Universal Messaging Operations Guide

Version 10.7

October 2020

This document applies to Software AG Universal Messaging 10.7 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-OG-107-20230713

Table of Contents

About this Documentation

Online Information and Support

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at http://documentation.softwareag.com.

Software AG Empower Product Support Website

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

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

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

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

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

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

Software AG TECHcommunity

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

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

Data Protection

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

1 Introduction

Purpose

The purpose of this document is to provide information to support:

- **daily operations**, by clearly describing all the tasks which need to be carried out to keep the Universal Messaging system running and thus prevent unplanned outages
- **special operations**, e.g. outage planning and all special tasks from shutdown to startup
- **monitoring**, by describing different KPIs to be tracked to measure the performance and general health of the Universal Messaging system
- **troubleshooting**, by ensuring that all required information is available. Especially in the case of problems that might arise, this document contains the information required by operations staff to recover from the situations and also collect diagnostic data required by the Software AG support team to find the root cause and recommend changes and/or prevent it from recurring.
- **all other active and proactive tasks**, e.g. purging, archiving and other cleanup tasks performed to keep the UM system running within the expected performance limits, and prevent unplanned outages

Audience

This guide is intended for the IT team delivering a solution involving the Universal Messaging product, and the IT Operations team which manages the daily operations of a solution using the Universal Messaging product.

2 Installation

Each Universal Messaging realm server has a directory called `data` where the realm's state is stored. By default the data directory is located at `<InstallDir>\UniversalMessaging\server\<InstanceName>\data`. It is recommended to keep the data directory outside the <InstallDir> installation directory tree and then point the realm server to this directory.

In the remainder of this guide, the following notation is used:

<InstallDir> means the disk root location of your Universal Messaging installation

<InstanceName> means the name of the realm server

<DataDir> or "**data directory**" means `<InstallDir>\UniversalMessaging\server\<InstanceName>\data`, or the data directory path with which the realm was created/started.

Note:
For Universal Messaging v10.1, ensure that the "Template Applications" option is selected when installing the product. Selecting this option ensures that the Health Checker, as well as other tools and samples, get installed, which are required for effectively monitoring and administering the Universal Messaging server. For Universal Messaging v10.3 onwards, the "Template Applications" are always installed, so this recommendation does not apply.

3 Startup and Shutdown / Planned Outages and Changes

This section covers how to start up and shut down the Universal Messaging server. It also looks at aspects to be taken into account while planning an outage for server maintenance.

Organization of a Startup

For details on how to start the Universal Messaging server, and different modes (Automatic and Manual) in which it can be started, refer to the section *Starting the Realm Server* of the Universal Messaging *Installation Guide*.

Note:

To check that the realm server has started properly, examine the realm server log file `nirvana.log` in the <DataDir> location for text similar to the following:

```
[Tue Nov 27 13:02:27.538 GMT 2018] [ServerStarterThread] Startup:
Realm Server Startup sequence completed
```

Organization of a Shutdown

For details on how to stop the Universal Messaging server, refer to the section *Stopping the Realm Server* of the Universal Messaging *Installation Guide*.

Note:

To check that the realm server has shut down properly, examine the realm server log file `nirvana.log` in the <DataDir> location for text similar to the following:

```
[Tue Nov 27 13:00:30.312 GMT 2018] [ServerStarterThread] Shutdown:
Realm Server completed shutdown sequence, process exiting normally
--------- Log File Closed ---------
```

Maintenance Window (Planned Outage)

The following points have to be considered before shutting down the Universal Messaging realm or cluster for a scheduled maintenance.

- Clients (other than Integration Server clients) which are publishing the messages are either stopped or suspended.
- In the case of Integration Server clients, ensure that the "Enable CSQ" option under the Producer Settings of the Universal Messaging connection alias settings is enabled. This will ensure that when the Integration Server publishes documents using this connection alias, it writes the documents to the client-side queue if the Universal Messaging server is unavailable. Integration Server will publish the messages written to the client-side queue when the Universal Messaging server is back online. Client-side queuing can also be enabled by passing "true" to the `useCSQ` flag while invoking the `pub.jms:send` and `pub.jms:sendAndWait` services. The "Maximum CSQ Size (messages)" property of the Universal Messaging connection alias settings should be set to an appropriate value depending on the publishing rate and planned outage window size.

  For related information, refer to the description of *Creating a Universal Messaging Connection Alias* in the chapter *Configuring Integration Server for webMethods Messaging* in the *Integration Server Administrator's Guide*. Refer also to the descriptions of `pub.jms:send` and `pub.jms:sendAndWait` in the *JMS Folder* section of the *Built-In Services Reference Guide*, which is included in the Integration Server documentation set.
- All the channels and/or queues are drained or emptied before stopping or suspending the clients consuming the messages.

Note:
The above points are not recommendations but rather points to be considered while designing the system as well as planning for a scheduled maintenance.

Application of Fixes

Consider the following points when applying a fix to a realm or a cluster of realms:

- Before applying the fix, perform the following steps:
  1. Back up the realm data and configurations, following the backup procedure.
  2. Shut down the realm, following the maintenance window procedure.
- Apply the fix to the realm using Software AG Update Manager.
- After applying the fix, perform the following steps:
  - Start the realm.
  - Perform a configuration health check by running the HealthChecker tool. The checks to perform are:
    - DurableSubscriberLargeStoreCheck
    - FixLevelCheck
    - JNDIStatusCheck
    - JoinMismatchCheck
    - ResourcesSafetyLimitsCheck

- ServerProtectionConsistencyCheck

Rolling Update of Servers in a Cluster

A rolling update of the Universal Messaging servers in an active/active cluster enables you to apply a fix to a server in the cluster while the other servers remain online. You shut down only the server to which you are applying the fix and not the whole cluster.

Consider the following information before starting the procedure:

- You must apply the fix to all servers in the cluster.
- Software AG recommends that you first apply the fix to the secondary nodes and then to the primary node.

To install a fix to the servers in an active/active cluster, perform the following steps on each node:

1. Shut down the node.

   The node disconnects from the cluster.

2. Apply the fix to the node.
3. Start the node.

   The node reconnects to the cluster and is operational again.

Clients connected to the cluster should be able to tolerate a brief loss of connection:

- When the node to which a client is connected is shut down, the client loses connection to the node and the cluster. Then the client must re-establish the connection.
- When the primary node is shut down, the whole cluster briefly goes offline until a new primary node is elected.

4 Backup and Recovery

This section covers the details about the data which needs to be backed up, and how to restore and recover the state of the system in the case of a failure.

Conceptual Explanation and Description

The state of a realm server is stored in the realm's <DataDir> directory. The realm server's state includes:

- Configurations
- JNDI Naming entries
- Stored channel/queue data

Backup

A realm server can be restored from a copy of the realm's <DataDir> directory.

Backup and recovery procedures can vary depending on the type of Universal Messaging deployment and also the recovery requirements. Universal Messaging can be deployed in Active/Passive or Active/Active cluster modes.

Note:

In Active/Passive deployment, the active and inactive nodes share the same storage (they point to the same <DataDir> directory), but only one realm server is online at any moment in time. In this case, typically a shared network drive would be used as the <DataDir> directory, so that all the cluster nodes (active as well as inactive) can access the data.

In Active/Active deployment, cluster nodes replicate resources among themselves, and maintain the state of the resources across all cluster nodes. Operations such as configuration changes, transactions and store management are orchestrated by the master node. The master node broadcasts the requests to the other cluster nodes to ensure that all the servers are in sync. If a cluster node disconnects and reconnects, all the states and data are recovered from the master node. In this case, the state replication or duplication is achieved by the Universal Messaging server by replicating its state on all the nodes of the cluster. Hence all the cluster nodes can be configured with the <DataDir> directory pointing to their respective local disks (recommended by Software AG).

