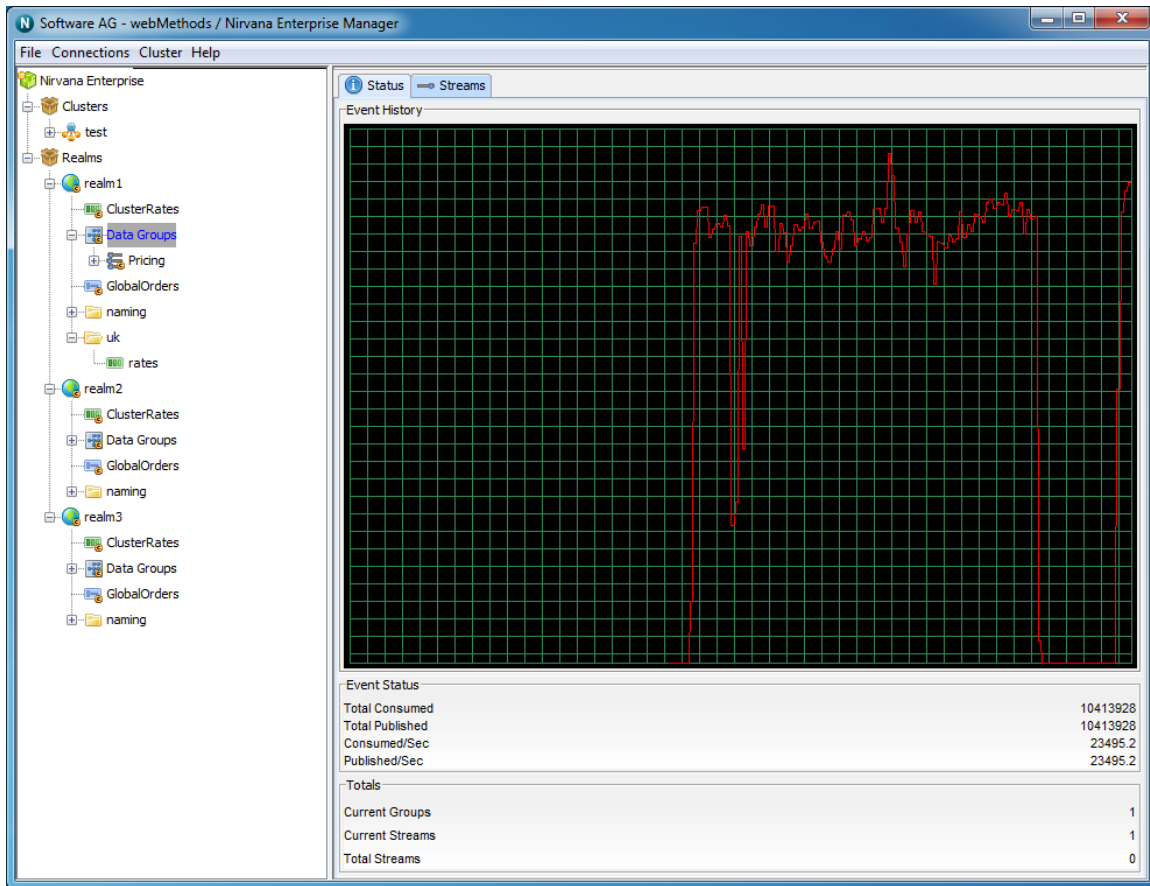
The Status panel for The 'Data Groups' node contains information regarding the publish and consumed events on Data Groups as well as the number of Data Groups and Data Streams currently connected

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

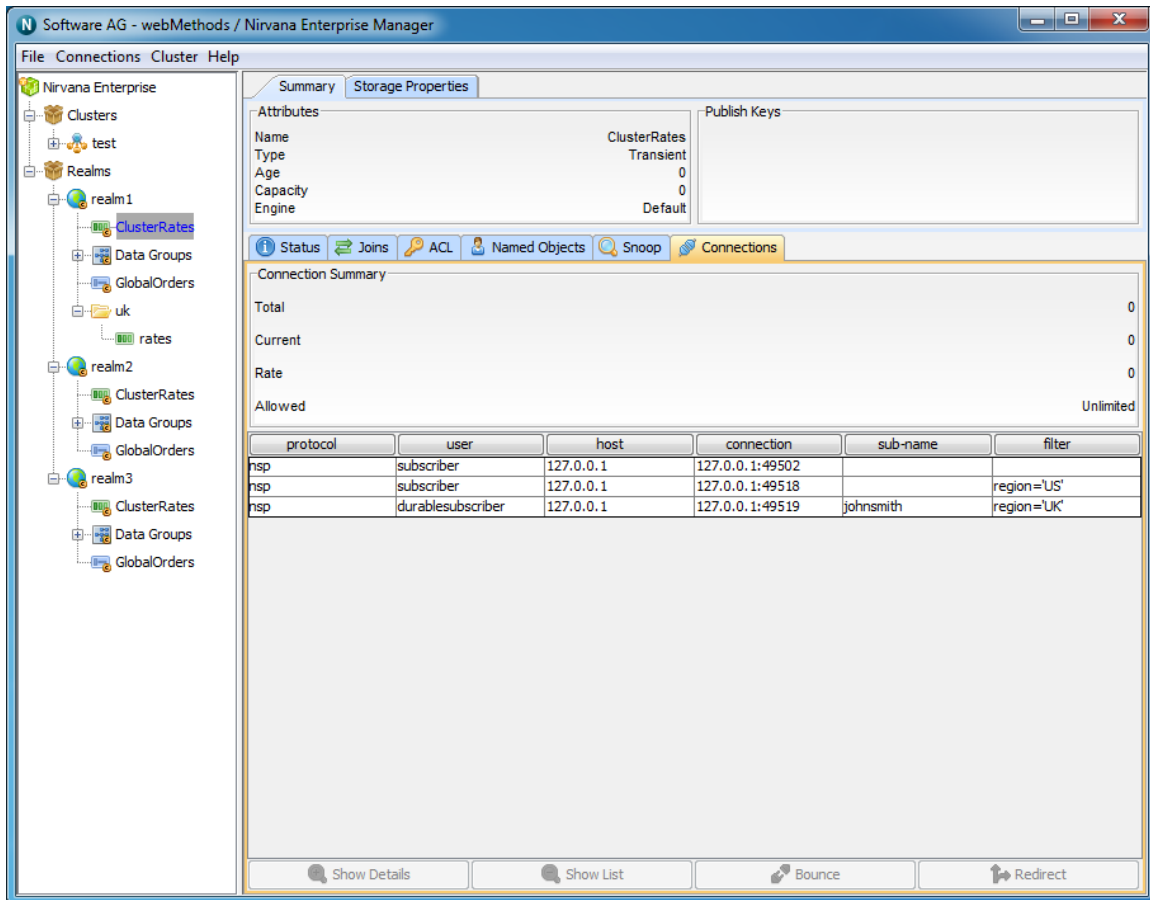
Channel Connections

When a Universal Messaging client connects to a Realm Server, the server maintains information on the connection (see "[Connection Information](#)" on page 347) 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:

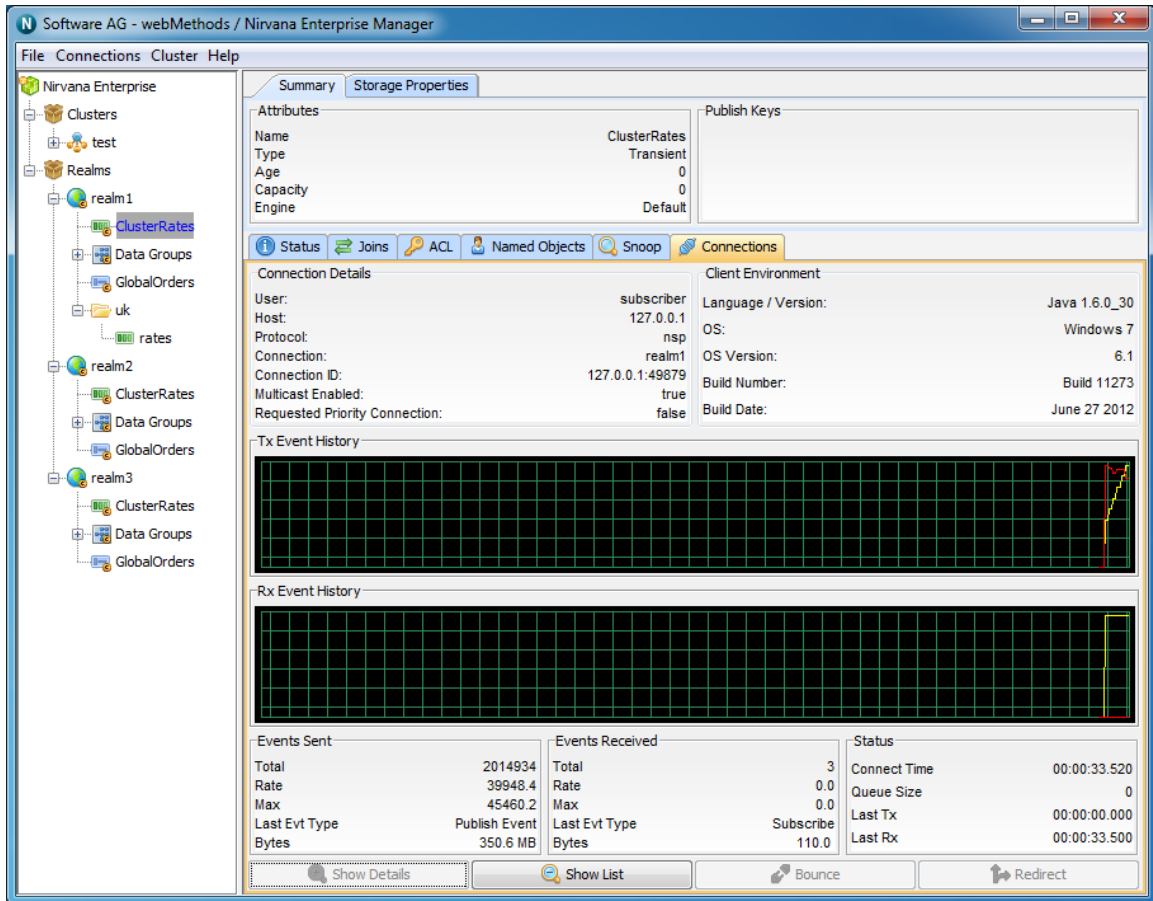
- **Protocol** - The protocol used in the connection
- **User** - The name of the user connected
- **Host** - The host machine that the user is connecting from
- **Connection** - The local connection id, defined as hostname:local_port
- **Sub-Name**- The named object (see "[Channel Named Objects](#)" on page 98) reference if one has been provided
- **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 named object 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 a the connection.

