Informatica Ultra Messaging (Version 6.7.1)

Guide for Persistence

Informatica Ultra Messaging Guide for Persistence

Version 6.7.1
August 2014

Copyright (c) 2009-2014 Informatica Corporation. All rights reserved.

This software and documentation contain proprietary information of Informatica Corporation and are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright law. Reverse engineering of the software is prohibited. No part of this document may be reproduced or transmitted in any form, by any means (electronic, photocopying, recording or otherwise) without prior consent of Informatica Corporation. This Software may be protected by U.S. and/or international Patents and other Patents Pending.

Use, duplication, or disclosure of the Software by the U.S. Government is subject to the restrictions set forth in the applicable software license agreement and as provided in DFARS 227.7202-1(a) and 227.7702-3(a) (1995), DFARS 252.227-7013©(1)(ii) (OCT 1988), FAR 12.212(a) (1995), FAR 52.227-19, or FAR 52.227-14 (ALT III), as applicable.

The information in this product or documentation is subject to change without notice. If you find any problems in this product or documentation, please report them to us in writing.

Informatica, Informatica Platform, Informatica Data Services, PowerCenter, PowerCenterRT, PowerCenter Connect, PowerCenter Data Analyzer, PowerExchange, PowerMart, Metadata Manager, Informatica Data Quality, Informatica Data Explorer, Informatica B2B Data Transformation, Informatica B2B Data Exchange Informatica On Demand, Informatica Identity Resolution, Informatica Application Information Lifecycle Management, Informatica Complex Event Processing, Ultra Messaging and Informatica Master Data Management are trademarks or registered trademarks of Informatica Corporation in the United States and in jurisdictions throughout the world. All other company and product names may be trade names or trademarks of their respective owners.

Portions of this software and/or documentation are subject to copyright held by third parties, including without limitation: Copyright DataDirect Technologies. All rights reserved. Copyright © Sun Microsystems. All rights reserved. Copyright © RSA Security Inc. All Rights Reserved. Copyright © Ordinal Technology Corp. All rights reserved.Copyright © Aandacht c.v. All rights reserved. Copyright Genivia, Inc. All rights reserved. Copyright Isomorphic Software. All rights reserved. Copyright © Meta Integration Technology, Inc. All rights reserved. Copyright © Intalio. All rights reserved. Copyright © Oracle. All rights reserved. Copyright © Adobe Systems Incorporated. All rights reserved. Copyright © DataArt, Inc. All rights reserved. Copyright © ComponentSource. All rights reserved. Copyright © Microsoft Corporation. All rights reserved. Copyright © Rogue Wave Software, Inc. All rights reserved. Copyright © Teradata Corporation. All rights reserved. Copyright © Yahoo! Inc. All rights reserved. Copyright © Glyph & Cog, LLC. All rights reserved. Copyright © Thinkmap, Inc. All rights reserved. Copyright © Clearpace Software Limited. All rights reserved. Copyright © Information Builders, Inc. All rights reserved. Copyright © OSS Nokalva, Inc. All rights reserved. Copyright Edifecs, Inc. All rights reserved. Copyright Cleo Communications, Inc. All rights reserved. Copyright © International Organization for Standardization 1986. All rights reserved. Copyright © ej-technologies GmbH. All rights reserved. Copyright © Jaspersoft Corporation. All rights reserved. Copyright © is International Business Machines Corporation. All rights reserved. Copyright © yWorks GmbH. All rights reserved. Copyright © Lucent Technologies. All rights reserved. Copyright (c) University of Toronto. All rights reserved. Copyright © Daniel Veillard. All rights reserved. Copyright © Unicode, Inc. Copyright IBM Corp. All rights reserved. Copyright © MicroQuill Software Publishing, Inc. All rights reserved. Copyright © PassMark Software Pty Ltd. All rights reserved. Copyright © LogiXML, Inc. All rights reserved. Copyright © 2003-2010 Lorenzi Davide, All rights reserved. Copyright © Red Hat, Inc. All rights reserved. Copyright © The Board of Trustees of the Leland Stanford Junior University. All rights reserved. Copyright © EMC Corporation. All rights reserved. Copyright © Flexera Software. All rights reserved. Copyright © Jinfonet Software. All rights reserved. Copyright © Apple Inc. All rights reserved. Copyright © Telerik Inc. All rights reserved. Copyright © BEA Systems. All rights reserved. Copyright © PDFlib GmbH. All rights reserved. Copyright © Orientation in Objects GmbH. All rights reserved. Copyright © Tanuki Software, Ltd. All rights reserved. Copyright © Ricebridge. All rights reserved. Copyright © Sencha, Inc. All rights reserved.

This product includes software developed by the Apache Software Foundation (http://www.apache.org/), and/or other software which is licensed under various versions of the Apache License (the "License"). You may obtain a copy of these Licenses at http://www.apache.org/licenses/. Unless required by applicable law or agreed to in writing, software distributed under these Licenses is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the Licenses for the specific language governing permissions and limitations under the Licenses.

This product includes software which was developed by Mozilla (http://www.mozilla.org/), software copyright The JBoss Group, LLC, all rights reserved; software copyright © 1999-2006 by Bruno Lowagie and Paulo Soares and other software which is licensed under various versions of the GNU Lesser General Public License Agreement, which may be found at http:// www.gnu.org/licenses/lgpl.html. The materials are provided free of charge by Informatica, "as-is", without warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose.

The product includes ACE(TM) and TAO(TM) software copyrighted by Douglas C. Schmidt and his research group at Washington University, University of California, Irvine, and Vanderbilt University, Copyright (©) 1993-2006, all rights reserved.

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (copyright The OpenSSL Project. All Rights Reserved) and redistribution of this software is subject to terms available at http://www.openssl.org and http://www.openssl.org/source/license.html.

This product includes Curl software which is Copyright 1996-2013, Daniel Stenberg, <daniel@haxx.se>. All Rights Reserved. Permissions and limitations regarding this software are subject to terms available at http://curl.haxx.se/docs/copyright.html. Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

The product includes software copyright 2001-2005 (©) MetaStuff, Ltd. All Rights Reserved. Permissions and limitations regarding this software are subject to terms available at http://www.dom4j.org/ license.html.

The product includes software copyright © 2004-2007, The Dojo Foundation. All Rights Reserved. Permissions and limitations regarding this software are subject to terms available at http://dojotoolkit.org/license.

This product includes ICU software which is copyright International Business Machines Corporation and others. All rights reserved. Permissions and limitations regarding this software are subject to terms available at http://source.icu-project.org/repos/icu/icu/trunk/license.html.

This product includes software copyright © 1996-2006 Per Bothner. All rights reserved. Your right to use such materials is set forth in the license which may be found at http:// www.gnu.org/software/ kawa/Software-License.html.

This product includes OSSP UUID software which is Copyright © 2002 Ralf S. Engelschall, Copyright © 2002 The OSSP Project Copyright © 2002 Cable & Wireless Deutschland. Permissions and limitations regarding this software are subject to terms available at http://www.opensource.org/licenses/mit-license.php.

This product includes software developed by Boost (http://www.boost.org/) or under the Boost software license. Permissions and limitations regarding this software are subject to terms available at http:/ /www.boost.org/LICENSE_1_0.txt.

This product includes software copyright © 1997-2007 University of Cambridge. Permissions and limitations regarding this software are subject to terms available at http:// www.pcre.org/license.txt.

This product includes software copyright © 2007 The Eclipse Foundation. All Rights Reserved. Permissions and limitations regarding this software are subject to terms available at http:// www.eclipse.org/org/documents/epl-v10.php and at http://www.eclipse.org/org/documents/edl-v10.php.

This product includes software licensed under the terms at http://www.tcl.tk/software/tcltk/license.html, http://www.bosrup.com/web/overlib/?License, http:// www.stlport.org/doc/ license.html, http:// asm.ow2.org/license.html, http://www.cryptix.org/LICENSE.TXT, http://hsqldb.org/web/hsqlLicense.html, http:// httpunit.sourceforge.net/doc/ license.html, http://jung.sourceforge.net/license.txt , http://www.gzip.org/zlib/zlib_license.html, http://www.openldap.org/software/release/

license.html, http://www.libssh2.org, http://slf4j.org/license.html, http://www.sente.ch/software/OpenSourceLicense.html, http://fusesource.com/downloads/license-agreements/fuse-message-broker-v-5-3- license-agreement; http://antlr.org/license.html; http://aopalliance.sourceforge.net/; http://www.bouncycastle.org/licence.html; http://www.jgraph.com/jgraphdownload.html; http://www.jcraft.com/jsch/LICENSE.txt; http://jotm.objectweb.org/bsd_license.html; . http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231; http://www.slf4j.org/license.html; http://nanoxml.sourceforge.net/orig/copyright.html; http://www.json.org/license.html; http://forge.ow2.org/projects/javaservice/, http://www.postgresql.org/about/licence.html, http://www.sqlite.org/copyright.html, http://www.tcl.tk/software/tcltk/license.html, http://www.jaxen.org/faq.html, http://www.jdom.org/docs/faq.html, http://www.slf4j.org/license.html; http://www.iodbc.org/dataspace/iodbc/wiki/iODBC/License; http://www.keplerproject.org/md5/license.html; http://www.toedter.com/en/jcalendar/license.html; http://www.edankert.com/bounce/index.html; http://www.net-snmp.org/about/license.html; http://www.openmdx.org/#FAQ; http://www.php.net/license/3_01.txt; http://srp.stanford.edu/license.txt; http://www.schneier.com/blowfish.html; http://www.jmock.org/license.html; http://xsom.java.net; http://benalman.com/about/license/; https://github.com/CreateJS/EaselJS/blob/master/src/easeljs/display/Bitmap.js; http://www.h2database.com/html/license.html#summary; http://jsoncpp.sourceforge.net/LICENSE; http://jdbc.postgresql.org/license.html; http://protobuf.googlecode.com/svn/trunk/src/google/protobuf/descriptor.proto; https://github.com/rantav/hector/blob/master/LICENSE; http://web.mit.edu/Kerberos/krb5-current/doc/mitK5license.html; http://jibx.sourceforge.net/jibx-license.html; and https://github.com/lyokato/libgeohash/blob/master/LICENSE.

This product includes software licensed under the Academic Free License (http://www.opensource.org/licenses/afl-3.0.php), the Common Development and Distribution License (http://www.opensource.org/licenses/cddl1.php) the Common Public License (http://www.opensource.org/licenses/cpl1.0.php), the Sun Binary Code License Agreement Supplemental License Terms, the BSD License (http:// www.opensource.org/licenses/bsd-license.php), the new BSD License (http://opensource.org/licenses/BSD-3-Clause), the MIT License (http://www.opensource.org/licenses/mit-license.php), the Artistic License (http://www.opensource.org/licenses/artistic-license-1.0) and the Initial Developer's Public License Version 1.0 (http://www.firebirdsql.org/en/initial-developer-s-public-license-version-1-0/).

This product includes software copyright © 2003-2006 Joe WaInes, 2006-2007 XStream Committers. All rights reserved. Permissions and limitations regarding this software are subject to terms available at http://xstream.codehaus.org/license.html. This product includes software developed by the Indiana University Extreme! Lab. For further information please visit http://www.extreme.indiana.edu/.

This product includes software Copyright (c) 2013 Frank Balluffi and Markus Moeller. All rights reserved. Permissions and limitations regarding this software are subject to terms of the MIT license.

This Software is protected by U.S. Patent Numbers 5,794,246; 6,014,670; 6,016,501; 6,029,178; 6,032,158; 6,035,307; 6,044,374; 6,092,086; 6,208,990; 6,339,775; 6,640,226; 6,789,096; 6,823,373; 6,850,947; 6,895,471; 7,117,215; 7,162,643; 7,243,110; 7,254,590; 7,281,001; 7,421,458; 7,496,588; 7,523,121; 7,584,422; 7,676,516; 7,720,842; 7,721,270; 7,774,791; 8,065,266; 8,150,803; 8,166,048; 8,166,071; 8,200,622; 8,224,873; 8,271,477; 8,327,419; 8,386,435; 8,392,460; 8,453,159; 8,458,230; and RE44,478, International Patents and other Patents Pending.

DISCLAIMER: Informatica Corporation provides this documentation "as is" without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of noninfringement, merchantability, or use for a particular purpose. Informatica Corporation does not warrant that this software or documentation is error free. The information provided in this software or documentation may include technical inaccuracies or typographical errors. The information in this software and documentation is subject to change at any time without notice.

NOTICES

This Informatica product (the "Software") includes certain drivers (the "DataDirect Drivers") from DataDirect Technologies, an operating company of Progress Software Corporation ("DataDirect") which are subject to the following terms and conditions:

1. THE DATADIRECT DRIVERS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.
2. IN NO EVENT WILL DATADIRECT OR ITS THIRD PARTY SUPPLIERS BE LIABLE TO THE END-USER CUSTOMER FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL OR OTHER DAMAGES ARISING OUT OF THE USE OF THE ODBC DRIVERS, WHETHER OR NOT INFORMED OF THE POSSIBILITIES OF DAMAGES IN ADVANCE. THESE LIMITATIONS APPLY TO ALL CAUSES OF ACTION, INCLUDING, WITHOUT LIMITATION, BREACH OF CONTRACT, BREACH OF WARRANTY, NEGLIGENCE, STRICT LIABILITY, MISREPRESENTATION AND OTHER TORTS.

Part Number: UM-PER-67100-0001

Table of Contents

Preface

The *Ultra Messaging Guide for Persistence* is written for Ultra Messaging administrators and application developers. It describes Ultra Messaging Persistence functionality. This guide assumes that you are familiar with Ultra Messaging streaming concepts.

Informatica Resources

Informatica My Support Portal

As an Informatica customer, you can access the Informatica My Support Portal at http://mysupport.informatica.com.

The site contains product information, user group information, newsletters, access to the Informatica customer support case management system (ATLAS), the Informatica How-To Library, the Informatica Knowledge Base, Informatica Product Documentation, and access to the Informatica user community.

Informatica Documentation

The Informatica Documentation team takes every effort to create accurate, usable documentation. If you have questions, comments, or ideas about this documentation, contact the Informatica Documentation team through email at infa_documentation@informatica.com. We will use your feedback to improve our documentation. Let us know if we can contact you regarding your comments.

The Documentation team updates documentation as needed. To get the latest documentation for your product, navigate to Product Documentation from http://mysupport.informatica.com.

Informatica Web Site

You can access the Informatica corporate web site at http://www.informatica.com. The site contains information about Informatica, its background, upcoming events, and sales offices. You will also find product and partner information. The services area of the site includes important information about technical support, training and education, and implementation services.