Backing up the realm data

- Shut down the Universal Messaging realm(s). This is required, as taking a backup of the data directory <DataDir> while the realm is running could result in backed up data being inconsistent/corrupt, thus rendering it useless.
- Take the backup of the following folders:
  - <DataDir>\RealmSpecific, containing realm specific configuration files
  - <DataDir>\naming, containing JNDI entries
  - all the files and folders under <DataDir>, *if your recovery requirements demand the channel/queue data to be recovered as well*
  - <InstallDir>\UniversalMessaging\server\ <InstanceName>\bin, containing scripts and configuration files (only if they were customized)
- In the case of an Active/Active deployment, take the backup of all the above directories for each realm server in the cluster. All the nodes in the cluster need to be backed up in the same time frame. The primary advantage of backing up the contents of the <DataDir> directory of all the realms of a cluster is to retain configuration information which is not synchronized between nodes of the cluster. The following information is not synchronized across the cluster:
  - Interfaces
  - SSL Certificate/Truststore
  - non-clustered resources (queues/channels)
  - Configured Plugins
  - Cluster private keys
  - Server JVM options
- Make a baseline the realm configuration by exporting it into a XML representation using either Enterprise Manager or the command line tool.

Command Details

```
Usage:
runUMTool ExportRealmXML -rname=<rname> -filename=<filename> [optional_args]
Examples:
  To export all the information:
  runUMTool ExportRealmXML -rname=nsp://localhost:9000
    -filename=test.xml -exportall=true (This will export all the information)
  To export only some information:
  runUMTool ExportRealmXML -rname=nsp://localhost:9000
    -filename=test.xml -realms=true -realmconfiguration=true
    -channels=true -queues=true
Required arguments:
  rname : Connection URL to the realm you want to export.
  filename : File name where the information will be exported.
Optional Parameters:
  exportall : Export all information for the chosen realm.
  username : Your Universal Messaging server username.
```

```
password : Your Universal Messaging server password.
```

This can also be achieved by via Enterprise Manager by right clicking on the realm and selecting 'Export Realm to XML' and checking the box for 'Export All'.

Snapshot the VM

If virtual machines (VM) are in use, taking a periodic snapshots of the VMs will come in useful when the realms have to be rolled back to a known good state. It is preferable to do this when UM is in an offline state as it ensures that all of UM's persisted state is consistent (assuming it's part of the snapshot).

If you are using VM snapshots with UM clustering it is imperative to deploy the infrastructure such that UM cluster communication is isolated from all storage, live migration and other infrastructure management traffic. It is highly recommended that it is on physically separate hardware.

Schedule

Taking regular backups is good practice. In addition to the regular backups, it is advisable to take a backup before making any changes to the realms or before attempting the recovery operations.

Restore/Recover From Realm data backup

- Shut down the Universal Messaging realm(s) which are being restored
- Delete everything from the realm's <DataDir> directory
- Restore the folders under the <DataDir> directory from the backup
  - <DataDir>\RealmSpecific, containing realm specific configuration files
  - <DataDir>\naming, containing JNDI entries
  - all the other folders under <DataDir>, **if your recovery requirements demand the channel/queue data to be recovered as well**
- Restore the contents of <InstallDir>\UniversalMessaging\server\<InstanceName>\bin from the backup, only if they were customized
- Start the realm(s)

Note:

For an Active/Active cluster, it may not be required to recover all the nodes of the cluster. Restore only the node which is perceived to be having a problem. If the problem appears to be with the cluster as a whole, then it is necessary that all the nodes of the cluster are shut down (the order of the nodes does not matter) before each node is restored and restarted.

All realms should be restored to the backup taken at the same time.

Important:

Always collect the diagnostic data before a realm is recovered. For an Active/Active cluster, the following steps have to be performed for each realm in the cluster:

1. Take a backup of the whole <DataDir> directory
2. Run the 'Realm Information Collector' tool

5 Monitoring

Universal Messaging provides a set of command line tools that allow you to perform many of the common actions available. Some of these tools can be used to monitor different aspects of a realm which indicate its health. Below is the description of these tools and their usage.

Environment State Check

Periodically (every 1 minute) run the Health Checker tool for checking the environment status.

Command Details

```
Usage:
  runUMTool HealthChecker -rname=<rname> -check=EnvironmentStateCheck
Examples:
  runUMTool HealthChecker -rname=nsp://localhost:9000
     -check=EnvironmentStateCheck
Required arguments:
  rname : Name of a realm to check.
```

Refer to the section *Running a Configuration Health Check* section of the Universal Messaging *Administration Guide* for more information on the HealthChecker tool.

Output of each run can be parsed to raise alerts for any line starting with WARN or ERROR. The tool checks what percentage of memory is taken by events from the whole heap memory. If the percentage is between 70 and 80, or between 80 and 90, or above 90, an appropriate warning will be displayed.

Sample output of the EnvironmentStateCheck command:

```
ENVIRONMENT STATE CHECK
Environment State [umserver]
INFO: [umserver] Connections: 2
INFO: [umserver] Queued threads: 0
INFO: [umserver] Vended threads: 62
INFO: [umserver] Total threads: 64
INFO: [umserver] Total memory (MB): 981
INFO: [umserver] Used memory (MB): 101
INFO: [umserver] Free memory (MB): 880 (89.69%)
INFO: [umserver] Total direct memory (MB): 1024
INFO: [umserver] Used direct memory (MB): 0
INFO: [umserver] Free direct memory (MB): 1024 (100%)
INFO: [umserver] Max heap memory (MB): 981
INFO: [umserver] Memory allocated for events (MB): 0
```

Store State Check

Periodically (every 1 minute) run the Health Checker tool for checking the store status.

The following checks have to be enabled in the HealthChecker:

- StoreMemoryCheck
- StoreMismatchCheck
- StoreWarningsCheck

Command Details

```
Usage:
runUMTool HealthChecker -rname=<rname>
  -check=StoreMemoryCheck, StoreMismatchCheck, StoreWarningsCheck
Examples:
runUMTool HealthChecker -rname=nsp://localhost:9000
  -check=StoreMemoryCheck, StoreMismatchCheck, StoreWarningsCheck
Required arguments:rname : Name of a realm to check.
```

Refer to the section *Running a Configuration Health Check* of the Universal Messaging *Administration Guide* for more information on the HealthChecker tool.

Output of each run can be parsed to raise alerts for any line starting with WARN or ERROR.

Cluster State

Check the cluster state by a given RNAME, which is part of a cluster.

Command Details

```
Usage:
runUMTool ClusterState -rname=<rname> optional_args]
Examples:
runUMTool ClusterState -rname=nsp://localhost:9000
Required arguments:
rname : Name of a realm, which is part of a cluster.
Optional Parameters:
username : Your Universal Messaging server username.
password : Your Universal Messaging server password.
```

As seen in the sample output below, the statuses of the cluster nodes can be parsed and appropriate alerts can be raised.

Sample output of ClusterState command

```
-------------------------------------
Cluster Name: Cluster1
-------------------------------------
Cluster Nodes:
Node name: umserver (Master)
Realm rnames: nhp://10.42.96.207:9000/
  nhp://fe80:0:0:0:2de9:64df:4bbf:d85c%14:9000/
Is node clustered: true
```

```
Node name: umserver2 (Slave)
Realm rnames: nhp://10.42.96.207:9000/
  nhp://fe80:0:0:0:2de9:64df:4bbf:d85c%14:9000/ nhp://10.42.96.207:9001/
  nhp://fe80:0:0:0:2de9:64df:4bbf:d85c%14:9001/
Is node clustered: true
--------------------------------------
Cluster Statuses
--------------------------------------
Server name: umserver
Server status: online
Cluster state: Master
Broadcast time: 0
Client Request size: 0
Comms Queue size: 0
Queue size: 0
Response time: 0
--------------------------------------
Server name: umserver2
Server status: online
Cluster state: Slave
Broadcast time: 0
Client Request size: 0
Comms Queue size: 0
Queue size: 0
Response time: 0
--------------------------------------
```

Alternative Java Sample

The sample code `nClusterWatch.java` found in `<InstallDir>\UniversalMessaging\java\examples\com\pcbsys\nirvana\nAdminAPI\apps\nClusterWatch.java`, demonstrates how the Java Admin API can be used to monitor the Cluster State.

Monitor Channels

Monitors the channels and queues in a realm and prints totals.

