Universal Messaging Release Notes

Version 10.15

October 2022

This document applies to Software AG Universal Messaging 10.15 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.

Document ID: NUM-RN-1015-20240308

Table of Contents

About this Documentation

Online Information and Support

Product Documentation

You can find the product documentation on our documentation website at https://documentation.softwareag.com.

In addition, you can also access the cloud product documentation via https://www.softwareag.cloud. Navigate to the desired product and then, depending on your solution, go to "Developer Center", "User Center" or "Documentation".

Product Training

You can find helpful product training material on our Learning Portal at https://learn.softwareag.com.

Tech Community

You can collaborate with Software AG experts on our Tech Community website at https://techcommunity.softwareag.com. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software AG news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at https://github.com/softwareag and https://containers.softwareag.com/products and discover additional Software AG resources.

Product Support

Support for Software AG products is provided to licensed customers via our Empower Portal at https://empower.softwareag.com. Many services on this portal require that you have an account. If you do not yet have one, you can request it at https://empower.softwareag.com/register. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

Data Protection

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

1 Overview

The *Release Notes* for Universal Messaging describe the changes introduced with the current product release.

Documentation roadmap

Universal Messaging provides documentation in the following formats:

- HTML
- PDF

The following table describes the different guides that are available.

Title	Description
Administration Guide	Describes how to perform administration tasks for your Universal Messaging environment, by using either using a graphical user interface or an API.
Clustering Guide	Describes how to work with Universal Messaging clusters.
Concepts	Describes the underlying concepts of the Universal Messaging product.
Developer Guide	Describes how to develop client applications for enterprise, mobile clients or web clients.
Installation Guide	Describes how to install the product.
Operations Guide	Describes how to perform daily operations, special operations, monitoring, troubleshooting, archiving and cleanup tasks in order to keep the Universal Messaging system running smoothly.
Reference Guide	Provides programmer-level API documentation for the supported APIs and languages.
Release Notes	Describes new features and changes since the previous release.

License Types and Feature Sets

To use Universal Messaging in a production environment, you require one of the following licence types:

NUMWF/NUMTF
Universal Messaging Fully Featured / TC Universal Messaging Fully Featured.

NUMWI
Universal Messaging for Integration Active/Passive.

NUMWS
Universal Messaging for Integration Active/Active.

Note:
You can only use this license type if you also have the NUMWI license type.

The following table shows the feature sets available with each of the license types:

Feature Name	Description	NUMWF / NUMTF	NUMWI	NUMWS
Max Resources	Can create channels and queues	Unlimited / Specified	Unlimited	
Mixed Channels	Can create channels that contain messages that are persistent or reliable	x	x	
Reliable Channels	Can create channels that contain messages that are reliable (current event ID persisted to disk)	x	x	
Persistent Channels	Can create channels where all events are stored on disk	x	x	
Queues	Messaging paradigm where only one consumer can read a message from a queue. If more than one consumer is subscribed to a queue then the messages are distributed in a round-robin fashion.	x	x	
JMS Clients	Clients can use the JMS API	x	x	
Java Clients	Clients can use the native Java API	x	x	
Enterprise Clients for C++ (deprecated), C#	Clients can use additional native APIs (C++, C#)	x	x	
Zones	Organize servers within one namespace	x	x	
Joins	Joins can be used to forward messages between servers or clusters	x	x	
Publish Keys	Uniquely identify messages on a channel, used for merging or replacing	x	x	
Snoop	Peek at messages on a queue or topic	x	x	
Messaging Priority	Server can reorder messages based on priority	x	x	
Horizontal Scaling	Clients can seamlessly publish/consume events to/from multiple independent realms and clusters using a single connection	x	x	

Feature Name	Description	NUMWF / NUMTF	NUMWI	NUMWS
Active/Active Clustering	Clusters are formed of multiple active servers for high availability	x		x
Web Clients	Web browser-based clients	x		
Mobile Clients	Mobile-based clients (Apple iOS, Google Android)	x		
Plugins	Server can expose additional functionality (e.g. files, REST) over HTTP interfaces	x		
HTTP Support	Connect over HTTP interfaces	x		
AMQP	Use the AMQP protocol for messaging with queueing	x		
MQTT	Use the MQTT lightweight protocol for embedded devices	x		
Shared Memory	Low latency interface communication on the same machine	x		
Multicast	Reliable multicast over UDP for efficient routing of messages	x		

Note:
Universal Messaging ships with a trial license, which allows the full-feature server to run for a maximum of 90 days from first run. During the installation procedure, you can either choose to use the trial license, or you can specify the name of the production license file. If you start off with the trial license and later decide to move to a production license, follow the instructions in the section *Upgrading from a Trial to a Production License* of the *Installation Guide*.

2 What's New in Universal Messaging 10.15

Universal Messaging 10.15 is the successor of Universal Messaging 10.11.

The Universal Messaging 10.15 release notes include new features, enhancements, changes, deprecated and removed functionality.

New Features

The following features have been added in Universal Messaging 10.15:

- **Support for determining the server state within a cluster**

  You can determine the state of a Universal Messaging server within a cluster in the following ways:

  - Using the newly added command-line tool `GetServerClusterState`.
  - Using the newly added endpoints in the Health Monitor Plugin `IsMaster` or `GetClusterState`.

- **File size limit on multi-file disk stores**

  Files in multi-file disk stores now have a size limit. When the maximum size per file is reached Universal Messaging creates a new file to persist the next events. You can use a new realm configuration property, `MaxSpindleFileSize`, to specify the maximum file size in bytes.

  For more information about how to configure multi-file disk stores and the new property, see the section "About Multi-File Disk Stores" in the *Concepts* guide.

- **Bi-directional admin client and server compatibility**

  Universal Messaging 10.15 and higher supports bi-directional compatibility of Enterprise Manager and the Java Admin API for release versions 10.15 and higher.

  For more information about the functionality, see the section "Bi-directional Admin Client and Server Compatibility" in the *Concepts* guide.

- **Bi-directional compatibility of server-to-server communication**

  Universal Messaging 10.15 and higher supports bi-directional compatibility of server-to-server communication within clusters, zones, and joins for release versions 10.15 and higher.

- **Server log level configuration for Log4j2**

  You can now use the new `LogLevelOverride` realm configuration property to configure the server log level when the server uses the Log4j2 logging framework. This parameter is only supported when the server is running as a Docker container.

  If you set `LogLevelOverride` to `true`, you can configure the `fLoggerLevel` property. `false` restores the configuration from log4j2.xml and makes `fLoggerLevel` ineffective in the Enterprise Manager.

  For more information about the logging configuration, see "Realm Configuration" in the Administration Guide.

  The new functionality is provided with the Universal Messaging 10.15 Fix 2.

- **com.softwareag.um.client.missed_keep_alives client property**

  A new client property that specifies the number of missed server-side keep-alive intervals before the client closes a connection. The property is introduced in Universal Messaging 10.15 Client Fix 4.

  For more information about the property, see the topic "Client Parameters" in the *Concepts* guide.

- **MaxNoOfConnectionsPerUserName realm property**

  A new realm configuration property that defines the number of concurrent connections to a Universal Messaging server per user. The property is introduced in Universal Messaging 10.15 Fix 5. For more information about using the property, see the section "Realm Configuration" in the *Administration Guide*.

- **AuthenticationTimeLogThreshold property**

  A new realm configuration property that defines the number of concurrent connections to a Universal Messaging server per user. The property is introduced in Universal Messaging 10.15 Fix 10. For more information about using the property, see the section "Server Behavior when Authenticating Connections" in the *Operations Guide*.

- **New HealthChecker Tool Check**

  The new `JoinLastEIDMismatchCheck` checks for joins that are ahead of the channel's last EID. The check is introduced in Universal Messaging 10.15 Fix 12. For more information, see the section "Running a Configuration Health Check" in the *Administration Guide*.

- **User Connections JMX bean and metrics**

  Universal Messaging exposes a new JMX bean, named Universal Messaging User Connections, which enables access to data related to the number of current sessions that each user has established to the server. You can use the related NumberOfConnections JMX metric and sag_um_connections_per_user Prometheus metric for monitoring. The functionality is available with Universal Messaging 10.15 Fix 12.

  For more information about the metrics and using JMX, see the topic "Using JMX to Monitor Universal Messaging" in the *Administration Guide*.

- **com.softwareag.um.client.server_keep_alive_leeway client property**

  A new client property that specifies the additional time that it can take a Universal Messaging server to reply to a keepalive sent by a client. The property is introduced in Universal Messaging 10.15 Fix 14.

  For more information about the property, see the topic "Client Parameters" in the *Concepts* guide.

Changed Features

The following Universal Messaging features already available in the previous product release have been changed in Universal Messaging 10.15:

- **C++ Libraries**

  Note the following changes relating to the C++ libraries that Universal Messaging provides:

  - The C++ libraries for Universal Messaging use OpenSSL libraries that have now been upgraded to version 3.0.5 from version 1.1.1k.
  - The C++ libraries for Universal Messaging use POCO libraries that have now been upgraded to version 1.11.3 from version 1.10.1.
  - OS X is no longer supported.

  If you experience any problems with the new version of the C++ client libraries for Universal Messaging, first consider recompiling your client applications against the new version of the libraries and compilers.

- **Defining users exempt from authentication**

  Now you explicitly define users that are exempt from authentication by configuring the `-DNirvana.auth.exempt` property. Previously, users exempt from authentication were loaded by default from a `-DSECURITYFILE` file, which is not used any longer.

  In addition, exempt users both with and without a password should be able to log in to Universal Messaging. Previously, authentication failed when an exempt user provided an invalid password.

  For more information about defining users exempt from authentication and configuring the `-DNirvana.auth.exempt` property, see the section "Configuring Authentication and Client Negotiation" in the *Concepts* guide.

- **JMX namespaces for topics and queues**

  By default, JMX namespaces are now created using a flat structure, for example, `destinationName=/q1/q2/q3`. In previous Universal Messaging versions, by default, JMX namespaces for topics and queues have a folder tree structure.

  You can use a new realm configuration property, `FlatStoreJMXBeanNamespace`, to define if JMX namespaces for topics and queues have a flat structure or a folder tree structure.

  For more information about the property, see the section "Realm Configuration" in the *Administration Guide*.

- **MaxNoOfConnections realm property**

  A realm configuration property that defines the total number of concurrent connections to a Universal Messaging server. Now a user with *both* the **Override Connection Count** realm permission and an admin connection can override this property. Previously, users with full permissions could override the property. The change is introduced in Universal Messaging 10.15 Fix 5. For more information about using `MaxNoOfConnections`, see the section "Realm Configuration" in the *Administration Guide*.

- **Realm Information Collector**

The Realm Information Collector command-line diagnostic tool has been enhanced to collect the following additional information about a server:

- Java Service Wrapper log files even if you configured an absolute file path for the `wrapper.logfile` property in the Sever_Common.conf file.
- Log4j2 configuration, and also log files configured through file appenders in the log4j2.xml configuration file.
- The log header of the Universal Messaging server.

For more information about the Realm Information Collector, see the section "The 'Realm Information Collector' Diagnostic Tool" in the *Administration Guide*.

- **Queue batching**

  In Universal Messaging 10.15 and higher, the server disregards event batching for queues requested by clients and sets it to 1 event instead. The modified behavior affects Java clients using the com.pcbsys.nirvana.client.nQueueReaderContext.setPeekBatchSize(int) API.

- **com.pcbsys.nirvana.client.nConstants#setReconnectAfterAccessChange method behavior**

  If you set setReconnectAfterAccessChange(..) to `false`, if the client receives `nSecurityException` for cluster formation or for exceeding the connection limit, these exceptions are treated as recoverable and the client will try to reconnect to the Universal Messaging server. For any other security exceptions, the client will make no reconnection attempts.

  For more information about setReconnectAfterAccessChange(..), see the com.pcbsys.nirvana.client.nConstants class reference in the Java Client API documentation.

- **Support for plugins depending on the license type**

  Universal Messaging now supports plugins for all UM license types regardless of the `Plugins` key value in the license file. Previously, plugin support was available only for the NUMWF/NUMTF license type. The change is introduced in Universal Messaging 10.15 Fix 13.

- **Server behavior when handling duplicate stores for a store name**

  Starting with Universal Messaging 10.15 Fix 14, the Universal Messaging server does not start if there are duplicate stores for a particular store name present on the server when you work with case-insensitive data folders, for example, on a Windows operating system. Contact Software AG Global Support about how to proceed.

  When you start the Universal Messaging server, it checks for duplicate stores and shuts down if it finds such stores, for example, `testQueue` and `TestQueue`. The server also writes the following warning message in the nirvana.log file and in the console output: "`WARNING: Multiple stores found for store name: <Store name>. Listing the stores for this store name: <Store 1><Store 2>….`"

  In addition, you can no longer create duplicate stores for a particular store name when you work with case-insensitive data folders.

- **Base image for Universal Messaging container images**

Starting with Universal Messaging 10.15 Fix 16, the Universal Messaging container images on the Software AG Container Repository use Red Hat UBI 9 as a base image. Images up to 10.15 Fix 16 use Red Hat UBI 8 as a base image.

Deprecated Features

The following Universal Messaging features have been deprecated in Universal Messaging 10.15:

- **Java Client APIs**

  The following Java APIs are deprecated:

```
nQueue#createAsyncReaderPrevious(nQueueReaderContext context)
nQueue#createAsyncTransactionalReaderPrevious(nQueueReaderContext context)
nQueue#destroyReaderLocally(nQueueReader reader) ,destroyReader(nQueueReader reader)
nQueue#destroyReader(nQueueReader reader)
nSession#isConnectionVirtual()
nSession#enableThreading(boolean enabled)
nSession#enableThreading()
nSession#setHTTPURLParameter(String parameter)
nRealm#getHostVal()
nRealm#getPortVal()
nRealm#getHTTPPortVal()
```

- **Client APIs related to the removal of queue reader objects**

  The following C# and C++ APIs are deprecated:

```
nQueue#destroyReader(nQueueReader reader)
nQueue#destroyReaderLocally(nQueueReader reader)
```

Removed Features

Removed Functionality

The following Universal Messaging functionality has been removed in Universal Messaging 10.15:

- **Legacy JMX**

  The legacy JMX functionality, which you enabled by using the `EnableLegacyJMX` realm configuration property, has been removed. Legacy JMX was deprecated in Universal Messaging 10.7.

- **Data group and data stream functionality**

  The full data group and data stream functionality, and respective client and administration APIs in Java, C#, C++, and JavaScript, have been removed.

  When an older-version client connects to a Universal Messaging server 10.15, the client will receive the following exceptions:

  - For data stream calls:

```
com.pcbsys.nirvana.client.nSecurityException: SECURITY: Data Streams are not
supported
at
com.pcbsys.nirvana.base.clientimpl.singleconnection.ClientConnectionManagerImpl.requestHandshake
(ClientConnectionManagerImpl.java:773)
at
com.pcbsys.nirvana.base.clientimpl.singleconnection.ClientConnectionManagerImpl.connect
(ClientConnectionManagerImpl.java:669)
at
com.pcbsys.nirvana.base.clientimpl.singleconnection.ClientConnectionManagerImpl.initialise
(ClientConnectionManagerImpl.java:418)
at com.pcbsys.nirvana.client.nSession.init(nSession.java:269)
at com.pcbsys.nirvana.client.SimpleSessionTestWithDataStreams.main
(SimpleSessionTestWithDataStreams.java:17)
```

- For data group calls:

```
com.softwareag.um.tools.UMToolCommon processExceptionAndTerminate
SEVERE: Security error. [Data Groups/Streams not supported on Server]
com.pcbsys.nirvana.client.nSecurityException: Data Groups/Streams not supported
 on Server
at
com.pcbsys.nirvana.base.nExceptionEventProcessor.checkIfExceptionIsCommitOrRollBackOfEvents
(nExceptionEventProcessor.java:82)
at com.pcbsys.nirvana.base.nExceptionEventProcessor.checkSecurityException
(nExceptionEventProcessor.java:65)
at
com.pcbsys.nirvana.base.clientimpl.singleconnection.nDataGroupManagerImpl.findOrCreateGroup
(nDataGroupManagerImpl.java:524)
at com.pcbsys.nirvana.client.nSession.createDataGroup(nSession.java:3014)
at
com.softwareag.um.tools.datagroup.CreateDataGroup.execute(CreateDataGroup.java:54)
at com.softwareag.um.tools.UMToolCommon.executeTool(UMToolCommon.java:94)
at com.softwareag.um.tools.UMToolCommon.executeTool(UMToolCommon.java:77)
at
com.softwareag.um.tools.datagroup.CreateDataGroup.main(CreateDataGroup.java:65)
```

- **MacOS C++ client**
- **jmx_exporter.yaml configuration file**

  The jmx_exporter.yaml configuration file, which was deprecated in Universal Messaging 10.11, has been removed. Instead, to configure JMX with Universal Messaging, use the replacement file, jmx_sag_um_exporter.yaml. The jmx_exporter.yaml file was removed with Universal Messaging 10.15 Fix 10.

Removed Properties

The following Universal Messaging properties have been removed in Universal Messaging 10.15:

- **AutoCreatedStoreSpindleSize**

  A realm configuration property that is part of the Protocol MQTT Config group. This property became obsolete after the introduction of a file size limit on multi-file disk stores.

