# Universal Messaging Reference Guide

Version 10.1

October 2017

This document applies to Universal Messaging Version 10.1 and to all subsequent releases.

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

Copyright © 2013-2017 Software AG, Darmstadt, Germany and/or Software AG USA Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors.

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

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

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

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

**Document ID: NUM-RG-101-20171103**

# Table of Contents

# Overview

The Reference Guide contains the following sections:

- Glossary
- The Javadoc for the Client APIs

# 1 Glossary

This glossary provides an overview of technical terms used in the product documentation.

## ACL

Universal Messaging's Access Control List (ACL) controls client connection requests and subsequent Universal Messaging operations. By default, a realm will always perform access control checks.

The Universal Messaging realm has an ACL associated with it. The ACL contains a list of subjects and the operations that each subject can perform on the realm.

Each channel, queue and service also has an associated ACLs that defines subjects and the operations the subjects can perform. Each type of acl entry has a number of flags that can be set to true or false in order to specify whether the subject can or can't perform the operation.

## Channel

A channel is the distribution mechanism for an asynchronous *publish/subscribe* messaging model.

In this model, the publisher and consumer of an event (or "message") are decoupled, but are both connected to a common channel which exists within the Universal Messaging realm server.

The publisher publishes its data as events to the channel. As messages arrive on a channel, the server automatically sends them to all consumers subscribed to the channel.

Universal Messaging supports multiple publishers and consumers on a single channel.

Note: the terms "*channel*" and "*topic*" are used interchangeably throughout this documentation.

## Consumer

A consumer is a Universal Messaging client that receives events from a Universal Messaging channel, queue or datagroup.

Note: the terms "*consumer*" and "*subscriber*" are used interchangeably throughout this documentation.

## DataGroup

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

DataGroups provide an alternative to channels/topics for publish/subscribe. DataGroups are essentially groups of consumers to which publishers can send events; more specifically, DataGroup members are either individual consumers or other (nested) DataGroups. A consumer which is a member of a DataGroup is known as a DataStream.

Messages published to a DataGroup will be sent to all members of the group. There can be multiple publishers associated with a single DataGroup, and DataGroup membership can be managed by any Universal Messaging client that has permissions to do so.

DataGroups are designed to support large numbers of consumers whose subscriptions are typically fluid in nature. The addition or removal of consumers from DataGroups can be entirely transparent from the consumer perspective.

### An Example DataGroup Structure

Imagine a Foreign Exchange organization that provides different tiers of currency price data to users on different desks at their various customers. Valued customer desks might be provided with gold level data, while less valued desks might receive silver or bronze tiered data:

- EURUSD_Gold
  - Customer_Desk_A
    - *User1*
    - *User2*
  - Customer_Desk_B
    - *User3*
    - *User4*
- EURUSD_Silver
  - Customer_Desk_C
    - *User5*
  - Customer_Desk_D
    - *User6*
- EURUSD_Bronze
- GBPUSD_Gold
  - Customer_Desk_A
    - *User1*
    - *User2*

A suitable DataGroup structure to represent this arrangement might be to create eight DataGroups:

- EURUSD_Gold
- EURUSD_Silver

- EURUSD_Bronze
- GBPUSD_Gold
- Customer_Desk_A
- Customer_Desk_B
- Customer_Desk_C
- Customer_Desk_D

Then, structure these DataGroups as follows:

- EURUSD_Gold
  - Customer_Desk_A
  - Customer_Desk_B
- EURUSD_Silver
  - Customer_Desk_C
  - Customer_Desk_D
- EURUSD_Bronze
- GBPUSD_Gold
  - Customer_Desk_A

Price data would be published directly to the four DataGroups EURUSD_Gold, EURUSD_Silver, EURUSD_Bronze and GBPUSD_Gold (though, optimally, a publisher may well choose not to publish to the empty DataGroup EURUSD_Bronze). By virtue of this structure, prices published to EURUSD_Gold, for example, will be delivered to both the DataGroups Customer_Desk_A and Customer_Desk_B.