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

Secondly, 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.

Queue Status

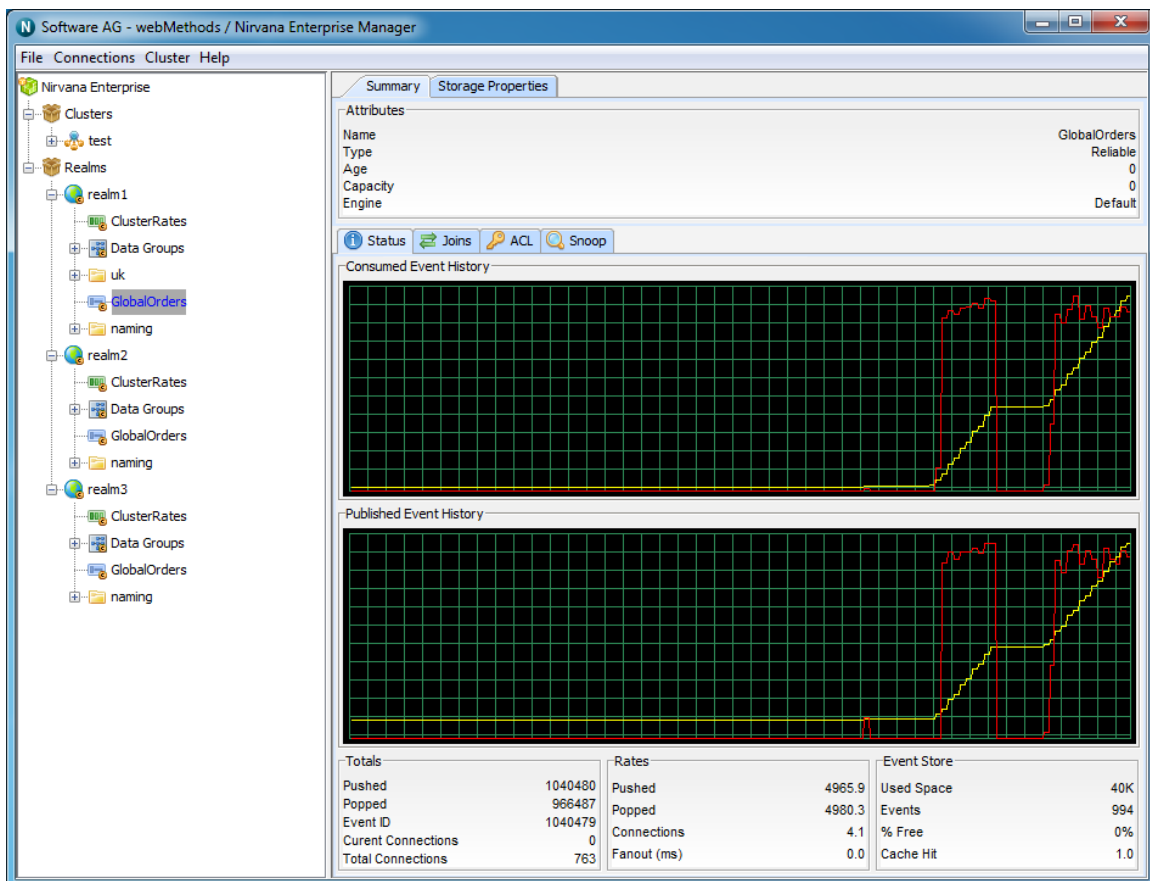
Every time a queue object is selected from the namespace, the first panel to be displayed on the right hand side of the Enterprise Manager panel is the 'Status' panel.

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 actual 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 cluster 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 $(\text{total} - \text{previous total}) / (\text{interval } 1000 \text{ milliseconds})$
- **Consumed** - The current rate of events popped from the queue, calculated as $(\text{total} - \text{previous total}) / (\text{interval } 1000 \text{ milliseconds})$
- **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 $(\text{used space} - (\text{total space used by all purged or aged events}))$)
- **Cache Hit** - The %age of events popped from the queue event cache as opposed form the actual physical store if persistent or mixed

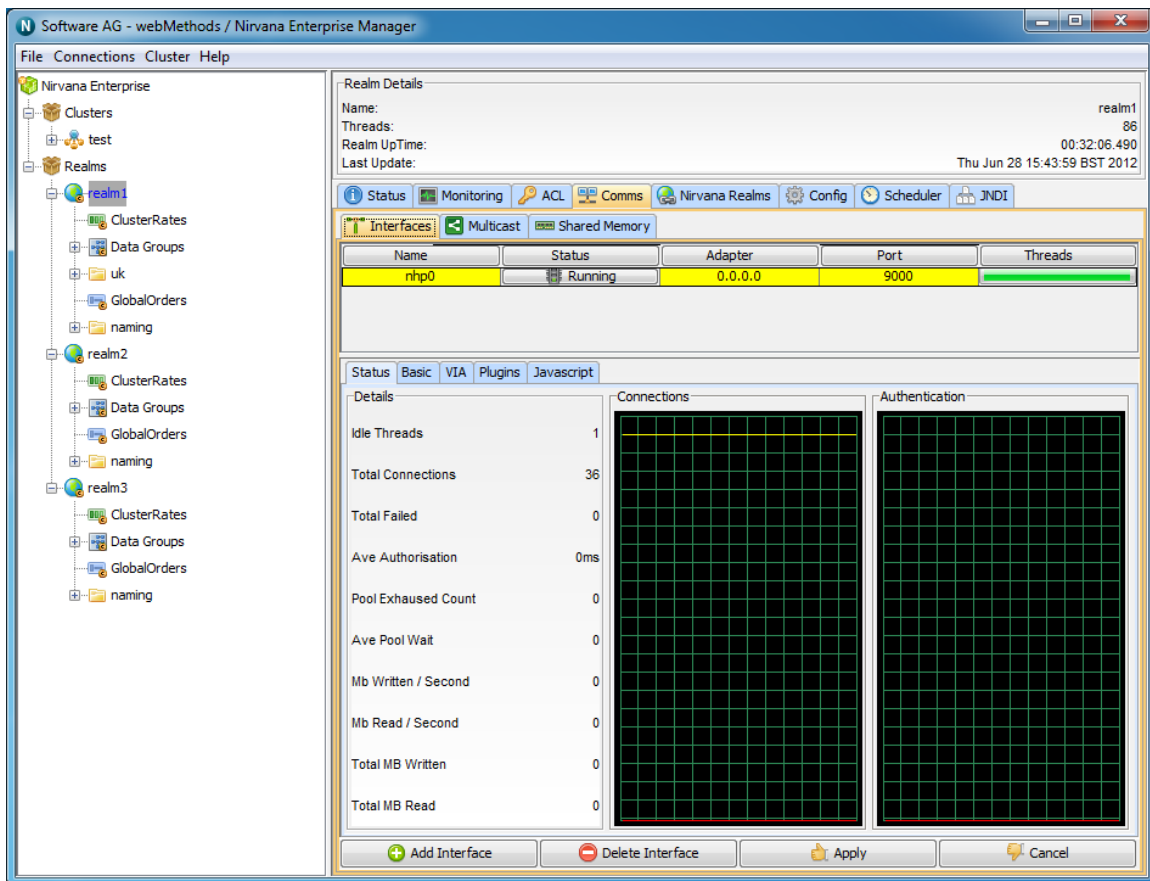
Interface Status

Universal Messaging interfaces (see "[Universal Messaging Enterprise Manager Comms: TCP Interfaces, IP Multicast and SHM](#)" on page 185) 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 192) 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).

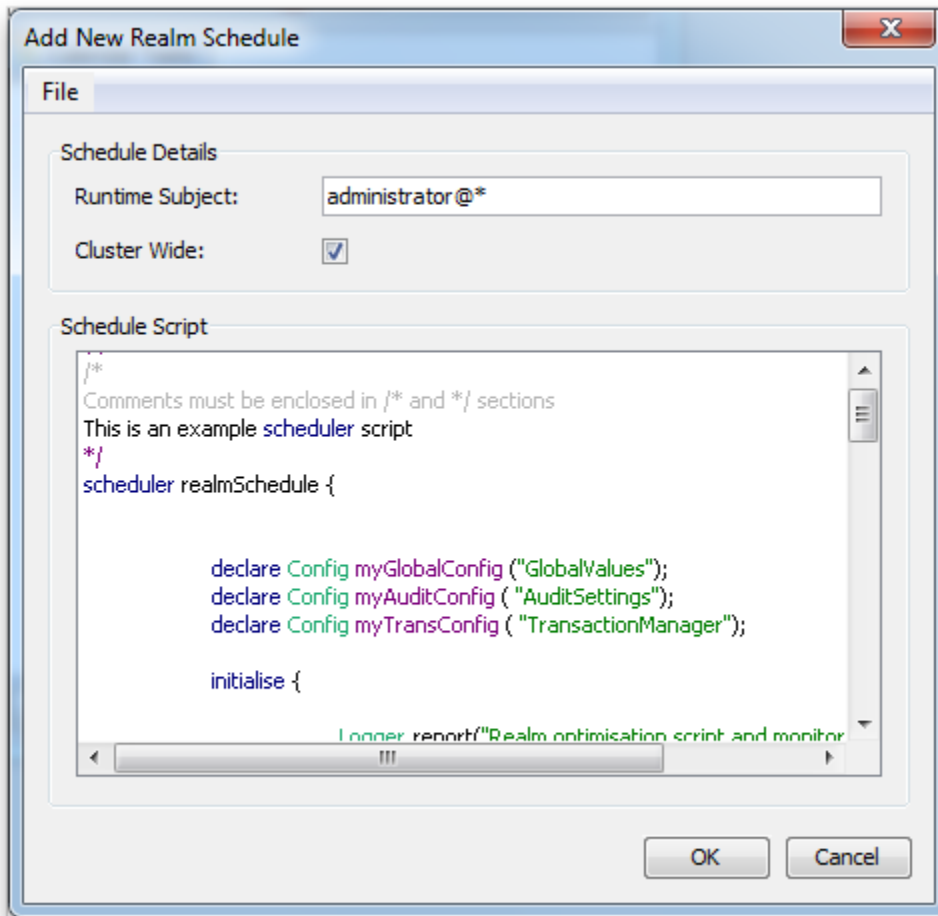
Scheduler view

Universal Messaging Enterprise Manager allows managing, deploying and editing server side scripts for execution in the Universal Messaging scheduler. The scripts consist of initial tasks, triggered tasks and / or calendar tasks as illustrated in the following sections.