- **EnableCaching**

A realm configuration property that is part of the Global Values configuration group.

- **EnableConsumerStateMonitor**

  A realm configuration property that is part of the DurableConfig group. `EnableConsumerStateMonitor` enabled or disabled checks for unhealthy store consumers. Now the server automatically checks for such consumers.

- **EnableLegacyJMX**

  A realm configuration property that is part of the JVM Management configuration group.

- **NHPScanTime and NHPTimeout**

  Realm configuration properties that are part of the Global Values configuration group.

  `NHPScanTime` controlled how often the server checks HTTP connections that are not part of another monitoring pool, for example, HTTP 1.1 connections, which are long-lived. `NHPTimeout` defined how long an HTTP connection can be idle before the connection is closed. This behavior has been removed in favor of using a generic monitor pool for all connections. You can control the connection timeout by using the `IdleDriverTimeout` property, part of the **Connection Config** group.

Removed APIs

The following Universal Messaging APIs, classes, and methods have been removed in Universal Messaging 10.15:

- **C++ Client APIs**

  The following C++ APIs are removed:

```
nChannelAttributes#setClusterWide(bool flag)
nChannelAttributes#isClusterWide()
nNamedObject#isPersistent()
nProtobufEvent#setProperties(nEventProperties newProperties)
nStoreProperties#getEnableReadBuffering()
nStoreProperties#setEnableReadBuffering(bool flag)
nStoreProperties#getReadBufferSize()
nStoreProperties#isRequestPriorityConnection()
nStoreProperties#setRequestPriorityConnection(bool requestPriorityConnection)

Removed classes: nRegisteredEventUpdateListener,nRegisteredEventListener
```

- **C# Client APIs**

  The following C# APIs are removed:

```
nChannelAttributes#setClusterWide(bool flag)
nChannelAttributes#isClusterWide()
nNamedObject#isPersistent()
nProtobufEvent#setProperties(nEventProperties newProperties)
nStoreProperties#getEnableReadBuffering()
nStoreProperties#setEnableReadBuffering(bool flag)
nStoreProperties#getReadBufferSize()
nStoreProperties#isRequestPriorityConnection()
```

```
nStoreProperties#setRequestPriorityConnection(bool requestPriorityConnection)

Removed class: nRegisteredEventUpdateListener
```

- **Java Client APIs**

  The following Java Client APIs are removed:

```
nAbstractChannel#updateProtobufDefinitions(byte[][] descriptors)
nChannel#createIterator(nDurable name, String selector, int windowSize)
nChannel#createIterator(nDurable name, String selector, int windowSize, boolean
autoAck)
nChannelAttributes#setClusterWide(boolean flag)
nChannelAttributes#getFullName()
nConstants#setEnabledPriorityQueues(boolean enabled)
nConstants#getEnabledPriorityQueues()

nConstants#setPriorityQueueCount(int queueCount)
nConstants#getPriorityQueueCount()

nEventPropertiesIterator#next(boolean convert)
nEventPropertiesHelper#getKey(Object pair)
nEventPropertiesHelper#getKey(Object pair, boolean convert)
nEventPropertiesHelper#getValue(Object pair)
nEventPropertiesHelper#getValue(Object pair, boolean convert)
nEventProperties#nEventProperties(String serializedString)

nNamedObject#ack(long[] eids, boolean isSynchronous)
nNamedObject#ack(long[] eids)
nNamedObject#isPersistent()
nProtobufEvent#setProperties(nEventProperties newProperties)

nQueueReaderContext#canMaintainPriority()
nQueueReaderContext#setMaintainPriority(boolean flag)

nSession#setReadThreadAsDaemon(boolean isDaemon)
nSession#addConnectionQueueListener(nConnectionQueueListener listener, boolean
highWaterMark,
boolean lowWaterMark, boolean access, boolean push, boolean block, boolean unblock)

nSessionAttributes#isRequestPriorityConnection()
nSessionAttributes#setRequestPriorityConnection(boolean requestPriorityConnection)

nSharedDurableAttributes#getFilter()
nSharedDurableAttributes#setFilter(String filter)

nStoreProperties#getEnableReadBuffering()
nStoreProperties#setEnableReadBuffering(boolean flag)
comnStoreProperties#getReadBufferSize()
nStoreProperties#setReadBufferSize(long bufferSize)

nTransactionFactory#getPublishBufferSize()
nTransactionFactory#setPublishBufferSize(int size)
```

- **C++ Administration APIs**

  All C++ Administration APIs are removed. The following C++ applications are also removed:

```
DataGroupDeltaDelivery
DataGroupPublisher
```

```
DataStreamListener
addchanacl
addqueueacl
addrealmacl
LoginMonitor
authserver
Watchers
connectionwatch
dataGroupsManager
delchanacl
deleteDataGroup
delnodeacl
delqueueacl
delrealmacl
dumpACLs
modchanacl
modqueueacl
modrealmacl
ntop
realmconfig
```

- **C# Administration APIs**

  All C# Administration APIs are removed. The following C# applications are also removed:

```
addchanacl
addqueueacl
addrealmacl
addserviceacl
authserver
connectionwatch
delchanacl
delqueueacl
delrealmacl
dumpACL
modchanacl
modqueueacl
modrealmacl
nDiff
ntop
PasswordAuthentication
realmconfig
RealmMonitor
RealmStatusMonitor
removeServiceACL
XMLExport
XMLImport
```

- **Java Administration APIs**

  The following Java Administration APIs are removed:

```
nRealmNode#changeTransactionChannel(final int type, final long age, final int
capacity)
nRealmNode#changeStatusChannel(final int type, final long age, final int capacity)
nRealmACLEntry#canManageP2PServices()
nRealmACLEntry#setManageP2PServices(boolean val)
RealmACLEntry#isCreateP2PService()
RealmACLEntry#setCreateP2PService(boolean value)
RealmGroupACLEntry#isCreateP2PService()
```

```
RealmGroupACLEntry#setCreateP2PService(boolean value)
nContainer#modACLEntry(nACLEntry entry, boolean allChannels, boolean allQueues,
boolean allServices)
nContainer#modACLEntry(nACLEntry entry, boolean allChannels, boolean allQueues,
boolean allServices, boolean realm)
nContainer#addACLEntry(nACLEntry entry, boolean allChannels, boolean allQueues,
boolean allServices)
nContainer#addACLEntry(nACLEntry entry, boolean allChannels, boolean allQueues,
boolean allServices, boolean realm)
nContainer#removeACLEntry(nACLEntry entry, boolean allChannels, boolean allQueues,
 boolean allServices)
nContainer#removeACLEntry(nACLEntry entry, boolean allChannels, boolean allQueues,
 boolean allServices,boolean realm)
nContainer#setACL(nACL acl, boolean allChannels, boolean allQueues, boolean
allServices)
nRealmNode#addRealmACLEntry(nACLEntry entry)
nRealmNode#removeRealmACLEntry(nACLEntry entry)
nRealmNode#modRealmACLEntry(nACLEntry entry)
nLeafNode#isClusterWide()
nLeafNode#copy(nNode p_toNode, String p_toName, boolean p_waitForComplete, nACL
acl, nCommandStatus cmdStatus)
nLeafNode#copy(String p_toName, boolean p_waitForComplete, nACL acl, nCommandStatus
 cmdStatus)
nDurableNode#isClusterWide()
nClusterNode#addConnectionListenerToAllServiceNodes(final nConnectionListener
listener)
nAdminSession#AddAclGroupMember(String group, fSubject member)
nAdminSession#findStatusChannel()
nAdminSession#findTXChannel()
nAdminSession#changeTransactionFile(int type, long age, int capacity)
nAdminSession#changeStatusFile(int type, long age, int capacity)
ImportExportParametersBuilder#processScheduler()
ImportExportParametersBuilder#setSchedulerProcessing(boolean flag)
nClusterSite#getNodes()
nTopicNode#getDurables()
Removed classes: nRealmAdmin, nAddRealmAclEntryWithRealmAdmin, nServiceACLEntry,
nTXChannel, nStatusChannel
```

Note:
Instead of using the `nRealmAdmin` API, you can use the `nRealmNode` API to achieve the same results.

- **nJMS APIs**

  The following nJMS APIs are removed:

```
ConnectionFactoryImpl#getEnableSharedDurable()
ConnectionFactoryImpl#setEnableSharedDurable(boolean value)
ConnectionFactoryImpl#getEnableSerialDurable()
ConnectionFactoryImpl#setEnableSerialDurable(boolean value)
ConnectionFactoryImpl#setEnableSharedPriority(boolean value)
```

  Note:
  You can use `getDurableType()` and `setDurableType(nDurableAttributes.nDurableType type)` to get and set durable types.

- **Java nAdmin.apps APIs**

The following Java APIs are removed:

```
Removed classes: nModifyStatusChannel, nModifyTransactionChannel
```

- **Admin APIs related to the removal of support for local stores in a cluster**

  The following C# APIs are removed:

```
nAdminSession#findStatusChannel()
nAdminSession#findTXChannel()
nAdminSession#changeTransactionFile(int type, long age, int capacity)
nAdminSession#changeStatusFile(int type, long age, int capacity)
nRealmNode#changeTransactionChannel(int type, long age, int capacity)
nRealmNode#changeStatusChannel(int type, long age, int capacity)
Removed class: nStatusChannel
```

  The following C++ APIs are removed:

```
nRealmNode#changeStatusChannel(int type, long age, int capacity)
nRealmNode#changeTransactionChannel(int type, long age, int capacity)
```

- **Data group and data stream APIs**

  The following Java client APIs, and the respective APIs in C# and C++, are removed:

```
nSession.init(nDataStreamListener streamListener)
nSession.getStreamId()
nSession.pauseStream()
nSession.resumeStream()
All nSession.writeDataStream(...) APIs
All nSession.writeDataGroup(...) APIs
All nSession.commitRegisteredEventsToDataGroups(...) APIs
All nSession.createDataGroup() APIs
All nSession.deleteDataGroup() APIs
All nSession.getDefaultDataGroup() APIs
All nSession.getDataGroups() APIs
Removed classes: nDataStream, nDataGroup, nDataGroupListener
```

  The following administration APIs in Java, and the respective APIs in C# and C++, are removed:

```
nRealmNode.getDefaultDataGroup(nDataStreamListener streamListener)
nRealmNode.getDataGroupListener()
nRealmNode.getDataGroups()
nRealmNode.findDatagroup(String name)
Removed classes: nDataStreamNode, nDataGroupNode, nDataGroupsContainer
```

Documentation Changes

This section describes major changes in the documentation for the 10.15 release:

- The information about running, administering, and monitoring Universal Messaging containers has been expanded and moved from the *Installation Guide* to a new topic, "Running, Administering, and Monitoring Universal Messaging Containers", in the *Operations Guide*.
- You can find information about using Nginx as a reverse proxy server for Universal Messaging in the topic "Using Nginx with Universal Messaging" in the *Administration Guide*.

- The information about using JMX to monitor and manage Universal Messaging has been enhanced and moved from the *Concepts* guide to a new topic, "Using JMX to Monitor Universal Messaging", in the *Administration Guide*.
- The topic "Monitoring Universal Messaging Using Prometheus" has been moved from the *Operations Guide* to the *Administration Guide*.

3 What's New in Universal Messaging 10.11

Universal Messaging 10.11 is the successor of Universal Messaging 10.7.

Universal Messaging 10.11 includes new features, enhancements, and changes as described in the following topics.

New Features

The following features have been added in Universal Messaging 10.11:

- **Support for creating Docker images for Universal Messaging using Software AG Installer**

  You can use Software AG Installer to create Docker images for a Universal Messaging server on Linux operating systems. The Installer client creates a Docker image with a Universal Messaging server entry point and support for command-line tools similar to the images provided in the Software AG repository on Docker Hub. For a detailed description of this feature with usage examples, see the documentation about creating Docker images with Software AG Installer on Linux in the *Using Software AG Installer* guide.

- **Support for exporting Protobuf definitions from a store**

  You can export Protobuf definitions from a Universal Messaging store in the following ways:

  - Using a new command-line tool named `ExportProtobufDefinitions`.
  - Using the Enterprise Manager. A new option, **Export Protobuf Definitions**, has been added to the context menu for a store.

- **Server support for TLS 1.3**

  For more information about configuring SSL/TLS in Universal Messaging, see the section "Security > Using SSL > Server SSL Configuration" in the *Concepts* guide.

- **Support for JAAS authentication using HTTP headers**

  You can configure Universal Messaging to use custom HTTP headers from the client HTTP connection request for JAAS authentication. To enable JAAS authentication using HTTP headers, you configure a new system property, `com.softwareag.um.server.authentication.http.extraHeaders`.

  For more information about JAAS authentication with HTTP headers, see the section "Security > Authentication > JAAS Authentication with HTTP Headers" in the *Concepts* guide.

- **Support for authentication when clients do not provide a password**

  You can use JAAS authentication with HTTP headers to authenticate clients that did not provide a password. To enable authentication without a password, you configure a new system property, `com.softwareag.um.server.authentication.simpleAuthenticationEnabled`.

  For more information about authenticating clients that did not provide a password, see the section "Security > Authentication > Enabling Authentication Without a Password" in the *Concepts* guide.

- **Support for client SSL certificates in PKCS12 format**

- **Bi-directional client compatibility**

  Universal Messaging is backwards and forwards compatible with the Universal Messaging Java client, JMS, and Resource Adapter. In addition to clients connecting to the same or higher-release servers, Universal Messaging enables Java clients version 10.7 and higher to connect to lower-release servers. The feature was introduced in Universal Messaging 10.7.

  For more information about the functionality and compatible client and server versions, see the section "Deployment > Bi-directional Client and Server Compatibility" in the *Concepts* guide.

- **New release identifiers**

  You can use the following objects and APIs on the Java client, which were added in version 10.7, to identify the release version of the components that are in use in your landscape:

  - com.pcbsys.nirvana.client.ReleaseDetails - object that provides release and URL details.
  - com.pcbsys.nirvana.client.ReleaseIdentifier - object that provides a list of release identifiers.
  - com.pcbsys.nirvana.client.nSession APIs:
    - getClientReleaseIdentifier() - returns the current client release.
    - getServerReleaseIdentifier() - returns the current server release or, in case of a horizontal scalability (HS) connection, the release versions of the HS servers.
    - getConnectionProtocol() - returns the negotiated release between the server and client components.

  For more details, see the Universal Messaging Java Client API documentation.

- **Horizontal scalability (HS) behaves consistently in mixed environments**

  Now that Universal Messaging is backwards and forwards compatible with Java clients, for consistency in a mixed server environment, you must restrict the HS URL of your application to use the lowest server release version available in the HS landscape. For the purpose, you can use a new parameter, `hsReleaseVersion`, as part of the HS URL. This functionality has been delivered with Universal Messaging 10.7 Fix 1.

  For more information about the functionality and about configuring the `hsReleaseVersion` parameter, see the section "Horizontal Scalability > Horizontal Scalability Behavior in a Mixed Landscape" in the *Concepts* guide.

- **A new command-line tool, ServerReleaseVersion, that generates HS URLs containing the hsReleaseVersion parameter**

  You can use the `ServerReleaseVersion` tool to obtain the release versions for a list of Universal Messaging servers or for servers that are part of a Horizontal Scalability (HS) URL. For HS servers, the tool generates an HS URL that contains the `hsReleaseVersion` parameter.

  For more information about using the tool, see the section "Command Line Administration Tools > The ServerReleaseVersion Tool" in the *Administration Guide*.

- **Support for NHP, NHPS interfaces with AMQP, MQTT protocols**

Universal Messaging can use the NHP and NHPS interfaces to communicate with MQTT and AMQP protocol clients. This makes configuring Universal Messaging easier because the NHP and NHPS interfaces can support all protocols of the server on a single interface.

- **com.softwareag.um.client.missed_keep_alives client property**

  A new client property that specifies the number of missed server-side keep-alive intervals before the client closes a connection. The property is introduced in Universal Messaging 10.11 Fix 12.

  For more information about the property, see the topic "Client Parameters" in the *Concepts* guide.

- **MaxNoOfConnectionsPerUserName realm property**

  A new realm configuration property that defines the number of concurrent connections to a Universal Messaging server per user. The property is introduced in Universal Messaging 10.11 Fix 13. For more information about using the property, see the section "Realm Configuration" in the *Administration Guide*.

- **New HealthChecker Tool Check**

  The new `JoinLastEIDMismatchCheck` checks for joins that are ahead of the channel's last EID. The check is introduced in Universal Messaging 10.11 Fix 14. For more information, see the section "Running a Configuration Health Check" in the *Administration Guide*.

Changed Features

The following Universal Messaging features already available in the previous product release have been changed in Universal Messaging 10.11:

- **C++ libraries**

  The 10.11 release includes the following changes related to the C++ libraries that Universal Messaging provides:

  - The C++ libraries for Universal Messaging on Windows are now compiled with Visual Studio 2019.
  - The C++ libraries for Universal Messaging on Linux are now compiled with gcc compiler version 8.3.1 (Red Hat 8).
  - The C++ libraries for Universal Messaging use OpenSSL libraries that have now been upgraded to version 1.1.1k from version 1.1.1f.
  - The C++ libraries for Universal Messaging use POCO libraries that have now been upgraded to version 1.10.1 from version 1.9.0.

  If you experience any problems with the new version of the C++ client libraries for Universal Messaging, first consider recompiling your client applications against the new version of the libraries and compilers.