Command Details

```
Usage:
runUMTool MonitorChannels -rname=<rname> [optional_args]
Examples:
runUMTool MonitorChannels -rname=nsp://localhost:9000
  -channelname=channel0 -format=plaintext
runUMTool MonitorChannels -rname=nsp://localhost:9000
  -channelname=queue1 -format=plaintext
Required arguments:
rname : URL of the realm to monitor channels and queues for.
Optional Parameters:
channelname : Name of a specific channel or queue to monitor
format : Format to print output in (plaintext/xml/json)
username : Your Universal Messaging server username.
password : Your Universal Messaging server password.
```

As seen in the sample output below, channel and queue statuses of the cluster nodes can be parsed and appropriate alerts can be raised.

Sample output of the MonitorChannels command

```
Name : channel0
Total Events Published : 10000
Total Events Consumed : 0
Last Event ID : 10277
Current Connections : 0
Total Connections : 0
Used Space : 781K
Events : 10000
Memory Usage : 1M
% Free : 0%
Cache Hit : 0.0
```

Alternative Java Sample

Sample code using Java Admin API to monitor Channel and Queue depths

```
package com.pcbsys.nirvana.nAdminAPI.apps;

import com.pcbsys.nirvana.client.nSessionAttributes;
import com.pcbsys.nirvana.nAdminAPI.nContainer;
import com.pcbsys.nirvana.nAdminAPI.nLeafNode;
import com.pcbsys.nirvana.nAdminAPI.nNode;
import com.pcbsys.nirvana.nAdminAPI.nRealmNode;

import java.util.ArrayList;
import java.util.Enumeration;
import java.util.List;

/**
 * Scans the provided Realm for Channels and Queues and displays their
 * attributes (Current Depth, Total Published and Total Consumed)
 * every 10 seconds.
 *
 * Expects Realm Name (nsp://hostname:port) as runtime argument
 */
public class GetChannelsAndQueuesInfo {

    private nRealmNode realmNode;
    private List<nLeafNode> channels = new ArrayList<>();
    private List<nLeafNode> queues = new ArrayList<>();

    public GetChannelsAndQueuesInfo(String realmName)
    throws Exception {
        realmNode = new nRealmNode(new nSessionAttributes(realmName));

        scanRealmForChannelsAndQueues(realmNode.getNodes());
    }

    /**
     * Recursively scans the Realm namespace for Channels and Queues
     *
     * @param realmNamespaceNodes
     */
    private void scanRealmForChannelsAndQueues(
        final Enumeration realmNamespaceNodes) {
        while (realmNamespaceNodes.hasMoreElements()) {
            final nNode child = (nNode) realmNamespaceNodes.nextElement();
```

```
            if (child instanceof nLeafNode) {
                final nLeafNode leafNode = (nLeafNode) child;

                if (leafNode.isChannel()) {
                    channels.add(leafNode);
                } else if(leafNode.isQueue()) {
                    queues.add(leafNode);
                }
            }
            else if (child instanceof nContainer) {
                scanRealmForChannelsAndQueues(((nContainer) child).getNodes());
            }
        }
    }

    public nRealmNode getRealmNode() {
        return realmNode;
    }

    public List<nLeafNode> getChannels() {
        return channels;
    }

    public List<nLeafNode> getQueues() {
        return queues;
    }

    public static void main(String[] args) throws Exception {
        if(args.length == 0) {
            throw new Exception("Realm Name startup argument is missing.");
        }

        GetChannelsAndQueuesInfo getChannelsAndQueuesInfo =
            new GetChannelsAndQueuesInfo(args[0]);
        System.out.println();
        System.out.println("Connected to Realm : " +
           getChannelsAndQueuesInfo.getRealmNode().getRealm().getName());
        System.out.println();


        while(true) {
            StringBuilder displayString = new StringBuilder();

            displayString.append(
              "Channels (Name | Current Depth | Total Published | Total Consumed)
\n");
            displayString.append(
              "-------------------------------------------------------------------
\n");
            for (nLeafNode oneLeaf : getChannelsAndQueuesInfo.getChannels()) {
                printLeafNode(displayString, oneLeaf);
            }

            displayString.append(
              "\nQueues (Name | Current Depth | Total Published | Total Consumed)
\n");
            displayString.append(
             "------------------------------------------------------------------ \n");
            for (nLeafNode oneLeaf : getChannelsAndQueuesInfo.getQueues()) {
```

```
                printLeafNode(displayString, oneLeaf);
            }
            displayString.append(
              "================================================================");
            System.out.println();

            System.out.println(displayString);

            Thread.sleep(10000);
        }
    }

    private static void printLeafNode(StringBuilder displayString,
        nLeafNode oneLeaf) {
        displayString.append(oneLeaf.getAbsolutePath())
                     .append(" | ")
                     .append(oneLeaf.getCurrentNumberOfEvents())
                     .append(" | ")
                     .append(oneLeaf.getTotalPublished())
                     .append(" | ")
                     .append(oneLeaf.getTotalConsumed())
                     .append("\n");
    }
}
```

Identify Large Durable Outstanding Events

Identifies channels containing Durables with a large number of outstanding events.

Command Details

```
Usage:
runUMTool IdentifyLargeDurableOutstandingEvents
  -rname=<rname> -threshold=<threshold>
  [optional_args]
Examples:
runUMTool IdentifyLargeDurableOutstandingEvents
  -rname=nsp://localhost:9000 -threshold=100
Required arguments:
rname : URL of the realm to list the details of all the channels within.
threshold : Long value representing the tolerated number of outstanding events.
Optional Parameters:
username : Your Universal Messaging server username.
password : Your Universal Messaging server password.
```

Periodic Logging of Server Status

The Universal Messaging server writes status information to the log file at regular intervals. The default interval can be configured using the `StatusBroadcast` realm configuration property, and the default value is 5 seconds).

For information on realm configuration properties, see the section *Realm Configuration* in the Enterprise Manager part of the *Administration Guide*.

Sample status log message

```
ServerStatusLog> Memory=3577, Direct=3925, EventMemory=0,
  Disk=277070, CPU=0.2, Scheduled=29, Queued=0,
  Connections=5, BytesIn=12315, BytesOut=19876,
  Published=413, Consumed=1254, QueueSize=0,
  ClientsSize=0, CommQueueSize=0
```

The log file can be parsed to extract the server status and take appropriate preemptive actions if these parameters are deviating from set thresholds.

For more information, refer to the section *Periodic Logging of Server Status* section of the Universal Messaging *Concepts Guide*.

Other Parameters to Monitor

Apart from the parameters mentioned above, some more system parameters which need to be monitored are:

- **Disk utilization** : Any rapid increase in the disk usage should be tracked and alerted. An appropriate threshold needs to be set and monitored.
- **CPU utilization**
- **Memory utilization**
- **Network activity**

6 Monitoring Containers

Using Prometheus

Prometheus is an open-source monitoring tool. It provides a component called the *JMX Exporter* that can be used to gather data from MBeans of a JMX target. The JMX Exporter is included in the Universal Messaging product delivery as the jar file jmx_prometheus_javaagent.jar. You can run the JMX Exporter as a Java Agent to export a set of Universal Messaging time series data that can be analyzed by Prometheus.

For general details about Prometheus, visit the web site https://prometheus.io/. If you want more details about the JMX Exporter, visit the web page https://github.com/prometheus/jmx_exporter/tree/master/example_configs.

To use Prometheus in Universal Messaging, you must perform the following actions:

1. Enable JMX in Universal Messaging. For more information about enabling JMX, see the section "JMX User Guide > Enabling JMX on the server" in the *Concepts* guide.
2. Enable the JMX Exporter agent. For more information, see "Enable the JMX Exporter Agent" on page 28.
3. Set up the JMX Exporter port. For more information, see "Set Up the JMX Exporter Port" on page 29.
4. Connect with Prometheus. For more information, see "Connect with Prometheus" on page 29.

For more information about using JMX with Universal Messaging, see the "JMX User Guide" section in the *Concepts Guide*.

Enable the JMX Exporter Agent

When you start a Universal Messaging realm instance, the JMX Exporter agent is not running by default. To enable the agent, you must modify the Server_Common.conf file or Custom_Server_Common.conf file in the *Software AG_directory* \UniversalMessaging\server*instance_name*\bin directory.