- "[Adding Scheduler Scripts](#)" on page 299
- "[Editing Scheduler Scripts](#)" on page 300
- "[Scheduler Script Initial Tasks](#)" on page 301
- "[Scheduler Script Triggered Tasks](#)" on page 302
- "[Scheduler Scripts Calendar Tasks](#)" on page 303

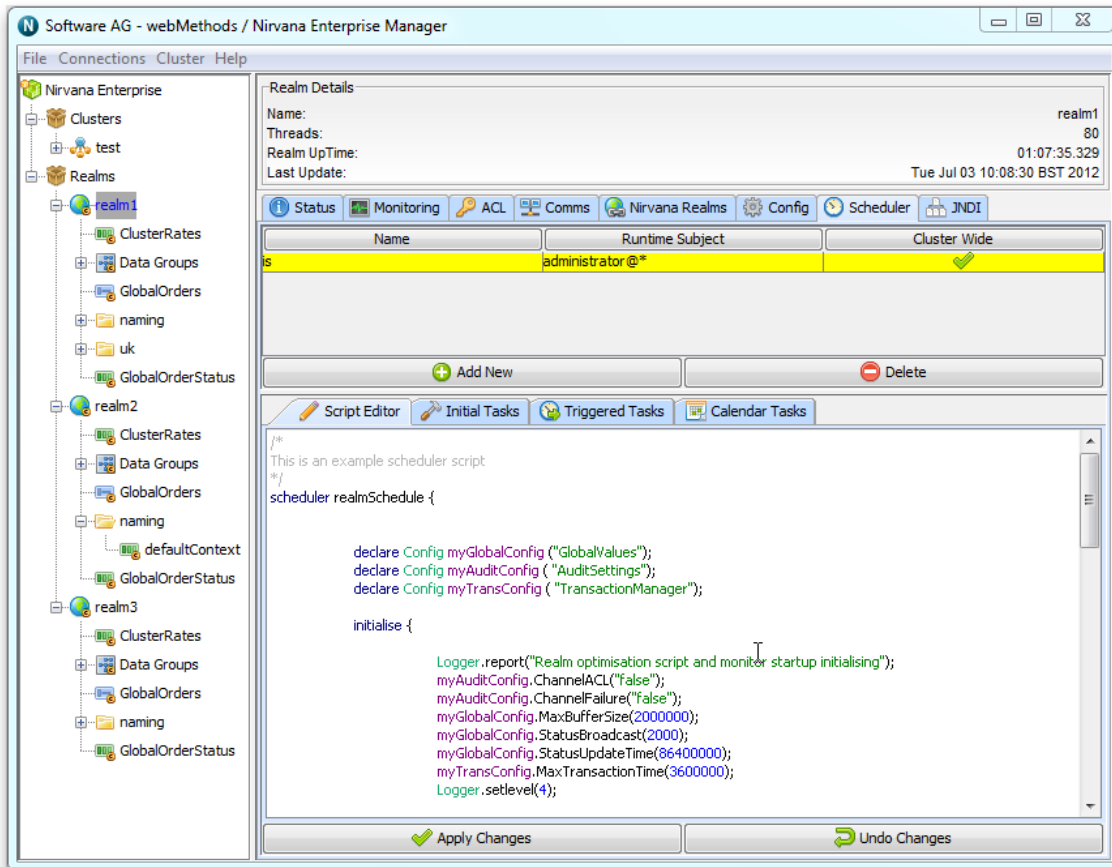
Adding Scheduler Scripts

The Universal Messaging Enterprise Manager allows deployment of scheduler scripts that execute on the Universal Messaging realm's scheduler. The editor features syntax highlighting to facilitate script editing.



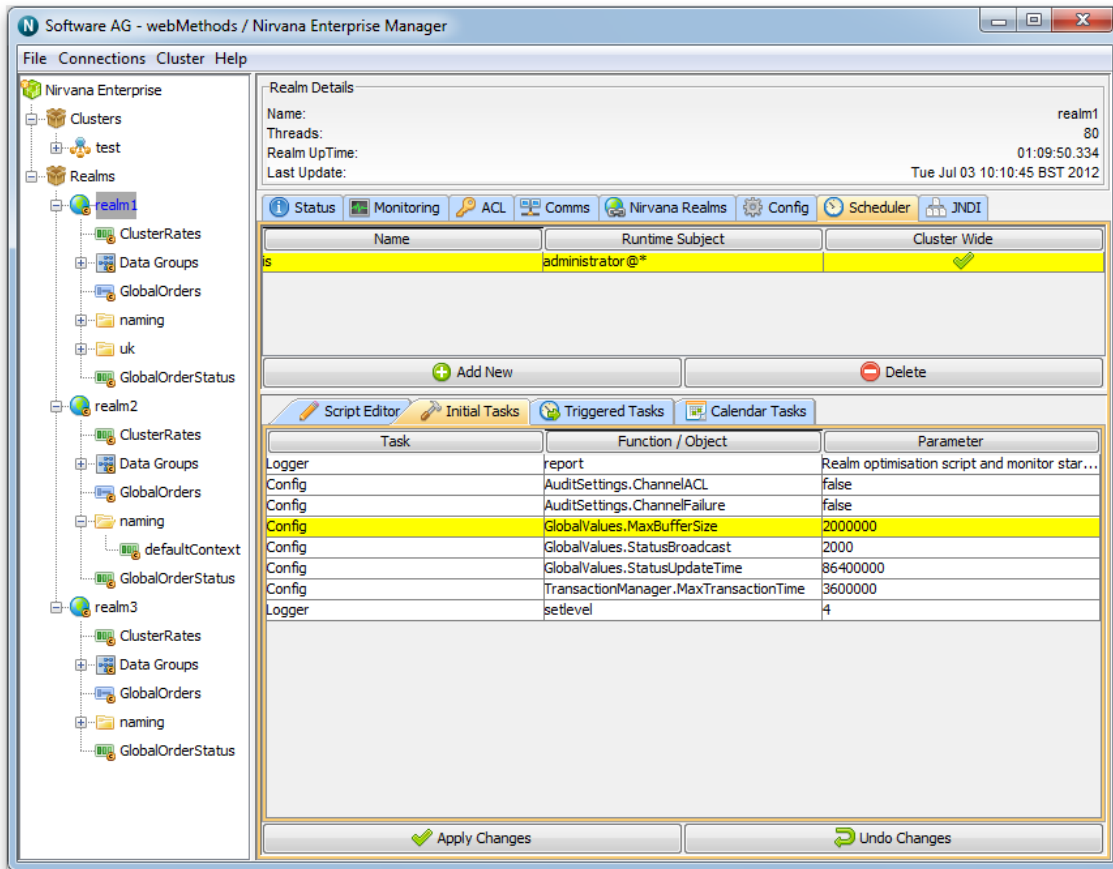
Editing Scheduler Scripts

The Universal Messaging Enterprise Manager allows viewing and editing of scripts that are already deployed on the realm from any location.



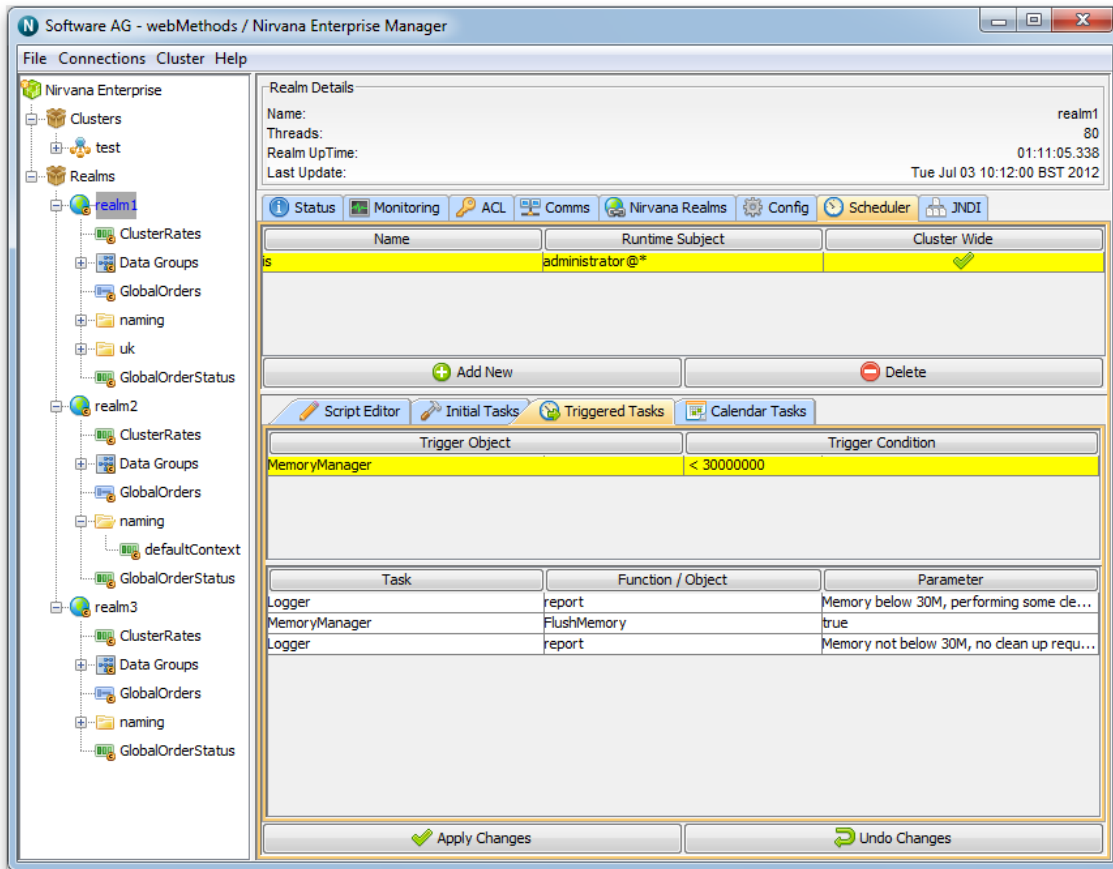
Scheduler Script Initial Tasks

Each scheduler script has an initialisation section that allows variable definition as well as execution of tasks such as realm configuration changes, custom log messages, interface configuration etc. These can be viewed after they are parsed and validated from the Enterprise Manager.