Informatica How-To Library

As an Informatica customer, you can access the Informatica How-To Library at http://mysupport.informatica.com. The How-To Library is a collection of resources to help you learn more about Informatica products and features. It includes articles and interactive demonstrations that provide solutions to common problems, compare features and behaviors, and guide you through performing specific real-world tasks.

Informatica Knowledge Base

As an Informatica customer, you can access the Informatica Knowledge Base at http://mysupport.informatica.com. Use the Knowledge Base to search for documented solutions to known technical issues about Informatica products. You can also find answers to frequently asked questions, technical white papers, and technical tips. If you have questions, comments, or ideas about the Knowledge Base, contact the Informatica Knowledge Base team through email at KB_Feedback@informatica.com.

Informatica Support YouTube Channel

You can access the Informatica Support YouTube channel at http://www.youtube.com/user/INFASupport. The Informatica Support YouTube channel includes videos about solutions that guide you through performing specific tasks. If you have questions, comments, or ideas about the Informatica Support YouTube channel, contact the Support YouTube team through email at supportvideos@informatica.com or send a tweet to @INFASupport.

Informatica Marketplace

The Informatica Marketplace is a forum where developers and partners can share solutions that augment, extend, or enhance data integration implementations. By leveraging any of the hundreds of solutions available on the Marketplace, you can improve your productivity and speed up time to implementation on your projects. You can access Informatica Marketplace at http://www.informaticamarketplace.com.

Informatica Velocity

You can access Informatica Velocity at http://mysupport.informatica.com. Developed from the real-world experience of hundreds of data management projects, Informatica Velocity represents the collective knowledge of our consultants who have worked with organizations from around the world to plan, develop, deploy, and maintain successful data management solutions. If you have questions, comments, or ideas about Informatica Velocity, contact Informatica Professional Services at ips@informatica.com.

Informatica Global Customer Support

You can contact a Customer Support Center by telephone or through the Online Support.

Online Support requires a user name and password. You can request a user name and password at http://mysupport.informatica.com.

The telephone numbers for Informatica Global Customer Support are available from the Informatica web site at http://www.informatica.com/us/services-and-training/support-services/global-support-centers/.

CHAPTER 1

Introduction

This chapter includes the following topics:

Overview

In addition to high performance streaming, Ultra Messaging also provides persistence by implementing a configurable daemon that runs persistent stores.

Persistence

A system implementing UMP persistence comprises any number of sources, receivers, and persistent stores. Ultra Messaging's unique design provides Parallel Persistence, which refers to the ability of a persistent store or stores to run independently of sources and receivers and in parallel with messaging. The persistence store does not interfere with message delivery to receiving applications. Parallel Persistence adds several key features missing in other messaging solutions.

- A fault recovery ability
- The capacity to continue operation during specific types of failures

Fault recovery refers to the system's ability to recover from a failure of any system component (source, receiver or store). Under certain circumstances, Ultra Messaging can even recover from multiple failures and multiple cascading failures.

CHAPTER 2

Concepts

This chapter includes the following topics:

Overview

This section illuminates important UMP concepts and features.

Contained in UMP are all of the features and capabilities of Ultra Messaging's high performance, message streaming. This document explains persistence capabilities only. For specific information about Ultra Messaging's high performance streaming, see the *Ultra Messaging Concepts Guide*.

Also available to UMP is the Ultra Messaging Manager. UMM provides a GUI that simplifies the creation of UM XML configuration files and also allows you to assign application configurations to specific users, also created in the UMM GUI. The UMM Daemon runs this feature, offering a UMM GUI API to support custom GUIs and uses a MySQL database to store configurations. See the *Ultra Messaging Manager Guide*.

Persistence

In discussing UMP , we refer to specific recovery from the failures of sources, receivers, and persistent stores. Failed sources can restart and resume sending data from the point at which they stopped. Receivers can recover from failure and begin receiving data from the point immediately prior to failure. This process is sometimes called durable subscription. Persistent stores can also be restarted and continue providing persistence to the sources and receivers that they serve. UMP is not designed to address ongoing, corrupting agents. Rather, if one of its components fails, the design of UMP 's persistence enables it to continue supporting its ongoing operations at some level.

UMP offers persistence in the following two modes.

- **Source-paced Persistence (SPP)** - In the default mode, the consumption of messages by receivers does not impact the rate a UM source can send messages. Sources send messages simultaneously to receivers and the persistent store. (See "Normal Operation" on page 17.)

- **Receiver-paced Persistence (RPP)** - In RPP, sources also send messages to receivers and the persistent store in parallel, but the store retains RPP messages until all RPP receivers acknowledge consumption. If the store fills up with messages awaiting acknowledgment from all registered RPP receivers, the store stops accepting new messages from the source until RPP receivers acknowledge older messages and space becomes available. Late Join is not avaliable with RPP. (See “RPP Normal Operations” on page 23.)

Persistent Store

UMP uses a daemon to persist source and receiver state outside the actual sources and receivers themselves. This is the UMP Persistent Store. The store can persist state in memory as well as on disk. State is persisted on a per-topic, per-source basis by the store. UMP stores need not be a single entity. For fault tolerance purposes, it is possible to configure multiple stores in various ways. See “Adding the UMP Store to a Source” on page 36, “ UMP Stores” on page 69, “ Store Configuration Considerations” on page 76, Chapter 9, “Man Pages” on page 83 and Chapter 10, “Configuration Reference for Umestored” on page 86.

Registration Identifier

UMP identifies sources and receivers with Registration Identifiers, also called Registration IDs or RegIDs. A RegID is a 32-bit number that uniquely identifies a source or a receiver to a store. This means that RegIDs are also specific to a store and can be reused between individual stores, if needed. No two active sources or receivers can share a RegID or use the same RegID at the same time. This point is critical: since UMP enables your application to use and handle RegIDs very freely, you must use RegIDs carefully to avoid destructive results. See “Adding Fault Recovery with Registration IDs” on page 37 and “Registration Identifiers” on page 46. RegIDs can also be managed easily with the use of Session IDs. See “Managing RegIDs with Session IDs” on page 47.

Delivery Confirmation

UMP provides feedback to sources upon notification that a receiver has consumed a given piece of data, in other words, that it has received and processed a message. This feedback is called Delivery Confirmation. See also “Confirmed Delivery” on page 52. For information about source retention policies, see “ Source Message Retention and Release” on page 50.

Release Policy

Sources and persistent stores retain data according to a release policy, which is a set of rules that specifies when a message can be reclaimed. Each rule would allow any message that complies with the rule to be reclaimed. However, a message must comply with all rules before it can be reclaimed. Conversely, any message not complying with all rules will not be reclaimed. A source or store retains messages until its retention policy dictates the message may be removed. Sources and stores use slightly different retention policies based on their individual roles. For more information on retention policies, see “ Source Message Retention and Release” on page 50.

Message Stability

Sources send messages to both receivers and to stores. Messages become stable once the message has been persisted at the store or a set of stores. The number of messages that can be sent by a source has no relation to the number of its messages that have been stabilized unless “ UMP Flight Size” on page 18 is enabled. In addition, UMP informs the application when messages are stabilized, enabling the application to take any desired action. See “ Source Message Retention and Release” on page 50.

Round-Robin Store Failover

Stores can also experience failures from which they may or may not recover. A source can be configured to move to a second store if the first store fails and can not recover in time. Round-robin store behavior describes the behavior of a source moving through a list of stores, using a single store at any one time, with several specified backups available to it in case the single store fails.

See also “ Sources Using Round-Robin Store Configuration” on page 70 and “Round-Robin Store Usage” on page 70

Note: The Round-Robin Store failover configuration is deprecated.

Quorum/Consensus Store Failover

In addition to a source being configured for round-robin store behavior, several stores can be configured for simultaneous operation. In this situation, a single store or even a handful of stores can fail without impacting the source and receivers. As long as a quorum of the configured stores is accessible, messaging operation generally continues uninterrupted. (UMP defines a Quorum as a majority.)

CHAPTER 3

Architecture

This chapter includes the following topics:

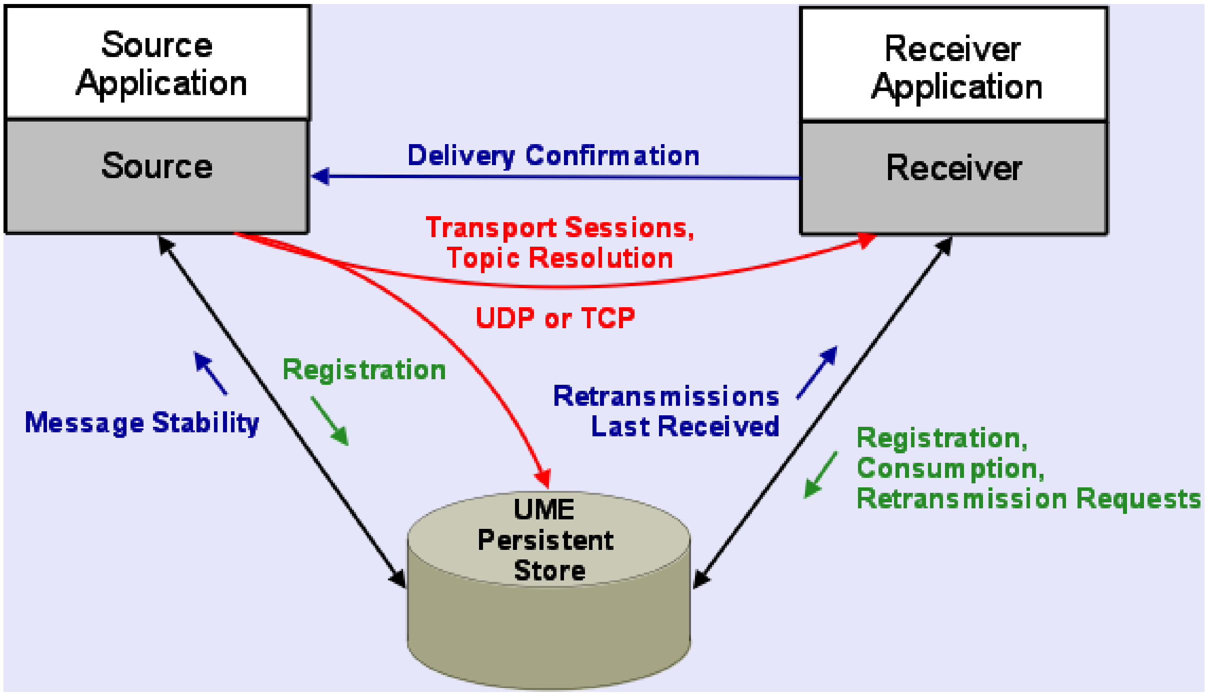
Overview

The same Ultra Messaging API may be used for stream-based messaging or persistent messaging. The `umestored` daemon can be configured as a persistent store providing consistent and efficient operation across persistent messaging systems.

Persistence Architecture

As shown in the diagram, UMP provides messaging functionality as well as persistent operation. See Figure 1 on page 6 for an overview of UMP architecture.

Figure 1. Persistence Architecture



The highlights of this architecture are:

- Sources communicate with stores
- Receivers communicate with stores
- Sources communicate with receivers

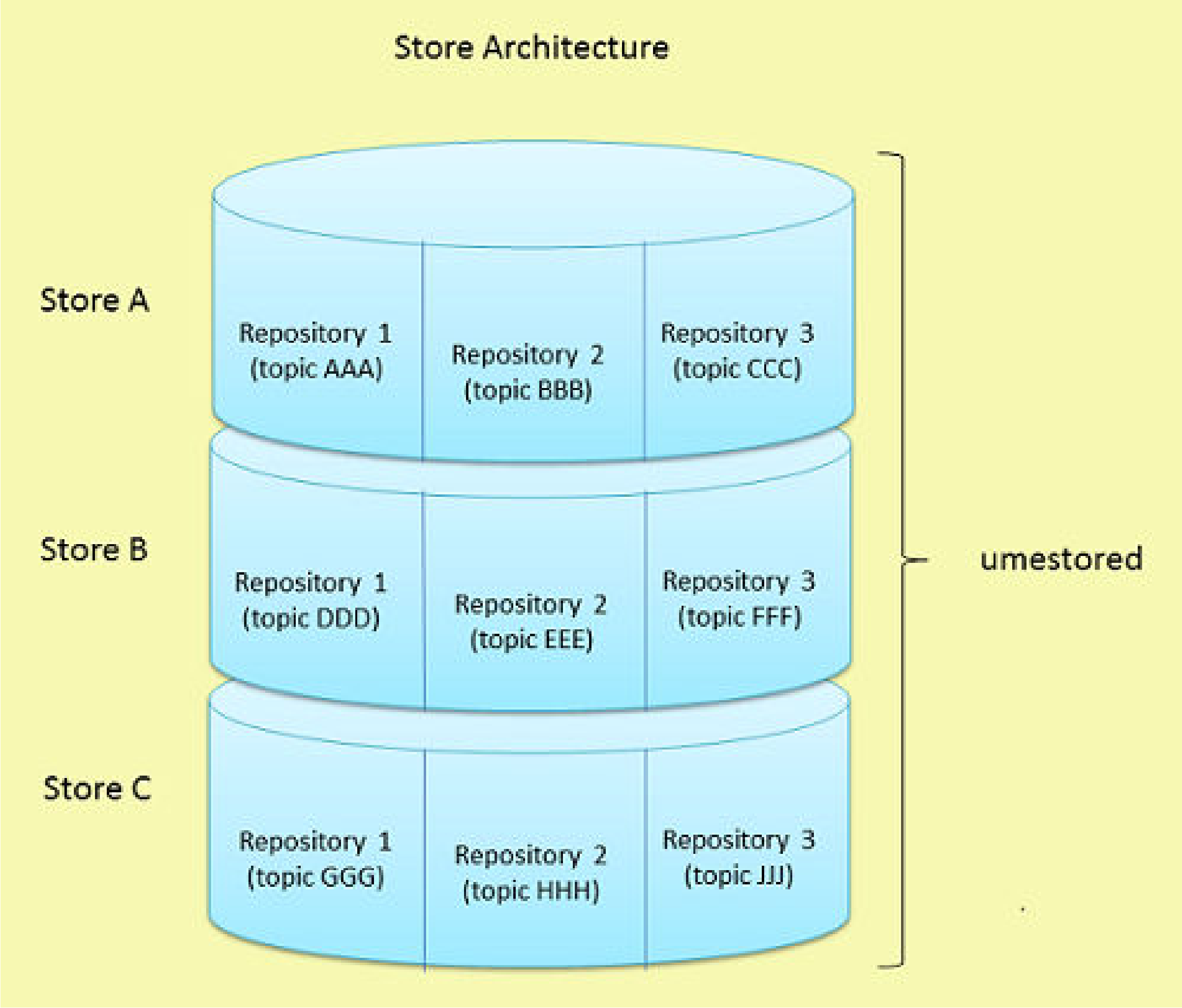
Note: The persistent store does not lie in the middle of the data path between source and receivers. Along with other enhancements, this feature, called Parallel Persistence, gives UMP a significant performance edge over any other persistent messaging product.