End user clients with DataStream-enabled sessions can be added to arbitrary DataGroups - and thus receive data from those DataGroups. So, for example, adding User1 and User2 as DataStream members of the DataGroup Customer_Desk_A, both users would, by virtue of the above structure, receive price data via their inherited membership of both EURUSD_Gold and GBPUSD_Gold.

### Dynamic Changes

Let us now assume that our organization decides to upgrade the level of EURUSD price data sent to users on Customer_Desk_C from silver to gold.

Achieving this with DataGroups is extremely simple. All we need to do is remove the Customer_Desk_C DataGroup from the EURUSD_Silver DataGroup, and add it to the EURUSD_Gold DataGroup instead, resulting in the following new structure:

- EURUSD_Gold
  - Customer_Desk_A
  - Customer_Desk_B

    - Customer_Desk_C
- EURUSD_Silver
    - Customer_Desk_D
- EURUSD_Bronze
- GBPUSD_Gold
    - Customer_Desk_A

### Even More Dynamic Changes

Let's take our example a little further. Assume that the publisher of the gold EURUSD price data stops for some reason, and that the organization decides customers who were receiving this data should, for the time being, fall back to silver EURUSD level prices instead.

One way of doing this would be to move the three DataGroups Customer_Desk_A, Customer_Desk_B and Customer_Desk_C out of the EURUSD_Gold DataGroup and into the EURUSD_Silver DataGroup.

An even simpler way, however, would be to just move the EURUSD_Gold DataGroup itself into the EURUSD_Silver DataGroup, resulting in the following structure:

- EURUSD_Silver
    - EURUSD_Gold
        - Customer_Desk_A
        - Customer_Desk_B
        - Customer_Desk_C
    - Customer_Desk_D
- EURUSD_Bronze
- GBPUSD_Gold
    - Customer_Desk_A

This structure could be used while the gold level publisher of EURUSD prices remains inactive. Once the publisher of the gold level data restarts, the EURUSD_Gold DataGroup can be removed from the EURUSD_Silver DataGroup (for if it were not removed, the users expecting gold level EURUSD prices would now receive both gold and silver level prices).

### Other Notes

It is typical (though not necessary) for a process which publishes data to DataGroups to be independent of the process that manages DataGroup structures and DataGroup membership of end-user DataStreams. This allows the logic for these two very different responsibilities to be separated.

### DataStream

A Universal Messaging client (typically, but not necessarily, a simple consumer) may initialise a DataStream-enabled session, making it eligible for membership in one or more DataGroups. Such a client is considered a DataStream.

Note that DataStreams do not determine the DataGroups of which they are members; their membership is determined by clients with the appropriate permissions (see DataGroup).

### Dictionary

Event Dictionaries provide an accessible and flexible way to store any number of message properties for delivery within a Universal Messaging event.

Event Dictionaries are quite similar to a hash table, supporting primitive types, arrays, and nested dictionaries.

Filtering allows subscribers to receive only specific subsets of a channel's events by applying the server's advanced filtering capabilities to the contents of each event's dictionary.

### Enterprise Manager

Universal Messaging's Enterprise Manager is a powerful GUI management tool that allows you to control, configure and administer all aspects of any Universal Messaging realm or clusters of realms.

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

Enterprise Manager is completely implemented using the Universal Messaging Management API, so any of its features can be easily integrated into bespoke or 3rd party systems management services.

### Event (Message)

An event is the message object in which a publisher inserts data to be published. Events are published to either a Universal Messaging channel, queue or datagroup. From there, it is passed on to consumers. Events are language agnostic, which means that clients using different languages can interact seamlessly.

An event may be a simple byte array or contain more complex structures such as dictionaries, Google Protocol Buffers, JSON or JMS events.

Note: the terms "event" and "message" are used interchangeably throughout this documentation.

### Filtering

Universal Messaging provides a server side filtering engine that allows only events meeting certain criteria to be delivered to consumers.