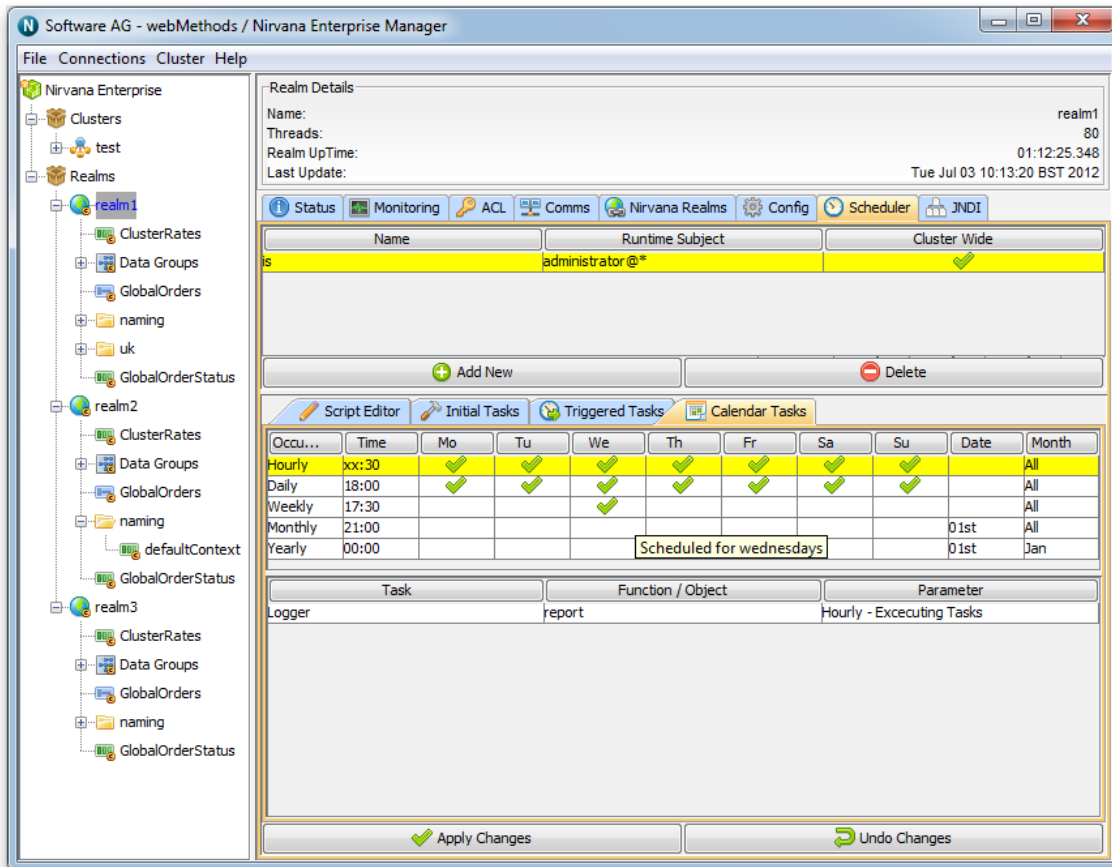
Scheduler Scripts Triggered Tasks

Each scheduler script can optionally contain a triggered tasks section. Triggered tasks are tasks that are executed when certain conditions are met during the operation of the Universal Messaging realm. For example the image below shows a triggered task that is executed if the realm log level is set to 0 (very verbose) which results in a custom log message as well as setting the log level to 1.



Scheduler Scripts Calendar Tasks

Each scheduler script can optionally have a calendar tasks section. Calendar tasks are triggered by calendar events as defined in the scheduler script. The Enterprise Manager image below shows a calendar tasks that writes a custom log message every 30 minutes of every hour, of every day or month.



Channel view

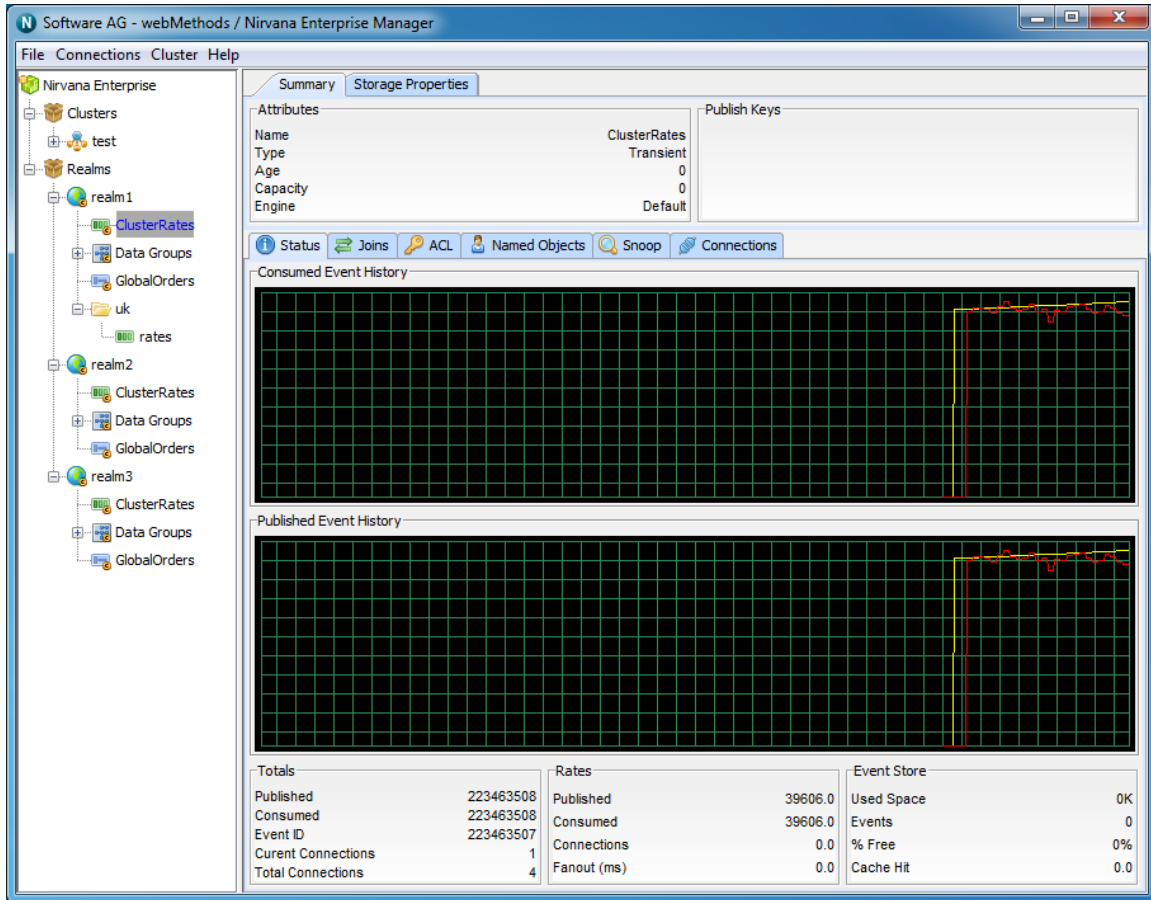
Universal Messaging Enterprise Manager allows monitoring, administration and configuration of every aspect of a Universal Messaging channel, as illustrated in the following sections.

- ["Channel Status" on page 305](#)
- ["Channel Access Control List \(ACL\)" on page 305](#)
- ["Channel Joins" on page 306](#)
- ["Channel Connections" on page 307](#)
- ["Channel Named Objects" on page 308](#)
- ["Channel Event Snooping" on page 309](#)

For more information on these screens please see the Management Information section.

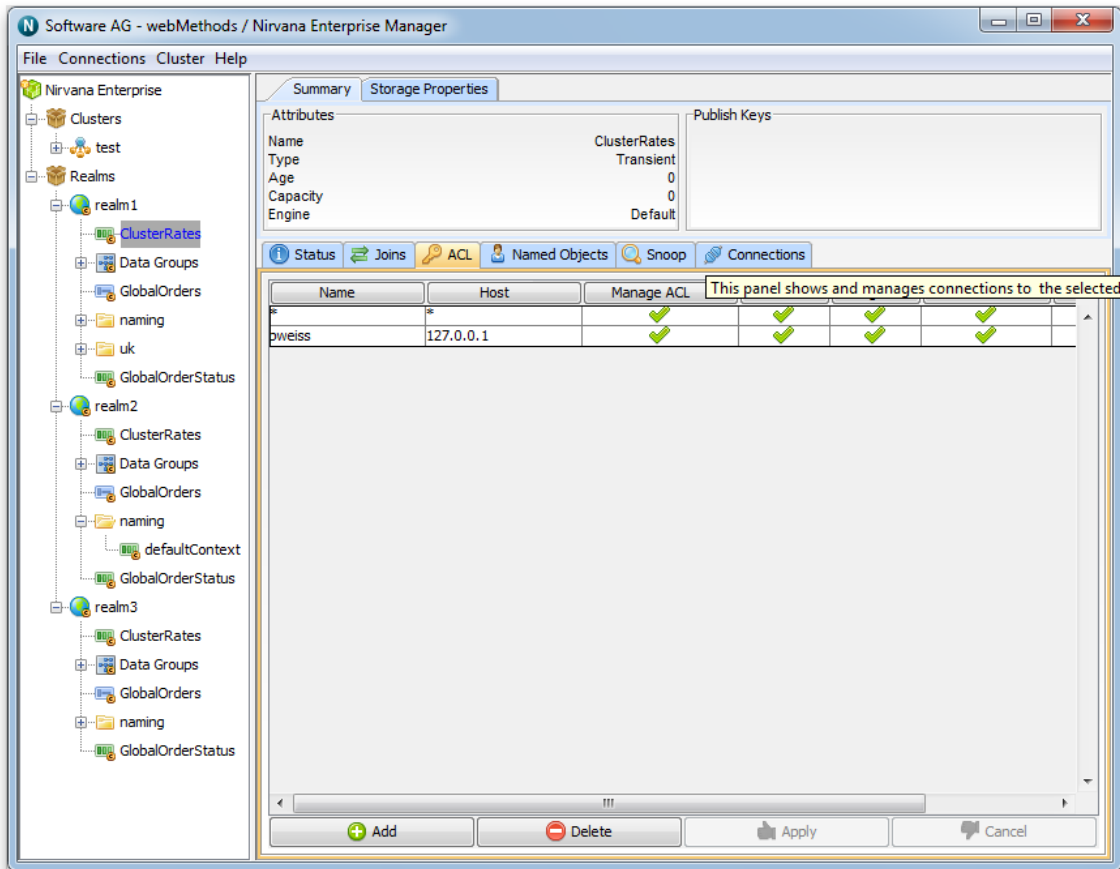
Channel Status

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.