Restriction: The persistent store is not supported on the OpenVMS® platform. UM applications running on the OpenVMS® platform, however, can access a persistent store running on a different platform, such as Microsoft Windows® or Linux.

Persistent Store Architecture

The `umestored` daemon runs the UMP persistent store feature. You can configure multiple stores per daemon using the `<store>` element in the `umestored` XML configuration file. See Chapter 10, “Configuration Reference for Umestored” on page 86. Individual stores can use separate disk cache and disk state directories and be configured to persist messages for multiple sources (topics), which are referred to as, source repositories. UMP provides each `umestored` daemon with a Web Monitor for statistics monitoring. See Chapter 11, " Ultra Messaging Web Monitor" on page 98.

Figure 2. Store Architecture



This section discusses the following topics.

- "Source Repositories" on page 7
- "Persistent Store Fault Tolerance" on page 9

Source Repositories

Within a store, you configure repositories for individual topics and each can have their own set of `<topic>` level options that affect the repository's type, size, liveness behavior and much more. If you have multiple sources sending on the same topic, UMP creates a separate repository for each source. UMP uses the repository options configured for the topic to apply to each source's repository. If you specify 48MB for the size of the repository and have 10 sources sending on the topic, the persistent store requires 480MB of storage for that topic.

A repository can be configured as one of the following types.

- `no cache` - the repository does not retain any data, only state information

- `memory` - the repository maintain both state and data only in memory
- `disk` - the repository maintains state and data on disk, but also uses a memory cache.
- `reduced-fd` - the repository maintains state and data on disk, also uses a memory cache but uses significantly fewer File Descriptors. Normally a store uses two File Descriptors per topic in addition to normal UM file descriptors for transports and other objects. The reduced-fd repository type uses 5 File Descriptors for the entire store, regardless of the number of topics, in addition to normal UM file descriptors for transports and other objects. Use of this repository type may impact performance.

You can configure any combination of repository types within a single store configuration.

Note: If you run a store with all `disk` or `reduced-fd` type repositories, then restart the store with memory type repositories and do not clear out the `disk-cache-directory` and `disk-state-directory`, the memory repositories revert automatically to disk repositories.

Note: With UMP Version 5.3, the UMP store daemon has Standard C++ Library dependencies for Unix packages. The `libstdc++` must also be included in LD_LIBRARY_PATH. See *Section 3. Code* for more infromation.

Repository Thresholds and Limits

Repositories are designed as circular buffers. When age or size thresholds are met for a topic, the repository removes or overwrites messages in order to prevent reaching its configured limit, which keeps space available for new messages. UMP provides UM configuration options and store configuration options to control threshold and limit behavior.

UM configuration options control source repositories for all the sources sending within the context. The default for these options, listed below, are 0 (zero) which makes the like-name option for the repository in the `umestored` XML configuration file active.

- *`ume_repository_disk_file_size_limit`* (source)
- *`ume_repository_size_limit`* (source)
- *`ume_repository_size_threshold`* (source)

See *Ultra Messaging Persistence Options* in the *UM Configuration Guide*.

Note: The above configuration options can be altered for individual source by calling **lbm_src_topic_attr_setopt()** before you allocate the topic.

The `umestored` configuration options for source/topic repositories explained below can also be used to control threshold and limit behavior. See "Topic Element" on page 90 for complete information about the following repository options.

Important: Whether you use the UM configuration options mentioned above or the source repository options explained below to control source repository threshold and limit behavior, remember the values you configure apply to a single source sending to the store. If you use the default repository size limit of 48 MB and you have 1,000 sources sending to the store, UMP creates a store with 1,000 source repositories of 48 MB each, which requires a store with approximately 48 GB of memory. And if you use the default disk file size limit of 100 MB and you have 1,000 sources sending to the store, UMP creates a store with 1,000 source repositories of 100 MB each, which requires a store with disk storage capacity of approximately 100 GB.

Memory Repository

A `memory` type source repository has three configuration options that manage its size relative to its capacity.

- `repository-age-threshold` - This value determines how long the repository retains messages. The repository deletes any message older than this configured value.

- `repository-size-threshold` - The size in bytes that a repository can reach before it begins to delete the oldest retained messages. If the repository size falls below the threshold, it stops deleting old messages.
- `repository-size-limit` - The maximum size in bytes for the repository. Once this limit is reached, the repository stops accepting new messages. The age and size thresholds should be set at levels that guarantee the size limit is never met. You should consider how fast the source sends messages, the size of the messages and the reliability of the receivers. For example, more reliable receivers mean less recovery instances, which could mean a younger age threshold.

Disk or Reduced-fd Repositories

A `disk` or `reduced-fd` type source repository maintains a memory cache in addition to the actual disk storage. It continually persists messages from the memory cache to the disk, and uses the memory cache for receiver recovery first before performing disk reads to access needed messages. It has four configuration options that manage its size relative to its capacity.

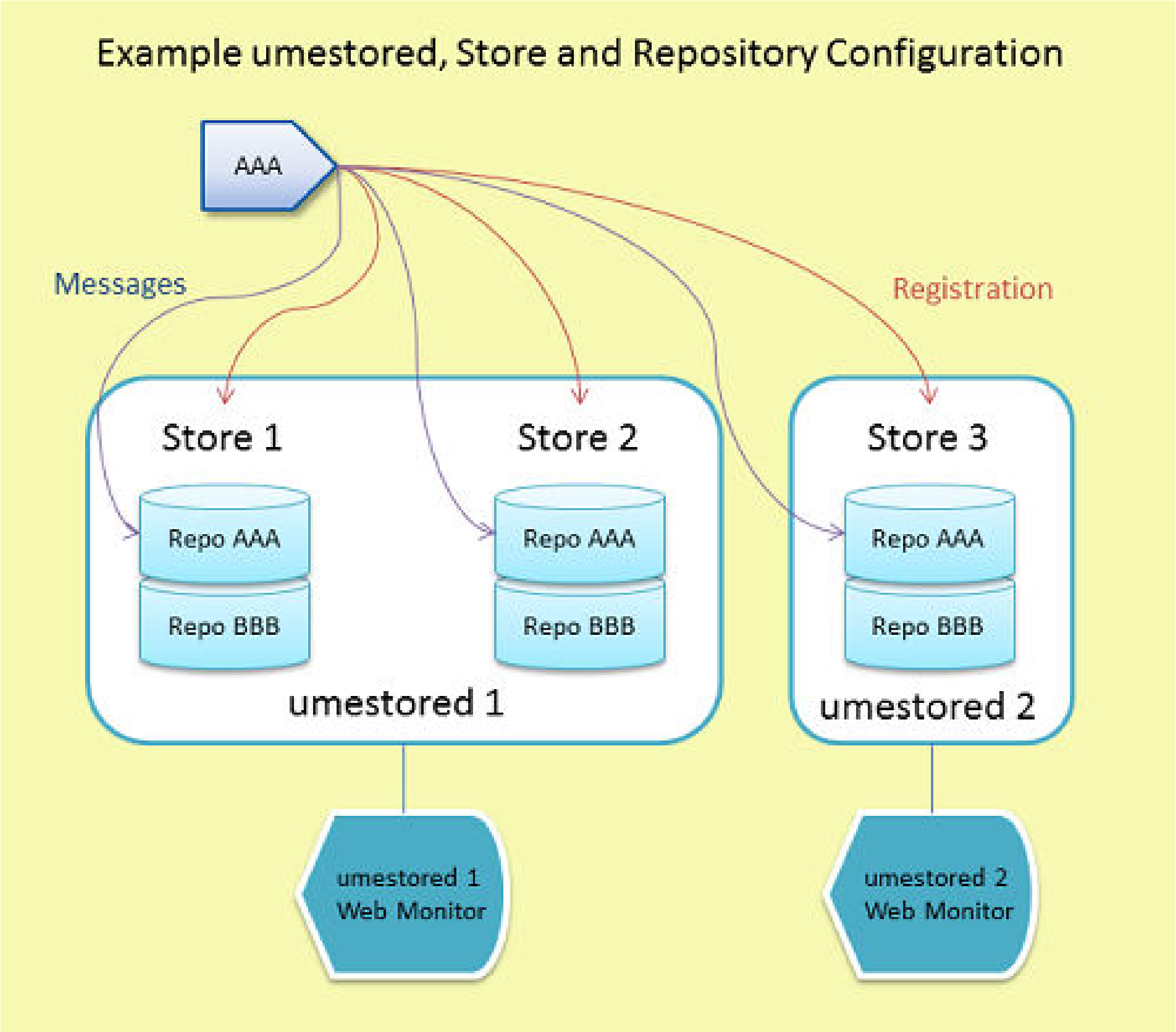
- `repository-age-threshold` - This value determines how long the disk repository retains messages in its memory cache. The repository deletes any message from memory cache older than this configured value. These messages could have been persisted to disk and may be available for recovery.
- `repository-size-threshold` - The size in bytes that a repository can reach before it begins to delete the oldest retained messages. These messages could have been persisted to disk and may be available for recovery. If the disk repository memory cache size falls below the threshold, it stops deleting old messages.
- `repository-size-limit` - The maximum size in bytes for the disk repository's memory cache. Once this limit is reached, the repository stops accepting new messages. The age and size thresholds should be set at levels that guarantee the size limit is never met. You should consider how fast the source sends messages, the size of the messages and the reliability of the receivers. For example, more reliable receivers mean less recovery instances, which could mean a younger age threshold.
- `repository-disk-file-size-limit` - The maximum disk space (in bytes) for the disk repository. Once this limit is reached, the repository overwrites old messages with new messages. Overwriting old messages is not necessarily a negative situation provided you disk file size is adequate. However, if messages needed for recovery are not in either the memory cache or the disk file, you may need to increase the disk file size to ensure that overwritten messages are no longer needed for receiver recovery.

Persistent Store Fault Tolerance

Sources and receivers register with a store and use individual repositories within the store. Sources can use redundant repositories configured in multiple stores in either a Round Robin or Quorum/Consensus arrangement for fault tolerance. Stores and repositories have no indication of these arrangements.

The following diagram depicts an example Quorum/Consensus configuration of stores and repositories. These stores could also be run by a single `umestored` daemon or one daemon for each store.

Figure 3. Example Store Configuration



See “ Store Configuration Considerations” on page 76 and also “Stores Element” on page 88 for more about store configuration.

Identifying Persistent Stores

You can identify stores with either a `domainID:interface:port`, `interface:port` or a name. Using only `interface:port` is more feasible in smaller implementations where the smaller number of possible IP addresses is easier to manage. Larger implementations, especially those that span topic resolution domains using UM Routers, are better served with stores identified by a name or `domainID:interface:port`.

UM automatically resolves and maintains a mapping between a store name and a single topic resolution domain, IP address and port. UM also automatically resolves store names if the store is located across one or more UM Routers in a different topic resolution domain.

The following lists other specifics of store identification.

- Store sends ads at startup and in response to queries from sources.
- If a store receives a context name advertisement that matches its own store name, umestored issues a warning in the store's log.
- Sources using named stores issue an information message to the application every time a resolved context name changes its `DomainID:IPaddress:port`.

Using a Single Interface and Port

Configure store for a single interface and port.

1. Identify the store with only the `interface:port`, specified in `umestored` configuration file.

   ```
   <store name="newyork-1" port="14567" interface="10.29.3.16">
   ```

2. Add the `interface:port` to *`(source) ume_store`* so sources can find and register with the store.

   ```
   source ume_store 10.29.3.16:14567
   ```

To run the store on a different machine for any reason, you must change both the `umestored` XML configuration file and the UM configuration file.

Using a Range of Interfaces

Configure a store with a range of IP addresses.

1. Identify the store with a range of interfaces specified in the `umestored` configuration file.

   ```
   <store name="newyork-1" port="14567" interface="10.29.3.16/25">
   ```

2. Add the ***active*** interface to *`(source) ume_store`* so sources can find and register with the store. You can only specify one interface in the configuration file.

   ```
   source ume_store 10.29.3.16:14567
   ```

To run the store on a different machine, you must only change the interface specified in the *`(source) ume_store`* UM configuration option, provided you use one of the interfaces in the range specified in in the `umestored` configuration file.

Using a Store (context) Name

Configure a store with a name instead of just IP:port.

"0.0.0.0 (INADDR_ANY)" or no value is the default for the store's `interface` attribute.

1. Identify the store with a `context-name` that resolves to the interface and port - or range of interfaces and port - specified in the `umestored` configuration file

   ```
   <store name="newyork-1" port="14567" interface="0.0.0.0">
   <ume-attributes>
      <option name="context-name" type="store" value="NEWYORK-1"/>
   </ume-attributes>

   OR

   <store name="newyork-1" port="14567" interface="10.29.3.16">
   <ume-attributes>
      <option name="context-name" type="store" value="NEWYORK-1"/>
   </ume-attributes>

   OR

   <store name="newyork-1" port="14567" interface="10.29.3.16/25">
   <ume-attributes>
      <option name="context-name" type="store" value="NEWYORK-1"/>
   </ume-attributes>
   ```

2. Add the store's `context-name` to *`(source) ume_store_name`* so sources can find and register with the store.

   ```
   source ume_store_name NEWYORK-1
   ```

You do not have to make any configuration changes to run NEWYORK-1 on another machine, provided the new interface matches one of those specified in the umestored configuration file. This includes running the store in a different topic resolution domain.

CHAPTER 4

Operational View

This chapter includes the following topics:

Introduction

If your application is running with the UM configuration option, *`request_tcp_bind_request_port`* set to zero, request port binding has been turned off, which also disables UMP .

Persistence Operations

Sources, receivers, and stores in UMP interact in very controlled ways. This section illustrates the flow of network traffic between the components during three modes of operation and also provides a reference of UMP Events.