To enable the JMX Exporter agent

1. Open Server_Common.conf in a text editor and search for the line referring to the JMX Exporter agent. The line looks similar to this:

```
wrapper.java.additional.<n>=
   -javaagent:<path1>/jmx_prometheus_javaagent.jar=0.0.0.0:9200:
   <path2>/jmx_exporter.yaml
```

2. If the line is commented out, uncomment it.
3. Restart the Universal Messaging realm instance.

Updat the jmx_exporter File when Applying Fixes

Universal Messaging fix packages can contain an updated jmx_exporter.yaml template file to resolve problems with the JMX Exporter agent. In these cases, you must manually update each Universal Messaging server instance to apply the changes to the JMX Exporter agent.

Important:
A new version of the jmx_exporter.yaml file delivered in a fix overwrites any changes you prevously made to the template. If you altered your local version of this file, Software AG recommends that you always keep a backup file with your changes.

To apply the updated jmx_exporter.yaml file to a server instance after installing a Universal Messaging fix

1. Copy the jmx_exporter.yaml file from *Software AG_directory*\UniversalMessaging\server\templates to the *Software AG_directory*\UniversalMessaging\server*instance_name*\bin directory. If you made your own changes to this file, you must transfer them to the new version.

2. Restart the Universal Messaging server instance.

Set Up the JMX Exporter Port

To configure a port number for the JMX Exporter agent, you must modify the Server_Common.conf file or Custom_Server_Common.conf file of a Universal Messaging realm instance in the *Software AG_directory* \UniversalMessaging\server*instance_name*\bin directory. The Server_Common.conf file contains a line that refers to the JMX Exporter agent. The line looks similar to the following:

```
wrapper.java.additional.<n>=
   -javaagent:<path1>/jmx_prometheus_javaagent.jar=0.0.0.0:9200:
   <path2>/jmx_exporter.yaml
```

In the example shown, the port number of the JMX Exporter agent is `9200`.

If you want to change the port, replace `9200` with your preferred port and restart the Universal Messaging realm instance.

Connect with Prometheus

Consider the following prerequisites for connecting the JMX Exporter agent of a Universal Messaging server instance with Prometheus:

- You have enabled the JMX Exporter agent.
- You have installed a runnable version of Prometheus and have access to the configuration file prometheus.yaml in that environment.

To connect the JMX Exporter agent with Prometheus

1. Open the prometheus.yaml configuration file in a text editor.
2. Under the `scrape_configs` section, add the address of the running JMX Exporter.

   Here is an example of how the `job_name` section should look:

```
- job_name: 'um_realm'
    scrape_interval: 5s
    static_configs:
      - targets: [127.0.0.1:9200]
```

3. Restart Prometheus.
4. Open a browser and navigate to the Prometheus address. The default address is `http://localhost:9090/`.
5. From the navigation bar, go to **Status > Targets**.

   You should see the new endpoint with job name "um_realm" as in the above example. The state of the job should be Active.

Modify Filters in jmx_exporter.yaml

The Server_Common.conf file of a Universal Messaging realm instance refers to the jmx_exporter.yaml file, which is the configuration file for the JMX Exporter agent. The jmx_exporter.yaml file contains filters of different JMX beans attributes. You can change the names of those filters.

By default, Prometheus queries are available in the Graph section of the product.

To edit the jmx_exporter.yaml file and change the names of filters

1. Open the jmx_exporter.yaml file in a text editor.
2. Go to the `rules` section. You will see filters for each JMX attribute, for example:

```
#Heap Memory Usage - HeapMemory
- pattern: com.softwareag.um.server<type=Broker, brokerName=(\S*)><>(HeapMemory)
  name: HeapMemory
  help: "Server Statistics"
  type: COUNTER
```

3. To change the filter name of a metric, edit the `name` property accordingly. For example, to change the name of the metric `HeapMemory` to `ServerHeapMemory`, change the `name` property to `ServerHeapMemory`.

Query Topics and Queues

If you want execute a query from Prometheus for some attribute of a Universal Messaging channel (JMS topic) or queue, you supply a query of the following form in the query field:

```
<AttributeName>_<ChannelName>
```

where `<AttributeName>` is the channel or queue bean attribute name and `<ChannelName>` is the name of the channel or queue.

For example, if you already created a channel "C1" and want to get the `NumberOfEvents` attribute, enter "NumberOfEvents_C1" in the query window.

Switch to Flat Topic and Queue Namespaces

By default, JMX namespaces for topics and queues are created using a folder tree structure, for example, `Folder1=q1,Folder2=q2,destinationName=queuename`, which might cause issues for some monitoring solutions.

For this reason, you can switch to flat namespaces for JMX topics and queues by using the `FlatStoreJMXBeanNamespace` realm configuration property.

In addition, you must edit the jmx_exporter.yaml file, which is the configuration file for the JMX Exporter agent. The file is located in the *Software AG_directory* \UniversalMessaging\server*instance_name*\bin directory.

To switch to flat namespaces

1. In Enterprise Manager, select a realm and go to **Config > JVM Management**.
2. Set the **FlatStoreJMXBeanNamespace** property to **true**.
3. Disable the **EnableJMX** property to stop the JMX server.
4. Edit the `pattern` parameter for all topic and queue attributes in the jmx_exporter.yaml file as follows:

   Old version of `pattern`:

   ```
   - pattern: com.softwareag.um.server<type=Broker, brokerName=(\S*),
   destinationType=Queue,
   (.*)><>(TotalPublished)
   ```

   New version of `pattern`:

   ```
   - pattern: com.softwareag.um.server<type=Broker, brokerName=(\S*),
   destinationType=Queue,
   destinationName=(\S*)><>(TotalPublished)
   ```

   You must replace `(.*)` in the old file version with `destinationName=(\S*)` in the new file version.

5. Set **EnableJMX** to **true** again.

After you you enable the functionality, a JMX queue namespace, for example, will have the format `destinationName=/q1/q2/q3`.

7 Handling Multiple Connections

Handling Many Connections on the Client Side

When a client Java application establishes many simultaneous sessions, or nSession instances, to Universal Messaging, you might encounter the following issues:

- The client application might run out of memory because the many Java threads that are created consume a lot of memory.
- The client application might return the error `"java.net.SocketException: Too many open files"` because it has reached the maximum allowed count of open files, or sockets, on the operating system.

Resolve Out of Memory Issues

Each non-multiplex nSession to Universal Messaging creates at least three Java threads:

- A connection read thread (`UM-Connection-Reader`) - responsible for reading events received from the Universal Messaging server.
- A connection write thread (`fProcessThreadedQueue`) - responsible for sending events to the Universal Messaging server.
- One or more event dispatch threads (`UM-Event-Processing-Pool`) - responsible for processing events that are read from the connection. These threads are available only when threading is enabled on the session. Threading is enabled by default and a thread is created when the session is started. For more information about threading, see the Universal Messaging Java API reference documentation for nSession.enableThreading(boolean).

The Java virtual machine (JVM) allocates a certain amount of memory to each thread stack. For this reason, when many threads are created for many sessions, the client application might run out of memory.

You can perform the following actions to reduce memory consumption:

- Reduce the amount of memory allocated for a thread stack by using the `-Xss` command-line option of the JVM. For more information about the option, see the documentation of your JVM vendor.
- Decrease the number of event dispatch threads used for every session by configuring the `com.softwareag.um.client.useSharedThreadPool` and `com.softwareag.um.client.SharedThreadPoolMaxSize` client system properties.

  Normally, the Universal Messaging Java client uses a dedicated thread pool of event dispatch threads for every session. Instead, you can configure the client to use a single, shared thread pool for this type of threads for all client sessions by using the following properties.

  - `com.softwareag.um.client.useSharedThreadPool` - enables usage of a shared thread pool for processing incoming nConsumeEvent events for each client session. Values are `true` or `false` (default). By default, each client session creates and uses a separate thread pool for processing received nConsumeEvent events. When this system property is set to `true`, all client sessions will use a shared thread pool. Using a shared thread pool for all client

sessions may have a performance impact depending on the size of the shared thread pool and the number of client sessions.

Note:
When each client session uses a separate thread pool, closing the session interrupts any event listener threads that are blocked during processing. When all client sessions share a thread pool, closing a session does not interrupt any event listener threads that are blocked during processing. Thus, the blocked event listener does not release a shared thread after closing the client session. This behavior impacts the processing of incoming events, because the shared pool will not have available threads. Software AG recommends to enable the shared pool for all client sessions only if the event listener of the client application ensures that the processing of the events is fast and graceful.