- **MaxNoOfConnections realm property**

A realm configuration property that defines the total number of concurrent connections to a Universal Messaging server. Now a user with *both* the **Override Connection Count** realm permission and an admin connection can override this property. Previously, users with full permissions could override the property. The change is introduced in Universal Messaging 10.11 Fix 13. For more information about using `MaxNoOfConnections`, see the section "Realm Configuration" in the *Administration Guide*.

- **Support for plugins depending on the license type**

  Universal Messaging now supports plugins for all UM license types regardless of the `Plugins` key value in the license file. Previously, plugin support was available only for the NUMWF/NUMTF license type. The change is introduced in Universal Messaging 10.11 Fix 17.

Deprecated Features

The following Universal Messaging features have been deprecated in Universal Messaging 10.11:

- **C++ client API**
- **Client APIs related to the removal of support for local stores in a cluster**

  The following Java APIs are deprecated:

```
nAdminSession#findStatusChannel()
nAdminSession#findTXChannel()
nAdminSession#changeTransactionFile(int type, long age, int capacity)
nAdminSession#changeStatusFile(int type, long age, int capacity)
nStatusChannel - the class itself
nTXChannel - the class itself
nDurableNode#isClusterWide()
nLeafNode#isClusterWide()
nRealmNode#changeTransactionChannel(final int type, final long age, final int
capacity)
nRealmNode#changeStatusChannel(final int type, final long age, final int capacity)
nChannelAttributes#setClusterWide(boolean flag)
nChannelAttributes#isClusterWide()
```

  The following C# APIs are deprecated:

```
nChannelAttributes#setClusterWide(bool flag)
nChannelAttributes#isClusterWide()
nAdminSession#findStatusChannel()
nAdminSession#findTXChannel()
nAdminSession#changeTransactionFile(int type, long age, int capacity)
nAdminSession#changeStatusFile(int type, long age, int capacity)
nStatusChannel - the class itself
nRealmNode#changeTransactionChannel(int type, long age, int capacity)
nRealmNode#changeStatusChannel(int type, long age, int capacity)
```

  The following C++ APIs are deprecated:

```
nChannelAttributes#setClusterWide(bool flag)
nChannelAttributes#isClusterWide()
nRealmNode#changeStatusChannel(int type, long age, int capacity)
nRealmNode#changeTransactionChannel(int type, long age, int capacity)
```

- **Cluster-wide flag for various Universal Messaging resources**
  - Command-line administration tools:

    ```
    CreateChannel
    CreateQueue
    SubscribeChannelAsyncDurable
    SubscribeChannelDurable
    ```

  - Client samples:

    ```
    makeChannel
    makeQueue
    namedChannelIterator
    namedsubscriber
    ```

  - C# samples:

    ```
    makechan
    makechannels
    makequeue
    namedChannelIterator
    namedsubscriber
    ```

  - The "ClusterWide" JMX property for stores
- **Data group and data stream functionality**

  The following Java client APIs, and the respective APIs in C# and C++, are deprecated:

  ```
  nSession.init(nDataStreamListener streamListener)
  nSession.getStreamId()
  nSession.pauseStream()
  nSession.resumeStream()
  All nSession.writeDataStream(...) APIs
  All nSession.writeDataGroup(...) APIs
  All nSession.commitRegisteredEventsToDataGroups(...) APIs
  All nSession.createDataGroup() APIs
  All nSession.deleteDataGroup() APIs
  All nSession.getDefaultDataGroup() APIs
  All nSession.getDataGroups() APIs
  Deprecated classes: nDataStream, nDataGroup, nDataGroupListener
  ```

  The following administration APIs in Java, and the respective APIs in C# and C++, are deprecated:

  ```
  nRealmNode.getDefaultDataGroup(nDataStreamListener streamListener)
  nRealmNode.getDataGroupListener()
  nRealmNode.getDataGroups()
  nRealmNode.findDatagroup(String name)
  Deprecated classes: nDataStreamNode, nDataGroupNode, nDataGroupsContainer
  ```

Removed Features

The following Universal Messaging features have been removed in Universal Messaging 10.11:

Removed Functionality

The following Universal Messaging functionality has been removed in Universal Messaging 10.11:

- **Enterprise Manager Scheduler**

  The Scheduler functionality in the Enterprise Manager, which enabled you to schedule tasks using a specific scheduler language, has been removed.

  `RealmSchedulerSet` is no longer exported. If you perform an import from an older server version, the `RealmSchedulerSet` XML section will be ignored.

- **GetEventsInfo command-line administration tool**

  The `GetEventsInfo` functionality has been replaced by the `ExportEventsFromOfflineMemFile` tool.

- **Support for local stores in a cluster**

  You can create only cluster-wide stores on the realms in a cluster. When you add realms to a new or existing cluster, all local stores are either migrated to cluster stores or deleted.

- **Client-side JAAS authentication**

  You can use the server-side SASL and JAAS components as a replacement.

- **Fanout archive**

Removed Properties

The following Universal Messaging properties have been removed in Universal Messaging 10.11:

- **System properties related to client JAAS authentication**

  The following properties have been removed:

  ```
  Nirvana.enable.legacy.jaas.auth
  Nirvana.auth.server.jaascompat
  ```

- **TimeoutSynchronousConsumerOnMaster realm configuration property**

Removed APIs

The following Universal Messaging APIs, classes, and methods have been removed in Universal Messaging 10.11:

- **Python client API**
- **XML document type APIs**

  The following Java client APIs have been removed:

  ```
  nConsumeEvent#nConsumeEvent(String tag, byte[] data, boolean dom)
  ```

```
nConsumeEvent#nConsumeEvent(String tag, Document adom)
nConsumeEvent#nConsumeEvent(nEventProperties properties, Document adom)
nConsumeEvent#isDom()
nConsumeEvent#getDocument()
nConsumeEventFactory#create(String tag, byte[] data, boolean isDom,
long TTL, boolean isPersistent)
nConsumeEventFactory#create(nEventProperties props, Document dom, long TTL,
boolean isPersistent)
nConsumeEventFactory#create(String tag, Document dom, long TTL, boolean isPersistent)
nChannel#publish(String tag, Document adom)
nChannel#publish(String tag, Document adom, nMessageSigner signer)
nQueue#push(String tag, Document adom)
nQueue#push(String tag, Document adom, nMessageSigner signer)
nTransaction#publish(String tag, Document adom)
```

The sample applications `XMLPublish` and `XMLSubscribe` have also been removed.

Additionally, support for XML has been discontinued from the REST plugin.

- **APIs related to the Scheduler functionality**

  The following administration APIs in Java, and the respective APIs in C#, have been removed:

  ```
  nAdminSession.addScheduler(String source, String subject, boolean isClusterWide)
  nAdminSession.delScheduler(String name, boolean cluster)
  nSchedulerConfiguration nAdminSession.getSchedulerConfig()
  nSchedulerConfiguration class
  nSchedulerManager nRealmNode.getSchedulerManager()
  nSchedulerManager class
  nScheduler class
  nRealmNode.delSchedulerNode(final nScheduler node)
  ```

- **Client-side DisconnectOnClusterFailure control flag**

  The client-side `DisconnectOnClusterFailure` control flag and the following client APIs that use it have been removed. Now all non-administrative client applications are disconnected on cluster failure.

  The following Java APIs under `com.pcbsys.nirvana.client` have been removed:

  ```
  nSession#init(boolean disconnectIfClusterFails)
  nSession#init(boolean disconnectIfClusterFails, nDataStreamListener streamListener)
  nSessionAttributes#isDisconnectOnClusterFailure()
  nSessionAttributes#setDisconnectOnClusterFailure(boolean disconnectOnClusterFailure)
  ```

  The following C# APIs have been removed:

  ```
  nSession#init(bool fails)
  nSession#init(bool fails, nDataStreamListener streamListener)
  ISessionAttributes#DisconnectIfClusterFails
  ```

- **Client APIs related to the removal of support for local stores in a cluster**

  The following Java APIs have been removed:

  ```
  nDurableAttributes#isClustered()
  nDurableAttributes#setClustered(boolean clustered)
  ```

The following C# APIs have been removed:

```
nDurableAttributes#isClustered()
nDurableAttributes#setClustered(bool clustered)
nLeafNode#isClusterWide()
```

The following C++ APIs have been removed:

```
nChannel#createNamedObject(const std::string& name, longlong startEID,
bool persistent, bool clusterWide)
nChannel#createNamedObject(const std::string& name, bool persistent, bool
clusterWide)
nNamedObject#isClusterWide()
```

- **Persistent flag for durables and the related nDurableAttributes.setPersistent and isPersistent methods**

  Now the `persistent` flag for durables is implicitly set by the API depending on whether the channel on which the durable exists supports persistence. Currently channels of type Mixed and Persistent support persistence. For these channel types, the flag is implicitly set to `true`, while for all other channel types the flag defaults to `false`.

  The following Java APIs, and the respective C# and C++ APIs, have been removed:

  ```
  nDurableAttributes.isPersistent()
  nDurableAttributes.setPersistent(boolean persistent)
  nDurableNode.isPersistent()
  ```

  In addition, the `persistent` parameter has been removed from the following command-line administration tools:

  ```
  SubscribeChannelDurable
  SubscribeChannelAsyncDurable
  ```

- **Multiplex sessions in C# and C++ client APIs**

  The following C# APIs have been removed:

  ```
  nSession nSessionFactory.createMultiplexed(nSession session,
  NetworkCredential newCreds = null)
  ```

  The following C++ APIs have been removed:

  ```
  nSession* createMultiplexed(nSession *pSession);
  nSession* createMultiplexed(nSessionAttributes *sessionAttributes);
  nSession* createMultiplexed(nSessionAttributes *sessionAttributes, std::string
  userName);
  ```

- **Message signer APIs**

  The following Java APIs, and the respective APIs in C# and C++, have been removed:

  ```
  nChannel.publish(String tag, Document adom, nMessageSigner signer)
  nChannel.publish(nConsumeEvent e, nMessageSigner signer)
  nChannel.addSubscriber(nSignedEventListener nel, nDurable durable, String selector,
  nMessageValidator validator,boolean AutoAck)
  nChannel.addSubscriber(nSignedEventListener nel, nDurable durable, String selector,
  ```

```
nMessageValidator validator,int windowSize)
nChannel.addSubscriber(nSignedEventListener nel, nMessageValidator validator)
nChannel.addSubscriber(nSignedEventListener nel, long eid, nMessageValidator
validator)
nChannel.addSubscriber(nSignedEventListener nel, String selector, nMessageValidator
 validator)
nChannel.addSubscriber(nSignedEventListener nel, String selector, long eid,
nMessageValidator validator)
nChannel.addSubscriber(nSignedEventListener nel, nDurable name, String selector,
nMessageValidator validator, boolean AutoAck, int windowSize)
nQueue.createFragmentReader(final nQueueReaderContext context)
nQueue.createTransactionalFragmentReader(final nQueueReaderContext context)
nQueue.push(String tag, Document adom, nMessageSigner signer)
nQueue.push(final nConsumeEvent e, final nMessageSigner signer)
nQueueReaderContext(final nSignedEventListener aListener, final nMessageValidator
 aValidator)
nQueueReaderContext(final nSignedEventListener aListener, final nMessageValidator
 aValidator,
final int windowSize)
nQueueReaderContext(final nSignedEventListener aListener, final nMessageValidator
 aValidator,
final String selector)
nQueueReaderContext(final nSignedEventListener aListener, final nMessageValidator
 aValidator,
final String selector, final int windowSize)
nQueueReaderContext(final nSignedEventListener aListener, final nMessageValidator
 aValidator,
final String selector, final int windowSize, boolean maintainPriority)
nQueueReaderContext(final nSignedEventListener aListener, final nMessageValidator
 aValidator,
final String selector, final int windowSize, boolean maintainPriority, long timeout)
nQueueReaderContext.getValidator()
nQueueReaderContext.setValidator(final nMessageValidator validator)
nTransactionAttributes(nAbstractChannel aChannel, long aTimeToLive,
nMessageSigner signer)
nTransactionAttributes(nAbstractChannel aChannel, long aTimeToLive,
nMessageSigner signer, long timeout)
byte[] nConsumeEvent.sign(nMessageSigner signer)
Object nConsumeEvent.validate(nMessageValidator validator)
nCertificateSigner class
nCertificateValidator class
nMessageSigner class
nPassphraseSigner class
nMessageValidator class
nPassphraseValidator class
nSignedEventListener class
```

- **Compressed APIs**

  The following Java API classes, and the respective C# and C++ API classes, have been removed:

```
nConsumeEventCompressReader
nConsumeEventCompressWriter
nConsumeEventFragmentReader
nConsumeEventFragmentWriter
nQueueSyncFragmentReader
nQueueSyncTransactionFragmentReader
```

  The `PublishCompressed` and `SubscribeCompressed` command-line tools have also been removed.

- **APIs related to the fanout archive functionality**

  The following Java APIs, and the respective C# and C++ APIs, have been removed:

  ```
  nChannelAttributes.getFanoutArchiveTarget()
  nChannelAttributes.setFanoutArchiveTarget(String name)
  nStoreProperties.getFanoutArchiveTarget()
  nStoreProperties.setFanoutArchiveTarget(String name)
  ```

  In addition, the `fanoutarchivetarget` parameter has been removed from the following command-line tools:

  ```
  CreateQueue
  CreateChannel
  GetChannelInfoOutput
  ```

- **com.pcbsys.nirvana.nJMS.roundRobin package**

Documentation Changes

- *Enterprise Developer's Guide for Python* has been removed.
- The *Scheduling* documentation in the *Administration Guide* has been removed.
- A new section, "Handling Multiple Connections", has been added to the *Operations Guide*. It provides information about how to handle multiple connections both on the client side and server side, and how to resolve some related issues that you might encounter.
- The information about using Prometheus in Universal Messaging has been moved from the *Concepts* guide to the section "Monitoring Containers" in the *Operations Guide*.

4 What's New in Universal Messaging 10.7

Universal Messaging 10.7 is the successor of Universal Messaging 10.5.

Universal Messaging 10.7 includes new features, enhancements, and changes as described in the following topics.

New features in v10.7

The following Universal Messaging features have been added in Universal Messaging 10.7:

- **JMS Engine automatic purging of individual events**

  The automatic purging feature of the JMS engine has been extended to include automatic purging of individual events.

  By default, automatic purging of individual events in a channel is activated.

  For details, see the section *Automatic purging of individual events* in the *Developer Guide*.

- **Enterprise Manager - Adding client API build number in realm connections list**

  The Enterprise Manager display showing the list of client connections for a realm has been extended to include an additional column **build number**. This column lists the version of the client API that was used to build the client application.

  For details, see the section *Realm Connections* in the *Enterprise Manager* part of the *Administration Guide*.

- **Support for Command Central asset inventory commands**

  You can view and install Universal Messaging assets from an asset repository, using the Command Central asset inventory commands.

  Important:
  The Command Central asset inventory commands are a preview feature that is subject to change in the future without any deprecation announcements. This preview has limited functions and is not intended for use in a production environment. If you want to provide feedback for this preview feature, go to the Command Central area on the Software AG TechCommunity website.

  For details, see the section *Asset Deployment* in the *Command Central* part of the *Administration Guide*.

- **New command line tools**

  The following command line tools have been added:

Tool name	Description
`AddHealthMonitorPlugin`	This tool adds an HTTP REST endpoint to the URL of the session to which the realm server is connected, allowing you to access information about health checks that run on the server at periodic intervals. See the entry *Health Monitor Plugin* below for details.
`ExportEventsFromOfflineMemFile`	This tool dumps events from the .mem files of a Universal Messaging realm server store to XML or JSON format. The events can be re-imported using the `RepublishEventsFromOfflineFile` tool.
`ListInterfaces`	This tool lists the details of all of a realm's interfaces.
`RepublishEventsFromOfflineFile`	This tool imports events from XML or JSON files that were created by the `ExportEventsFromOfflineMemFile` tool and republishes them to the specified Universal Messaging realm server store. The tool can also import events from a single .mem file or from a directory which contains .mem files.

  For details, see the sections *Syntax: Interface Tools* and *Syntax: Recovery Tools* in the *Command Line Administration Tools* part of the *Administration Guide*.

- **Getting the current status of the realm service/daemon**

  The command line command `nserverdaemon -q` has been added to allow Windows users to query the current status of the Universal Messaging server.

  For details, see the section *Getting the current status of the realm service/daemon* in the *Installation Guide*.

- **Event lifecycle logging (trace logging)**

  Event lifecycle logging, also called *trace logging*, is used for creating detailed log files that describe the flow of events as they pass from client to Universal Messaging server, or vice versa, through channels or queues.

  For details, see the section *Event Lifecycle Logging* in the *Concepts* guide.

- **Product usage metering**

  Product usage metering is a feature that allows Universal Messaging to collect metrics relating to the usage of the product, such as total number of events sent and received, and total number of bytes sent and received. If a transaction-based product license is activated, the metrics can be used as a basis of transaction-based pricing.

  For details, see the section *Product Usage Metrics* in the *Concepts* guide.

- **Health Monitor Plugin**

  A new health monitor plugin is available, which adds an HTTP REST endpoint to the URL of the session to which the realm server is connected. This allows clients to query the current state of the realm server. The endpoint defines the "liveness" of the server. The plugin returns the result of the health checks that run on the server at periodic intervals. You can add the plugin using either the Enterprise Manager or a command line interface.