- "Source Registration" on page 13
- "Normal Operation" on page 17
- "Receiver Recovery" on page 19

Source Registration

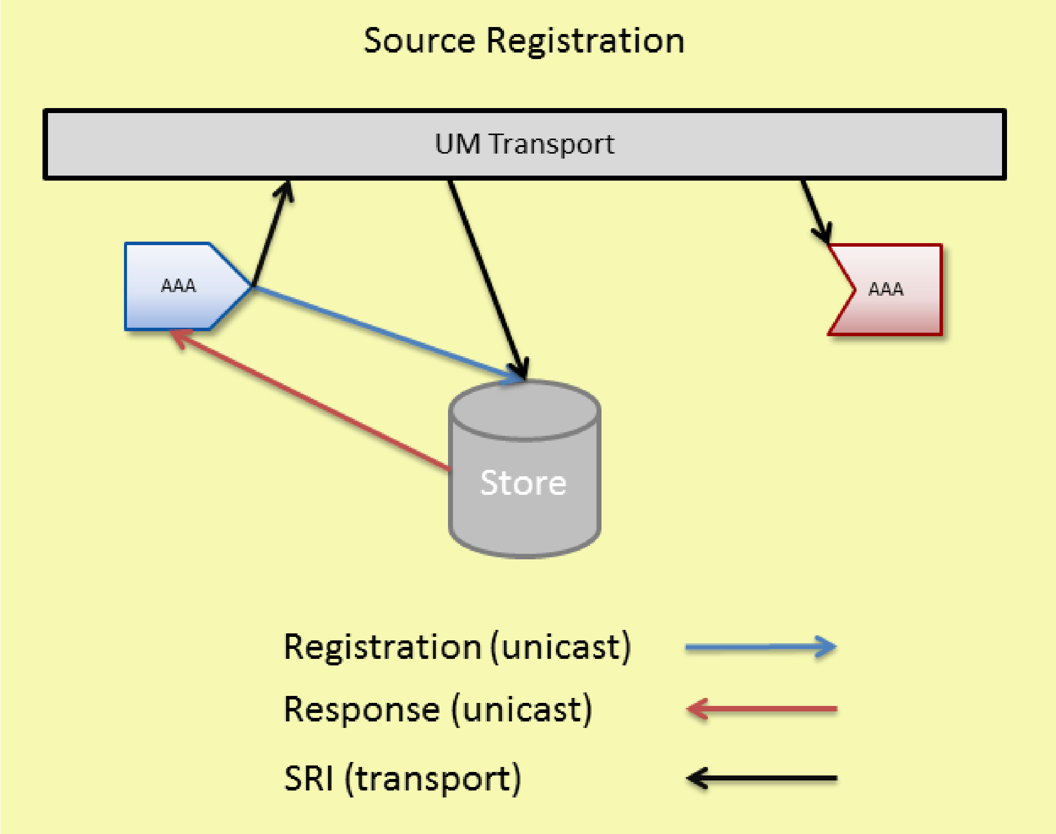
UM sources heavily influence the UMP registration process. Sources send out registration information to enable receivers to register with stores and also monitor store liveness. If stores become unresponsive, or if communication among sources, stores and receivers becomes impaired, the source directs re-registration.

The following outlines the major events in the source registration process with the store.

1. Source advertises topic over topic resolution transport
2. (optional) Source queries for and resolves store name
3. Source registers with store by unicast
4. Source sends SRI over configured transport

The following diagram illustrates network flow during the registration process.

Figure 4. Source Registration Process



Sources can find the correct store(s) to register with from the values configured for it in *`ume_store`* or *`ume_store_name`*. The configuration option *`ume_store`* contains the IP address, TCP port, registration ID, and group index for the store(s) to be used by the source. The configuration option *`ume_store_name`* contains the names of the stores to be used by the source. *`ume_store_name`* requires that the store name is configured with the context-name attribute in the store's XML configuration file. See "Identifying Persistent Stores" on page 10 and the "Store Element" on page 88.

Sources unicast registrations to the store. The store unicasts responses back to the source. Registrations are on a per topic per source basis. Stores use RegIDs to identify sources and receivers. After registration sources may send data.

After the source successfully registers with all the stores for which it is configured, the source issues a Registration Complete event and sends a Source Registration Information (SRI) record over the configured UM transport session.

For multiple stores, the source determines when to issue a Registration Complete event based on the settings for the `ume_retention_intragroup_stability_behavior` and `ume_retention_intergroup_stability_behavior` options.

The source sends the SRI at the rate set by `ume_sri_inter_sri_interval` until it reaches the maximum number of SRIs set by `ume_sri_max_number_of_sri_per_update`.

Source Registration Information (SRI)

Packet sent over the UM transport by a source that contains store information that a receiver needs to register with the store.

An SRI contains the following store information.

- Domain ID
- IP address
- TCP port
- store index for all the stores with which the source registered
- group index for all the stores with which the source registered
- the source's Registration ID
- SRI overall version number and a separate version number for each store

The SRI contains one overall version number and a separate version number for each store. If stores become unresponsive and the source must re-register when the store returns, the source increases the SRI version number and the version numbers for the stores it re-registered with. The highest SRI version number indicates the most current registration information. If a receiver gets an SRI with a higher version number than the version number it has, the receiver examines the individual store version numbers and re-registers with the those stores that have higher individual version numbers.

Receiver Registration

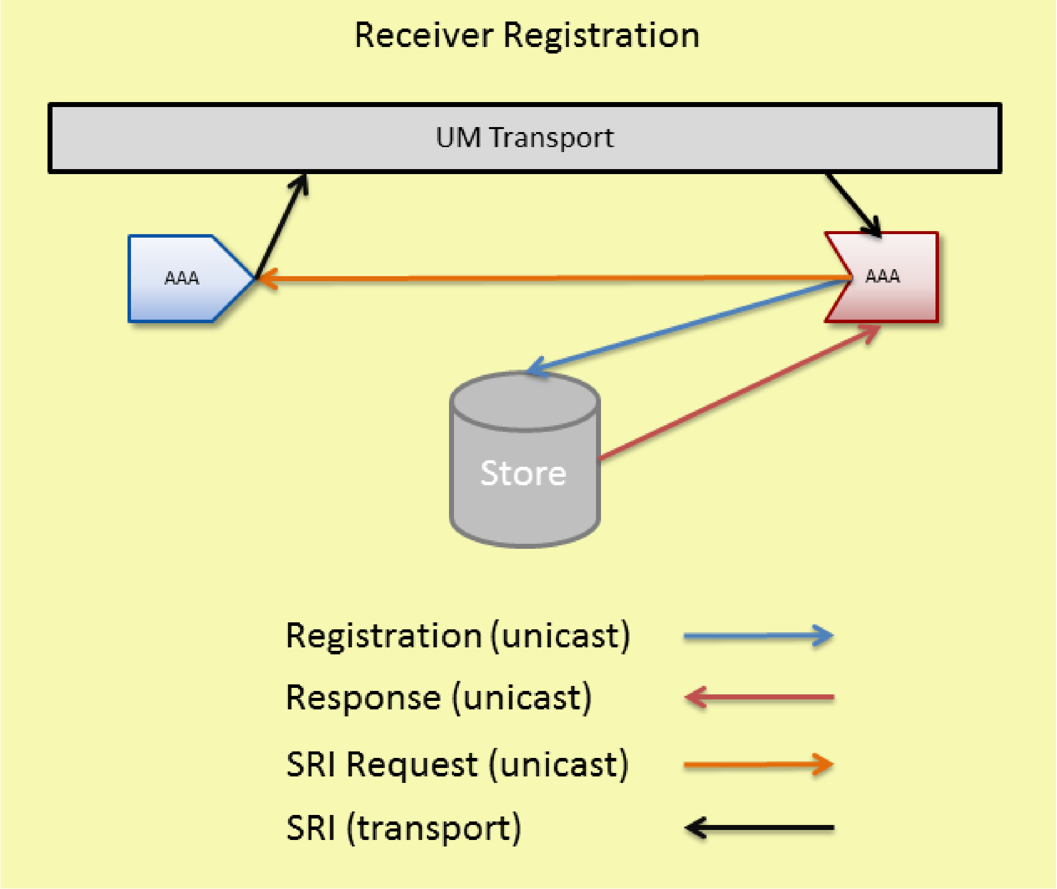
Receivers register with a store or stores after receiving a SRI packet from the source sending on the receiver's topic.

Receiver must receive an SRI before they can register with the store or stores. The following lists the major events in the receiver registration process.

1. Receiver resolves topic over topic resolution transport.
2. If source is not sending SRIs, receiver sends SRI request by unicast.
3. Receiver receives SRI over its transport.
4. Receiver registers with store(s) by unicast.

The following diagram illustrates network flow during the registration process.

Figure 5. Receiver Registration Process



Any receivers who have resolved their topic and joined the transport session when the source sends out SRIs can register with the store. Any receivers joining the transport session when the source is not sending SRIs can request an SRI from the source if they find that the UMP flag is set in the source's TIR during topic resolution. The source responds with a SRI record.

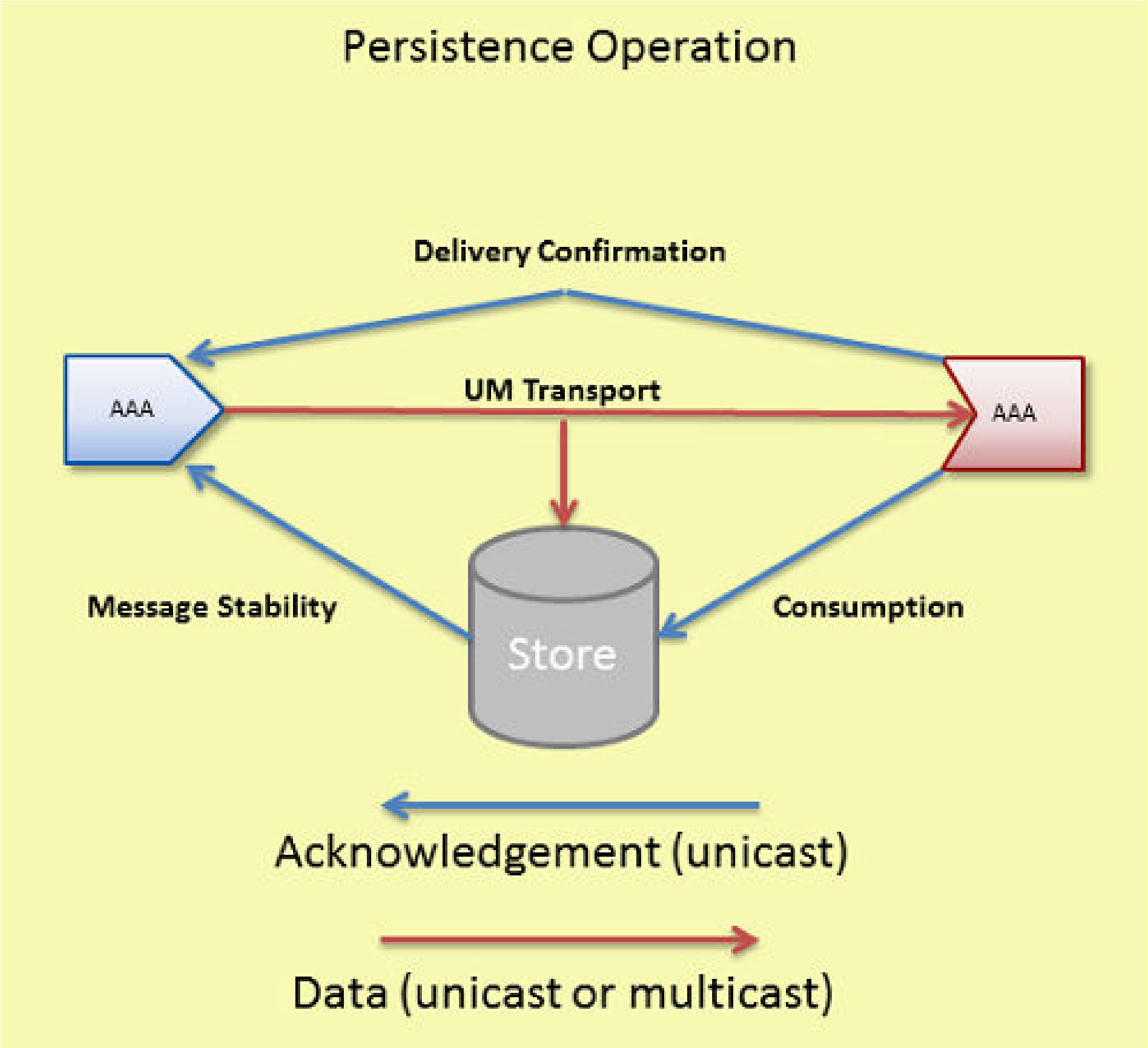
Receivers unicast registrations to the store. The store unicasts responses back to the receivers. Stores use RegIDs to identify sources and receivers. After registration, receivers may handle recovery and send acknowledgements.

Note: If a UMP receiver's initial registration fails, it does not become an Ultra Messaging receiver.

Normal Operation

Figure 6 on page 17 illustrates the normal operation of data reception and acknowledgement and also shows how UMP attains Parallel Persistence. The source sends message data to receivers and stores in parallel.

Figure 6. Normal Operation



1. Sources transmit data to receivers and stores at the same time over UM multicast or unicast transport protocols.
2. As the store receives and persists messages, the store unicasts acknowledgements, (message stability control messages), to the source letting it know of successful reception and storage.
3. As receivers process and consume messages they unicast acknowledgments to the store letting the store know of successful consumption of data.
4. If the source desires delivery confirmation, the receiver unicasts acknowledgements directly to the source letting the source know of message consumption as well.

Normal operation and recovery can proceed at the same time. In addition, as a receiver consumes retransmitted messages, the receiver sends normal acknowledgements for consumption and confirmed delivery (if requested by the source).

Note: A store can be configured with different storage limits for each repository. If the repository reaches this limit, the repository releases the oldest message in order to persist a new message. This behavior occurs for a memory repository as well as a disk repository. If a repository releases a message that one or more receivers have not consumed (sent a consumption notification), the repository logs a single warning message in the store log file per receiver per registration.

UMP Flight Size

UMP supports a flight size mechanism that tracks messages in flight from a particular source and responds when a send would exceed the configured flight size (*ume_flight_size* and/or *ume_flight_size_bytes*). You can configure *ume_flight_size_behavior* to either:

- block any sends that would exceed the flight size or,
- allow the sends while notifying your application.

UMP considers a sent message in flight until the following two conditions are met.

1. The source receives the configured number of stability acknowledgements from the store(s).
2. The source has received the configured number of delivery confirmation notifications. (See *ume_retention_unique_confirmations* .)

If configuring both *ume_flight_size* and *ume_flight_size_behavior* , UMP uses the smaller of the two flight sizes on a per send basis.

ume_flight_size	ume_flight_size_bytes	Result
Exceeded	Exceeded	ume_flight_size_behavior executes
Exceeded	Not Exceeded	ume_flight_size_behavior executes
Not Exceeded	Exceeded	ume_flight_size_behavior executes
Not Exceeded	Not Exceeded	No flight size sending restriction

When using stores in a Quorum/Consensus configuration, *intragroup* and *intergroup* stability settings affect whether UMP considers a messages in flight. Consider a case with three stores in a single QC group, and two receivers. Given the default configuration, until a source receives a stability notification from two of the three stores, UMP considers a given message in-flight. In addition, if you set *ume_retention_unique_confirmations* to 2, that same message would be considered in flight until the source receives two stability notifications AND two delivery confirmation notifications. See also “ Sources Using Quorum/Consensus Store Configuration” on page 71.