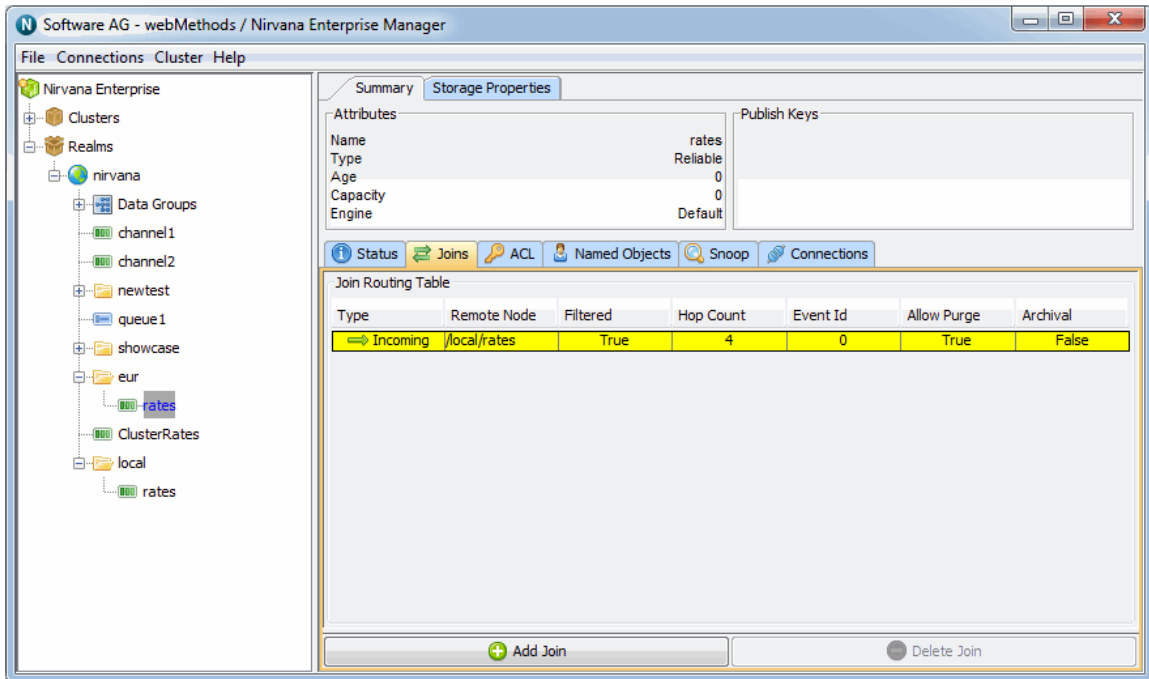
Channel Access Control List (ACL)

Universal Messaging offers complete control over security policies. Universal Messaging stores security policies locally or 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.



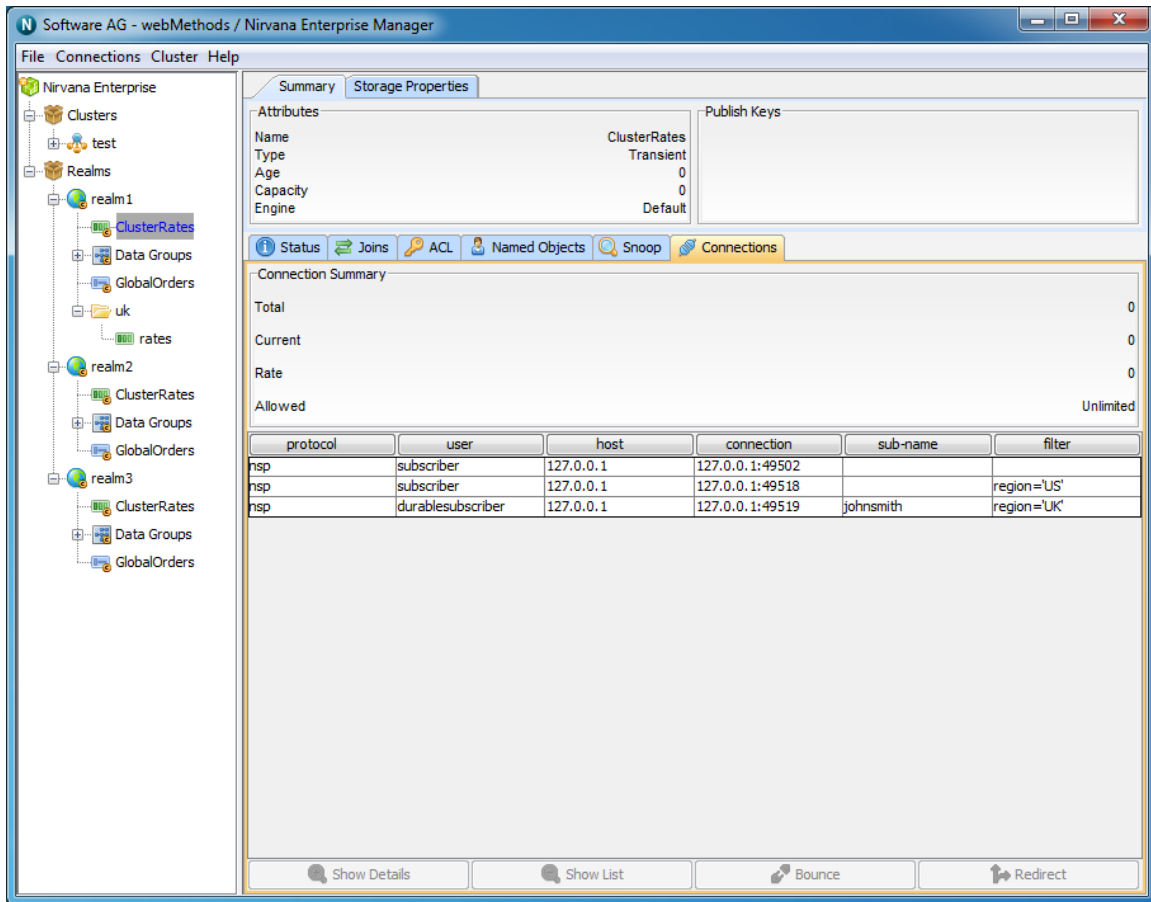
Channel Joins

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.