  Information on the use of the plugin is available in the section *Health Monitor Plugin* in the *Administration Guide*.

- **Active/Active Cluster Mode "Replication"**

  Universal Messaging Active / Active clustering now supports a new mode - ***Replication***. This mode ensures that non-admin client connections can be established only on the master node, and the other cluster nodes are used only to replicate the master's state. In this mode, nodes that are not the master node will reject all non-admin connections. This is similar to when the client's ***FollowTheMaster*** (FTM) flag is set, so this mode behaves like a server-side FTM.

  To achieve this behavior in clients of earlier Universal Messaging versions, the FTM flag had to be set for each client. Also this capability was available only in the Java client API.

  To enable the new Replication mode, a new property `ClusterMode` is available in the **Cluster Config** configuration group of the realm server properties. Setting `ClusterMode` to "Replication" activates the Replication mode.

  If you are using C# or C++ client version 10.3 or 10.5, make sure that the client is upgraded with the latest client libraries, before enabling Replication mode. Also, C#/C++ clients in v10.1 and earlier cannot make use of the feature.

  In general, the new Replication mode might be chosen to resolve performance or cluster synchronization issues where we have Universal Messaging clients connected to all cluster nodes simultaneously.

  For details, see the topic *Cluster Mode Configuration* in the section *Client Connection* in the *Concepts* guide.

- **Bi-directional client compatibility**

  Universal Messaging is now backwards and forwards compatible with the Universal Messaging Java client, JMS, and Resource Adapter. In addition to clients connecting to the same or higher-release servers, Universal Messaging now enables Java clients version 10.7 and higher to connect to lower-release servers.

  For more information about the functionality and compatible client and server versions, see the section "Deployment > Bi-directional Client and Server Compatibility" in the *Concepts* guide.

- **New release identifiers**

  You can use the following objects and APIs on the Java client to identify the release version of the components that are in use in your landscape:

  - com.pcbsys.nirvana.client.ReleaseDetails - object that provides release and URL details.

- com.pcbsys.nirvana.client.ReleaseIdentifier - object that provides a list of release identifiers.
- com.pcbsys.nirvana.client.nSession APIs:
  - getClientReleaseIdentifier() - returns the current client release.
  - getServerReleaseIdentifier() - returns the current server release or, in case of a horizontal scalability (HS) connection, the release versions of the HS servers.
  - getConnectionProtocol() - returns the negotiated release between the server and client components.

For more details, see the Universal Messaging Java Client API documentation.

- **Prefetch APIs for synchronous message consumers**

  Synchronous consumers now use a client-defined *prefetch* size parameter to define the maximum number of events which the consumer can receive in a single call. When events are delivered in batches, the whole batch of events is returned to the consumer and no events stay cached client-side.

  In Universal Messaging versions prior to v10.7, the client had no control over the number of events returned in a single call; instead, the number of events returned by the server was controlled by the defined window and batch sizes. Also if events were delivered in batches some of them were cached in the client without this being transparent, as the APIs were returning only one event at a time.

  In Universal Messaging v10.7 we refer to a client-defined prefetch size, which replaces both window-size and batch parameters, to describe the maximum number of events returned to a synchronous consumer in a single call.

  The window size used up to Universal Messaging v10.5 is removed in Universal Messaging v10.7 clients.

  Note:
  While the new synchronous consumers will not be using a window size, clients from before the prefetch feature's changes will still be doing so. The new server will honor the window size (and the v10.5 queue batching) for older clients. This was done to preserve the behavior of old consumer applications until they can be migrated.

  Note:
  This new prefetch processing only applies to **synchronous** consumers. The server-defined window size still applies in Universal Messaging v10.7 for **asynchronous consumers**.

  For a description of the new prefetch API, including a list of the new APIs, see the section *Using the Prefetch API* in the *Concepts* guide.

  For information about the deprecated APIs for the old window-style processing for synchronous consumers, see the topic *Window-related APIs for synchronous consumers* in the section *Deprecated features in v10.7* in these Release Notes.

- **MaxNoOfConnectionsPerUserName realm property**

A new realm configuration property that defines the number of concurrent connections to a Universal Messaging server per user. The property is introduced in Universal Messaging 10.7 Fix 14. For more information about using the property, see the section "Realm Configuration" in the *Administration Guide*.

Changed features in v10.7

The following Universal Messaging features already available in the previous product release have been changed in Universal Messaging 10.7:

- **Realm configuration parameter `PauseServerPublishing`**

  The realm server configuration property `PauseServerPublishing`, which you can use to pause the publishing of new events from clients to the realm server, has moved from the "Advanced" category to the "Basic" category. This means that if you are viewing the set of realm server properties in the Enterprise Manager, you will now find the property under the **Global Values** configuration group under the **Basic** heading.

  The purpose of the property remains unchanged.

  See the description of the realm configuration parameter `PauseServerPublishing` in the section *Realm Configuration* of the *Administration Guide*. See also the description of the Pause Publishing feature in the section *Pause Publishing* of the *Concepts* guide

- **Changed default value for client logging level**

  The default log level for client applications is now 7 (OFF). This value can be changed by using the parameter `LOGLEVEL` when the client application is started from the command line.

  See the section *Client Parameters* in the *Concepts* guide for information on setting client application parameters.

- **Out-of-memory configuration has been modified**

  The order of the realm server configuration parameters `FlowControlWaitTimeOne` and `FlowControlWaitTimeThree` has been swapped, so that `FlowControlWaitTimeOne` is triggered at a threshold of 70% (previously 90%) of memory capacity, and `FlowControlWaitTimeThree` at 90% (previously 70%). The threshold for `FlowControlWaitTimeTwo` remains unchanged at 80%.

  The processing when the realm server configuration parameter `EmergencyThreshold` is exceeded has changed. Previously, the server would aggressively remove as many client requests as necessary in order to reduce the memory requirements to under the `EmergencyThreshold` value. Now, the server blocks client requests for a long period of time, namely 24 days.

  The realm server configuration parameter `ThrottleAllPublishersAtThreshold` has been removed.

  For details, see the section *Out-of-Memory Protection* in the *Concepts* guide.

- **Automated scan of free disk space**

The way in which the realm server performs automated scans of free disk space has changed. Previously, this was managed by the server startup parameters `DISK_USAGE_SCAN_ENABLE`, `DISK_USAGE_FREE_THRESHOLD` and `DISK_USAGE_SCAN_INTERVAL`.

Now, functionality managed by these parameters is provided by the new realm server configuration parameters `DiskScanEnable`, `DiskUsageFreeThreshold` and `DiskUsageScanInterval`.

By moving this configuration from the server startup parameters to the realm server configuration parameters, it is now possible to change the configuration while the realm server is running.

Additionally, the old server startup parameter `EXIT_ON_FREE_SPACE_ERROR` has been removed.

Note:

The server startup parameter `DISK_USAGE_SCAN_ENABLE` is still available in v10.7, but marked as deprecated. See the section *Deprecated features in v10.7* below for additional information.

Under certain circumstances, the value of `DISK_USAGE_SCAN_ENABLE` can override the value of `DiskScanEnable`. The server startup parameters `DISK_USAGE_FREE_THRESHOLD` and `DISK_USAGE_SCAN_INTERVAL` are no longer available in v10.7.

For details of the new parameters, see the section *Realm Configuration* in the *Administration Guide*.

- **Client disconnect when server leaves the cluster**

  An option has been added to configure the Universal Messaging server to disconnect client sessions if the server falls out of the cluster. Up to now, such an option was available only client side using the APIs `nSessionAttributes.setDisconnectIfClusterFails()` and `nSession.init(bool disconnectIfClusterFails)`, and there was no global option to set this behavior for all client connections of a server.

  The option doesn't affect Admin connections.

  To enable the feature use the realm configuration property `DisconnectWhenNotReady`. This property existed in previous product versions, but was used in a slightly different way. Similarly, the purpose of the realm configuration property `DisconnectWait` has been slightly modified to fit in with the new `DisconnectWhenNotReady` handling.

  For details of `DisconnectWhenNotReady` and `DisconnectWait`, see the configuration group *Cluster Config* in the section *Realm Configuration* in the *Administration Guide*.

- **JMX Handling and Prometheus support**

  There have been updates in the way that Universal Messaging manages JMX beans. Also, it is now possible to export JMX metrics for subsequent analysis by Prometheus, which is an open source monitoring system for time series data.

  Several realm configuration properties have been modified or added:

  - The purpose of `EnableJMX` has been modified slightly, so that it only applies to the JMX handling provided in Universal Messaging v10.7 and upwards.

- There is a new property `EnableLegacyJMX`, which enables the JMX handling that was available in Universal Messaging versions up to and including v10.5.
- There is a new property `JMXRMIPort` for defining the port number to be used for JMX RMI connections.

In Universal Messaging v10.7, tools such as JConsole that can display JMX metrics from Universal Messaging can show the new view **com.softwareag.um.server** in addition to the known view **NirvanaRealmServer**. The view **com.softwareag.um.server** is visible in JConsole if you enable `EnableJMX`, and the view **NirvanaRealmServer** is visible in JConsole if you enable `EnableJMX`.

- **MaxNoOfConnections realm property**

  A realm configuration property that defines the total number of concurrent connections to a Universal Messaging server. Now a user with *both* the **Override Connection Count** realm permission and an admin connection can override this property. Previously, users with full permissions could override the property. The change is introduced in Universal Messaging 10.7 Fix 14. For more information about using `MaxNoOfConnections`, see the section "Realm Configuration" in the *Administration Guide*.

Deprecated features in v10.7

The following Universal Messaging features are now deprecated in Universal Messaging 10.7. Features listed as deprecated are still available in the product, but will be removed in a future release.

- **Command line tool GetEventsInfo**

  This tool has been deprecated in Universal Messaging v10.7 and will be removed from the product in a future release.

  The functionality of `GetEventsInfo` is covered by the new command line tool `ExportEventsFromOfflineMemFile` introduced in v10.7.

- **Client-side authentication with JAAS**

  The client side authentication with JAAS has been deprecated in Universal Messaging v10.7 and will be removed from the product in a future release. This deprecation affects the use of the system property `Nirvana.auth.client.jaaskey` for setting up client-side authentication.

- **Deprecated APIs and Methods**
  - **Methods setHeader and getHeader in nConsumeEvent API**

    The methods `setHeader` and `getHeader` in the `com.pcbsys.nirvana.base.nConsumeEvent` API have been deprecated. You should access `nHeader` fields only through the setters and getters exposed in the `nConsumeEvent` API.

  - **nRealmAdmin API**

    This API has been deprecated in Universal Messaging v10.7 and will be removed from the product in a future release.

The functionality of the `nRealmAdmin` API is covered by the existing `nRealmNode` API.

- **Client side disconnectOnClusterFailure control flag**

  The client side `DisconnectOnClusterFailure` control flag and all APIs that use it have been deprecated. In future, all non-administrative client applications will be disconnected on cluster failure, so the client side control flag is redundant.

  The following Java APIs under `com.pcbsys.nirvana.client` have been deprecated:

  ```
  nSessionAttributes#isDisconnectOnClusterFailure()
  nSessionAttributes#setDisconnectOnClusterFailure(boolean
  disconnectOnClusterFailure)
  ```

  The following C++ API has been deprecated:

  ```
  nSession.init(bool disconnectOnClusterFailure)
  ```

  The following C# API has been deprecated:

  ```
  nSession.init(bool fails)
  ```

- **Method setProperties in nProtobufEvent API**

  The method `setProperties` in the `nProtobufEvent` client APIs for Java, C# and C++ has been deprecated. If the method is used from now on it will do nothing.

  If old clients are connected to a v10.7 server and properties are set explicitly on a Protobuf Event they will not be processed.

- **APIs for Fragment/Compress for Readers and Writers**

  All APIs for Fragment/Compress Readers and Writers have been deprecated in Java, C# and C++.

  The affected APIs are:

  ```
  nQueueSyncFragmentReader
  nQueueSyncTransactionFragmentReader
  nConsumeEventCompressWriter
  nConsumeEventCompressReader
  nConsumeEventFragmentReader
  nConsumeEventFragmentWriter
  ```

  Also deprecated are any tools or samples which use these APIs, and any other public API related to these APIs.

- **XML document type APIs**

  In a previous product version, some XML document type APIs were deprecated. In the current product version, all remaining XML document type APIs have been deprecated.

  Also deprecated are any samples or tools that use these APIs, such as the REST Plugin, and any other public APIs related to XML document type APIs.

- **nMessageSigner APIs**

All nMessageSigner APIs have been deprecated in Java, C# and C++.

The affected APIs are:

```
nCertificateSigner
nCertificateValidator
nMessageValidator
nPassphraseValidator
nSignedEventListener
nMessageSigner
nPassphraseSigner
```

Also deprecated are any tools or samples which use these APIs, and any other public API related to these APIs.

- **Window-related APIs for synchronous consumers**

  Due to the introduction of the new *prefetch* processing for synchronous consumers (see the *New features* section above), various APIs associated with server-side *window* processing for synchronous consumers have been deprecated.

  For a list of the deprecated APIs, see the section *Using the Prefetch API* in the *Concepts* guide.

- **Multiplexing sessions**

  The feature allowing multiplexing of sessions has been deprecated in Universal Messaging v10.7 and will be removed from the product in a future release.

  For related information see the section *Multiplexing Sessions* in the *Concepts* guide.

- **Server startup parameter** DISK_USAGE_SCAN_ENABLE

  The server startup parameter `DISK_USAGE_SCAN_ENABLE` has been deprecated in Universal Messaging v10.7 and will be removed from the product in a future release.

  For related information see the section *Server Parameters* in the *Concepts* guide.

- **Pre-compiled sample applications**

  Universal Messaging pre-compiled sample applications are deprecated as of version 10.7. Note that the sample source code will remain available and can be used for reference when coding, but the compiled binaries will be removed from the product distribution in the next release. Please use the Universal Messaging command line administration tools instead.

  For related information see the sections *Running the Sample Applications* and *Command Line Administration Tools* in the *Administration* guide.

- **Display of JMX metrics using the view NirvanaRealmServer**

  Tools such as JConsole that can display JMX metrics from Universal Messaging can show the view **NirvanaRealmServer**. This view is deprecated in Universal Messaging v10.7 and will be removed from the product in a future release.

Notice of deprecation of Solaris v.11.3 support

The original provider of the operating system Solaris (Oracle) strongly recommends to upgrade from version 11.3 to 11.4 as of now.

In order to provide for enough time for Software AG customers to react to the upgrade policy of Oracle for Solaris versions after v.11.3, production environments based on Software AG products of this October 2020 release will in general continue to work without the immediate need to upgrade.

However, Software AG customer support will only handle issues for Software AG products that can be reproduced in a Solaris v.11.4 environment.

All newer versions of Software AG products AFTER the October 2020 release will no longer work with Solaris v.11.3.

Software AG therefore strongly recommends to address necessary migration steps timely.

Removed Features in v10.7

The following Universal Messaging features have been removed in Universal Messaging 10.7:

- **Realm server configuration property `JMSEngineIndividualPurgeEnabled`**

  The realm server configuration property `JMSEngineIndividualPurgeEnabled`, which was introduced in v10.5 as a short-term measure to switch on/switch off the JMS engine's functionality for automatic purging of individual events on a channel, has been removed. Now, automatic purging of individual events in a channel is by default activated.

  For related information, see the entry *JMS Engine automatic purging of individual events* in the above section *New features in v10.7*.

- **Server system parameter `JMSEngineIndividualPurgeEnabled`**

  The server system parameter `globalIndexedIteratorWindowSize` has been removed. It was used as a temporary mechanism in early releases of v10.3 but was otherwise undocumented.

Documentation Changes

- **Migrating from IPv4 to IPv6**

  A section has been added in the *Administration Guide* to summarize points to look out for in the configuration of Universal Messaging if you are moving from an IPv4 infrastructure to an IPv6 infrastructure.

  For details, see the section *Migrating from IPv4 to IPv6* in the *Administration Guide*.

- **Availability of Docker images**

  Docker support for Universal Messaging now covers the following:

  - There are container images in Docker Hub for the Universal Messaging server and associated command line administration and monitoring tools.

- There are scripts in GitHub for building a Universal Messaging image to run in a Docker container.

For related information, see the section *Using Docker* in the *Installation Guide*.

- **Server startup parameter for log rolling depth**

  There is a server startup parameter `LogFileDepth` that specifies the maximum number of log files to keep on disk when using log rolling. The oldest log files will be deleted when new log files are created.

  This parameter was undocumented in previous releases.

  Information on the use of the server startup parameter is available in the section *Server Parameters* in the *Concepts* guide.

- **Command Central documentation in** Administration Guide

  The Command Central documentation for Universal Messaging in the *Administration Guide* has been restructured to follow the workflow of managing a product in Command Central. In addition, the topics about using the command-line interface to manage Universal Messaging have been reorganized based on functionality.

5 What's New in Universal Messaging 10.5

Universal Messaging 10.5 is the successor of Universal Messaging 10.4.

Universal Messaging 10.5 includes new features, enhancements, and changes as described in the following topics.

New features in v10.5

The following Universal Messaging features have been added in Universal Messaging 10.5:

- **Support for purging events on a queue using Command Central**

  It is now possible to purge a range of event IDs or a single event ID on a queue.