Note: The UMP flight size mechanism operates on a per message basis, not a per fragment basis.

Note: The UMP flight size bytes mechanism operates with only payload data. UM or network overhead is not included in the byte count.

Blocking Message Sends That Exceed the Flight Size

By default, when a source sends a message that exceeds it's flight size, the call to send blocks. For example, suppose the flight size is set to 1. The first send completes but before the source receives a stability notification or delivery confirmation, it initiates a second call to send. If the source uses a blocking send, the send call blocks until the first message stabilizes. If the source uses a non-blocking send, the send returns an LBM_EWOULD_BLOCK.

Notification of Message Sends That Exceed the Flight Size

Alternatively, *`ume_flight_size_behavior`* can be set to notify your application when a message send surpasses the flight size. A send that exceeds the configured flight size succeeds and also triggers a flight size notification, indicating that the flight size has been surpassed. Once the number of in-flight messages falls below the configured flight size, another flight size notification source event is triggered, this time, informing the application that the number of in-flight messages is below the source's flight size.

Receiver Recovery

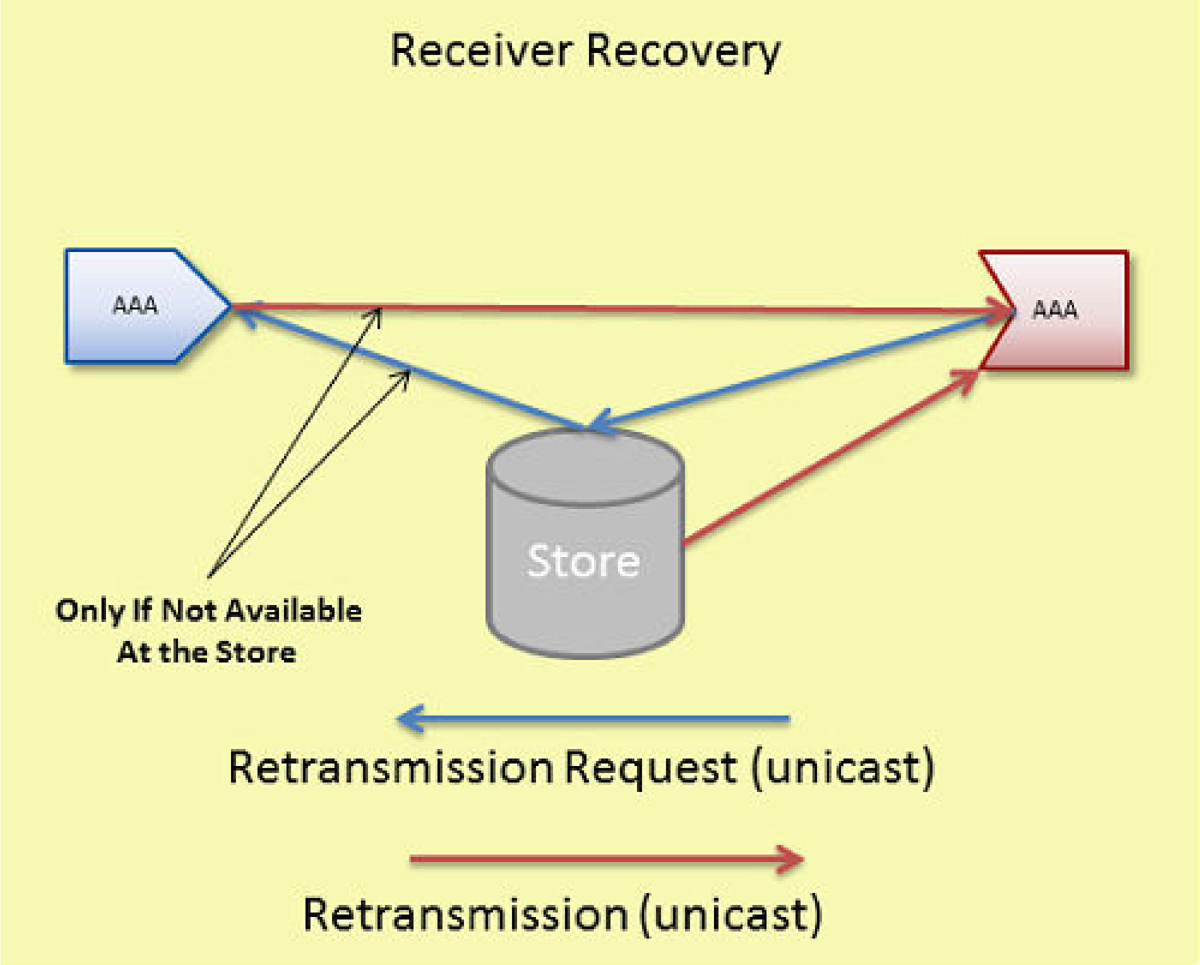
Normal loss retransmission over the UM transport operates identically in UMP as it does in UMS. The source retransmits messages in response to NAKs from the receiver. Stores do not participate in this transport-level loss retransmissions.

The most elementary method of non-transport message recovery requires the receiver to restart and re-register with the store. Through re-registration, the receiver discovers the lowest message sequence number it did not receive, and subsequently requests retransmissions of all messages not received, starting from this low sequence number. For more on this process see, “Receiver Recovery” on page 73

Note: Another method of recovery, Off Transport Recovery (OTR), does not require the receiver to restart if it detects a gap in the topic message sequence numbers it has received. See the *UM Concepts Guide, UMS Features, Off Transport Recovery (OTR)* for more information. For more reliable operation, Informatica recommends enabling OTR with UMP, especially when using UM Routers.

The following figure illustrates receiver recovery:

Figure 7. Message Recovery



Receivers unicast retransmission requests. If the store has the message, it unicasts the retransmission to the receiver. If it does not have the message and is configured to forward the request to the source , it unicasts the retransmission request to the source. If the source has the message, it unicasts the retransmission directly to the receiver. See also "UMP Message Loss Recovery" on page 74

UM store sends retransmissions from a thread separate from the main context thread so as not to impede live message data processing. The `<store>` configuration option, `retransmission-request-processing-rate`, sets the store's capacity to process retransmission requests. The retransmission thread processes requests off a retransmission queue which is set at 4 times the size of `retransmission-request-processing-rate`. The following UM Web Monitor statistics record retransmission activity. See UM Web Monitor "Store Page" on page 100.

- Retransmission requests received rate
- Retransmission requests served rate
- Retransmission requests dropped rate
- Total retransmission requests dropped since store startup

Receiver-paced Persistence Operations

Receiver-paced Persistence (RPP) refers to different message retention behavior for designated receivers. You enable RPP with UM configuration options. No special API calls are needed. RPP differs from UMP's default source-paced persistence in the following ways.

- The repository must be configured to allow RPP and sources and receivers must be configured to request RPP behavior during registration.
- Sources can modify specific repository configuration options that pertain to RPP.
- The repository retains RPP messages until all RPP receivers acknowledge consumption. The repository maintains an accurate count of all RPP receivers.
- Late Joining receivers cannot receive all previously sent topic messages, only those unconsumed by all RPP receivers. Late Joining receivers can always start at the current message retained by the repository, defined as the earliest message not consumed by all RPP receivers.
- Sources must also configure their flight size in bytes, and optionally, in message count. By using a total bytes flight size, the store can keep track of exactly how must space it has available and not send stability acknowledgements if new messages would exceed the available space, which would endanger the receipt of all messages by all RPP receivers. See “ UMP Flight Size” on page 18.

In addition, a disk write delay interval for the repository, available for Source-paced Persistence as well, improves performance by preventing unnecessary disk activity.

This section discusses the following topics.

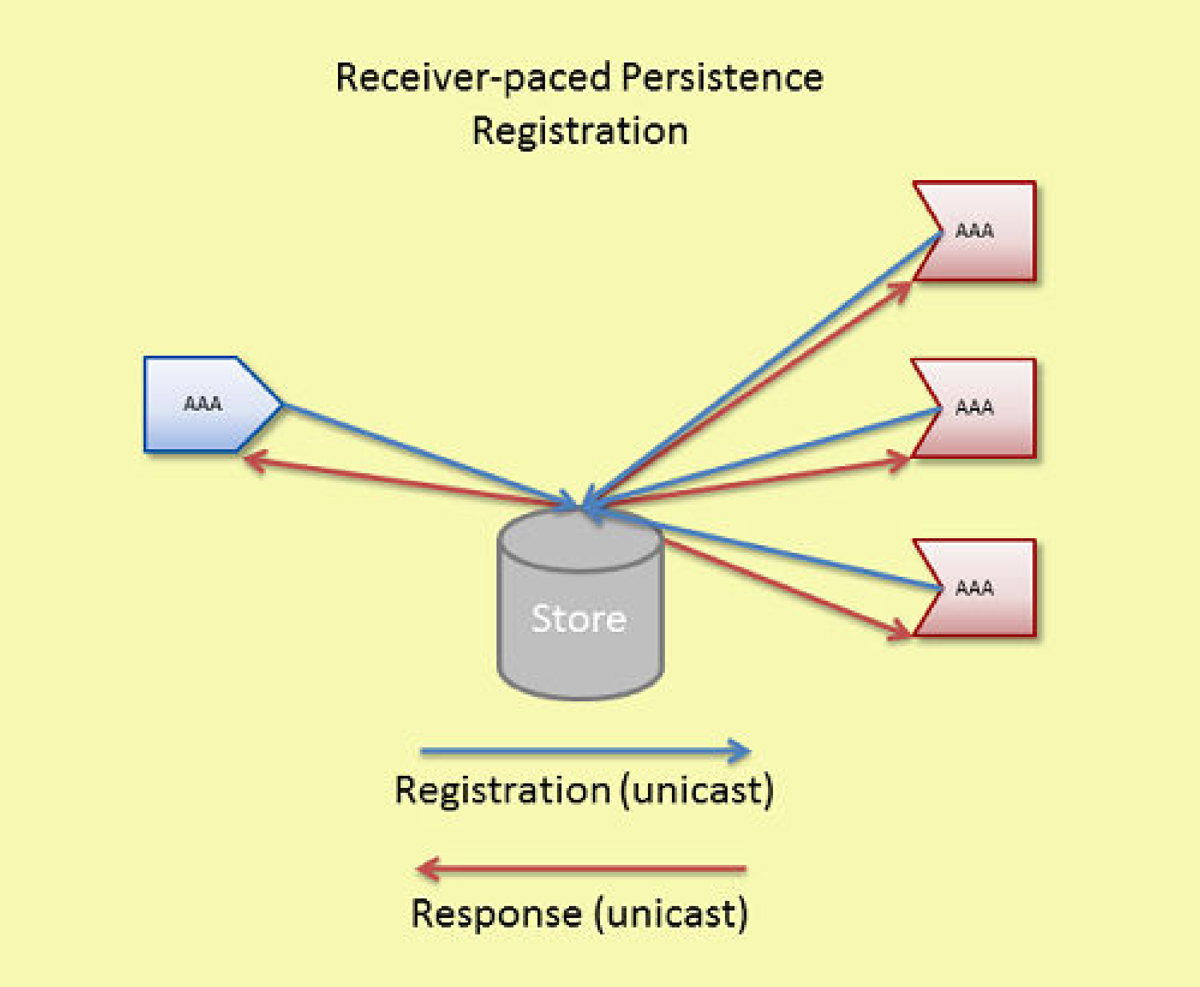
- “RPP Registration” on page 21
- “RPP Normal Operations” on page 23
- “RPP Message Recovery” on page 27
- “RPP Deregistration” on page 27
- “Implementing RPP” on page 27
- “Example RPP Configuration Files” on page 28
- “RPP Cross Feature Functionality” on page 30

RPP Registration

If a source sets *ume_receiver_paced_persistence*, its topic becomes a RPP topic. When the source registers with the store, the source's repository also becomes a RPP repository. Receivers registering with a store on the RPP topic become RPP receivers.

Note: As with Source-paced Persistence, RPP sources send Source Registration Information (SRI) packets to RPP receivers over the configured UM transport. RPP Receviers must wait for this information before they can unicast registration requests to the store. See “Source Registration” on page 13 and “Receiver Registration” on page 15 for more information.

Figure 8. RPP Registration



A source registration request includes the following.

- Designation of a RPP topic (LBMC_UME_PREG_FLAG_REGISTER flag)
- Reconfigured repository configuration option values. Possible options are the 3 repository size options, `repository-allow-ack-on-reception`, `repository-disk-write-delay` and `source-flight-size-bytes-maximum`.
- Re-registration must request same configuration options or the store rejects the request.

Receiver registration request includes its designation as a RPP receiver (LBMC_UME_PREG_FLAG_REGISTER flag).

The repository's registration response to both a source and a receiver acknowledges RPP mode.

Late Registering Receiver

Late joining receivers that register after the first RPP topic message has been sent cannot receive any messages sent prior to their registration, except for messages not yet consumed by all RPP receivers. This behavior also applies to the very first receiver of a RPP group that registers after the source sends the first message. Any messages published prior to RPP receiver registration are not available for recovery.

Early Exiting Receiver

Should a registered receiver's activity timer expire and be declared by the repository to be inactive, the repository retains all messages published since the receiver's last acknowledged message (or initial sequence number if no messages were acknowledged) until its receiver state lifetime expires and the repository deletes the receiver state information. Deleting receiver state removes all knowledge of the receiver from the repository. As a result, the repository also deletes all messages being held solely for this receiver.

Should an early exiting receiver reregister (or otherwise become active) before the expiration of its state lifetime, that receiver can recover all messages retained for that receiver.

UMP Version RPP Compatibility Matrix

The following table indicates the result of registration requests across UMP versions.

Version/Object	**Pre-ver. 5.3 Store**	**Ver. 5.3 RPP Store**	**Ver. 5.3 Non-RPP Store**
Pre 5.3 Source	Granted	Rejected *	Granted *
5.3 RPP Source	Granted - Source Error	Granted *	Rejected *
5.3 Non-RPP Source	Granted	Rejected *	Granted *
Pre 5.3 Receiver	Granted	Rejected	Granted
5.3 RPP Receiver	Granted - Receiver Error	Granted	Rejected
5.3 Non-RPP Receiver	Granted	Rejected	Granted

- Granted - Source Error indicates that the store granted the registration but the source detected that RPP behavior was not acknowledged by the store.
- Granted - Receiver Error indicates that the store granted the registration but the receiver detected that RPP behavior was not acknowledged by the store.
- * Refers only to the re-registration of a source with an existing source repository because the source determines the repository's behavior for new registrations.