- `com.softwareag.um.client.SharedThreadPoolMaxSize` - specifies the maximum size of the shared thread pool. The default is `5` threads.

Resolve Socket Exceptions

The connection protocol used for each session to Universal Messaging impacts the total number of sockets that are created. An nSession using the nsp or nsps protocol, that is, the Universal Messaging socket protocol or SSL socket protocol, uses a single socket for communication with the Universal Messaging server. An nSession using the nhp or nhps protocol, that is, the Universal Messaging HTTP or HTTPS protocol, uses two sockets for communication with the server.

An application creating many sessions might fail with a `"java.net.SocketException: Too many open files"` error if it creates so many sockets that it reaches the maximum allowed count of open files, or sockets, on the operating system.

To resolve this issue, you might need to perform any or a combination of the following actions:

- Increase the limit of maximum allowed files or sockets.
- Consider if your application requires that many sessions.
- Check if the application creates sessions but fails to close them after they are no longer needed.
- Check for TCP/IP port exhaustion.

  This issue might occur even when the application closes sessions when they are no longer needed. If the application creates and then closes a lot of sessions in a short time, it might cause TCP/IP port exhaustion because operating systems usually keep sockets in a waiting status (TIME WAIT) for some time after sockets are released by the application. In this case, you might need to do the following:

  - Expand the range of ports available for use by your application.
  - Reduce the time the operating system keeps those sockets in a waiting status after they are released.

Handling Many Connections on the Server Side

To handle many simultaneous inbound connections to Universal Messaging, you might need to do the following:

Use NSP or NSPS Protocol

When handling many inbound connections that do not require the HTTP or HTTPS protocol, we recommend using the *Universal Messaging Socket Protocol (nsp)* or the *Universal Messaging SSL Protocol (nsps)*.

For more information about what native communication protocols Universal Messaging supports, see *Native Communication Protocols* in the Concepts Guide.

Adjust Interface Attributes

The Universal Messaging server does not allocate individual threads for each incoming connection. Instead, the server uses a thread pool named Accept Threads to accept the connection and a separate thread pool named Select Threads to monitor data to be read or written to the socket.

When a Universal Messaging server is handling many incoming connections at the same time, you might need to adjust several attributes of the server interface to which the client connects. You configure the attibutes on the **Comms > Interfaces** tab for a server in the Enterprise Manager as follows:

- **Accept Threads**. Increase the **Accept Threads** value, otherwise some of the connections will be queued in a backlog queue waiting to be accepted.
- **Backlog**. Increase the capacity of the **Backlog** queue, otherwise when the capacity of the backlog queue is reached, additional connection requests will be denied.

  Note:
  On Linux, you might need to set the kernel parameter `net.core.somaxconn`. For Linux 2.2 and greater, the parameter specifies the maximum queue length for completely established sockets waiting to be accepted.

  For additional operating-system-specific tuning, see *Tuning the Linux Operating System* in the Concepts Guide.
- **Auth Time**. Increase the time to authenticate incoming connection requests if connection authentication is slow and causes the server to close the connection.
- **Select Threads**. Keep the **Select Threads** value low even if the interface is handling a lot of connections. The number of select threads should not typically exceed the number of cores available.

  Select Threads transfers reading or writing of network data to separate read or write thread pools. You configure the maximum count of threads for the read and write thread pools in the **ReadThreadPoolMaxSize** and **WriteThreadPoolMaxSize** properties on the **Config > Thread Pool Config** panel in the Enterprise Manager. Read pool threads are used for processing

incoming client requests, while write pool threads are involved in writing data to client connections.

For more information about the attributes of an interface, see *Basic Attributes for an Interface* in the Administration Guide.

Resolve "java.lang.OutOfMemoryError Direct buffer memory" Errors

Universal Messaging uses in-memory buffers to handle network traffic. By default, these network buffers use the Java direct (off-heap) memory. To configure the Java direct memory for a realm in the Enterprise Manager, you go to **Config > Connection Config > UseDirectBuffering**.

If the connections handle a lot of traffic or if there are too many connections, the server might return the error `"java.lang.OutOfMemoryError Direct buffer memory"`. In this case, you should increase the amount of direct memory allocated for the Universal Messaging server process by using the `-XX:MaxDirectMemorySize=<size>` Java command-line argument.

For information about `-XX:MaxDirectMemorySize`, see the Oracle documentation on Java command-line options.

For more information about how to troubleshoot the `"java.lang.OutOfMemoryError Direct buffer memory"` error and configure direct memory, see "Troubleshooting" on page 47.

Handle a Slow Network when Clients are Across WAN

In case of a slower network when clients are across WAN, Software AG recommends that you increase the socket read and write buffer size based on bandwidth-delay product.

Be aware that increasing the read and write buffer size impacts the direct memory allocated by the server. You might need to adjust your direct memory accordingly.

Restricting Connections on the Server Side

You can restrict both total connections to the Universal Messaging server and the number of connections for a specific user by setting the `MaxNoOfConnections` and `MaxNoOfConnectionsPerUserName` properties, respectively. You configure the properties on the **Connection Config** tab for a server instance in the Enterprise Manager:

- **MaxNoOfConnections**. Sets the total number of concurrent connections to the server. Valid values range from `-1` (default), which means an unlimted number of connections, to `2147483647`.
- **MaxNoOfConnectionsPerUserName**. Sets the number of concurrent connections that a specific user can make to the server. Valid values range from `-1` (default), which means an umlimited number of connections, to `2147483647`.

Restrict Connections per User

To restrict the number of concurrent connections per user on the server, you configure the `MaxNoOfConnectionsPerUserName` property. The user name, which is part of the `username@host` subject, is the one provided when authenticating the session, typically when the session is created

using nSessionFactory.create(). If the user name is not provided, the default user is used, which is the operating system user running the operating system process of the client application, or the certificate subject (CN) when SSL with client authentication is used.

After you set this property, the server rejects any connections that are created after the connection limit is reached. The rejected session returns an nSecurityException with the reason why it is rejected.

For example, you can set `MaxNoOfConnectionsPerUserName` to `5` for the user testuser. If testuser tries to create 6 connections, after reaching the connection limit of 5, any upcoming connection will be rejected and the client will receive the following exception:
"`com.pcbsys.nirvana.client.nSecurityException: Server has rejected the connection due to reaching maximum allowable connections for user: testuser@127.0.0.1 current connections: 5 user limit: 5`".

Restrict Total Connections

To restrict the total number of concurrent connections to the server regardless of user, you configure the `MaxNoOfConnections` property. After you set this property, the server rejects any connections for any user after the connection limit is reached. The rejected session returns an nSecurityException with the reason why it is rejected.

For example, if you set `MaxNoOfConnections` to `52`, for any user who tries to connect after the limit is reached, the client will receive the following exception:
"`com.pcbsys.nirvana.client.nSecurityException: Server has rejected the connection due to reaching maximum allowable global connections for user: user0@127.0.0.1 current connections: 52 limit: 52`".

Restrict Both Connections per User and Total Connections

To restrict both total connections and the connections made per user, you configure both the `MaxNoOfConnections` and `MaxNoOfConnectionsPerUserName` properties, respectively. `MaxNoOfConnections` must always have a higher value than `MaxNoOfConnectionsPerUserName`. If the value of `MaxNoOfConnections` is lower, when the server reaches the limit for total connections it will stop accepting connections, although the limit set in `MaxNoOfConnectionsPerUserName` has not been reached.

Server Behavior when Authenticating Connections

The Universal Messaging server has denial-of-service (DoS) attack checks enabled by default. If the connections pending authentication within the **Auth Time** configured for an interface exceed the value of the `MaxUnauthorisedCount` property, the server rejects any upcoming connections for this host and reports "potential denial of service" errors. The host is either the remote IP address of the connecting socket or the load balancer host if a load balancer is used.

The `MaxUnauthorisedCount` realm configuration property specifies the maximum number of unauthorized connections per host. You configure `MaxUnauthorisedCount` in the **Thread Pool Config** group on the **Config** tab in the Enterprise Manager.