  This means that purging of events on queues is possible through Enterprise Manager, Command Central and the client API, in the same way as for purging events on channels.

- **Monitoring queue-level connections**

  It is now possible to monitor both asynchronous and synchronous subscription to a specific queue similar to the monitoring of connections to durable subscribers. This is available programmatically via `nQueueNode`, similar to `nDurableNode`. Both Enterprise Manager and the Command Central UI have been updated to show these statistics; they are almost identical to how connections for durable subscribers are shown.

  In Enterprise Manager they can be seen under a specific queue in the **Consumer Info** tab. In Command Central they are under in the **Administration** tab when you click on a specific queue and go to `Consumer Info` tab. Additionally there are few new queue properties present in the **Status** tab of the queue under **Consumer Status**.

- **Individual purging of events on shared/serial durables**

  The JMS Engine of Universal Messaging now has the ability to automatically purge individual events from channels whose consumers are exclusively using durable subscriptions of type *Shared* or *Serial*.

  The individual JMS Engine purge is deactivated by default. To activate it, set the new realm configuration property "Event Storage"-> "JMSEngineIndividualPurgeEnabled" to "true".

  Note:

We are planning to change this default setting from deactivated (boolean value "false") to activated (boolean value "true") in a software fix that is scheduled for release shortly after the initial release of v10.5.

Changed features in v10.5

The following Universal Messaging features already available in the previous product release have been changed in Universal Messaging 10.5:

- **Health Checker error and warning messages**

  Some messages which were previously reported as error messages during a health check of a live cluster have now been changed to warning messages. This is because small synchronization delays across the live cluster can occur that cause the health checker to detect discrepancies, although the cluster is working correctly. Such discrepancies are therefore now classified as warnings rather than errors.

- **Re-worked queue implementation**

  In v10.5 the queue implementation in Universal Messaging has been reworked. This was done in order reduce internal complexity regarding queue usage, and to increase their stability and maintainability.

- **Queue subscriber filtering**

  In previous product versions, queue subscriber filtering was always enabled, which led to a certain amount of processing overhead. In v10.5 this has been changed, so that the feature can be activated or deactivated by using the new realm property `QueueSubscriberFiltering` in the **Event Storage** configuration group.

  By default, this property is set to false, meaning that queue subscriber filtering is deactivated. If you set this property to true, this activates the feature whereby events published to a queue can be read by selected subscribers only.

  Subscriber name/host filtering is activated at the API level by using the `setSubscriberName()` method of `nConsumeEvent`.

  Even if you set a name/host on an `nConsumeEvent` (using `setSubscriberName`), this would not have any effect unless the new realm property `QueueSubscriberFiltering` is set to true.

Deprecated features in v10.5

The following Universal Messaging features are now deprecated in Universal Messaging 10.5. Features listed as deprecated are still available in the product, but will be removed in a future release.

- **API nQueueReaderContext method setMaintainPriority()**

  For single node usage, the priority assignment for queue asynchronous consumers has been deprecated. This concerns only native queue asynchronous consumers, which set the maintain priority to "true" in their nQueueReaderContext.

Additionally, cluster support for this method has been removed. The method still exists, but for cluster usage it has no effect.

- **Fanout archive**

  The fanout archive functionality is a legacy feature that has various limitations. It will be removed from the product in the next releases.

- **"Persistent" flag for durable objects**

  The "Persistent" flag for durable objects is now deprecated. When the flag is removed in a later product version, the persistence of the durable will be inherited by the persistence support of the durable's channel. If the parent channel supports persistent events, its durables will always be created as persistent, whereas if the parent channel does not support persistent events, its durables will always be created as non-persistent.

- **"Cluster-wide" flag for durable objects**

  When the flag is removed in a later product version, the cluster-wide property of the durable will not be defined explicitly. The durable will be created as cluster-wide if its channel is a cluster-wide channel, whereas the durable will be created as non-cluster-wide if its channel is non-cluster-wide.

Removed Features in v10.5

The following Universal Messaging features have been removed in Universal Messaging 10.5:

- **API for Microsoft Silverlight**

  The Universal Messaging client API for Microsoft Silverlight has been removed.

- **API for Windows Phone**

  The Universal Messaging client API for Windows Phone has been removed.

- **Removal of store types**

  Support for the channel/queue types Simple, Paged, Transient and Offheap has been removed.

  Note:

  If a Universal Messaging client from an earlier product version connects to the version 10.5 server, the client might wish to create a store (channel or queue) that has a store type of Simple, Paged, Transient or Offheap. In this case, these store types will be remapped to one of the existing ones as follows:

  - Simple -> Reliable
  - Transient -> Reliable with a TTL (time to live) of 1 millisecond
  - Off-heap -> Reliable
  - Paged -> Mixed

  Also, log message will be visible in the `nirvana.log` file.

Example: If a v10.3 client tries to create a Transient channel, the v10.5 server will instead create a Reliable channel, and will also set the TTL to 1 millisecond in order to provide functionality that is similar to what the client would expect. A log message similar to the following will be visible in the `nirvana.log` file:

```
UserCreate> Warning :
   Store <store_name> is of type TRANSIENT which is not supported anymore.
Instead RELIABLE type will be used with TTL set to 1 ms.
```

For further information on the migration of store types that have been removed, see the section "Migration Notes" on page 99 in these Release Notes.

- **Removal of durable types "Priority" and "Shared-Queued"**

  The durable types "Priority" and "Shared-Queued" have been removed.

  Clients connections from previous product versions that use these durable types will be automatically mapped to other types if they connect to a v10.5 server:

  - "Priority" will be mapped to "Serial".
  - "Shared-Queued" will be mapped to "Shared".

- **Removal of `nNamedObject`-based API from the `nChannel` API**

  The previously deprecated `nNamedObject`-based API has been removed from the `nChannel` public API. If you have client applications that use `nNamedObject`, you will need to change them to use the `nDurable` API instead.

  For further information on the migration of applications that use `nNamedObject`, see the section "Migration Notes" on page 99 in these Release Notes.

- **Client API for Java**

  The option `-autoconvert` has been removed from the Java client API. This option converted clustered transient channels and queues to mixed channels and queues when you imported a realm configuration from an XML file. The option has also been removed from the sample program `nimportrealmxml` that is delivered with the product.

- **Realm configuration properties**

  The following list shows the realm configuration properties that are no longer available. The names are given in the form *<category>: <property>*, where *<category>* is the category to which the property belongs, and *<property>* is the property name.

  - `Event Storage : QueueDeliveryPersistencePolicy`
  - `Fanout Values : SyncQueueDelay`
  - `Fanout Values : SyncQueuePublisher`

  The `syncQueuePublisher` and `syncQueueDelay` properties provided a throttling mechanism for queues and shared-queued durables.

When you set `syncQueuePublisher` to `true`, Universal Messaging started to slow down publishers for a queue or shared-queued durable when all consumers for this destination had received their full window of events and had not yet acknowledged or rolled back any received events.

You configured this throttling mechanism globally, that is, it had effect for all queues and shared-queued durables on the server.

With the removal of the two properties, Universal Messaging no longer provides this throttling mechanism. This means that if the publishing rate for a queue or shared durable is faster than the consumption rate, events start accumulating in the respective store on disk or in memory, depending on store type.

For this reason, using non-persistent events with queues and shared durables might deliver suboptimal performance if the publishing rate is higher than the consumption rate for a store. In such cases, events accumulate in heap memory until the server memory protection mechanism is activated if you have enabled it. The mechanism throttles publishers and enables consumers to process events, reducing the overall publishing throughout.

If you exported a realm configuration containing any of the removed properties to an XML file in a previous product version, when you import the XML file into a newer-version realm, the properties are ignored.

For more information about the server memory protection mechanism, see *Out-of-Memory Protection* in the *Concepts Guide*.

- **Transient queue events**

  The transient event flag of an `nConsumeEvent(setTransient())` for events published to a queue is no longer supported and will be ignored. As a consequence, transient events will always be stored on the queue, even if there are no active subscribers.

Documentation Changes

- **Enterprise Manager documentation**

  The Universal Messaging Enterprise Manager documentation in the *Administration Guide* has been reorganized to be more task-oriented.

Additional Information

You can run Universal Messaging in a Docker environment.

See the section *Using Docker* in the *Installation Guide* for further information.

6 What's New in Universal Messaging 10.4

Universal Messaging 10.4 is the successor of Universal Messaging 10.3.

Universal Messaging 10.4 includes new features, enhancements, and changes as described in the following topics.

New features in v10.4

The following Universal Messaging features have been added in Universal Messaging 10.4:

- **Support for channel and queue snoop in Command Central**

  You can snoop on events on a Universal Messaging channel or queue as well as purge events from a channel or queue, using the Command Central web user interface and command-line interface.

  For information, see the sections *Snooping on Channels*, *Snooping on Queues*, *Commands for Snooping on Channels*, and *Commands for Snooping on Queues* in the *Administration Guide*.

- **Support for publishing events on channels and queues in Command Central**

  You can publish events, including Protobuf events, on a Universal Messaging channel or queue, using the Command Central web user interface and command-line interface.

  For information, see the sections *Publishing Events* and *Event Publishing Commands* in the *Administration Guide*.

- **Support for .NET Standard 2.0**

  The APIs `Nirvana DotNet.dll`, `Nirvana nAdminAPI.dll` and `Nirvana.Reactive.dll` have been tested to be compatible with .NET Standard 2.0. For the verification, the ".NET Portability Analyzer" tool from Microsoft is used.

- **Support for deleting clusters using the Tools Runner application**

  The Tools Runner application (`runUMTool`) has been extended with a new command `DeleteCluster`. This command allows you to delete an existing cluster.

  For information, see the topic *Syntax: Cluster Tools* in the section *Command Line Administration Tools* of the *Administration Guide*.

Changed features in v10.4

The following Universal Messaging features already available in the previous product release have been changed in Universal Messaging 10.4:

- **Multiplex session authentication**

  There is a change in the way user names are handled for client authentication in a multiplex session.

  For details, refer to the topic *Multiplex session authentication* in the section *Multiplexing Sessions* in the *Concepts* guide.

- **Horizontal Scalability**

  Universal Messaging sessions using the "horizontal scalability" feature now support automatic (re-)connection to realm servers in the scalability set that are offline/unavailable when the session is created or initialized.

  Also, HS sessions support connecting to various connection interfaces, e.g. NHP/NHPS/NSP/NSPS, and you can use a combination of these in a single HS RNAME.

  For details, refer to the topic *Usage Notes for Horizontal Scalability* in the section *Horizontal Scalability* in the *Concepts* guide.

- **Updated System.Reactive Libraries for .Net Reactive samples**

  The .Net Reactive Extensions (Rx) sample programs have been updated to use System.Reactive libraries v3.1.1. In the previous product release, version 1.0 of these libraries was used.

  The sample programs are located under in folder `<InstallDir>\UniversalMessaging\docnet\examples`.

- **Using AMQP in clustered environments**

  Universal Messaging now supports AMQP 1.0 in clustered environments.

  At the moment, automatic failover of AMQP clients is not supported. If the AMQP connection fails it is the responsibility of the AMQP client application to reconnect to another node in the cluster.

  Note:
  For the initial release of V10.4, the new functionality is provided as a software fix that you need to install after the standard product installation procedure has completed. To install the fix, you can use either Command Central or the Software AG Update Manager. If you wish to use the Update Manager, general instructions are provided in the document *Using Software AG Update Manager* that you can find on the Software AG documentation web site.

Deprecated features in v10.4

The following Universal Messaging features are now deprecated in Universal Messaging 10.4. Features listed as deprecated are still available in the product, but will be removed in a future release.

- **The store type "transient"**

  The channel/queue type "transient" is deprecated in v10.4 and will be removed in a future version of the product.

- **Realm configuration properties**

  The following realm configuration property is deprecated:

  - `SyncPingSize`

  The realm configuration properties are described in the section *Realm Configuration* in the *Administration Guide*.

Removed Features in v10.4

If features have been removed in Universal Messaging 10.4, they are listed in this section.

There are currently no removed features in v10.4.

Documentation Changes

- **Operations Guide**

  The product documentation set has been extended to include a new *Operations Guide*. This guide gives guidance on how to perform day-to-day operation and maintenance tasks, and includes suggestions on how to deal with standard questions in the areas of troubleshooting and maintenance.

- **Dead Event Store**

  The description of how the dead event store is used for channel processing has been updated.

  For details, refer to the topic *Dead Event Store restrictions for durable subscriptions* in the section *Channel Attributes* in the *Concepts* guide.

- **Java Service Wrapper**

  Several components of Universal Messaging make use of the Java Service Wrapper, which is an application provided by developed by Tanuki Software, Ltd. Information about the Universal Messaging components that use the Java Service Wrapper, and how to configure it, have been added to the documentation.

  For details, refer to the section *Configuring the Java Service Wrapper* in the *Administration* guide.

- **Startup and Shutdown Procedures**

Documentation has been added to better describe the procedures involved in starting up and shutting down a Universal Messaging Server.

For details, see the sections *Starting the Realm Server* and *Stopping the Realm Server* in the *Installation Guide*.

- **Additional descriptions for Clustering**

  Some topics such as the following have been added in the Clustering section of the Administration Guide.

  - Roles and Responsibilities for Configuring an Active/Passive Cluster
  - Configuring a Universal Messaging Active/Passive Cluster on UNIX
  - Cluster Verification
  - Failover Mechanism in an Active/Passive Cluster
  - Virtual IP Address of an Active/Passive Cluster

- **Restrictions for Shared Memory connections**

  A note has been added to the documentation that Shared Memory (SHM) connections are not supported on Solaris with SPARC architecture.

  See the related note in the section *Shared Memory (SHM)* in the *Concepts guide.*

- **Rules for naming channels and queues**

  A section has been added to the documentation to describe the valid set of characters that can be used when naming a new channel or queue.

  For details, see the topic *Valid channel names* on the page *Creating Channels* in the Enterprise Manager section of the *Administration Guide*.

- **Removal of some client JAR files**

  In previous product versions, some client Jar files were removed by design from the product delivery, but this was not stated clearly. The following table indicates Jar files that were delivered with previous product versions, but which are not delivered as Jar files any more. Their classes have now either been folded into other Jar files, or have been removed from the product altogether.

Previously available JAR File	Description	Status
nJ2EE.jar	Provided Universal Messaging support for interacting with Application servers that support J2EE.	The Jar file is no longer available. The Universal Messaging product now contains a resource adapter for use with application servers. See the topic *Resource Adapter for JMS* in the Java section of the

Previously available JAR File	Description	Status
		Developer Guide for related information.
nAdminXMLAPI.jar	Provided Universal Messaging Configuration XML Import / Export functionality.	The Jar file is no longer available, but the contents have been folded into nAdminAPI.jar.
nP2P.jar	Provided Universal Messaging Peer-to-Peer functionality.	The P2P functionality was removed from the product in version 9.12. The Jar file is no longer available, and the contents have not been folded into any other Jar file.

For the list of currently delivered client JAR files, refer to the section *Client Jars* in the *Concepts* guide.

7 What's New in Universal Messaging 10.3

Universal Messaging 10.3 is the successor of Universal Messaging 10.2.

Universal Messaging 10.3 includes new features, enhancements, and changes as described in the following topics.

New features in v10.3

The following Universal Messaging features have been added in Universal Messaging 10.3:

- **Client System Properties for Secure Communication**

  The Universal Messaging client system properties for secure communication configure only the connections to Universal Messaging realms and have no impact on the connections established to other endpoints, unlike the standard Java Secure Socket Extension (JSSE) system properties. The following Universal Messaging client system properties have been added:

  - com.softwareag.um.client.ssl.certificate_alias
  - com.softwareag.um.client.ssl.enabled_ciphers
  - com.softwareag.um.client.ssl.keystore_password
  - com.softwareag.um.client.ssl.keystore_path
  - com.softwareag.um.client.ssl.ssl_protocol
  - com.softwareag.um.client.ssl.truststore_password
  - com.softwareag.um.client.ssl.truststore_path

  The properties are described in the section *Using the Universal Messaging Client System Properties for Secure Communication* in the *Concepts* guide.

- **Client configuration properties**

  The following client configuration parameters have been added:

  - com.softwareag.um.client.follow_the_master
  - com.softwareag.um.client.network_io_buffer_size
  - com.softwareag.um.client.session_disable_reconnect

- com.softwareag.um.client.write_handler

The parameters are described in the section *Client Parameters* in the *Concepts* guide.

- **Realm configuration properties for disk free space**

  The following realm server startup parameters are now documented. They allow the Universal Messaging server to take action if the amount of free disk space drops below a threshold level.

  - `DISK_USAGE_FREE_THRESHOLD`
  - `DISK_USAGE_SCAN_ENABLE`
  - `DISK_USAGE_SCAN_INTERVAL`

  These parameters were available in the previous product release, but were not documented.

  The parameters are described in the section *Server Parameters* in the *Concepts* guide.

- **Support for Configuring a Universal Messaging SHM Port in Command Central**

  For information, see the sections *Ports* and *SHM Ports* in the *Administration Guide*.

- **Support for Secure Communication Between Command Central and Universal Messaging**

  Command Central now connects automatically to a Universal Messaging server that listens only on an nhps or nsps interface. Command Central uses this interface to establish the connection, as well as the truststore and keystore configured in the interface.