Standard filtering, as defined by JMS, allows events to be evaluated based on the value of the dictionary keys prior to delivering the event to the consumer. Universal Messaging supports not only standard filtering, but also filtering based on arrays and nested dictionaries contained within event dictionaries. There is no limit to the depth of nested properties that can be filtered.

Universal Messaging events can contain not only an event dictionary and a tag, but also a byte array payload of data. Universal Messaging consequently supports a yet more advanced form of filtering based on the content of the byte array data itself. In addition, filtering is possible based on time and consumer location.

### Fragmentation

Although there is no specific limit to the size of events that can be published to Universal Messaging, from a network perspective it is usually more efficient to publish several smaller messages than one large one.

Publishers using the Universal Messaging Enterprise APIs can choose to transparently fragment large events into smaller chunks for publishing. The Universal Messaging client API will transparently reconstitute the event at delivery.

### Forever IFrame

Forever IFrame is a technique which allows a web server to stream data into a client browser. This is done through a hidden inline frame in the page source which is declared to be infinitely long. It is one of many push technologies under the umbrella term Comet.

Universal Messaging's API for JavaScript supports Forever IFrame, allowing it to communicate with a server using this delivery mode.

### LongPolling

LongPolling is a technique which allows a web browser running as a client to asynchronously receive updates from a server machine. It is one of many push technologies under the umbrella term Comet.

Universal Messaging's API for JavaScript supports LongPolling, allowing it to communicate with a server using this delivery mode.

### Publisher

A publisher is a Universal Messaging client that sends data/messages as events to a Universal Messaging channel, queue or datagroup.

Note: the terms "*publisher*" and "*sender*" are used interchangeably throughout this documentation.

### Subject

A subject corresponds to the user information for a publisher or subscriber's realm connection. Subjects are used when defining ACLs.

A subject is comprised of a *username* and a *host*:

- The username component of the subject is the name of the user taken from either the operating system of the machine they are connecting from, or the certificate name if they are using an SSL protocol.
- The host component of the subject is either the IP address or the hostname of the machine from which they are connecting.

The subject takes the form of `username@host`; for example:

```
johnsmith@192.168.1.2
```

### Topic

A topic is a JMS term which translates directly to a Universal Messaging channel. Consequently, the terms "*channel*" and "*topic*" are used interchangeably throughout this documentation.

### Queue

A queue is much like a channel; the primary difference is that only one consumer can read any individual event from a queue. Consumed events are immediately removed from the queue, and are no longer available for consumption by any other consumer. Thus, a queue guarantees that each event is delivered only once.

If more than one consumer is subscribed to a queue, then queued events are distributed amongst consumers in a round-robin fashion.

### Realm

A Universal Messaging realm is the name given to a single Universal Messaging server. Universal Messaging realms can support multiple network interfaces, optionally supporting different Universal Messaging protocols. Each such interface is represented by a URL, known as an RNAME. Thus a single realm server can have more than one RNAME.

Each Universal Messaging realm defines a namespace of its own resources (such as channels and queues), but it is possible to merge the namespaces of multiple realms into one large federated namespace for transparent client access to resources on different realms.

Universal Messaging also provides the ability to create clusters of realms that share common resources within the namespace.

### RNAME

An RNAME is used by Universal Messaging Clients to specify how a connection should be made to a Universal Messaging realm server. The URL describes a particular interface on the Universal Messaging realm server.

### WebSockets

WebSocket is a technology for providing full-duplex connections over a TCP socket within the web browser. The WebSocket API is currently being developed by W3C and the protocol standardised by IETF.

Universal Messaging's API for JavaScript currently supports WebSockets for all browsers which implement the standard.

# 2 API Documentation (Javadoc etc.) for Developers using the Client APIs

The Universal Messaging documentation set includes API documentation for Enterprise Client APIs, Web Client APIs and Mobile Client APIs.

The API documentation for the client APIs is available in the HTML version of this document.