In addition, you can configure the server to report warning messages if a connection takes more than a specified time to authenticate. To do so, you set the `AuthenticationTimeLogThreshold` system property in the Server_Common.conf file in the *Software AG_directory* \UniversalMessaging\server*instance_name*\bin directory as follows:

```
wrapper.java.additional.n=-DAuthenticationTimeLogThreshold=<time_in_milliseconds>
```

where *n* is a unique positive integer. The default value is `1000` milliseconds. Usually, the property should have a value of between `1000` and `30000`, but you can adjust it according to the requirements of your system.

8 Contacting Software AG Technical Support

Support from Software AG is most effective if the right information is provided about the question or problem. Providing the following information is necessary for the support team to diagnose and reproduce the reported issue.

- General Information
  - The version and fix level of Universal Messaging servers.
  - Versions of all related software, including operating system, network components, etc.
  - Fixes/patches which have been applied to the Software AG products and any recently made changes.
- Specific error numbers and/or messages, where applicable.
- Sequence of events that led to the error. Basically the steps to reproduce the issue.
- Diagnostic data
  - A zip archive of the <DataDir> directory
  - Diagnostic data collected by the "Realm Information Collector" tool

Realm Information Collector for collecting diagnostic data

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

Command Details

```
Usage:
runUMTool RealmInformationCollector -mode=<mode>
  -instance=<instance> [optional_args]
Examples:
runUMTool
  RealmInformationCollector -mode=live -instance=umserver,umserver2
  -include=heapdump,heapdumps
runUMTool RealmInformationCollector -mode=offline
  -instance=* -include=data,heapdumps
```

```
Required arguments:
mode : Operating mode, either 'offline' or 'live'.
instance : Specifies a comma-separated list of realm server instance names
   to collect information from. Specify '*' to include all available instances.
Optional Parameters:
outputfile : The directory or file to write generated archive to.
   If this argument is omitted, the tool will generate the archive
   in the current working directory.
exclude : Specifies a comma-separated list of collector names to exclude.
include : Specifies a  comma-separated list of collector names to include.
username : Your Universal Messaging server username.
password : Your Universal Messaging server password.
```

Important:

Where possible it is useful to collect the diagnostic data from a running server (basically execute the RealmInformationCollector in "live" mode). So, you should always run this tool before you perform any recovery/restore operations.

Please note that in "live" mode, the tool doesn't collect the contents of the <DataDir> directory, as this might cause failures on the server. If the <DataDir> directory needs to be collected as well, shut down the server after running the tool in live mode and take copy of the directory manually.

9 Logging

Server-Side Journal Logging

The Universal Messaging server log is located at *Software AG_directory* \UniversalMessaging\server*instanceName*\nirvana.log. If any issues occur, first check the nirvana.log file of the realm. This is often the most descriptive and helpful source for problem analysis.

You can change the logging level for a realm in the **Logging Config** group on the **Config** tab in Enterprise Manager. The `fLoggerLevel` property specifies the level of logging to be used. Lower numbers here produce more verbose and detailed log messages.

Important:
More logging can result in worse performance and latency of the realm because more logging information is generated and written to disk.

Using the RealmInformationCollector to Collect Server Logs

You can use the `RealmInformationCollector` command-line diagnostic tool to automatically collect log information about the Universal Messaging installation, the server, the Java service wrapper, the Universal Messaging plug-in for Platform Manager, and migration.

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

Syntax

The command for collecting log information has the following syntax:

```
runUMTool RealmInformationCollector -mode=live -instance=instanceName
-include=logName[,logName][,logName][,...]
-logsduration=ndnh
```

Arguments and Options

-mode=live
Required. The server must be running when you execute the command.

-instance=*instanceName*
Required. The name of the server instance for which you want to execute the command.

-include=*logName*
Required. A comma-separated list of the names of the logs that you want to collect. You can list all available logs or a set of logs. The logs to specify are:

- `logs`. Collects logs of a realm server. The location of the file is *Software AG_directory* \UniversalMessaging\server*instanceName*\data\nirvana.log. Additionally, it collects logs created by the trace logger, if present. The location of the trace logging directory is *Software AG_directory* /UniversalMessaging\server*instanceName*\data\traceLogging by default.

- `tanukilogs`. Collects Java service wrapper logs of a realm server. This file is located at *Software AG_directory* \UniversalMessaging\server*instanceName*\bin\UMRealmService.log.
- `installlogs`. Collects installation log files for a product. These files are located in *Software AG_directory* \install\logs.
- `spmlogs`. Collects logs for the Universal Messaging plug-in for Platform Manager.
- `migrationlogs`. Collects Software AG migration logs.

-logsduration=*n*d*n*h
Required. The duration in days and hours for which you want to collect log information.

Examples

- To collect all log information for the server with the instance name "umserver" for a period of two days and four hours:

```
runUMTool RealmInformationCollector -mode=live -instance=umserver
-include=logs,tanukilogs,installlogs,spmlogs,migrationlogs -logsduration=2d4h
```

- To collect installation and server logs for the server with the instance name "umserver" for a period of two days and four hours:

```
runUMTool RealmInformationCollector -mode=live -instance=umserver
-include=logs,installlogs -logsduration=2d4h
```

Configuring Logging in the Server_Common.conf File

You can configure server-side logging in the Server_Common.conf configuration file located in the *Software AG_directory* \UniversalMessaging\server*instanceName*\bin directory. You can specify the following parameters related to logging:

- `LOG_FRAMEWORK`. A third-party logging framework to use. Valid values are `LOGBACK`, `LOG4J2`, and `fLogger`. The default is `fLogger`.
- `LogFileDepth`. The number of log files to keep on disk when using log rolling. The oldest log files are deleted when new log files are created.
- `LOGLEVEL`. The current log level to use. The default is `5`.
- `LOGFILE`. A log file to which to write the log entries. The default is `System.out`.
- `LOGSIZE`. The maximum size in bytes of the log file before the log file is rolled. The default is `100000`.

You specify these parameters in the Server_Common.conf file in the following format:

```
wrapper.java.additional.n=-Dparameter=value
```

where *n* is a unique positive integer.

For example, you can have the following entry:

```
wrapper.java.additional.26=-DLOGLEVEL=3
```

Audit Logging

Each Universal Messaging realm server maintains a log file containing information about administration operations performed on the realm. The log entries are called Audit Events and are stored at *Software AG_directory* \UniversalMessaging\server*instanceName*\NirvanaAudit.mem. These audit events are useful for tracking historical information about the realm and who performed what operation and when.

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

Client-Side Logging

The Universal Messaging client API supports a variety of parameters that you can specify in the command line of any Universal Messaging client application. The `-D` parameters that you can use for client-side logging are listed in the following table.

Name	Description
`LOGLEVEL`	Specifies the current log level to use (`0`=highest log level, `7`=lowest log level). The default value is `7` (OFF).
`LOGFILE`	Specifies a client log file. The default value is `System.out`.
`LOGSIZE`	Specifies the size in bytes before the log file is rolled. The default value is `100000`.

In addition, you can use the `javax.net.debug=ssl` generic JSSE command-line option to debug SSL issues. This option is available through the Java Secure Socket Extension (JSSE) component of the Java SE platform.

The default client log name has the format ClientLog_*ModuleName*.log

10 Troubleshooting

Out of Memory

One of the common reasons for a realm to attempt a graceful shutdown is an OutOfMemoryException.

Cause

- The realm server is under heavy load where the current allocated maximum heap resource is not enough to handle this high load or volume.
- Reliable channels keep all their events in memory. If the events on these reliable channels are larger (in total size) than the amount of memory available to the realm server's JVM then it will run out of memory.
- Due to a memory leak issue in the realm server.