  In addition, you can provide custom truststore and keystore files by configuring JSSE system properties or Universal Messaging client system properties.

  For information, see the section *Securing Communication Between Command Central and Universal Messaging* in the *Administration Guide*.

Changed features in v10.3

The following Universal Messaging features already available in the previous product release have been changed:

- **C++ libraries**

  The following changes relating to the C++ libraries that Universal Messaging provides have taken place:

  - The C++ libraries for Universal Messaging on Windows are now compiled with Visual Studio 2015.
  - The C++ libraries for Universal Messaging on Linux are now compiled with gcc compiler version 4.8.5 20150623 (Red Hat 4.8.5-4).
  - The C++ libraries for Universal Messaging utilize OpenSSL libraries which have now been upgraded to 1.1.0h (the previous version was 1.0.2l).

- The C++ libraries for Universal Messaging utilize POCO libraries which have now been upgraded to 1.9.0 (previous version was 1.6.3).

Should you experience any problems with the new version of the C++ client libraries for Universal Messaging, first consider recompiling your client applications against the new version of the libraries/compilers.

- **Cleanup of shared durable stores at server startup**

  Occasionally, a shared durable store of type "Shared-Queued" can continue to exist after the shared durable subscription for which the store was created has been deleted. To ensure that such "orphaned" stores are removed, the Universal Messaging realm server checks for orphaned stores at each restart and deletes them. This can cause a short delay during the startup procedure of the realm server.

- **Configuration of the Python API**

  The procedure for configuring the Python API for Universal Messaging has been modified.

- **Editing JNDI Settings for connection factories**

  The method for editing JNDI settings for a Universal Messaging realm has changed. Now, the **JNDI** tab for a selected realm in the Enterprise Manager allows you to view and edit existing JNDI settings of connection factories. You can also add your own optional JNDI key, value and data type settings.

  The previous method for editing JNDI settings required you to do a *channel snoop* on the `/naming/defaultContext` channel and to use an "edit and republish" mechanism in the snoop panel. You can still use channel snoop to *view* the JNDI settings, but the "edit and republish" mechanism for the JNDI settings has been disabled (you can still enter new JNDI settings in the panel, but they will be ignored).

  Note that the "edit and republish" mechanism of channel snoop still works as in previous product versions for all channel events **except** for events on the `/naming/defaultContext` channel that represent JNDI settings.

  For related information on the new feature, see the section *Integration with JNDI* in the *Administration Guide*.

- **Format of timestamp field in log file entries**

  The format of the timestamp in log file entries has been changed. Previously, the time of day was shown as hours, minutes, seconds, in the format `hh:mm:ss`. Now, the time is shown as `hh:mm:ss.ttt`, where `ttt` represents thousandths of a second.

  See the section *Universal Messaging Enterprise Manager : Logs Panel* in the *Administration Guide* for examples.

- **Google Protocol Buffers**

  The Google Protocol Buffer library that Universal Messaging 10.3 uses has been updated from version 2.5.0 to version 3.6.0. The 3.6.0 version of Google protocol buffers supports both the *proto2* language syntax and the new *proto3* language syntax. The use of the new version brings

Universal Messaging into line with other Software AG products that use Google Protocol Buffers.

There is one known restriction: Universal Messaging v10.3 does not support server-side filtering of events based on the "Map" container type that is available with the new Google protocol buffers.

- **JVM behavior on Out of Memory Exception**

  In previous product releases, the Java client library triggered a JVM exit when an out of memory exception (OOME) occurred. In some updated versions of previous product releases, the client configuration parameter StopJVMonOOM was introduced to allow this behavior to be configurable.

  Now, in v10.3 the JVM never exits when an out of memory exception (OOME) occurs. Instead, the client library just logs the error, and the current session used by the client is automatically closed.

  Also, in v10.3 the client parameter StopJVMonOOM has been removed again, since the V10.3 behavior is not configurable - it is equivalent to a setting of " StopJVMonOOM=false" in v10.2, meaning that the JVM will not exit if an OOME occurs.

- **Limitations with Horizontal Scalability**

  The current implementation of Horizontal Scalability (HS) has some limitations when dealing with Universal Messaging realm servers that went offline during HS operation and are now online again. Similar restrictions apply if a realm server was not available at the start of the HS operation but has now come online.

  For details of these limitations, see the section *Usage Notes for Horizontal Scalability* in the *Concepts* guide.

- **Temporary limitation on window size of indexed durable subscriptions**

  For the initial release of v10.3, there is a limitation regarding the number of events per window that can be returned to clients that use durable subscriptions. This limitation is expected to be removed in later updates of v10.3. The limitation is as follows:

  By default, the iterator window size of indexed durable subscriptions is currently set to 1, even if you have specified a value of more than 1 for the window size.

  This applies to the following types of durable subscription:

  - Serial
  - Shared
  - Shared-Queued

  The lock can be overridden by setting the JVM argument `globalIndexedIteratorWindowSize` to "true". The default value of this parameter is "false".

  For related information, see the section *Using Durable Subscriptions with Multiple Clients* in the *Concepts* guide.

Deprecated features in v10.3

The following Universal Messaging features are now deprecated in Universal Messaging 10.3. Features listed as deprecated are still available in the product, but will be removed in a future release.

- **Client API**

  In the client API, `nChannelAttributes.getFullName()` has been deprecated. Users of this method should use `nChannelAttributes.getName()` instead.

- **Client configuration properties**

  The following client configuration properties are deprecated:

  - `CAKEYSTORE`
  - `CAKEYSTOREPASSWD`
  - `CKEYSTORE`
  - `CKEYSTOREPASSWD`

  The parameters are described in the section *Client Parameters* in the *Concepts* guide.

- **Creating non-clustered resources on clustered realm servers**

  The ability to create and use non-clustered resources on realm servers that are part of a cluster is deprecated.

  For example, channels of type "transient" are not intended to be used in a clustered environment, so it will no longer be possible to create a channel of type "transient" on a realm server that is part of a cluster.

  In the current product release it is possible to create such a non-clustered resource on a clustered realm server, but the behavior is unpredictable.

- **Priority and Shared-Queued Durable Subscriptions**

  The durable subscription types *Priority* and *Shared-Queued* are deprecated.

  For future applications we suggest you use the durable type *Shared* instead of Shared-Queued, and the durable type *Serial* instead of Priority.

  An overview of the durable subscription types is available in the section *Types of Durable Subscription* in the *Concepts* guide.

- **Realm configuration properties**

  The following realm configuration properties are deprecated:

  - `PriorityReadSpinLockMaxConnections`
  - `PriorityReadSpinLockTime`

- `PriorityReadType`

The realm configuration properties are described in the section *Realm Configuration* in the *Administration Guide*.

- **Storage properties of channels and queues**

  The following storage properties of channels and queues are deprecated:

  - `Enable Read Buffering`
  - `Read Buffer Size`

  The storage properties are described in the section *Storage Properties of Channels and Queues* in the *Concepts* guide.

Removed Features in v10.3

The following Universal Messaging features have been removed in Universal Messaging 10.3:

- The `EXIT_ON_FREE_SPACE_ERROR` server parameter has been removed in Universal Messaging version 10.1 and higher.
- **Client configuration parameters**

  The following client configuration parameters have been removed:

  - `StopJVMonOOM`

    See the item *JVM behavior on Out of Memory Exception* in these Release Notes for related information.

  The parameters are described in the section *Client Parameters* in the *Concepts* guide.

- **Legacy (i.e. global) protocol buffers**

  Support for configuring "legacy" (i.e. global) Google protocol buffers has been removed. In previous product versions, protobuf descriptors could be kept in a global directory, rather than setting them on each channel. Now, support for using a global directory has been removed, and only channel-level protocol buffers are supported.

  For this reason, several realm configuration parameters relating to protocol buffers have also been removed.

  The option `Protobuf Config: FilterProtobufEvents` has been removed, since filtering of protobuf events is the most natural behavior and now always occurs (previously, filtering was optional). If you do not want filtering, you should not configure protobuf descriptors for the channel.

  See the summary of the removed *Protobuf Config* properties in the list below.

- **Proxy forwarding**

  Support for proxy forwarding has been removed. The corresponding realm configuration parameters in the category "Proxy Forward Config" have been removed.

- **Realm configuration properties**

  The following list shows the realm configuration properties that are no longer available. The names are given in the form *<category>: <property>*, where *<category>* is the category to which the property belongs, and *<property>* is the property name. Property names are unique within a category, but the same property name can be present in different categories. Some properties are annotated with (*1), (*2) etc. There are references to notes that are described at the end of the list.

  - `Cluster Config : BufferSize` (*3)
  - `Cluster Config: EnableSites` (*2)
  - `Cluster Config: FilterEventsDuringRecovery` (*2)
  - `Cluster Config: SecureHandshake` (*2)
  - `Cluster Config: SeparateLog`

    Note: the spelling "SeperateLog" was used originally.
  - `Cluster Config: TransactionSync` (*2)
  - `Connection Config : BufferQueueSize` (*3)
  - `Connection Config : EnablePriorityMessaging` (*3)
  - `Connection Config : HandshakeTimeout` (*3)
  - `Connection Config : whPeakTrailDelay` (*3)
  - `Connection Config: MaxBufferSizeClientSideCheck` (*2)
  - `Connection Config: NIOSelectArray`

    After the removal of this property, UM will continue to behave as if this property were set to "false".
  - `Data Stream Config : FanoutTaskQueueSize` (*3)
  - `Data Stream Config : MaxSessionIdSize` (*3)
  - `Data Stream Config : ParallelFanoutThreshold` (*3)
  - `Data Stream Config: FanoutTraversalType`

    After the removal of this property, UM will continue to behave as if this property were set to "in-order traversal".
  - `Event Storage : AutoMaintainOnFileLimit` (*3)
  - `Event Storage : EnableStoreReadBuffering` (*3)
  - `Event Storage: AutoMaintainSystemStores` (*2)
  - `Event Storage: EnableBufferingKey` (*2)

- `Fanout Values : MaximumDelayInWrite` (*3)
- `Fanout Values : ParallelThreshold` (*3)
- `Fanout Values : ParallelUseGlobalPool` (*3)
- `Fanout Values : RoundRobinDelivery` (*3)
- `Fanout Values: ParallelBatchSize` (*3)
- `Global Values : ServerStateFlush` (*3)
- `Global Values : StatusUpdateTime` (*3)
- `Global Values: NanoDelayBase`

  After the removal of this property, UM will continue to behave as if the default value of this property (100000) were still in effect.
- `Global Values: ServerTime` (*2)
- `Inter-Realm Comms Config: WriteDelayOnFail` (*2)
- `Inter-Realm Comms Config: ZoneDefaultCanRecv` (*2)
- `Inter-Realm Comms Config: ZoneDefaultCanSend` (*2)
- `JVM Management: AutoThreadDumpOnExit` (*2)
- `JVM Management: ExitOnMemoryError` (*2)
- `Logging Config: DisplayPackageName`
- `Logging Config: customDebugTag` (*1)
- `Logging Config: customErrorTag` (*1)
- `Logging Config: customFatalTag` (*1)
- `Logging Config: customInfoTag` (*1)
- `Logging Config: customLogTag` (*1)
- `Logging Config: customTraceTag` (*1)
- `Logging Config: customWarnTag` (*1)
- `Protobuf Config : MaximumProtobufBuilders`
- `Protobuf Config : MinimumProtobufBuilders`
- `Protobuf Config : ProtobufDescriptorsInputDir`
- `Protobuf Config : ProtobufDescriptorsOutputDir`
- `Protobuf Config : UpdateDescriptorsInterval`
- `Protobuf Config : UseChannelLevelProtobufCache`

- `Protobuf Config: FilterProtobufEvents` (*2)
- `Protocol AMQP Config : AllowUserTransformation` (*3)
- `Protocol MQTT Config : DisconnectOnSecurityException` (*3)
- `Protocol MQTT Config : Timeout` (*3)
- `Proxy Forward Config: BufferSize`
- `Proxy Forward Config: FlushTimeout`

Notes:

- *1: The functionality of custom tags has been removed, so the properties have been removed accordingly.
- *2: After the removal of this property, UM will continue to behave as if the default value of this property ("true") were still in effect.
- *3: This property was made available in previous releases in preparation for possible future use, but its value was ignored, therefore the behavior of UM is not affected by the removal of this property.

If you exported a realm configuration containing any of these removed properties to an XML file in a previous product version, the XML file can still be imported into a newer realm, but in this case the properties will be ignored, regardless of the value that they were set to.

- **SOAP Plugin**

  The SOAP server plugin has been removed.

  This plugin was previously described in the section *Plugins* of the Enterprise Manager in the *Administration Guide*.

8 What's New In Universal Messaging 10.2

Universal Messaging 10.2 is the successor of Universal Messaging 10.1.

Universal Messaging 10.2 includes new features, enhancements, and changes as described in the following topics.

Updates for the HealthChecker tool

New checks

The set of available checks provided by the HealthChecker tool has been extended.

New checks are:

- `ServerProtectionConsistencyCheck`
- `XMLServerProtectionConsistencyCheck`
- `DurableSubscriberLargeStoreCheck`

Extended syntax allowing custom values

The syntax for running the HealthChecker has been extended to allow custom values to be specified by using additional parameters. For example, the `DurableSubscriberLargeStoreCheck` check examines the number of remaining events to be consumed in a shared durable, and if the number is greater than a certain threshold a warning will be displayed. The default value for the threshold is 1000, but the additional parameter `-threshold` allows you to specify a different value for the threshold.

For more information, refer to the section *Running a Configuration Health Check* in the *Administration Guide*.

New Diagnostic Tool - Realm Information Collector

The Realm Information Collector is a new command-line diagnostic tool that gathers files and live data from one or more Universal Messaging realm servers. The tool makes it easier for you to collect information that Software AG support may require to diagnose issues with Universal Messaging, but the information collected may also be useful for internal support within your organization.

For more information, refer to the section *The "Realm Information Collector" Diagnostic Tool* in the *Administration Guide*.

Heap Dump following a JVM "Out of Memory Error"

Universal Messaging now automatically generates a heap dump file when an `OutOfMemoryError` occurs in the Java Virtual Machine (JVM).

For more information, refer to the section *The Dump file for Out-of-Memory Errors (OOME)* in the *Installation Guide*.

JVM behavior on Out of Memory Exception

In the initial version of the current product release, the Java client library triggered a JVM exit when an out of memory exception (OOME) occurred. In some updated versions of the product release, the client configuration parameter StopJVMonOOM has been introduced to allow this behavior to be configurable.

For details, see the description of StopJVMonOOM in the section *Client Parameters* in the *Concepts* guide.

Documentation of Command Line Tools

For several product versions, Universal Messaging has provided a set of command line administration tools that display online help when called on the command line. The product documentation set has now been extended to include this online help.

For more information, refer to the section *Command Line Administration Tools* in the *Administrator Guide*.

Horizontal Scalability

In the 10.2 release we have introduced the *horizontal scalability* (HS) feature, which allows clients to seamlessly publish and consume events from multiple independent realms and clusters using a single connection. This feature is available for both the Universal Messaging native API for Java and the Universal Messaging API for JMS. It is enabled by using the newly defined horizontal scalability URL syntax.

HS is generally a client-side function, and a server that is used in a HS set can also be used (at the same time) as a normal standalone realm or cluster. However, we don't recommend using the same channels/queues in both HS and non-HS modes.

The *horizontal scalability* feature is a replacement of the JMS layer round-robin publishing feature which is now deprecated. Also, the term *round-robin connection factory* in the product documentation has been replaced by the term *horizontal scalability connection factory*.

For more information, refer to the section *Horizontal Scalability* in the *Concepts* guide.

Check for version compatibility of Universal Messaging server and administration API client

In previous product versions, there was no explicit version compatibility check between the Universal Messaging server and administration API clients connecting to the server, so for example an administration API client v9.12 could connect to a v10.1 server. In general, this resulted in an undefined behavior, but usually in a stream corruption, since cross-version administration API compatibility is not supported.

Starting from v10.2, the Universal Messaging server includes a version check that rejects connection requests from administration API clients whose protocol version differs from that of the server. The Universal Messaging server adds an entry in the log file `nirvana.log`, clearly describing the protocol mismatch.

If the Universal Messaging server detects a version mismatch, the server replies to the client with a security exception that is intended to be compatible across Universal Messaging protocol versions. If, however, the security exception is not compatible with the protocol version of the administration API client, the client re-throws the received security exception.

Changed behavior following file system I/O exceptions

In previous product versions, the Universal Messaging server would keep running even after experiencing serious file system I/O exceptions, such as being unable to create a file for channel persistence due to using an invalid filename. In such situations, the application would not be made aware that these exceptions had been received, and so would continue processing without the expected persistence guarantees being in place.

This issue has been resolved by modifying the server to shut down when a non-recoverable I/O exception is received. This fail-fast behaviour may impact server availability but will avoid the worse situation where an application incorrectly assumes that it has persistence guarantees that are not in fact provided.

Client API for C++

New functionality

In the client API for C++, you can now get the header of an `nConsumeEvent` and iterate through its values. This functionality is already available in the client API for Java.

This can be achieved by getting the header from the event and then getting an iterator from the header.

Important:
You have to dispose of the iterator after you have finished using it.