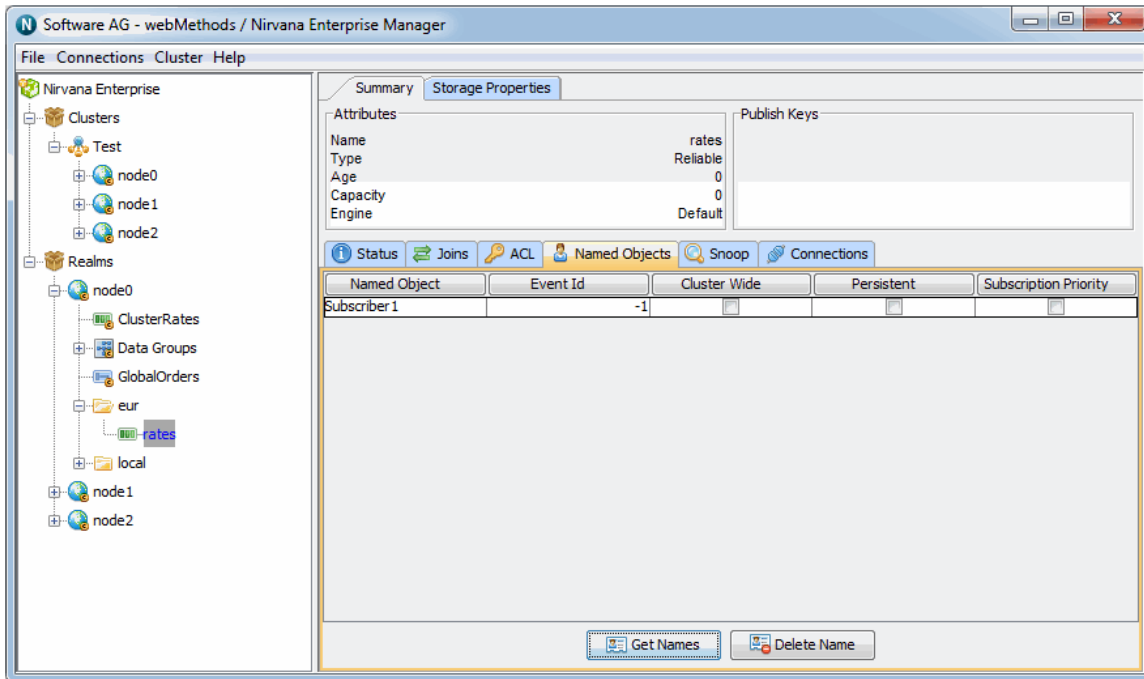
Channel Connections

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



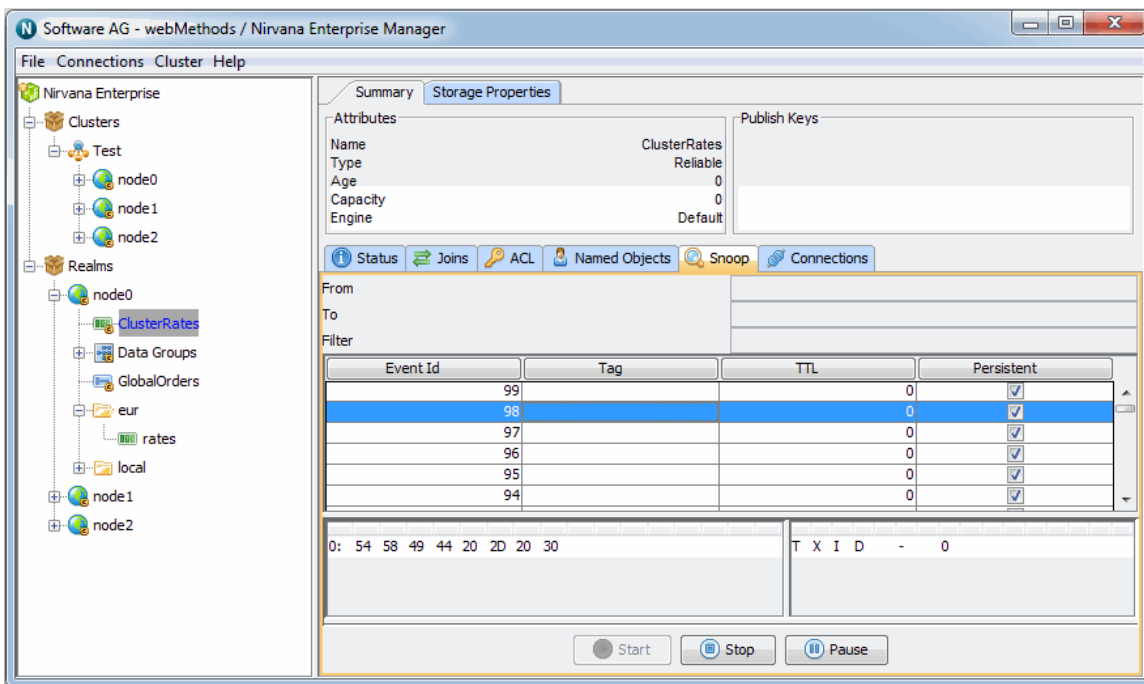
Channel Named Objects

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 named objects.



Channel Event Snooping

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



Queue view

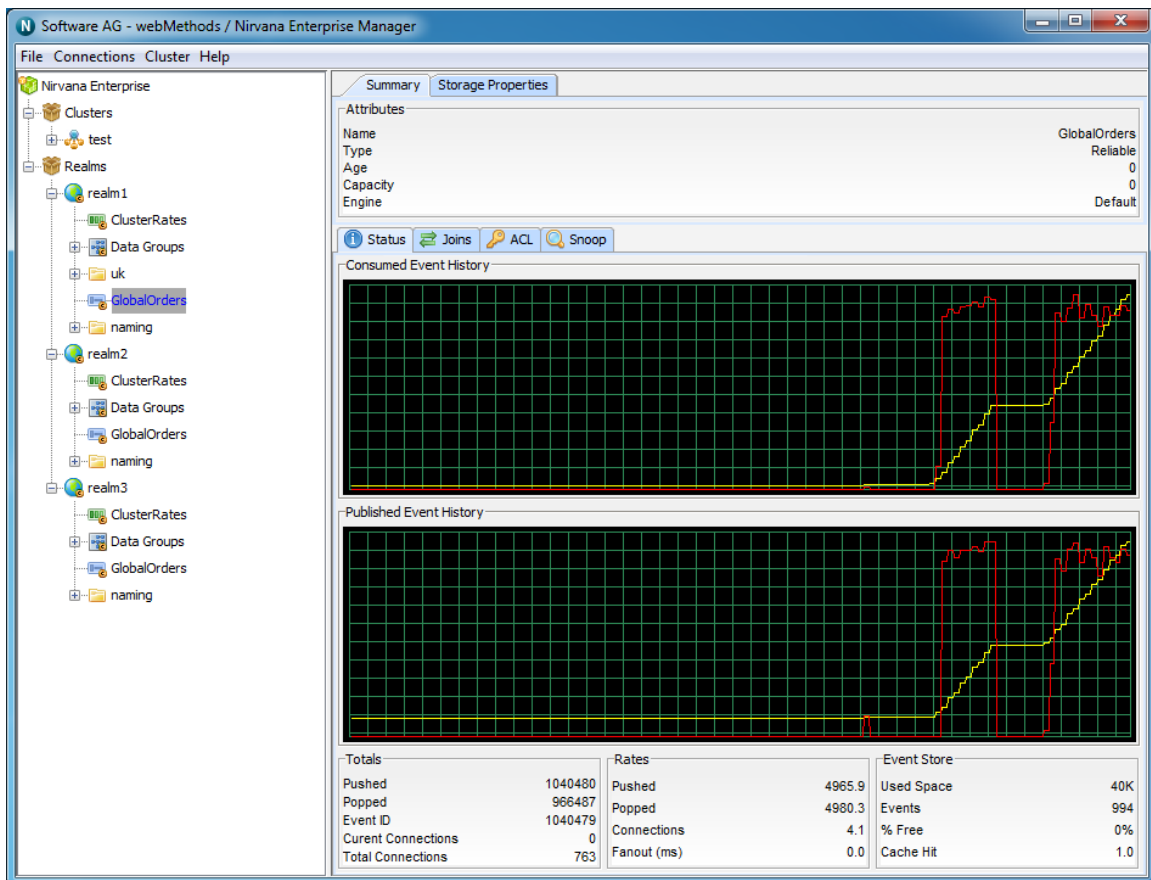
Universal Messaging Enterprise Manager allows monitoring, administration and configuration of every aspect of a Universal Messaging queue, as illustrated in the following sections.

- "Queue Status" on page 310
- "Queue Access Control List (ACL)" on page 311
- "Queue Joins" on page 311
- "Queue Event Snooping" on page 312

For more information on these screens please see the Management Information section.

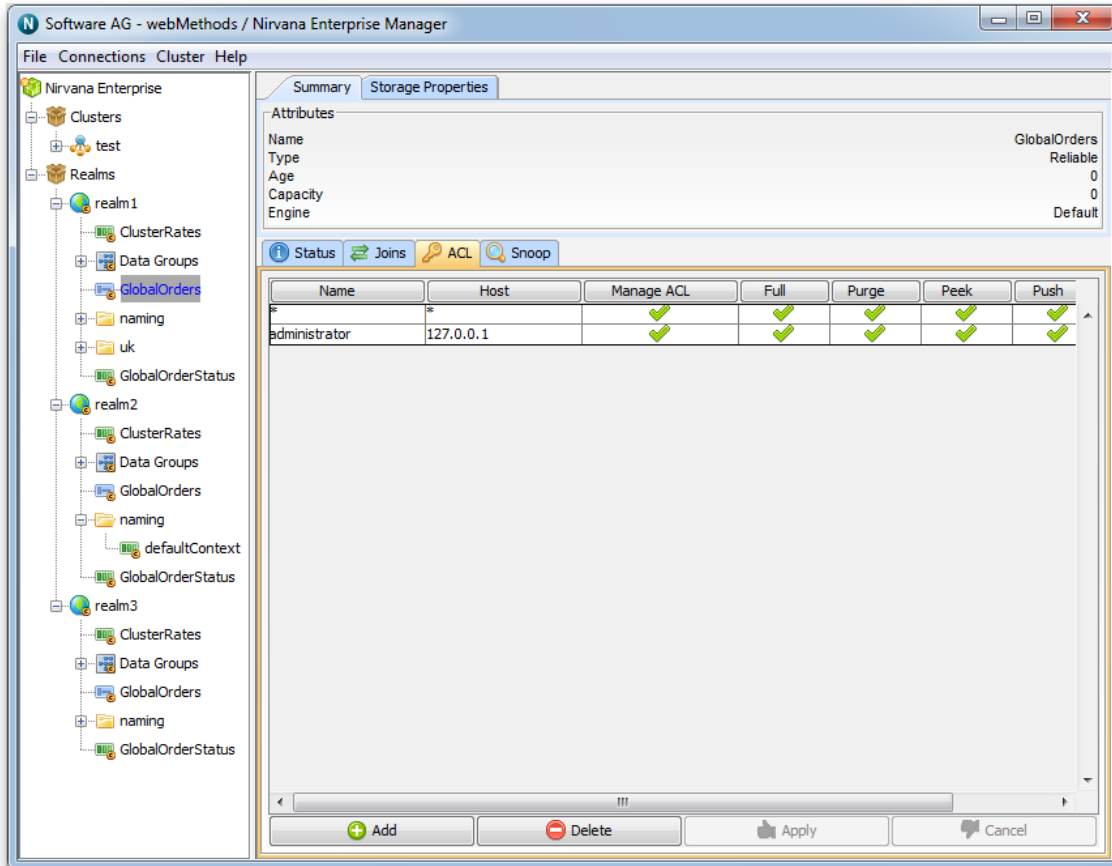
Queue Status

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



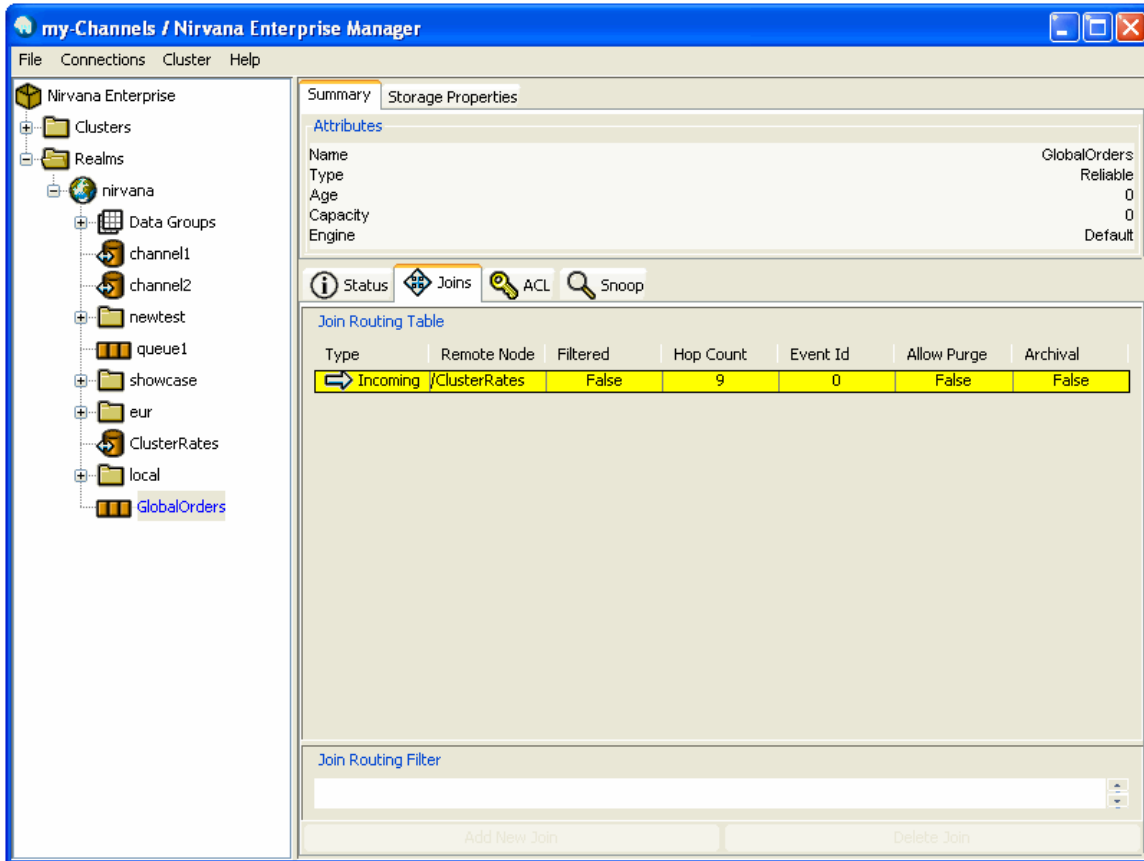
Queue Access Control List (ACL)

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.