Solution

- Consider using other channel types (for example, persistent for storing to disk)
- Purge events from channels either programmatically or by using channel attributes such as TTL or capacity.
- Servers with a large number of resources and handling large message loads may legitimately need a large amount of memory to operate. If high memory usage is due to high load, one suggestion is to increase the UM maximum heap size to a higher value so Universal Messaging has enough heap to handle the load. This value can be changed in an entry called "wrapper.java.maxmemory" in Server_Common.conf (under UniversalMessaging/server/<InstanceName>/bin), and then you need to restart the Universal Messaging realm.
- If the realm server is running on the RedHat Enterprise Linux 6 OS (RHEL 6), execute the command below to capture the output to a `debug.out` file during the time when memory is continuing to grow:

  ```
  pmap <realm_server_pid> > ./debug.out
  For example:
  pmap 17845 > ./debug.out
  ```

  Next, review the `debug.output` file from the above command. If there are many of the entries with `[anon]` as shown in the example below, this indicates the cause of the high memory usage

is due to the Linux/JVM bug which has been discussed in the following articles that provide more information and a work-around solution.

- https://serverfault.com/questions/341579/what-consumes-memory-in-java-process
- https://www.ibm.com/developerworks/community/blogs/kevgrig/entry/linux_glibc_2_10_rhel_6_malloc_may_show_excessive_virtual_memory_usage?lang=en

```
0000000000400000 4K r-x-- java0000000000600000 4K rw--- java
0000000002201000 4896K rw--- [ anon ] 00000005c0000000 8390912K rw--- [ anon ]
00000007c0240000 1046272K ----- [ anon ] 00007f6ab8000000 27108K rw--- [ anon ]
00007f6ab9a79000 38428K ----- [ anon ] 00007f6ac8000000 65492K rw--- [ anon ]
00007f6acbff5000 44K ----- [ anon ]
```

If the cause of high memory is due to JVM/Linux as described above, the system variable `MALLOC_ARENA_MAX` on the OS needs to be reviewed and adjusted to correct this issue. A suggestion is to set `MALLOC_ARENA_MAX=1` as a work-around to correct this growing memory.

Mitigation

- Monitor the realm server's JVM heap memory usage and take corrective actions.
- The following information needs to be gathered apart from the regular details, if further root cause analysis is needed:

1. Execute a pmap command as below to capture the output

```
pmap <realm_server_pid> > ./debug.out
For example: pmap 17845 > ./debug.out
```

2. Take the heap dump of the realm server during the time when memory is high. From the command line, navigate to the <InstallDir>/jvm/jvm/bin directory and execute the following command to generate the heap dump:

```
jmap -dump:format=b,file=<full_path>\<file_name>.hprof <RealmServer_pid>
```

For example, on Windows:

```
jmap -dump:format=b,file=C:\tmp\realm9300.hprof 13276
```

Unix:

```
./jmap -dump:format=b,file=/temp/realm9300.hprof 13276
```

java.lang.OutOfMemoryError Direct buffer memory

There is a case where the realm server will shut down when the direct memory configuration is not sufficient for the realm server to use while handling the load or transactions.

1. Review the nirvana.log file (under the <DataDir> directory) and verify to see if there is an error such as `"java.lang.OutOfMemoryError Direct buffer Memory"` before the realm server went down, for example:

```
[Tue Jul 25 12:20:07 PDT 2017],UserManager: User abc@<host>
Logged Out using nsp, Reason : java.lang.OutOfMemoryError Direct buffer memory,
session established for 26192 seconds, Session Id = 3c7fd6a000000000 ID:
```

```
<host>:37532
```

2. Verify the realm configuration to see whether realm server is configured for use with direct buffering or heap buffering, since this issue only happens when the realm server is configured to use direct buffering. From Enterprise Manager, under the "Config" tab, navigate to "Connection Config", and check "UseDirectBuffering". If it set to "true", this means the realm server will allocate DirectByteBuffers to use for network I/O (otherwise the realm server will use HeapByteBuffers instead).

   If "UseDirectBuffering" is set to true, and the realm server stopped with the above error in nirvana.log, this indicates there are two possibilities that can cause this issue:

   - The available direct memory which is allocated for this server is not sufficient.
   - Or, the direct memory is sufficient, however the garbage collection (GC) does not run frequently enough to free the memory.

3. Next, to find out how much memory is currently allocate for Direct Memory, open Server_Common.conf (under <InstallDir>/UniversalMessaging/server/<InstanceName>/bin) and check the current value of "MaxDirectoryMemorySize". For example, the following entry sets the size to 1 gigabyte:

   ```
   wrapper.java.additional.17=-XX:MaxDirectMemorySize=1G
   ```

4. To monitor the buffer usage, from Enterprise Manager, navigate to the "Monitoring" > "Threads" > "Pools" tab, then monitor the "Buffers Created" and "Buffers Reused" values to get a better idea of the buffer usage.

Cause

- Direct memory configuration is not sufficient for the realm server to use while handling the load or transactions.

Solution

There are three suggested options that can be used to resolve this issue depending on the customer's environment, configuration, and architecture.

- Increase the value for MaxDirectMemorySize (configured in Server_Common.conf) as long as there is enough memory (RAM) available on the system.
- Or, decrease the maximum heap memory setting (wrapper.java.maxmemory in Server_Common.conf). This will cause more frequent GC sweeps so it can release the allocated direct memory.
- Or, disable to use of direct memory. This can be done by setting "Config" > "Connection Config" > "UseDirectBuffering"=false in the Enterprise Manager.

  Note:
  This will adversely affect the performance of the UM to some extent. It also requires to increase the max heap size. The heap size should be at least equal to the current heap size plus the direct memory size.

Mitigation

- Monitor the memory usage of the realm server process and take corrective action.
- The following information needs to be gathered apart from the regular details, if further root cause analysis is needed:
  - The buffer usage from Enterprise Manager. Navigate to the tab "Monitoring" > "Threads" > "Pools", then monitor the "Buffers Created" and "Buffers Reused" values to get a better idea of the buffer usage.

Too many open files

The realm server can shut down due to an environment error, data/RealmSpecific/channels.nst_new (Too many open files).

Cause

- The realm server has exceeded the maximum file descriptors limit on the operating system.

Solution

- Increase the ulimit values based on the following requirements
  - ~2 socket connection per client
  - 1 per channel or queue
  - ~30 for realm server state synchronization

Mitigation

- Set appropriate ulimit values based on the number of clients, channels/queues

Cluster Not Forming

Cause

- The cluster cannot form if more than 50% of the realm servers forming the cluster are unavailable or not reachable
- Inter-realm server communication or network is broken

Solution

- Make sure that at least 2 realm servers in a 3 node cluster are online and reachable
- Make sure that all the realm servers are reachable from each other

Contact Software AG Technical Support if none of the above works.

Mitigation

- Monitor the cluster state

Realm node fails to start

The realm server uses the Tanuki Java Service wrapper to start. Behind the scenes, the Tanuki wrapper tracks the health of the JVM process by pinging it. If it does not get a response from the JVM started process within a certain preset time limit, the Tanuki wrapper throws the message `"Startup failed: Timed out waiting for singal from JVM"`. Then, the wrapper will either try to restart the JVM process again if "Automatic JVM restarts" is enabled, or the Tanuki wrapper will issue a shutdown if "Automatic JVM Restarts" is disabled.

1. Review UMRealmService.log : Navigate to the <InstallDir>/UniversalMessaging/server/<InstanceName>/bin directory, open UMRealmService.log and search for "Timed out waiting for signal from JVM". For example, the messages when the realm server fails to come up may look something like this:

```
wrapper | Startup failed: Timed out waiting for signal from JVM.
wrapper | JVM did not exit on request, termination requested.
wrapper | JVM received a signal SIGKILL (9).
wrapper | JVM process is gone.
wrapper | JVM exited after being requested to terminate.
wrapper | Automatic JVM Restarts disabled. Shutting down.
wrapper | <-- Wrapper Stopped
```

2. Check Server_Common.conf under the <InstallDir>/UniversalMessaging/server/<InstanceName>/bin directory to see if the entry "wrapper.startup.timeout" exists and is set to any value. If this entry does not exist, the default value set by Tanuki is 30 seconds.

Cause

The realm server could not start within the timeout setting of Tanuki wrapper.

Typical causes could be:

- Memory or CPU starvation
- Heavy swap usage

Solution

1. The quick solution is to add (if it does not exist) or update the entry in Server_Common.conf in the <InstallDir>/UniversalMessaging/server/<InstanceName>/bin directory, and set the timeout to a higher value (for example 300 seconds), then restart the realm server.

```
wrapper.startup.timeout=300
```

2. Once the realm server is up, review nirvana.log to find out the reason for the longer startup time

Mitigation

- None

Realm node fails to recover

A server which fails to recover will retry automatically and can often recover given enough time. Realms that are offline for a long time may take a long time to recover, as all information has to be re-synchronized from the master. Monitor the cluster state as noted in one of the previous sections. As long as the recovery is progressing and not looping on any particular message, the realm should be left to complete its recovery.