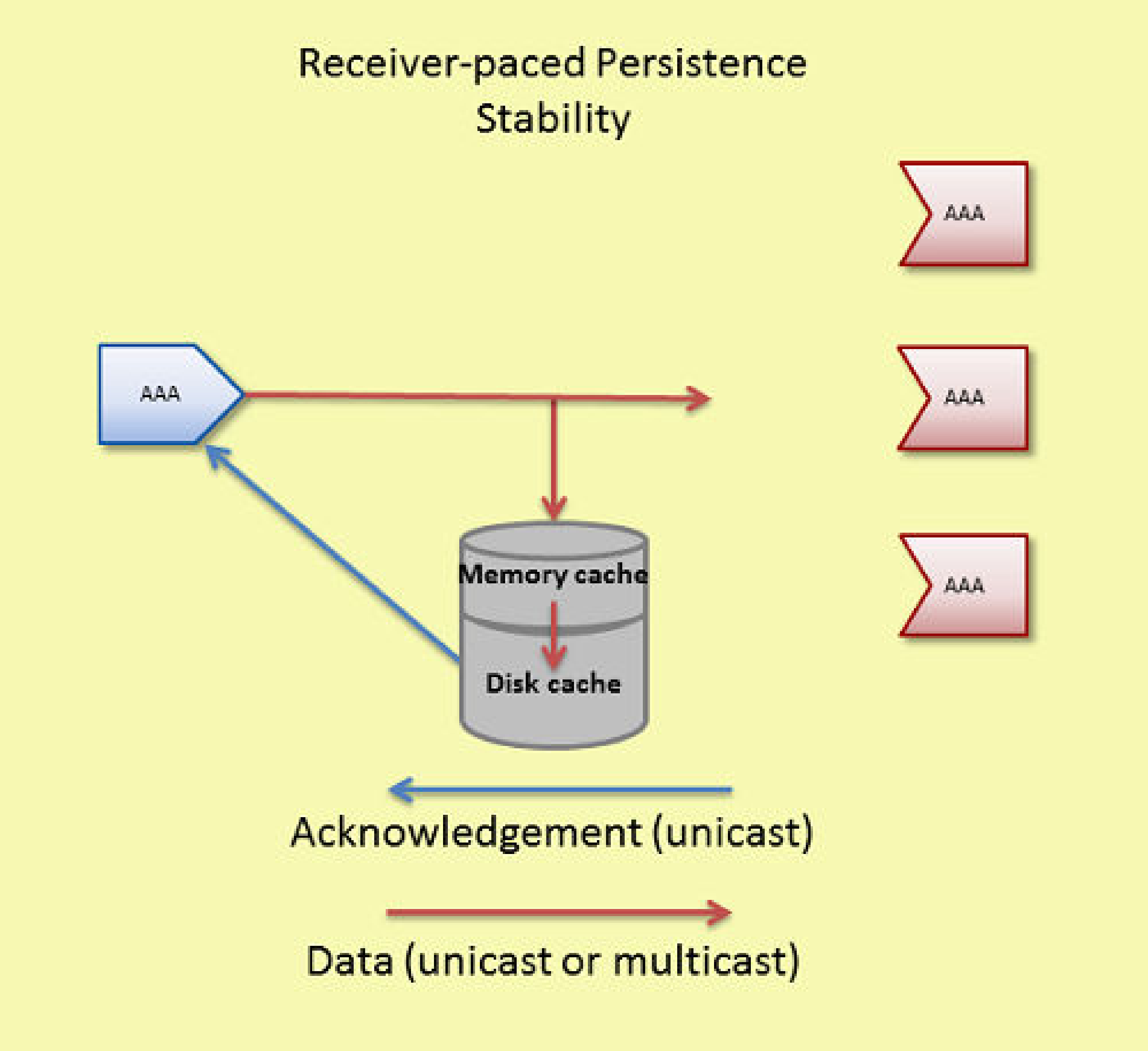
RPP Normal Operations

Since all RPP receivers must receive all messages, message overruns at the store or receiver must be prevented by regulating the sending pace of the source. The store uses the source's flight size (bytes) to regulate the source's speed, by withholding stability acknowledgements if the repository does not have at least one flight size available.

1. Sources transmit data to receivers and store repositories at the same time over UM multicast or unicast transport protocols.
2. When a disk repository receives a message, it holds the message in memory cache before it writes the message to disk. The repository sends a stability notification to the source after it writes the message to

disk. Memory repositories send the stability notice upon reception. See also Acknowledge on Reception and Receiver Acknowledgement and Flight Size below.

Figure 9. RPP Stability Acknowledgement

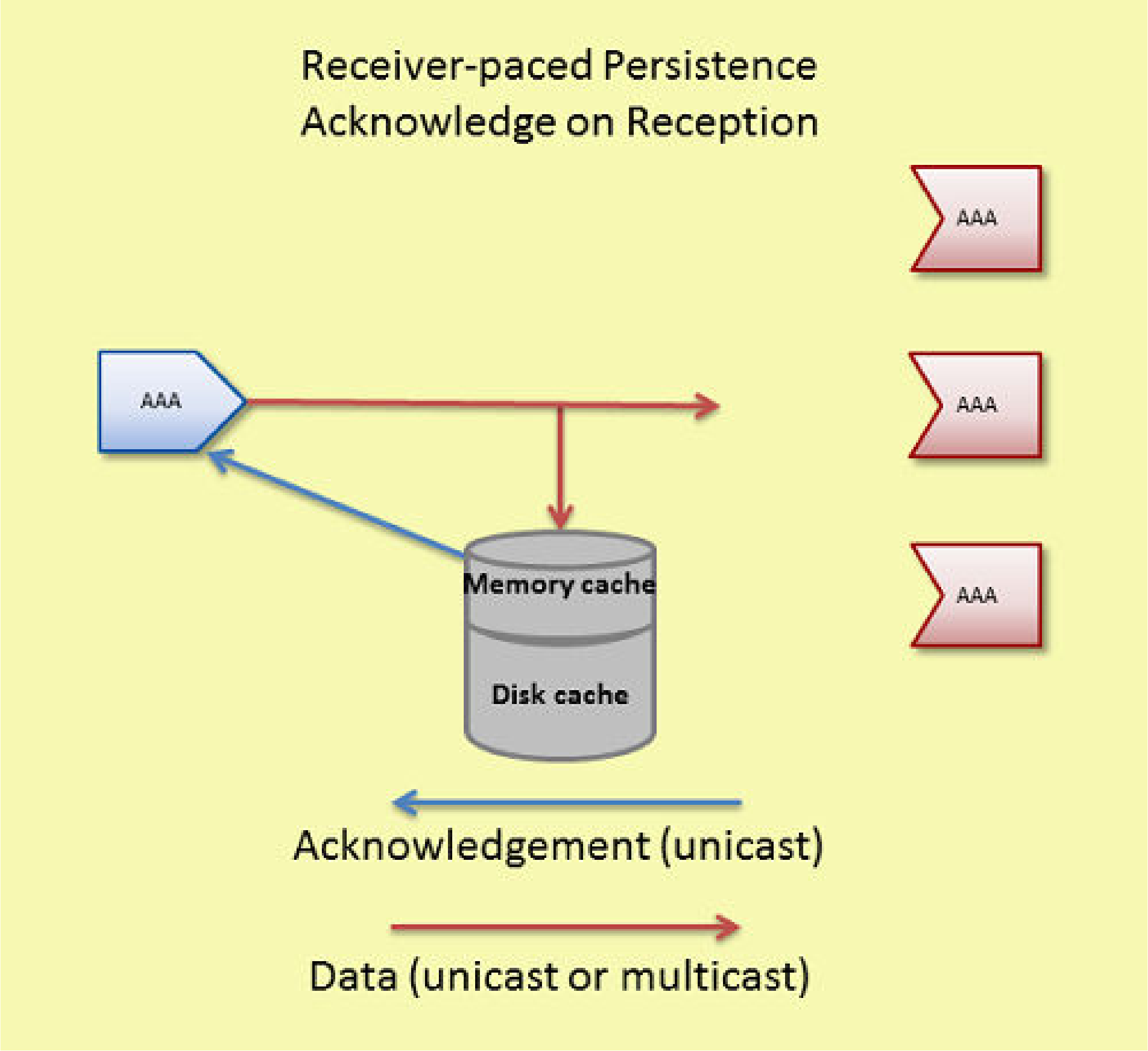

Receiver-paced Persistence
Stability
AAA
AAA
AAA
AAA
Memory cache
Disk cache
Acknowledgement (unicast)
Data (unicast or multicast)

3. If the source desires delivery confirmation, receivers unicast acknowledgements directly to the source letting the source know of message consumption as well.

The following also affect when a repository sends a stability acknowledgement to the source.

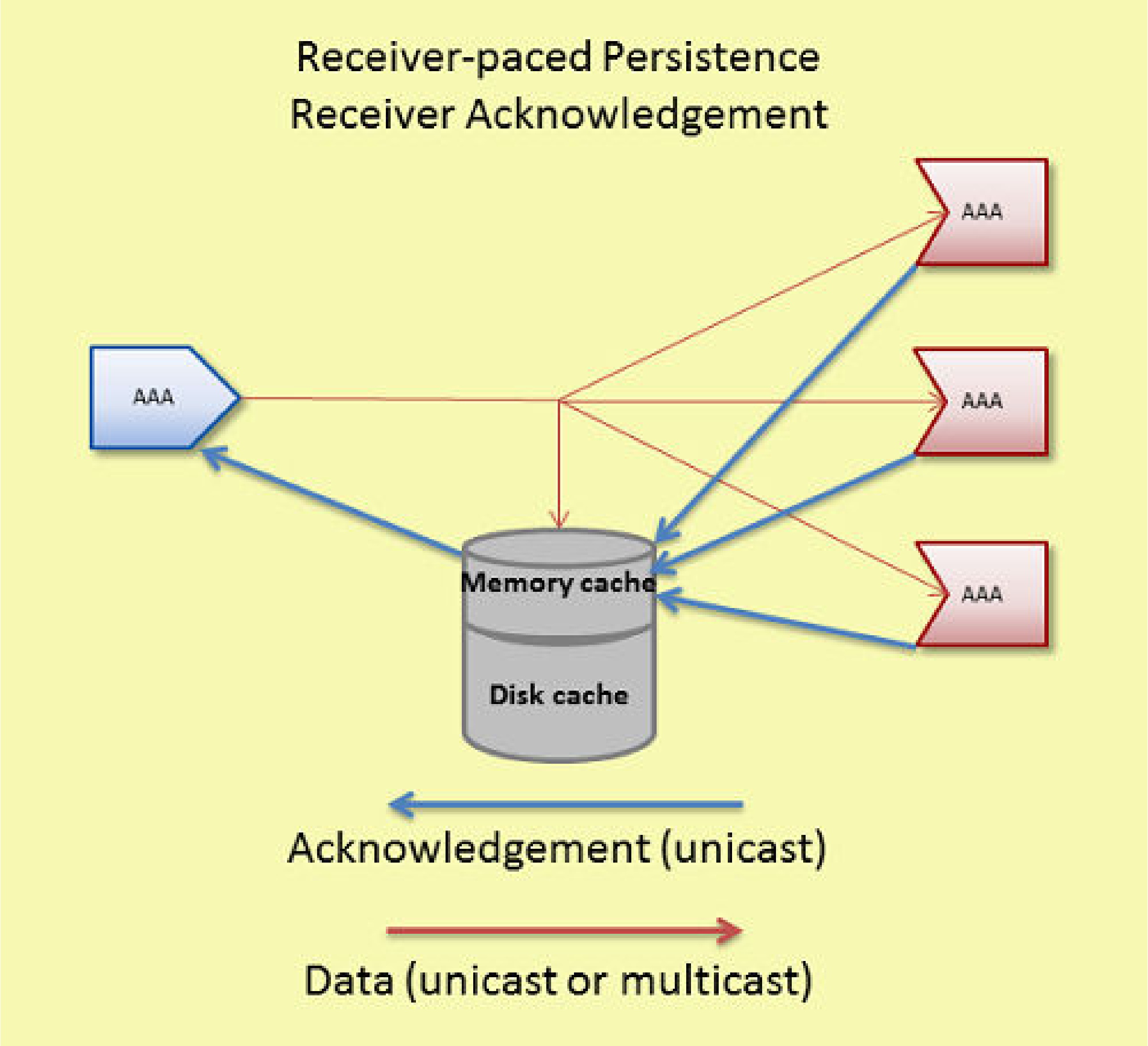
- Acknowledge on Reception - If you configure the repository for `repository-allow-ack-on-reception` and the source also sets *`ume_repository_ack_on_reception`* , the repository sends a stability acknowledgement to the source immediately upon reception. If the disk write has not already been initiated, UMP does not write the message to disk.

Figure 10. RPP Acknowledge on Reception


Receiver-paced Persistence
Acknowledge on Reception
AAA
AAA
AAA
AAA
Memory cache
Disk cache
Acknowledgement (unicast)
Data (unicast or multicast)

- Receiver Acknowledgement - If a repository receives acknowledgements from all receivers before writing the message to disk, it immediately sends a stability acknowledgement to the source. If the disk write has not already been initiated, UMP does not write the message to disk.

Figure 11. RPP Receiver Acknowledgement


Receiver-paced Persistence
Receiver Acknowledgement
AAA
AAA
AAA
AAA
Memory cache
Disk cache
Acknowledgement (unicast)
Data (unicast or multicast)

- Write Delay - The repository option, repository-disk-write-delay, allows the repository to hold messages in memory cache longer before persisting them to disk. This delay increases the probability that all RPP receivers acknowledge message consumption, eliminating the need to persist the message to disk.
- Flight size - A disk repository only sends stability acknowledgement to the source if its memory cache has at least one flight size (in both messages and bytes) available. A memory repository also sends stability acknowledgement if it has at least one flight size (in both messages and bytes) available. A lack of available space in the repository blocks the source until the repository reclaims the necessary storage space and sends a stability acknowledgement.

For memory store repositories, the behaviors Acknowledge on Reception, Receiver Acknowledgement and Write Delay do not apply.

RPP Message Recovery

An RPP source repository retains messages until all RPP receivers acknowledge receipt of the message. Therefore an RPP receiver can only recover messages that have not been consumed by all RPP receivers. It is important to note that an RPP receiver joining after other RPP receivers have already joined and after messages have already been sent can only be guaranteed to recover messages sent subsequent to its joining.

The Off Transport Recovery (OTR) method is also active for RPP receivers. OTR does not require the receiver to restart if it detects a gap in the topic message sequence numbers it has received. See the *UM Concepts Guide, UMS Features, Off Transport Recovery (OTR)* for more information.

RPP Deregistration

You can deregister either sources or receivers using deregistration APIs, (**lbm_src_ume_deregistration()**, **lbm_rcv_ume_deregistration()** and **lbm_wrcv_ume_deregistration()**). UM deletes the state of deregistered objects. If you deregister a RPP receiver, UMP automatically updates the number of receiver acknowledgements required to maintain RPP behavior. The store issues Deregistration Successful events for every source or receiver that deregisters. See “ UMP Events” on page 31.