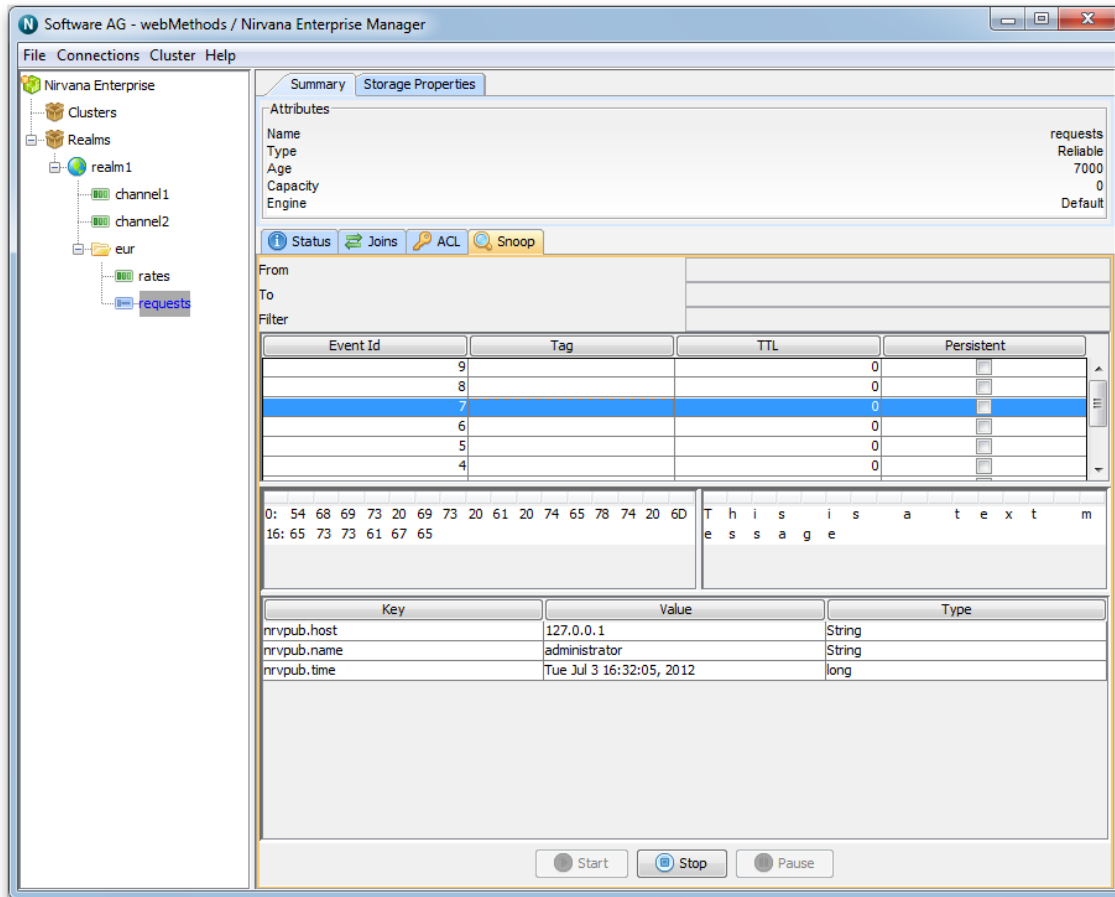
Queue Joins

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 Event Snooping

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



Peer 2 Peer view

Universal Messaging Enterprise Manager allows monitoring, administration and configuration of every aspect of a Universal Messaging Peer 2 Peer service, as illustrated in the following sections.

- "Service Status" on page 313
- "Service Access Control List (ACL)" on page 314
- "Service Connections" on page 315

For more information on these screens please see the Management Information section.

Service Status

The Universal Messaging Enterprise Manager allows monitoring of a peer 2 peer service's status in terms of publish & consume event totals/ rates as well as connection totals / rates.

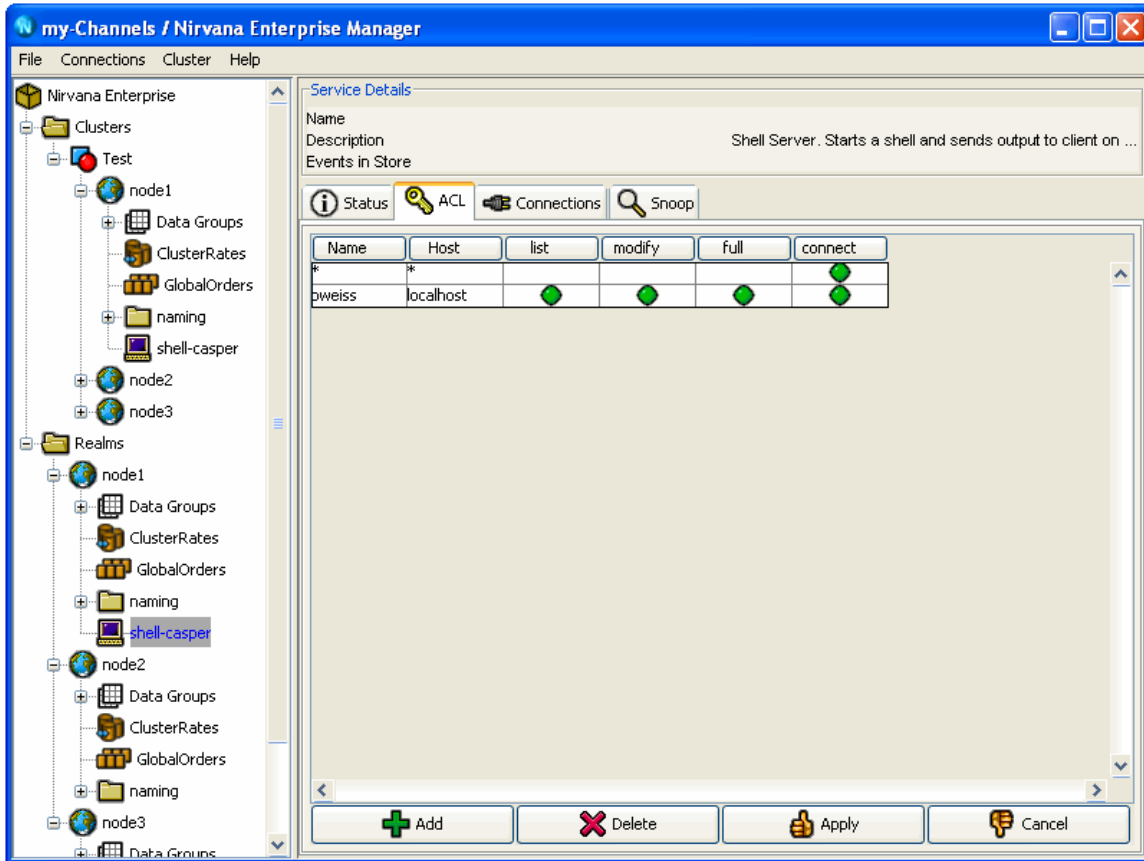
The screenshot displays the Nirvana Enterprise Manager interface. The left pane shows a tree view of the system structure, including Clusters (Test, node1, node2, node3) and Realms (node1, node2, node3). The right pane shows the Service Details for 'shell-casper', including its Name, Description, and Events in Store. Below this are two line graphs for Published and Consumed Event History, and a Totals/Rates table.

Totals		Rates	
Published	0	Published	0.0
Consumed	0	Consumed	0.0
Connections	4	Connections	0.0
Current	2		

Service Access Control List (ACL)

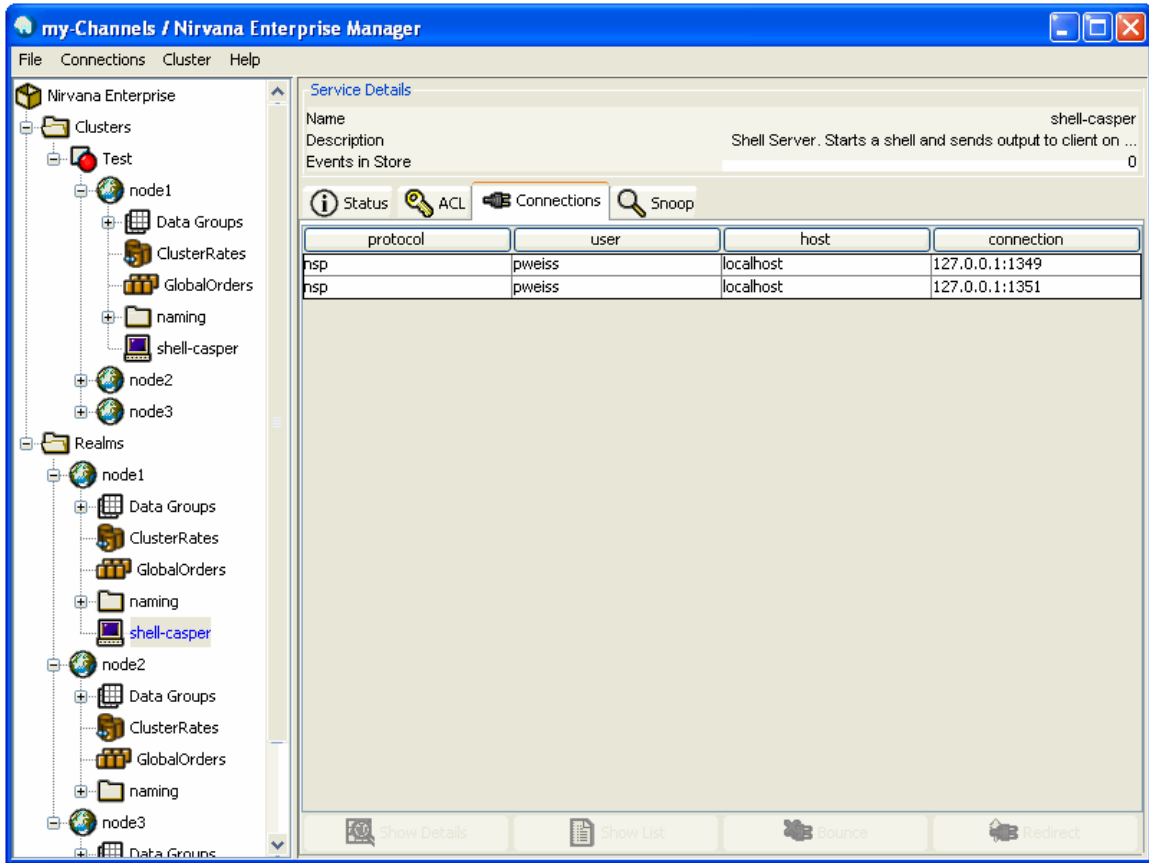
Universal Messaging offers complete control over security policies. Universal Messaging stores security policies locally or be driven by any external entitlements service.