Cause

A common cause of failure to recover is data corruption. This will appear in the log files as repeated occurrences of 'Cluster recovery failed... retrying' and 'going into recovery for channel' for the same channel.

For example:

```
[Wed Feb 18 10:20:49 EST 2017],Cluster recovery failed...retrying:
/cfp/core/maps/moduleHeartBeatMap First EID:540811 Last EID:541706
with 17 events First EID:516482 Last EID:541706 with 19 events
[Wed Feb 18 10:20:49 EST 2017],Going into recovery for channel
[/cfp/core/maps/moduleHeartBeatMap] because eid 541706 Status EID 541706
Channel Events: 17 Status Events:19
```

Storage file(s) can get corrupted for various reasons:

- Disk space ran out and the information cannot be stored to the file properly, or only partial information is stored which leads to corrupted information in the file(s).
- Disk drive issues which causes the file to be damaged or not accessible.
- Synchronization issues between storage files throughout the cluster.

For example, below are a few examples of the errors to indicate the storage file is corrupted or has some kind of problem which prevent the realm server from coming up. In the other case, the error may not be exactly the same, but if the same error with the same filename keeps repeating, this hints that the file is probably corrupted or has some issue.

1. The following exception indicates that there is an issue with the file `NirvanaClusterQueueCommitChannelb7cd0d8d1f8b5e.mem`, which prevents the realm server from starting:

   ```
   [Fri Apr 28 21:33:09 BST 2017],Mixed-Store>
   /UniversalMessaging/server/my_server/data/
     RealmSpecific/NirvanaClusterQueueCommitChannelb7cd0d8d1f8b5e.mem:
     Exception Follows:
   Fri Apr 28 21:33:09 BST 2017],Socket Stream reach EOF
   java.io.EOFException: Socket Stream reach EOF at
   com.pcbsys.foundation.io.fEventInputStream.readComplete(fEventInputStream.java:393)
   at
   com.pcbsys.foundation.io.fEventInputStream.readByteArray(fEventInputStream.java:300)
   at
   com.pcbsys.nirvana.base.events.nBasePublishEvent.performRead(nBasePublishEvent.java:430)
   ```

2. The following messages repeat every 1 minute in the log, and this indicates that there is an issue with the file

/namedsubqueues/xyz/top/um/productESB/NamedSubscriber4611686018427387905, which prevents the realm server from coming up.

```
[Sat Apr 22 00:38:24 BST 2017],Restarting restore for
/namedsubqueues/xyz/top/um/productESB/NamedSubscriber4611686018427387905
[Sat Apr 22 00:38:24 BST 2017],Cluster> Cluster Recovery started for
/namedsubqueues/xyz/top/um/productESB/NamedSubscriber4611686018427387905...
[Sat Apr 22 00:38:25 BST 2017],Restarting restore for
/namedsubqueues/xyz/top/um/productESB/NamedSubscriber4611686018427387905
[Sat Apr 22 00:38:25 BST 2017],Cluster> Cluster Recovery started for
/namedsubqueues/xyz/top/um/productESB/NamedSubscriber4611686018427387905
```

Solution

The data in the Universal Messaging channels is stored in `.mem` files on disk, in the <DataDir> directory of the realm. The contents of the channel can be found in a file called `<channelName><hashNumber>.mem`. The hash number is internally generated and is usually not significant for our purposes.

Follow these steps for recovering from the data corruption cases

- Stop (gracefully) the realm that is having problems recovering
- Take a backup of the <DataDir> directory
- Identify the .mem file for the channel on disk by the channel name
- Delete this .mem file
- Restart the realm

The data will be re-synchronized from the master, so no data will be lost for the channel.

Important:
Deleting a store erases the data from that slave node. If multiple nodes fail it may become impossible to recover data from the cluster. Always back up the contents of the data directory before performing any manual operation. Never modify the contents of the data directory while a realm is running.

Note:

If there are one or more .mem files that are drastically different from the store on the master realm, recovery can take a long time, as this .mem file has to be reconciled with the master store. This case can be seen by cluster log messages showing those specific channels repeatedly in recovery over an extended period. For example multiple occurrences of messages like the following would be diagnostic:

```
Cluster> Still in recovery for /various/messenger/generic
```

For these cases removing the .mem file from a realm will cause the entire store to be synchronized from the master and will speed up recovery. Proceed as outlined above for the data corruption case.

Contact Software AG Technical Support if none of the above works.

Mitigation

- If possible, shut down all the servers (gracefully). Killing the realm can cause data corruption and problems reforming the cluster.

Realm drops out of cluster

A realm goes offline and drops out of the cluster, with the following message indicating that the keep alive pings between the realms are not being returned:

```
nPhysicalKeepAlive : Terminating inactive cluster inter realm link
due to no data received
```

Sometimes disconnected messages appear, like:

```
[Tue Jun 13 23:38:56 EDT 2017],Disconnected from p2um1
```

Cause

- This may happen because of a slow response on the remote realm(s) due to load and/or resource constraints. By default a realm sends keep alive messages every 10 seconds and will log the message if there is no response in 35 seconds. Check the log on the remote realm(s) for signs of distress, warnings, and errors.
- Frequent GCs could result in message in `UMRealmService.log` like below, indicating that the JVM's response is very slow.

  ```
  STATUS | wrapper | 2017/06/18 23:38:50 |
     Pinging the JVM took 4 seconds to respond.
  ```

- The slow response of the JVM could be caused by CPU, memory, or other constraints in its environment.
- Cases of network errors are visible as java.io errors in the log files.
- In the case of socket/TCP problems, UM does not always get a clear message from the Java I/O libraries on exactly what went wrong. The Java libraries will sometimes just give a generic socket exception message, which shows up as the java.io.EOFException or similar. Check on the network using network diagnostics.

Solution

Identify the root cause from the logs as described in the previous section and try to resolve the issue.

Mitigation

Monitor the network and other resources and take appropriate corrective actions before a realm server goes unresponsive and drops out of the cluster.

Consumer Performance leading to "Consumer Warning" entries

In some cases, there can be "Consumer Warning" entries in the server log (logged in log level "Warning").

Cause

These log entries are generally related to poor performance on the consumer side. Typical symptoms are:

- Events piling up on a queue or channel with shared or serial durables.
- Slow consumption or publishing as a result of a consumer having incorrectly configured event filters.

Solution

The presence of these "Consumer Warning" log entries can give information about consumers that are not properly configured, such as:

- Consumers which are slow in acknowledging events, which can lead to events pile up.
- Consumers with incorrectly configured filters which can affect performance or lead to events piling up.

To be able to see the "Consumer Warning" log entries, the log level of the server should be set to "Warning" or more verbose.

Mitigation

Based on the "Consumer Warning" entries, check if your event consumption strategy can be optimized.

11 Purging and Archiving

Journal Log files

Universal Messaging provides automatic log file rolling. This means when the size of the log file reaches a certain threshold, the server closes the current log file and opens a new one. The maximum size for a log file is set to 100,000,000 bytes (approximately 100 MB) by default. This value can be changed by modifying the -DLOGSIZE property in the Server_Common.conf file under <InstallDir>/UniversalMessaging/server/ directory. You can also change this via Enterprise Manager's "Config" panel in the "Logging Config" group. The 'DefaultLogSize' property specifies the default size of the log in bytes.

Depending on the log level set and the level of activity, these logs files can grow quite large and occupy considerable amounts of disk space. It is recommend to archive and clean up the old journal log files periodically.

Audit Log files

Depending on what is being logged to the audit file, the file can grow quite large. As it is an audit and provides historical data, there is no automatic maintenance of the file. It is up to the realm administrators on when the file is archived. The 'Archive Audit' button in the Audit panel of the Enterprise Manager rolls the audit log file by renaming the existing audit file to a name with the current date and starts a new audit file. It is recommend to archive and clean up the archived audit log files periodically.