Applications should be cautious about using the deregistration APIs to deregister RPP sources or receivers. These APIs can be disruptive to RPP.

- **lbm_src_ume_deregistration()** also deletes any persisted RPP messages in the source's repository. A source application should only use **lbm_src_ume_deregistration()** if it uses delivery confirmation from the receiver and it knows all messages have been delivered. The source is blocked after deregistering and and must restart in order to register again with the RPP store.
- A receiver application using **lbm_rcv_ume_deregistration()** loses any connections to multiple sources sending on its topic. Any messages not yet confirmed for that receiver are unrecoverable. Receivers can still receive messages after deregistering, but cannot acknowledge message consumption. The receiver must restart in order to register again with the RPP store.
- A receiver application using **lbm_wrcv_ume_deregistration()** ends all of the sessions associated with the sources sending on the topics that match the wildcard pattern. Individual topic receivers can still receive messages after deregistering, but cannot acknowledge message consumption. The wildcard receiver must restart in order to register again with the RPP store.

Implementing RPP

Follow the procedure below to configure Receiver-paced Persistence.

1. Set *`ume_receiver_paced_persistence`* for sources and receivers in a UM configuration file. If only certain sources or receivers in a context are RPP, use **lbm_*setopt()**l in the source or receiver application or use Ultra Messaging Manager to specify RPP in an UM XML configuration file.
2. Set `repository-allow-receiver-paced-persistence` = 1 for the repository in the `umestored` XML configuration file.
3. Coordinate *`ume_flight_size_bytes`* between the repository and the source. Set the maximum flight size with the repository option, `source-flight-size-bytes-maximum`. Sources can reconfigure the repository's `source-flight-size-bytes-maximum` to a value less than or equal to the maximum.
4. Optional. Coordinate the *`ume_repository_ack_on_reception`* between the repository and the source. If the repository has `repository-allow-ack-on-reception` enabled (1), the source can choose to keep it enabled or turn it off (*`ume_repository_ack_on_reception`* = 0). If the repository has `repository-allow-ack-on-reception` disabled (0), the source cannot turn it on.

5. Optional. If the repository is a disk repository (repository-type = `disk` or `reduced-fd`), set the maximum write delay with the repository option, `repository-disk-write-delay`. Sources can reconfigure the repository's `repository-disk-write-delay` to a value less than or equal to the maximum configured for the repository with *`ume_write_delay`* .
6. Optional. Coordinate repository size options between the source and repository. If you wish to use the repository's values, you do not need to configure source configuration values. The repository sets a maximum for these three options. The source can reconfigure the repository's options with values less than or equal to the maximum configured for the repository using the following UM configuration options.
   - *`ume_repository_size_threshold`*
   - *`ume_repository_size_limit`*
   - *`ume_repository_disk_file_size_limit`*

Example RPP Configuration Files

The sample configuration files shown below show how a store configuration file establishes certain RPP option values and the source can reconfigure them via a UM configuration file. Although only two files appear below, this configuration represents two, single-store quorum/consensus groups and one UM context. A second `umestored` configuration file would be required for the store `store1rpp` containing options and values identical to `store0rpp`.

UM Configuration File

In the following example UM configuration file, RPP options appear in the section labeled, `##Persistence Options###`.

- The source uses the same repository size values as the store. In this case, you do not need to specify these option values again in the source's UM Configuration File. They appear in this file for the sake of completeness.
- The source reconfigures ume_flight_size_bytes to 1,000,000 bytes, which is less than the repository's 4 MB default. (The source can reconfigure this option to a value less than or equal to the repository's configured value.)
- The source reconfigures ume_write_delay from the default of 0 ms to 1000 ms or 1 second.
- The option, ume_session_id 5353, is commented out because this file specifies RegIDs 2929 and 2930, respectively, for the stores in the *`ume_store_name`* option. The option, *`ume_session_id`* , appears in this file as a reminder that you can use either RegIDs or Session IDs, but not both.

```
#Sample UM Configuration File, UMP Version 5.3
#Major Options
source transport lbtrm
# in order and reassembled
receiver ordered_delivery 1
#Multicast Resolver Network Options
context resolver_multicast_address 225.8.17.29
context resolver_multicast_interface 10.29.3.0/24
# Transport LBT-RM Netowrk Options
source transport_lbtrm_multicast_address 225.8.17.30
context transport_lbtrm_multicast_address_low 225.12.17.10
context transport_lbtrm_multicast_address_high 225.12.17.14
#Transport LBT-RM Operation Options
context transport_lbtrm_data_rate_limit 10000000
context transport_lbtrm_retransmit_rate_limit 5000000
# Transport LBT-RM Reliability Options
receiver transport_lbtrm_nak_initial_backoff_interval 40000
receiver transport_lbtrm_nak_initial_backoff_interval 500
receiver transport_lbtrm_nak_generation_interval 10000
```

```
##Turn off NAKs
receiver transport_lbtrm_send_naks 0
#Request Network Options
context request_tcp_port_low 55000
context request_tcp_port_high 55500

## Persistence Options ###
source ume_store_group 0:1
source ume_store_name store0rpp:2929:0
source ume_store_group 1:1
source ume_store_name store1rpp:2930:1
source ume_store_behavior qc
source ume_flight_size 500
source ume_flight_size_bytes 1000000
source ume_receiver-paced-persistence 1
source ume_repository_size_threshold 104857600
source ume_repository_size_limit 209715200
source ume_repository_disk_file_size_limit 1073741824
source ume_repository_ack_on_reception 1
source ume_write_delay 1000
receiver ume_receiver-paced-persistence 1
receiver ume_explicit_ack_only 1
source ume_proxy_source 1
#source ume_session_id 535353
context ume_source_liveness_timeout 4000
context ume_receiver_liveness_interval 1000
source ume_confirmed_delivery_notification 1
```

umestored Configuration File

In the following example store configuration file, RPP options appear in the section for the topic pattern, `ABC*` .

- The store has raised the `repository-size-limit` from the default of 48 MB to 200 MB, the `repository-size-threshold` from the default of 0 to 100 MB, and the `repository-disk-file-size-limit` from the default of 100MB to 1 GB.
- The store does not specify a `source-flight-size-bytes-maximum`, using the default of 4 MB.

```
<?xml version="1.0"?>
<ume-store version="1.2">
  <daemon>
    <log>/configs/stores/umestored1/umestored.log</log>
    <lbm-license-file>/bin/umq_exp_license.txt</lbm-license-file>
    <lbm-license-file>/bin/lbm_ume_umq_udx_rdma_license.txt</lbm-license-file>
    <lbm-config>/configs/lbm_4_store.cfg</lbm-config>
    <pidfile>/configs/stores/umestored1/umestored.pid</pidfile>
    <web-monitor>*:15404</web-monitor>
  </daemon>
  <stores>
    <store name="rpp-ump-test-store-1" port="14667">
      <ume-attributes>
        <option type="store" name="disk-cache-directory" value="/stores/store1/cache"/>
        <option type="store" name="disk-state-directory" value="/stores/store1/state"/>

        <option type="store" name="allow-proxy-source" value="0" />
        <option type="store" name="context-name" value="store1rpp"/>
      </ume-attributes>
      <topics>
        <topic pattern="ABC*" type="PCRE">
          <ume-attributes>
             <option type="store" name="repository-allow-receiver-paced-persistence"
value="1"/>
             <option type="store" name="repository-type" value="disk"/>
             <option type="store" name="repository-size-threshold" value="104857600"/>
             <option type="store" name="repository-size-limit" value="209715200"/>
             <option type="store" name="repository-disk-file-size-limit"
```

```
value="1073741824"/>
             <option type="store" name="repository-allow-ack-on-reception" value="1"/>
             <option type="store" name="repository-disk-write-delay" value="1000"/>
             <option type="store" name="receiver-new-registration-rollback" value="0"/>
             <option type="store" name="source-activity-timeout" value="120000"/>
             <option type="store" name="receiver-activity-timeout" value="30000"/>
             <option type="store" name="retransmission-request-forwarding" value="0"/>
          </ume-attributes>
        </topic>
      </topics>
    </store>
  </stores>
</ume-store>
```

RPP Cross Feature Functionality

UM Feature	Supported	Notes
UMP Proxy Sources	Yes	
UM Router	Yes	
UM Transports	Yes	
Multi-Transport Threads	No	UMP does not support Multi-Transport Threads
Off-Transport Recovery	Yes	
Late Join	Yes	With the new store option, Acknowledge on Reception, sources may not retain sufficient sent messages to provide an effective Late Join capability.
HF	Yes	
HFX	Yes	
Wildcard Receivers	Yes	
Message Batching	Yes	
Ordered Delivery	Yes	
Request/Response	Yes	
Multicast Immediate Messaging (MIM)	No	MIM messages are not persisted and have no impact on RPP.
Source Side Filtering	Yes	
Self-Describing Messaging (SDM)	Yes	
Pre-Defined Messaging (PDM)	Yes	
UM Spectrum	Yes	
Monitoring/Statistics	Yes	

UM Feature	Supported	Notes
Acceleration - DBL	Yes	
Acceleration - UD	Yes	
Implicit/Explicit Acknowledgements	Yes	
Registration ID/Session Management	Yes	
Fault Tolerance - Round Robin	No	If a RPP source attempts to register to a source repository configured for Round Robin fault tolerance, lbm_src_create() returns an error.
Fault Tolerance - Quorum Consensus	Yes	
UM SNMP Agent	Yes	
Ultra Messaging Manager	Yes	
Ultra Messaging Cache	Yes	
Ultra Messaging Desktop Services	No	

UMP Events

The Ultra Messaging API provides a number of events, callbacks, messages, functions, and settings. The API reference (*C API*, *Java API* or *.NET API*) can be used to see the true extent of the API. In order to design successful applications, though, a high level understanding of the events and callbacks is essential.

- Events - Source events occur on a per source basis.
- Callbacks - Source and receiver callbacks called directly from UMP internal operation and usually demands a return value be filled in and/or are informational in nature. Typically, applications do very little processing in callbacks.
- Messages - Messages to receivers can simply contain UMP information or have impact on operation.

Some specific languages, such as C, Java, or C# may have specific nuances for the various events and callbacks. But, by and large, an application should plan on having access to the items listed in the following sections. For details for a particular language, consult the Ultra Messaging API documentation (*C API*, *Java API* or *.NET API*).

Source Events

The following events and callbacks are available for source applications.

Event Name	Type	Description
`Store Registration Success`	Source Event	Delivered once a source has successfully registered with a single store. Event contains flags to show if the source is "old" (i.e. a re-registration) as well as the sequence number that the source should use as its initial sequence number when sending, and the store information
`Store Registration Complete`	Source Event	Delivered once a source has completed registration with the required store(s). This indicates the source may send as it desires. Event contains the consensus sequence number.
`Store Registration Error`	Source Event	Delivered once a source has received an error from the store indicating the requested registration was not granted. Event contains an error message to indicate what happened.
`Store Message Stable`	Source Event	Delivered once a message is stable at a single store. Event contains the message sequence number and indicates if the the message meets Intergroup and/or Intragroup stability requirements. Also includes the store information.
`Store Message Not Stable`	Source Event	Delivered once a message's *`ume_message_stability_lifetime`* has expired. The source no longer retransmits the message to the store.
`Delivery Confirmation`	Source Event	Delivered once a message has been confirmed as delivered and processed by a receiving application. Event contains the message sequence number as well as indications whether the message has met the unique confirmations requirement. Also contains the receiver's Registration ID or Session ID.
`Store Unresponsive`	Source Event	Delivered once a store is seen to be unresponsive due to failure or network disconnect. Event contains a message with more details suitable for logging. Sources using the unresponsive store as their only store (not in Round-Robin or Quorum/Consensus) will be prevented from sending until the store recovers.
`Store Message Reclaimed`	Source Event	Delivered once a message has passed through retention and is about to be released from memory or disk. Event contains the message sequence number. (Reclaim refers to storage space reclamation.)
`Store Forced Reclaim`	Callback	Indicates a message is being forcibly released because the memory size limit (*`retransmit_retention_size_limit`*) has been exceeded or the message's *`ume_message_stability_lifetime`* has expired. Event contains the message sequence number.
`Flight Size Notification`	Callback	Indicates that the number of in-flight messages for a source has exceeded or fallen below the configured flight size limit for a source. The event indicates if the flight size has been exceeded (OVER) by a new message send or that a message recently stabilized has reduced the number of in flight messages to less than the flight size limit (UNDER).
`RPP Source Registration Success`	Source Event	Delivered once a source has successfully registered with a single store as a RPP source. The event contains either the RegID or Session ID, the sequence number of the last message stored for the source and store information.

Event Name	Type	Description
`RPP Source Registration Failure`	Source Event	Delivered once a source has received an error from the store indicating the requested registration was not granted. Event contains an error message to indicate what happened.
`RPP Source Deregistration Success`	Source Event	Delivered once a source successfully deregisters from an individual store. The event contains either the RegID or Session ID, the sequence number of the last message stored for the source and store information.
`RPP Source Deregistration Complete`	Source Event	Delivered once UMP receives a successful deregistration event from all stores.

Receiver Events

The following callbacks and messages are available for receiver applications

Event Name	Type	Description
`Store Registration Success`	Message	Delivered once a receiver has successfully registered with a single store. Message contains flags to show if the receiver is "old" (i.e. Not a new registration) as well as the sequence number that the receiver should use as its low sequence number, and the store information. In addition, the event contains the source's Registration ID or Session ID and the receiver's Registration ID or Session ID.
`Store Registration Complete`	Message	Delivered once a receiver has completed registration with the store(s) required. This indicates the receiver may now receive data. Message contains the consensus sequence number.
`RPP Receiver Registration Success`	Message	Delivered once a receiver has successfully registered with a single store as a RPP receiver. Message contains either the RegID or Session ID, the sequence number of the last message stored for the source and store information.
`RPP Receiver Registration Failure`	Message	Delivered once a receiver has received an error from the store indicating the requested registration was not granted. Event contains an error message to indicate what happened.
`RPP Receiver Deregistration Success`	Message	Delivered once a receiver successfully deregisters from an individual store. The message contains either the RegID or Session ID for the receiver and the source, the sequence number of the last message stored for the source and store information.
`RPP Receiver Deregistration Complete`	Message	Delivered once UMP receives a successful deregistration event from all stores.
`Store Registration Error`	Message	Delivered once a receiver has received an error from the store indicating the requested registration was not granted. Message contains an error message to indicate what happened.
`Store Registration Change`	Message	Delivered once a change in store information is received from the source. The extent of the change is included in a message suitable for logging.