Example:

```
void someMethod(nConsumeEvent* event) {
 nHeader* header = event->getHeader();
 nHeaderIterator* headerIterator = header->getIterator();
```

```
 while (headerIterator->hasNext()) {
  // get the key and value
  std::string headerKey = headerIterator->getKey(); // nrvdead.orig.eid, nrvpub.time
 etc.
  fObject* headerValue = headerIterator->getValue();

  // do something with them
  .....

  // move the next value in the header
  headerIterator->next();
 }

 delete headerIterator;
}
```

Support for JAAS login with a client X.509 Certificate chain

Universal Messaging now supports authenticating users through the Software AG Security Infrastructure component (SIN) with a client X.509 certificate chain. This allows users to access a Universal Messaging server that requires authentication, over an SSL/TLS enabled interface with a client certificate.

For more information, refer to the section *Server JAAS Authentication with Software AG Security Infrastructure* in the *Concepts* guide.

Added Support for creating Windows service and UNIX daemon during Installation

The Software AG Installer now contains an option that allows you to install the default instance of the Universal Messaging server as a Windows service or a UNIX daemon.

In the Installer you can also specify that the Windows service or UNIX daemon will start up automatically when the host machine starts up.

The executables used to start the Universal Messaging server in console mode have been replaced by `nserver.bat` (Windows) and `nserver` (UNIX). The new scripts start the Universal Messaging server either as a command line console or as a service (Windows) or daemon (UNIX), depending on in the mode in which the server was installed.

Note:
As described in the section "Replacement of configuration file nserver.conf by nserverdaemon.conf" on page 77, the configuration file `nserver.conf` used in previous product releases has been removed. The new scripts do not use the configuration file `nserver.conf` - they use `nserverdaemon.conf` instead.

Support for registering a Windows service using Command Central CLI

Command Central command-line interface can now be used to register a Windows service when creating a Universal Messaging server instance.

For more information, refer to the section *Universal Messaging Realm Server Instance Management* in the *Administrator Guide*.

Support for configuring JVM options using Command Central

Command Central web user interface and command-line interface can now be used to configure JVM options.

For more information, refer to the section *JVM Options* and *Universal Messaging Commands* in the *Administrator Guide*.

Enterprise Manager Enhancements

- The Enterprise Manager now offers an option to convert clustered transient channels and queues to mixed channels and queues when you import a realm configuration from an XML file.

  For more information, refer to the section *Importing a Realm Configuration from an XML File* in the *Administrator Guide*.

- The Named Objects tab for a channel has been renamed to Durables tab, and the Get button has been removed. Now, the Enterprise Manager updates the durables table automatically when a durable is added or removed, or the attributes of a durable are changed.

  For more information, refer to the section *Viewing and Managing Durables for a Channel* in the *Administrator Guide*.

Client API for Java

The Java client API provides a new option, `-autoconvert`, to convert clustered transient channels and queues to mixed channels and queues when you import a realm configuration from an XML file.

For more information, refer to the section *Java Client: Import a realm's configuration information* in the *Developer Guide*.

Replacement of configuration file nserver.conf by nserverdaemon.conf

In previous product releases, the Universal Messaging server could be configured using either of the configuration files `nserver.conf` or `nserverdaemon.conf`. The file `nserver.conf` file was used if the server was started from the command line, and `nserverdaemon.conf` was used if the server was started as a Window service or UNIX daemon.

In the new release, the configuration file `nserver.conf` has been removed, and internal scripts that previously used `nserver.conf` use `nserverdaemon.conf` instead.

Use of terms "named object" and "durable subscription"

In previous product releases, we used the terms *named object* and *durable subscription* as synonyms. As of v10.2, we are phasing out the term *named object* and focusing on using the term *durable subscription* or its abbreviated form *durable*.

Documentation Corrections

MQTT Quality of Service (QoS) Level Support

In previous product releases, the documentation stated that Universal Messaging supports MQTT QoS levels 0, 1 and 2.

However, support is only provided for QoS levels 0 and 1. Connections requesting QoS level 2 will be downgraded to QoS level 1 at connection time, as allowed by the MQTT specification.

For more information, refer to the section *MQTT: An Overview* in the *Concepts* guide.

Deprecated features in v10.2

The following Universal Messaging features are now deprecated in Universal Messaging 10.2. Features listed as deprecated are still available in the product, but will be removed in a future release.

- **XML document type**

  Currently it is possible to publish and consume messages that are stored in XML format. This functionality is now deprecated.

  Note that this deprecation notice only applies if the entire message is in XML format. It does not apply if only the event payload is in XML format.

- **Simple, Paged and Offheap store types**

  The following store types (i.e. channel types and queues types) are now deprecated:

  - Simple
  - Paged
  - Offheap

- **SSL certificate generator**

  The Certificate Generator utility, that can be used to generate a self signed server certificate, a self signed client certificate and a trust store, is now deprecated.

- **Client API for Python**

  The Client API for Python is now deprecated.

- **Server plugins**

  The following server plugins are now deprecated:

  - Graphics
  - XML
  - SOAP

- Proxy passthrough
- Servlet

- **Round Robin publication via JMS**

  All APIs and classes under the package `com.pcbsys.nirvana.nJMS.roundRobin` are deprecated and the functionality has been placed into the package `com.pcbsys.nirvana.nJMS` using the Horizontal Scalability feature of Universal Messaging introduced in v10.2.

  Any existing JMS Round Robin scenarios will need to recreate or modify their JNDI environments to not use these classes but to use their equivalent in the package `com.pcbsys.nirvana.nJMS`.

  The term *round-robin connection factory* in the product documentation has been replaced by the term *horizontal scalability connection factory*.

- **Enterprise Manager Scheduler**

  The Scheduler feature of the Enterprise Manager is now deprecated.

Removed features in v10.2

The following features that were available in previous product releases have been removed in Universal Messaging 10.2.

- **TradeSpace demo**

  The TradeSpace demo that was available in previous versions has been removed from the product.

9 What's New In Universal Messaging 10.1

Universal Messaging 10.1 is the successor of Universal Messaging 10.0.

Universal Messaging 10.1 includes new features, enhancements, and changes as described in the following topics.

Updated and New Functionality in the client API for C#

- **Shared durable subscriptions now supported in client API for C#**

  The client API for C# now supports *shared durable subscriptions*. The client API for C# functionality used in previous product releases for creating named objects and named objects with priority has now been deprecated.

  The new API provides different public methods for interaction. For operations like creating, retrieving, deleting or unbinding a durable, the nDurableManager must be used. Every channel has a durable manager associated with it.

  For information how to use the new functionality, refer to the topic *Using Durable Objects* in the C# section of the *Developer Guide*. See also the section *Durable Subscriptions* in the *Concepts* guide.

- **Serial durable subscriptions now supported in client API for C#**

  The client API for C# now supports *serial durable subscriptions*.

  With a serial durable subscription, multiple subscribers can hold a subscription to the same named object and all the subscribers will process events in a serial manner.

  For information about the new functionality, refer to the topic *Using Durable Objects* in the C# section of the *Developer Guide*. See also the section *Durable Subscriptions* in the *Concepts* guide.

- **New public interface for committing and rolling back events**

  A new public interface has been added for committing and rolling back events. The methods are defined for the nDurable instance and the usage of the old API, e.g. calling ack() or rollback() on the received event's nConsumeEvent object, is not recommended since it does not fully support individual acknowledging and rolling back. To be able to apply these operations on a single event and not only on consecutive event IDs is a significant importance for the shared types.

- **Basic Authentication now supported in C# client API**

It is now possible to use SASL (e.g. plain text) authentication for the client API for C#. Previously, this functionality was only available using the client API for Java.

For details, refer to the section *Basic Authentication* in the C# section of the *Developer Guide*.

Serial durable subscriptions now supported in client API for Java and as an extension to the API for JMS

The client API for Java now supports serial durable subscriptions. The same functionality has been added as an extension to the Universal Messaging API for JMS.

With a serial durable subscription, multiple subscribers can hold a subscription to the same named object, and all the subscribers will process events in a serial manner.

For information about the new functionality, refer to the topic *Durable channel consumers and named objects* in the Java section of the *Developer Guide*. Refer also to the section *Durable Subscriptions* in the *Concepts* guide.

Periodic logging of the realm server status

The logging feature has been extended to allow the status of the realm server to be reported in the server log at regular intervals. The status includes metrics such as the amount of memory currently in use for active events, the amount of disk space in use, CPU load, number of active connections, total bytes sent and received.

For details, see the section *Periodic Logging of Server Status* in the *Concepts* guide.

New and Enhanced Command Central Capabilities

Command Central can now be used to configure Universal Messaging zones and security groups.

Command Central now supports enhanced port configuration options.

You can now configure round-robin JMS Connection Factories using Command Central, allowing messages to be distributed evenly across multiple Universal Messaging realms or clusters. For more information, see the Universal Messaging Administration Guide.

These capabilities can be accessed using the Command Central web user interface, command-line interface, REST API, and composite templates.

Changed handling of missing operator in filter expression

In v10.0 it was permissible to omit a logical operator between two selector clauses in a filter expression. In such cases, the omitted operator was treated implicitly as an "AND" operator. For example, instead of the correct form:

```
( Item1 = 'ABC' ) AND ( Item2 in ( 'Invoicing', 'Pending' ) )
```

it was possible to state:

```
( Item1 = 'ABC' ) ( Item2 in ( 'Invoicing', 'Pending' ) )
```

In 10.1, omitting the logical operator will be flagged as an error. Therefore, when you upgrade to 10.1, ensure that you modify your filter expressions accordingly.

Pause Publishing

The new *pause publishing* feature allows all client publishing to the server to be paused, across channels, queues and data groups. This pause will not affect the administration API, inter-cluster communication or joins.

The feature is activated by setting the server configuration property `PauseServerPublishing` to true. Then, clients trying to publish or commit will receive `nPublishPausedException`. Exception handlers from prior to the current product version will handle this as `nSessionPausedException`, which the new `nPublishPausedException` extends.

Information about this feature is available in the section *Pause Publishing* in the *Concepts* guide.

If client applications using APIs from previous product versions are used to publish events to a v10.1 server, the following changes will be observed:

Old API (prior to 10.1)	Behavior publishing to a v10.1 server
Native API for Java	If a client API from a previous product version is used to connect to a 10.1 server, Java native clients will receive `nBaseClientException` instead of `nPublishPausedException`.
API for JMS	The behavior here is the same as for 10.1, with the difference that the root JMSException will be `nBaseClientException` instead of `nPublishPausedException`.
API for C++/C#	The APIs for C++ and C# will throw `nUnexpectedResponseException` when publishing is paused.

Updates for MQTT

Configuration Properties for MQTT

A new set of realm configuration properties for MQTT has been added.

For the list of MQTT configuration properties see the topic *Realm Configuration* in the Enterprise Manager section of the *Administration Guide*.

Support for clustered channels

MQTT support for clustered channels has been added.

General improvements

Non-functional aspects such as performance, availability and scalability have been improved.

Server JAAS Authentication with Software AG Security infrastructure component

Universal Messaging can now use the Software AG Security Infrastructure component (SIN) to provide server JAAS authentication capabilities. The SIN component provides a variety of options for using different authentication back-ends and implementing flexible authentication scenarios.

The SIN module is now the default LDAP module for use with Universal Messaging.

SIN modules are delivered with the Universal Messaging distribution and are available on the Universal Messaging server classpath.

For details, see the section *Security* in the *Concepts* guide.

Configuring Universal Messaging for use with IBM WebSphere Application Server

The Universal Messaging installation contains a product-specific generic resource adapter for JMS. Universal Messaging can be configured to work with IBM WebSphere Application Server via this adapter.

For details, see the section *Resource Adapter for JMS* in the *Developer Guide*

Microsoft Edge is a supported Web browser for Javascript Communication Drivers

The list of web browsers that are supported for the Javascript communication drivers has been extended to include Microsoft Edge for several of the drivers.

For details, see the section *Communication Drivers* in the *Developer Guide*.

Corrected DLL names in the product documentation

Some sections of the documentation referred to the DLL files "Nirvana DotNet.dll" and "Nirvana Silverlight.dll" by the wrong names "Universal Messaging DotNet.dll" and "Universal Messaging Silverlight.dll" respectively. These have been corrected.

These DLLs are mentioned in the C# and VBA sections of the Developer Guide.

New Java system property "Nirvana.sasl.client.enablePrehash"

The new Java system property `Nirvana.sasl.client.enablePrehash` specifies whether to prehash the supplied password when using the CRAM-MD5 or Digest-MD5 mechanisms.

For details, see the section *Client-side Authentication* in the Java section of the *Developer Guide*.

Removed and deprecated features in 10.1

The following Universal Messaging features are now deprecated or have been removed in Universal Messaging 10.1:

- **Deprecated Client API Support for Microsoft Silverlight**

The Universal Messaging client API for Microsoft Silverlight is deprecated and will be removed from the product distribution in the next official release.

- **Removed realm configuration parameter "QueuedSharedDurableFilterBound"**

  The realm configuration parameter `QueuedSharedDurableFilterBound`, which was introduced in Universal Messaging 10.0, has been removed. This boolean parameter specified whether or not to bind a shared durable to the event filter of its latest user.

  The behavior in v10.1 without this parameter is the same as the 10.0 behavior when this parameter was set to "true", namely: once the shared durable is bound to a filter, any subsequent subscriptions that mismatch the filter will replace the filter and an asynchronous exception will be thrown to existing subscribers.

- **Removed realm configuration parameter "OutputBlockSize"**

  The realm configuration parameter `OutputBlockSize` has been removed.

- **Deprecated methods in C# Client API**

  The methods ack() and rollback() on the nConsumeEvent object of the received event is deprecated.

  See the section "Updated and New Functionality in the client API for C#" on page 81 above for information on the new API features for committing and rolling back events.

- **Deprecated Client API Support for Windows Phone**

  The Universal Messaging client API for Windows Phone is deprecated and will be removed from the product distribution in the next official release.

- **Deprecated Universal Messaging Directory Backend Authentication**

  The Internal User Repository and LDAP directory backend providers that were in use in Universal Messaging v10.0 for authentication have been deprecated. The directory backend providers are being replaced by the new functionality described in the section "Server JAAS Authentication with Software AG Security infrastructure component " on page 84 above.

- **Deprecated Administration APIs for C# and C++**

  The Administration APIs for C# and C++ are deprecated and will be removed from the product distribution in the next official release.

10 What's New In Universal Messaging 10.0

Universal Messaging 10.0 is the successor of Universal Messaging 9.12.

Universal Messaging 10.0 includes new features, enhancements, and changes as described in the following topics.

Durable subscribers can be browsed in Command Central

Command Central can now be used to browse individual messages in durable subscribers in a Universal Messaging server instance, including messages waiting to be received by Integration Server triggers. You can view the size of individual messages and some properties of each message. You can also view the payload of string messages. All or individual messages can be deleted from the durable subscriber (for some durable subscriber types).

Enhanced Command Central support for Universal Messaging

Command Central can now be used to configure realm ACLs for Universal Messaging, allowing you to control which users can perform what actions at the realm level. You can also use Command Central to configure Java system properties for Universal Messaging. For Universal Messaging clusters, Command Central now displays whether each running instance is a master or slave in the cluster.

These capabilities can be accessed using the Command Central web user interface, command-line interface, REST API, and composite templates.

Utility for migrating webMethods Broker gateways configurations to Universal Messaging

For webMethods Broker users who have configured territories and gateways and have large numbers of document types (topics) configured in the gateway connections, migrating these to equivalent remote joins in Universal Messaging previously involved a lot of manual effort. Now, you can use a new utility that can read the gateway configuration from webMethods Broker and automatically establish the same remote joins in Universal Messaging.

Servers can be instructed not to restrict incoming messages from specific clients in low memory situations

A client session can request to bypass the existing server-side low-memory throttling. This is intended to allow administrators or administrative services to continue to be able to connect in order to resolve the low-memory situation.

JNDI assets can now be stored in existing (non-Universal Messaging) JNDI providers

Users who wish to use a JNDI provider for binding JNDI assets such as Connection Factories and Destinations can now store their Universal Messaging JNDI assets in a JNDI provider of their choice. This capability is only available through the respective API.

Cluster side-by-side upgrades to new machines are easier to manage

You can now use a migration utility to migrate cluster configurations to machines that are on different hosts from the original cluster.

Shared durables re-architected to improve efficiency and robustness

The implementation of shared durables has been re-written to improve efficiency. The new implementation performs tracking of durable objects in-place on the channel, rather than relying on additional internal stores. This approach saves memory and reduces the complexity of the solution, improving robustness. A new API has been developed to manage durable objects through the Universal Messaging native API. The Universal Messaging JMS library has also been updated to utilize the new durable implementation.

Configuration using Enterprise Manager is simplified

Enterprise Manager hides a number of configuration settings behind an **Advanced** button. These hidden settings are ones that are rarely recommended to be modified.

Additional tools are now available showing how to use the API with datagroups and illustrating how to publish and subscribe

Tools are provided that allow management of datagroups and illustrate how to use publish and subscribe.

Robustness improvements for installations using the default out-of-the-box configuration

The default configuration for new installations has been reviewed and tested to ensure Universal Messaging is robust.

Named object ID is a concatenation of client ID and durable subscriber ID

Named objects IDs are now a concatenation of the client ID and the durable subscriber ID. Previously Universal Messaging just used the durable subscriber ID provided by the client.