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 and as peer 2 peer services use channels a similar model applies. For more information please see the Universal Messaging ACL's FAQ.



Service Connections

Channel subscribers are reported as channel connections and can be monitored / managed through the Universal Messaging Enterprise Manager. As Peer 2 Peer services use channels to exchange messages with service clients a similar model is used.



2 Universal Messaging Administration API

■ Introduction	318
■ Administration API Package Documentation	320
■ Namespace Objects	321
■ Realm Server Management	328
■ Security	335
■ Management Information	340

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 Admin API is available in the following languages:

- Java
- C#.NET
- C++

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, queues and P2P services.

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 %11 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 125), allowing control of the ACL permissions (see "[Access Control Lists](#)" on page 335)
- **nInterfaceManager** - The realm interface manager, allows me to add, remove, stop, start interfaces on a realm (see "[Interfaces](#)" on page 328)
- **nSchedulerManager** - the scheduler manager allows me to control scheduled tasks (see "[Scheduling](#)" on page 330) on the realm
- **nConfigGroup** - an enumeration of these corresponds to all configuration (see "[Config](#)" on page 331) 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 127) 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 347) 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 340) (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 344), 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
- **nServiceNode** - these correspond to p2p services.

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 321), 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 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.

nServiceNode (P2P Services)

Once you are familiar with the concept of the Universal Messaging Namespace, as discussed in the nRealmNode guide, you can then begin to use the other objects associated with a realm's Namespace.

This section discusses the nServiceNode which is used to administrate Nirvna P2P services. A general description of Universal Messaging P2P services is also available in the developer guide.

nServiceNode

The nServiceNode represents a Universal Messaging P2P service. The nServiceNode objects are generally stored under the root realm node. They can be obtained by a number of methods Firstly, by an enumeration of all child nodes from a realm node.

Java:

```
public void searchNodes(nContainer container) {
    Enumeration children = container.getNodes();
    while (children.hasMoreElements()) {
        nNode child = (nNode)children.nextElement();
        if (child instanceof nServiceNode) {
            nServiceNode service = (nServiceNode)child;
            System.out.println("Found Service : "+service.getName());
        }
    }
}
```

C#:

```
public void searchNodes(nContainer container) {
    System.Collections.IEnumerator children = realm.getNodes();
    while (children.MoveNext()){
        nNode child = (nNode)children.Current;
        if (child is nServiceNode) {
            nServiceNode service = (nServiceNode)child;
            Console.WriteLine("Found Service : "+service.getName());
        }
    }
}
```

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::CONTAINER)
        {
            searchNodes(((nContainer*)pNode)->getNodes());
        }
        else if (type == fBase::SERVICENODE)
        {
            nServiceNode *pSNode = (nServiceNode*)pNode;
            printf("Found Service : "+service->getName());
        }
    }
}
```

```
}
```

Or, directly from the realm node by calling :

Java:

```
Vector services = realm.getServicesList();
```

C#:

```
List services = realm.getServicesList();
```

C++:

```
services = realm->getServicesList();
```

Each `nServiceNode` will have a parent which is an instance of `nContainer`, however this is also an instance of an `nRealmNode` (i.e. the namespace root), since services do not contain any folder information in its name.

As services are created, they are added to the `nRealmNode`'s tree structure as `nServiceNodes`. 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.

Associated with each service node, is an `nServiceInfo` object that describes the details of each service node.

Each service 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 and P2P services. 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.

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 326](#)).

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 (see ["Channel Join" on page 327](#)) 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. 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 (see ["Channel Join" on page 327](#)) 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.

Channel joins can be created using the `nmakechanjoin` join sample application which is provided in the `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 321), 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 protocol 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.

For more information on Universal Messaging Administration, please see the API documentation, and the Enterprise Manager Guide.

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 321), 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 25](#)). 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 321](#)), 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 'GlobalValues' and set the 'fLoggerLevel' property:

Java, C#:

```
nConfigGroup grp = realm.getConfigGroup("fLoggerLevel");
nConfigEntry entry = grp.find("fLoggerLevel");
entry.setValue("0");
```

C++

```
nConfigGroup* grp = realm->getConfigGroup("fLoggerLevel");
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 25](#) 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 them 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

- Scalability - Large number of client connections can be managed across multiple servers within a cluster

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 form the RNAME values
    realms[x] = new nRealmNode(new nSessionAttribute(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.

nRealmNode

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" on page 334](#)) 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 321](#)). 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 3 different 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
- nServiceACLEntry - defines permissions for a subject on a Universal Messaging P2P service

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 336).

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();
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 Nirvana, 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 not 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 321), you can access 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 321) it is possible to get a reference to an `nLeafNode` (see "[nLeafNode \(Channels and Queues\)](#)" on page 323) 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 321), and an `nLeafNode` (see "[nLeafNode \(Channels and Queues\)](#)" on page 323) 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.

P2P Service Access Control List

When you have connected to a realm, and have a reference to an `nRealmNode` object (see "[nRealmNode](#)" on page 321), and an `nServiceNode` (see "[nServiceNode \(P2P Services\)](#)" on page 325) that corresponds to a p2p service, you can access the node's

acl object. This object contains a list of nServiceACLEntry objects that represent a subject and a set permissions for various operations on a p2p service.

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#:

```
nServiceNode service = realm.findNode("Universal Messaging-shell");
nACL acl = service.getACLs();
```

C++:

```
nServiceNode* service = realm->findNode("Universal Messaging-shell");
nACL* acl = service->getACLs();
```

Once you have the acl object, you can then add, remove or modify acl entries:

nServiceACLEntry

To find a specific acl entry from the service 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 service), I could use the following code:

Java, C#:

```
nServiceACLEntry entry = acl.find("Everyone");
entry.setFullPrivileges(false);
acl.replace(entry);
service.setACLs(acl);
```

C++

```
nServiceACLEntry* entry = acl->find("Everyone");
entry->setFullPrivileges(false);
acl->replace(entry);
service->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.

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 321](#)), 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 25](#)) 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 321), 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 observe 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 321), 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::ACLENTY) {
            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.

nServiceNode

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 321), and connecting to a realm, information is automatically delivered to the realm node from the realm.

This information is delivered as and when the state changes for one or all of the objects managed within a realm and it also delivers summary information periodically.

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 discussed the information that can be obtained through the nAdmin API for the nServiceNode object which is associated with a Universal Messaging P2P service:

Status Events

The nServiceNode extends nNode which is a subclass of Observable, so when the status information is received for a service node, (this occurs only when things change on the service, i.e. acl, connections, events published / consumed etc) the nServiceNode will trigger the update callback on any known Observers who are interested in receiving information about the service.

For example, if you write a class that implements the Observer interface, then the nServiceNode callback can be added as follows:

Java, C#:

```

Enumeration children = realm.getNodes();
while (children.hasMoreElements());
    nNode child = (nNode)children.nextElement();
    if (child instanceof nServiceNode) {
        child.addObserver(this);
    }
}
}

```

C++:

```

fSortedList nodes = pNode->getNodes();
fSortedList::iterator iterator;
for (iterator = nodes.begin(); iterator != nodes.end(); iterator++)

```

```

{
    nNode *pNde = (nRealmNode*)iterator->second;
    if (pNde->getType() ==fBase::SERVICENODE)
    {
        pNde->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 service node has changed.

When events occur on a service 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 service acl has been added or deleted:

Java, C#:

```

public void update(Observable obs, Object obj){
    if (obs instanceof nServiceNode) {
        if (obj instanceof nACLEntry) {
            nServiceNode srvc = (nServiceNode)obs;
            nACLEntry entry = (nACLEntry)obj;
            // acl modified / added / deleted
            System.out.println("Service"+srvc.getName()+"
acl event for "+entry.getSubject());
        }
    }
}

```

C++:

```

void ObservableMapping::update(Observable *pObs, void *pObj)
{
    if (obs->getType() == fBase::SERVICENODE) {
        if (obj->getType() == fBase::ACLENTY) {
            nServiceNode srvc = (nServiceNode*)obs;
            nACLEntry entry = (nACLEntry*)obj;
            // acl modified / added / deleted
            printf("Service %s",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 321), 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, the nLeafNode and the nServiceNode 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.

nServiceNode Connections

Universal Messaging provides the ability to issue notifications of connections to service nodes. Connections to service nodes correspond to p2p client connections, so when a user connects to a service or closes the service, 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.

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#:

```
service.addListener(this);
```

C++:

```
service->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 service connections are made or closed.

For information on monitoring realm connections using the enterprise manager or channel/queue connections please see the enterprise manager guide (see ["Realm Connections" on page 272](#) and ["Channel Connections" on page 292](#)). In addition monitor panels (see ["Top" on page 277](#)) are available to show TOP like functionality on realm usage.

For more information on Universal Messaging Administration, please see the API documentation, and the Enterprise Manager Guide.