Removed and deprecated features in 10.0

The following Universal Messaging features are now deprecated or have been removed in Universal Messaging 10.0:

- The **SharedDurableFilterBound** option, found in the **Durable Config** group in the administrative API and Enterprise Manager/Command Central, has been migrated to be a System Property (**-DQueuedSharedDurableFilterBound**). This new system property is deprecated and will be removed in a future release.

11 What's New In Universal Messaging 9.12

Universal Messaging 9.12 is the successor of Universal Messaging 9.10.

Universal Messaging 9.12 includes new features, enhancements, and changes as described in the following topics.

Durable subscribers monitoring and API improvements

Enterprise Manager has improved monitoring of durable subscribers and is now able to display more details for the durable subscribers, including details about the connections currently be used, the EIDs and the number of events outstanding in the queues. This information can now also be accessed via the administration API.

The client API for durable subscribers and named objects has been redesigned to improve performance, robustness and usability. The new durable subscribers API, available from the client API, maps to the existing durable subscribers functionality.

New and enhanced Command Central capabilities

You can now use Command Central to add, edit, delete, administer, and monitor channels (topics) and queues. In addition, you can monitor durable subscribers to easily detect and identify issues such as stalled triggers or processing backlogs.

These capabilities can be accessed using the Command Central web user interface, Command Central command-line interface, and REST API.

The Command Central web user interface now provides the following capabilities:

- Create and delete Universal Messaging server instances.
- Search for JNDI entries, channels, and queues.
- View, create, edit, and delete access control lists (ACLs).
- View create, edit, and delete joins for a channel or a queue.
- Delete durable subscribers.

Improved handling of low memory situations

New methods for protecting against out-of-memory situations have been introduced to increase the robustness of Universal Messaging under heavy load.

The "event usage" metric provides information on memory currently in use by on-heap events. This includes current on-heap event memory usage, the maximum memory currently available to the JVM, and the percentage of on-heap memory currently in use. These statistics enable monitoring of the current memory usage, allowing action to be taken accordingly.

Universal Messaging servers can now throttle producing connections while processing their events. At predefined, configurable thresholds of on-heap event memory usage, producer connections are throttled, enabling consumers to reduce the number of events on the connections while they are throttled. Connections are more strictly throttled as memory usage rises, helping to prevent out-of-memory situations.

Round-robin message publishing using connection factories for JMS

Horizontal scalability improvements have been introduced with the API for JMS, now allowing the configuration of round-robin connection factories. These factories allow clients to publish messages in a round-robin fashion, so that one message or transaction gets published to the first realm node or cluster, the next message to the next realm node or cluster, and so on.

These connection factories for JMS have the following limitations:

- Event consumption is not supported through these factories so, for example, message listeners cannot be registered and consumers cannot be created via the sessions created from these connection factories.
- The sessions created through these connection factories do not support distributed (XA) transactions.

For more information consult the JMS-related section of the product documentation.

Logging capabilities enhancements

Support has been added for utilizing the third party logging frameworks Logback and Log4J 2. Both of these testing frameworks offer improved throughput performance when compared to the existing Flogger engine.

Log file entries are now categorized by the component which generated the entry, e.g. Cluster Communications, Joins, etc.

Improved futureproofing for Universal Messaging clients

The client API is now officially supported for use with newer versions of the Universal Messaging server, which means that the 9.12 client API will be supported for use with future versions of UM server.

The client API has been extended in this release with features that were previously only in the administration API (which is not supported for use with newer versions of the server).

ACLs can now be set at store-creation time via the client APIs for Java and C++. This allows basic ACL control for stores without needing to use the administration API.

Setting ACLs at store-creation time has been a typical use for the administration API. These changes allow the client API to be used in a greater number of use cases. The client API is more lightweight than the administration API, and therefore switching to the client API can increase overall system performance and consumed bandwidth.

HTTP drivers support checking of "Origin" headers

The HTTP/WebSocket drivers have been updated to process the Origin header field according to standards proposed in RFC-6454, RFC-6455 and the W3C Cross Origin Resource Sharing document (https://www.w3.org/TR/cors).

Any nhp/nhps interfaces may have to have their CORS Allowed origins (located under the nhp/nhps Interface -> Javascript tab in Enterprise Manager) altered if an HTTP request has the Origin header field set. Previous versions of Universal Messaging had default values of "localhost, 127.0.0.1" assigned to the CORS Allowed origins field, and would process only host names as values to this field. The current W3C standards now expect any origin to be of the form "<scheme>://<host>:<port>"; for example, "localhost" is an incorrect value, while "http://localhost:11000" is a properly formatted value. The exception is a single value of "*", which indicates that all hosts are permitted access; note that the processing of this value has not changed with the update, and is now the default value in the CORS Allowed origins field whenever an nhp/nhps interface is created.

In addition, support for matching "http://example.com" and "http://example.com:80" as origins (as documented in RFC-6454) is currently not supported. You will need to explicitly white list hosts with *:80 as potential origins (if needed) in addition to others.

Warning of the effects of editing stores

When stores are edited, Universal Messaging deletes and recreates the store and this can disrupt active subscriptions. Enterprise Manager has been updated to display a warning message that the store will be recreated, before a channel edit is performed.

nInterfaceTool extended to allow editing of additional interface settings (e.g. autostart)

The nInterfaceTool has been extended to provide additional capabilities. For example, it now allows you to set interfaces to automatically start when the server starts.

Docker 1.10 support

The Docker kit for Universal Messaging now supports Docker 1.10.

Python and iOS client libraries included in installation

The client libraries for Python and iOS are now included as part of the installation.

UM-tools "runner" is installed as part of the "Template applications" module

The um-tools runner is now part of the installer "Template applications" module rather than the "Server" module. This allows you to install the um-tools runner without installing the server.

Updated version of OpenSSL

Universal Messaging now uses OpenSSL 1.0.2 instead of the previous version 1.0.1.

Platform changes in 9.12

Universal Messaging 9.12 runs on the platforms listed in the *Supported Platforms* document that is included in the Universal Messaging 9.12 documentation set. You can find this documentation set in the Software AG Documentation web site.

Check the above mentioned *Supported Platforms* document for details about the newer platform versions supported by Universal Messaging 9.12.

Removed and deprecated features in 9.12

The following Universal Messaging features are now deprecated or have been removed in Universal Messaging 9.12:

- Support for Flex has now been removed from Universal Messaging. Flex is a rich internet application (RIA) language which allows the development of complex web applications that run inside a browser. Flex is no longer supported by Adobe Systems Incorporated and has been transferred to the Apache Software Foundation.
- Support for P2P has now been removed from Universal Messaging. The API for P2P is a legacy API within Universal Messaging. It allows stream-based communication between two clients mediated by the Broker. This messaging system is no longer useful in the light of more recent and modern paradigms such as DataGroups.

12 What's New In Universal Messaging 9.10

Universal Messaging 9.10 is the successor of Universal Messaging 9.9.

Universal Messaging 9.10 includes new features, enhancements, and changes as described in the following topics.

Docker support

A Universal Messaging Packaging Kit for Docker is now part of the standard Universal Messaging installation on Linux. The kit was previously available only on TECHcommunity.

The Universal Messaging Packaging Kit can be found here:

```
<SAG Install Folder>/UniversalMessaging/server/<UM Server Name>/bin/docker
```

The kit includes the following Docker tools:

- Dockerfile for creating a Docker image from the Universal Messaging installation on Linux.
- Samples showing how to start the Universal Messaging server from the Docker image and run the sample applications of Universal Messaging within the image.
- The TradeSpace demo application shows how to use 'docker-compose' to set up and run the TradeSpace demo of Universal Messaging.

Note:
This delivery method can change in later releases of Universal Messaging.

Optimized persistent store

There is a new form of persistent store, enabled by setting the spindle size to a value greater than zero. This store format uses multiple files for each channel or queue and removes the overhead of Universal Messaging which has to "perform maintenance" on them. This new mechanism also allows stores to grow without any restriction on the JVM heap. The standard persistent store mechanism keeps an in-memory index which grows each time a message is added.

Improved Handling of Maximum Message Size

In previous releases, Universal Messaging restricted the maximum message size that the server will read in by the MaxBufferSize configuration property. Exceeding this value would cause the

connection to be disconnected, but the message had already been sent over the network and the reason for the disconnection was not obvious to the client. This check by the server remains but Universal Messaging now has a client side check so that the message is rejected before it is sent over the network and the user can handle the exception.

Clients can “follow the master node”

Clients can now be enabled to "follow the master" in a cluster. The client will initially connect to a server in the cluster but if that server is not the master, the client will be redirected to the master node. Connecting to the master node can provide better performance in some use cases.

Transactions over AMQP

The AMQP specification defines a number of transactional operations. Universal Messaging now supports AMQP transacted operations so that client applications can perform transactional work when communicating with the realm server over AMQP. For example, if an application communicates to the realm server using a JMS AMQP client library (e.g. Apache Qpid JMS client) it can take advantage of the local transaction functionalities defined in the JMS specification.

Universal Messaging does not currently support the AMQP Transactional Acquisition operation. However, this sets no limitations on using JMS transactions over AMQP.

Configuration profiles in Installer

It is now possible to select different configuration profiles using the Software AG Installer. We have provided two configurations: one tuned for typical webMethods use cases and one tuned for standalone use cases.

JNDI asset configuration in Command Central

Command Central now supports the creation and maintenance of JNDI connection factories, queues, and topics for Universal Messaging. These assets can be managed through the Command Central web user interface, command line interface, REST API, and within templates.

Enterprise Manager handles realm name conflicts more gracefully

You can no longer connect to a realm with the same name as the realm to which you are already connected. Enterprise Manager now shows the host and port in the realm tree to make it clear which realm is being referenced.

Additional sample applications

New sample applications are included that can be used to:

- Create clusters
- Create server interfaces
- Manage security groups

Additional documentation

Various items have been added to the product documentation:

- The documentation has been updated with a guide helping you configure your JVM in order to support FIPS 140-2.
- The documentation now includes best practice guidelines helping you to configure Universal Messaging to have separate interfaces for client and cluster communication. The guide describes how to restrict regular client communication but allow administrators to connect when a cluster is forming.
- Documentation is also provided describing the API's setMessageType() method that allows a client to convert a JMS message back to its original type.

Removal of Standalone Installer

Universal Messaging can no longer be installed using its own standalone Installer. It can now only be installed using the standard Software AG Installer or Command Central.

13 Migration Notes

If you are migrating from an earlier version of the product to a new version, there may be some migration issues that you should be aware of. Such issues are described in the document *Installation and Upgrade Information for Software AG Products* that is available on the Software AG documentation web site.

Additionally, we list some of the issues below.

Migrating queues that use the JMS engine from v9.x to v10.x

During the migration of Universal Messaging from version 9.x to version 10.x, any queues that were configured to use the JMS engine with version 9.x will be migrated successfully. However, from version 10.1 onwards, Universal Messaging doesn't support queues using the JMS engine. Therefore, on the first server restart in version 10.x, any queues that previously used the JMS engine will be permanently reconfigured to use the Universal Messaging default engine instead.

Migration of removed store types on v10.5 server startup

In Universal Messaging version 10.5, the store types Simple, Transient, Off-heap and Paged have been removed. If Universal Messaging v10.5 is installed to the same disk location as a previous version of the product, and there are any of these store types from the older version at this disk location, these store types will be mapped to other store types when the v10.5 server is started for the first time.

The mapping from removed store types to available store types is as follows:

- Simple -> Reliable
- Transient -> Reliable with a TTL (time to live) of 1 millisecond
- Off-heap -> Reliable
- Paged -> Mixed

For Paged stores, events might be persisted on disk and the Universal Messaging server is responsible for copying them to the migrated store. Please have in mind that this can lead to difference in the event IDs, but apart from that everything else will be available.

Also, note that there will be log entries for new stores being created and old ones being deleted.

Example

Assume we have a Paged channel called `PagedChannel`. This will be migrated to a channel of type Mixed via several steps :

1. A new temporary channel `PagedChannel_new` will be created as a copy of `PagedChannel`.
2. Events present on `PagedChannel` will be copied to `PagedChannel_new`.
3. The type of `PagedChannel` will be switched to MIXED.
4. Events will be copied from `PagedChannel_new` to `PagedChannel`, and `PagedChannel_new` will be removed.

The log entries will be like this:

```
[...] [main] Startup: Opening store PagedChannel_new
...
[...] [main] Startup: Opening store PagedChannel
...
[...] [main] Deleting store PagedChannel_new
[...] [main] Mixed-Store> junit2\data\StoreMigrationStandaloneTest100000_26010\
             data\PagedChannel_new6716f746102986.mem : File has been deleted
[...] [main] Deleting store complete PagedChannel_new
```

For related information, see the section "What's New in Universal Messaging 10.5" on page 51 that is specific to version 10.5.

Migration of client applications using the nNamedObject API

In v10.5, the previously deprecated `nNamedObject`-based API has been removed from the `nChannel` API. `nNamedObject` was used for coding applications that used durable types "Priority" and "Shared-Queued", and these durable types have also been removed in v10.5.

You will need to manually adapt any of your applications that used `nNamedObject` to use the `nDurable` API instead. The following table shows the mapping between `nNamedObject` and `nDurable`.

Type	NamedObject API	nDurable API

Named Object	channels.createNamedObject(name, startEid, persistent, isClusterWide, false);	long startEid = startEID;
		boolean isClusterWide = clusterWide;
		nDurable named = null;
		nDurableAttributes.nDurableType type =
		nDurableAttributes.nDurableType.Named;
		if (enablePriority) {
		  type = nDurableType.Serial;
		}
		nDurableAttributes attr =
		nDurableAttributes.create(type, name);
		attr.setPersistent(persistent);
		attr.setClustered(isClusterWide);
		attr.setStartEID(startEid);
		try {
		named = channels.getDurableManager().add(attr);
		}
		catch (nNameAlreadyBoundException exception) {
		return channels.getDurableManager().get(name);
		}

Named Priority Durable	channels.createNamedObject(name, startEid, persistent, isClusterWide, true);	long startEid = startEID;
		boolean isClusterWide = clusterWide;
		nDurable named = null;
		nDurableAttributes.nDurableType type =
		nDurableAttributes.nDurableType.Serial;
		nDurableAttributes attr =
		nDurableAttributes.create(type, name);
		attr.setPersistent(persistent);
		attr.setClustered(isClusterWide);
		attr.setStartEID(startEid);
		try {
		named = channels.getDurableManager().add(attr);
		}
		catch (nNameAlreadyBoundException exception) {
		return channels.getDurableManager().get(name);
		}

Shared Named Object	channels.createSharedNamedObject(name, persistent, isClusterWide, startEid);	// create the shared durable using new implementation.
		nDurableAttributes attr =
		
		nDurableAttributes.create(nDurableAttributes.nDurableType.Shared,
		 name);
		attr.setPersistent(persistent);
		attr.setClustered(isClusterWide);
		attr.setStartEID(startEid);
		// Need to check first in case shared durable already exists
		try {
		  return channels.getDurableManager().get(attr.getName());
		} catch (nNameDoesNotExistException ex) {
		  return channels.getDurableManager().add(attr);
		}

Deletion	channel.delNamedObject('myNamedObject')	channel.getDurableManager().delete("myDurable");

Deleting Local Stores when Migrating a Cluster to 10.11

When you migrate an active/active cluster that contains local stores to version 10.11, the local stores are deleted when you start the Universal Messaging 10.11 server instance for the first time.

In previous releases, it is possible to create non-cluster-wide stores in an active/active cluster. The stores are created with a cluster-wide flag set to `false`. These local stores exist only on the cluster node on which you created them and are not replicated to the other cluster nodes. Support for local stores has been removed in version 10.11 and they are deleted during migration.

You can obtain a list of all deleted local stores as part of the following message in the nirvana.log server log file:

```
Unsupported store configuration - non clustered stores in a clustered environment.
The following stores will be removed:
<list of the deleted local stores>
```

If the local stores contain important data, you can republish events to the 10.11 server instances using the `RepublishEventsFromOfflineFile` tool.

For more information about using the tool, see the section "Command Line Administration Tools > The RepublishEventsFromOfflineFile Tool" in the *Administration Guide*.

Client Session Behavior During Cluster Failure

In version 10.11 and higher, Universal Messaging client, non-admin sessions in an active/active cluster are always terminated if the server to which the client application connects is not in a good cluster state, that is, either master or slave state.

Prior to Universal Messaging 10.11, you could use the `DisconnectIfClusterFails` flag to configure the behavior of client sessions when the server is not in a good cluster state. The `DisconnectIfClusterFails` flag has been removed in Universal Messaging 10.11.

Previously, if you set the flag to `false` and the server was not in a good cluster state, the session would remain active even though the server node was not part of the cluster at that moment. In 10.11, with the removal of support for local stores in an active/active cluster, the flag has become redundant.

Consider the following information:

- Now client sessions always behave as if the `DisconnectIfClusterFails` flag is set to `true`.
- Clients that had the `DisconnectIfClusterFails` flag set to `false` will experience more disconnected sessions if the cluster state of the server changes.
- If the `DisconnectIfClusterFails` flag is set to `false` in lower-version clients that connect to Universal Messaging servers version 10.11 or higher, the flag is ignored and the session behaves as if the flag is set to `true`.

This behavior is valid only for client session that use the UM Client API. Admin sessions, which use the UM Admin API, can connect to the server regardless of its cluster state.