Event Name	Type	Description
`Store Retransmission`	Message	Retransmissions from recovery come in as normal messages with a flag indicating their status as a retransmission.
`Store Registration Function`	Callback	Called once a receiver receives store information from a source and UMP desires to know the RegID to use for the receiver. Callback passes the source RegID, the store information, and the source transport name. The return value is the RegID that UMP should request to use from the store.
`Store Recovery Sequence Number Function`	Callback	Called once registration is about to complete and the low sequence number must be determined. Callback passes the highest sequence number seen from the source and the consensus sequence number from the stores or sequence number from the store if using round-robin.

Context Events

The following events are available for the context of source and receiver applications.

Event Name	Type	Description
`Flight Size Notification`	Context Event	Indicates that the number of in-flight Multicast Immediate Messages has exceeded or fallen below the configured flight size limit. The event indicates if the flight size has been exceeded (OVER) by a new message send or that a message recently stabilized has reduced the number of in flight messages to less than the flight size limit (UNDER).

CHAPTER 5

Enabling Persistence

This chapter includes the following topics:

Overview

In this section, we explain how to build a persistence messaging application by starting with a minimum source and receiver and then adding UMP features incrementally. With the help of example source, this section explains the following operations.

- "Adding the UMP Store to a Source" on page 36
- "Adding Fault Recovery with Registration IDs" on page 37
- "Enabling Persistence Between the Source and Store" on page 38
- "Enabling Persistence in the Source" on page 38
- "Enabling Persistence in the Receiver" on page 39

Prerequisite: You should understand basic Ultra Messaging concepts such as Sources and Receivers and the basic methods for configuring them.

The following table lists all source files used in this section. The files can be found in the `/doc/example` directory. You can also acces these file via the *Sample Source Code* tab in the left panel, under *C Example Source Code*.

Object	Filename
Source Application	ume-example-src.c
Receiver Application	ume-example-rcv.c
Source Application 2	ume-example-src-2.c

Object	Filename
Receiver Application 2	ume-example-rcv-2.c
Source Application 3	ume-example-src-3.c
Receiver Application 3	ume-example-rcv-3.c
UMP Store Configuration File	ume-example-config.xml

Starting Configuration

We begin with the minimal source and receiver used by the *QuickStart Guide*. To more easily demonstrate the UMP features we are interested in, we have modified the QuickStart source and receiver in the following ways.

- Modified the source to send 20 messages with a one second pause between each message
- Modified the receiver to anticipate 20 messages instead of just one
- Assigned the topic, UME Example, to both the source and receiver
- Modified the receiver to not exit on unexpected receiver events

The last change allows us to better demonstrate basic operation and evolve our receiver slowly without having to anticipate all the options that UMP provides up front.

Example files for our exercise are:

Object	File
Source Application	ume-example-src.c
Receiver Application	ume-example-rcv.c

Note: Be sure to build `ume-example-rcv.c` and `ume-example-src.c`. Instructions for building them appear at the beginning of the source files.

Adding the UMP Store to a Source

The fundamental component of a UMP persistence solution is the persistent store. To use a store, a source needs to be configured to use one by setting *`ume_store`* for the source. We can do that with the following piece of code.

```
err = lbm_src_topic_attr_str_setopt(&attr, "ume_store", "127.0.0.1:14567");
```

This sets the UMP persistent store for the source to the store running at 127.0.0.1 on port 14567.

Note: If you desire to run a store on a different machine than where the source and receiver are run, then you should replace 127.0.0.1 with the IP address (not hostname) of the machine running the UMP persistent store.

Example files for our exercise are:

Object	Filename
Source Application	ume-example-src.c
Receiver Application	ume-example-rcv.c
UMP Store Configuration File	ume-example-config.xml

After adding the ume-store specification to the source, perform the following steps.

1. Create the cache and state directories. `$ mkdir umestored-cache ; mkdir umestored-state`
2. Start up the store. `$ umestored ume-example-config.xml`
3. Start the Receiver. `$ ume-example-rcv`
4. Start the Source. `$ ume-example-src`

You should see a message on the source that says:

```
INFO: Source "UME Example" Late Join not set, but UME store specified. Setting Late
Join.
```

This is an informational message from UMP and merely means Late Join was not set and that UMP is going to set it.

Notice that the receiver was not configured with any store information. That is because setting it on the source is all that is needed. The receiver learns UMP store settings from the source through the normal UM topic resolution process. Receivers don't need to do anything special to leverage the usage of a store by a source.

Adding Fault Recovery with Registration IDs

If the source or receiver crashes, how does the source and receiver tell the store that they have restarted and wish to resume where they left off? We need to add in some sort of identifiers to the source and receiver so that the store knows which sources and receivers they are.

In UMP , these identifiers are called Registration IDs or RegIDs. UMP allows the application to control the use of RegIDs as it wishes. This allows applications to migrate sources and receivers not just between systems, but between locations with true, unprecedented freedom. However, UMP requires an application to be careful of how it uses RegIDs. Specifically, an application must not use the same RegID for multiple sources and/or receivers at the same time.

Now let's look at how we can use RegIDs to provide complete fault recovery of sources and receivers. We'll first handle RegIDs in the simplest manner by using static IDs for our source and receiver. For the source, the RegID of 1000 can be added to the existing store specification by changing the string to

```
127.0.0.1:14567:1000
```

This yields the source code in ume-example-src-2.c

For the receiver, we accomplish this in two steps.

1. Set a callback function to be called when we desire to set the RegID to 1100. This is done by declaring the callback function, `app_rcv_regid_callback`, which will return the RegID value 1100 to UMP .

2. Inform the UMP configuration for the receiver to use this callback function. That is accomplished by setting the *ume_registration_extended_function* similar to example code below.

```
lbm_ume_rcv_regid_ex_func_t id;      /* structure to hold registration function
information */
id.func = app_rcv_regid_callback;    /* the callback function to call */
id.clientd = NULL;                   /* the value to pass in the clientd to the
function */
err = lbm_rcv_topic_attr_setopt(&attr, "ume_registration_extended_function", &id,
sizeof(id));
```

Once this is done, the receiver has the ability to control what RegID it will use. This yields the source code in ume-example-rcv-2.c.

With these in place, you can experiment with killing the receiver and bringing it back (as long as you bring it back before the source is finished), as well as killing the source and bringing it back.

The restriction to this initial approach to RegIDs is that the RegIDs 1000 and 1100 may not be used by any other objects at the same time. If you run additional sources or receivers, they must be assigned new RegIDs, not 1000 or 1100. Let's now take a more sophisticated approach to RegIDs that will allow much more flexibility

Enabling Persistence Between the Source and Store

Let's refine our source to include some desired behavior following a crash. Upon restart, we want our source to resume with the first unsent message. For example, if the source sent 10 messages and crashed, we want our source to resume with the 11th message and continue until it has sent the 20th message.

Accomplishing this graceful resumption requires us to ensure that our source is the only source that uses the RegID assigned to it. The same RegID should be used as long as the source has not sent the 20th message regardless of any crashes that may occur. The sources and receivers are primarily responsible for managing the RegIDs.

The following two sections explain the changes needed for the source and receiver, which become fairly easy due to the events that UMP delivers to the application during UMP operation.

Note: While the following sections are instructive about how UMP uses RegIDs to provide persistence, RegIDs can also be managed easily with the use of Session IDs. See “Managing RegIDs with Session IDs” on page 47.

Enabling Persistence in the Source

With the above mentioned behaviors in mind, let's turn to looking at how they may be implemented with UMP , starting with the source. We can summarize the changes we need by the following list.

1. At source startup, use any saved RegID information found in the file by setting information in the *ume_store* configuration variable.
2. After the store registration is successful, if a new RegID was assigned to the source, save the RegID to the file.
3. Set the message number to begin sending. Refer to the explanation below.

4. Send until message number 20 has been sent.
5. After message 20 has been sent, delete the saved RegID file.

For Step 3, if the source has just been initialized, the application starts with message number 1. If the source has been restarted after a crash, the application looks to UMP to establish the beginning message number because UMP will use the next sequence number. For this simple example, we can make the assumption that each message is one sequence number for UMP and that UMP starts with sequence number 0. Thus the application can set the message number it begins resending with the value of the UMP sequence number + 1.

Note: Using sequence numbers to set the message number is a good practice if you send messages smaller than 8K.

These changes yield the source code in ume-example-src-3.c.

Enabling Persistence in the Receiver

Let's also refine the receiver to resume where it left off after a crash. Just as with the source, the receiver can have the store assign it a RegID if the receiver is just beginning. Once the receiver receives the 20th message from the source, it can get rid of the RegID and exit. Because the receiver can receive some messages, crash, and come back, we should only need to look at a message and check if it is the 20th message based on the message contents or sequence number. UMP provides all the events to the application that we need to create these behaviors in the receiver.

The receiver changes are summarized below.

1. At receiver startup, use any saved RegID information found in the file for callback information when needed.
2. When RegID callback is called: Check to see if the source RegID matches the saved source RegID. If it does, return the saved receiver RegID. RegID matches the saved source RegID if so, return the saved receiver RegID.
3. After store registration is successful: If not using a previously saved RegID, then save the RegID assigned by the store to the source to a file, as well as the store information and the source RegID.
4. After the last message is received (message number 20 or UMP sequence number 19), end the application and delete the saved RegID file.

RegIDs in UMP can be considered to be per source and per topic. Thus the receiver does not want to use the wrong RegID for a different source on the same topic. To avoid this, we save the source RegID and even store information so that the `app_rcv_regid_callback` can make sure to use the correct RegID for the given source RegID. These changes yield the source code in *ume-example-rcvc-3.c*

The above sources and receivers are simplified for illustration purposes and do have some limitations. The receiver will only keep the information for one source at a time saved to the file. This is fine for illustration purposes, but would be lacking in completeness for production applications unless it was assured that a single source for any topic would be in use. To extend the receiver to include several sources is simply a matter of saving each to the file, reading them in at startup, and being able to search for the correct one for each callback invoked.

CHAPTER 6

Demonstrating Persistence

This chapter includes the following topics:

Overview

This section demonstrates the following events using the `ume-example` applications described in Chapter 5, “Enabling Persistence” on page 35.

- “Running UMP Example Applications” on page 41
- “Single Receiver Fails and Recovers” on page 42
- “Single Source Fails and Recovers” on page 43
- “Single Store Fails” on page 44

Note: While these four sections demonstrate how UMP uses RegIDs to provide persistence, RegIDs can also be managed easily with the use of Session IDs. See “Managing RegIDs with Session IDs” on page 47.

The following table lists all source files used in this section. The files can be found in the `/doc/example` directory. You can also acces these file via the *Sample Source Code* tab in the left panel, under *C Example Source Code*.

Object	Filename
Source Application 3	ume-example-src-3.c
Receiver Application 3	ume-example-rcv-3.c
UMP Store Configuration File	ume-example-config.xml

Perform the following tasks first.

1. Build *`ume-example-rcv-3.c`* and *`ume-example-src-3.c`* . Instructions for building them are at the beginning of the source files.

2. Create default directories, `umestored-cache` and `umestored-state` in the `/doc/UME` directory where the other ume-example files are located. Our sample XML store configuration file, `ume-example-config.xml`, doesn't specify directories for the store's cache and state files, so those will be placed in the default directories.
3. Start the store. `$ umestored ume-example-config.xml`

You should see no output if the store started successfully. However, you should find a new log file, `ume-example-stored.log`, in the directory you ran the store in. The first couple lines should look similar to below.

```
Fri Feb 01 07:34:28 2009 [INFO]: Latency Busters Persistent Store version 2.0
Fri Feb 01 07:34:28 2009 [INFO]: LBM 3.3 [UME-2.0] Build: Jan 31 2009, 02:10:43
( DEBUG license LBT-RM LBT-RU ) WC[PCRE 6.7 04-Jul-2006, appcb]
```

You'll also be able to view the store's web monitor. Open a web browser and go to:

```
     http://127.0.0.1:15304/
```

You should see the store's web monitor page, which is a diagnostic and monitoring tool for the UMP store. See Chapter 11, " Ultra Messaging Web Monitor" on page 98.

Running UMP Example Applications

With the store running, let's try our example source and receiver applications.

1. Start the Receiver. `$ ume-example-rcv-3.exe`
2. Start the Source. `$ ume-example-src-3.exe`

You should see output for the source similar to the following:

```
saving RegID info to "UME-example-src-RegID" - 127.0.0.1:14567:2795623327
```

You should see output for the receiver similar to the following:

```
UME Store 0: 127.0.0.1:14567 [TCP:169.254.97.160:14371][2795623327] Requesting RegID: 0
saving RegID info to "UME-example-rcv-RegID" - 127.0.0.1:14567:2795623327:2795623328
Received 15 bytes on topic UME Example (sequence number 0) 'UME Message 01'
Received 15 bytes on topic UME Example (sequence number 1) 'UME Message 02'
Received 15 bytes on topic UME Example (sequence number 2) 'UME Message 03'
Received 15 bytes on topic UME Example (sequence number 3) 'UME Message 04'
...
```

The example source sends 20 messages. After the 20th messages, both the source and receiver exit and print the message `removing saved RegID file...` So what just happened? Let's walk through the output line by line.

Source

```
saving RegID info to "UME-example-src-RegID" - 127.0.0.1:14567:2795623327
```

The source successfully registered with the store using its pre-configured store address and port of 127.0.0.1:14567. It didn't ask for a specific RegID from the store, so the store automatically assigned one to it. In this case, the store assigned the ID, 2795623327. Your source's ID will likely be different because stores assign random RegIDs.

If you run the test again, you'll notice the source application has written a file called `UME-example-src-RegID` that contains the same information the source printed on startup, namely the IP address and port of the store it registered with, along with its RegID assigned by the store.

Receiver

```
UME Store 0: 127.0.0.1:14567 [TCP:169.254.97.160:14371][2795623327] Requesting RegID: 0
saving RegID info to "UME-example-rcv-RegID" - 127.0.0.1:14567:2795623327:2795623328
```

The receiver has been informed of how to connect to the store by the source, and it also successfully registered with the store. The store's IP address and port are shown, followed by the source's unique identifier string (in this case, it's a TCP source on port 14371), and the source's RegID. The receiver then requests RegID 0 from the store, which is a special value that means *pick an ID for me* (Although not displayed, the source requested ID 0 when it started up as well).

In parallel with the source application, the receiver application writes its RegID with this store to the file, `UME-example-rcv-RegID`.

After sending 20 messages under normal, stable conditions, the source and receiver applications exit and remove their RegID files.

Single Receiver Fails and Recovers

Perform the following procedure with the store running to see what happens when a receiver fails and recovers.

1. Start the Receiver. `$ ume-example-rcv-3.exe`
2. Start the source. `$ ume-example-src-3.exe` Let it run for a few seconds so the receiver gets a few messages.

```
UME Store 0: 127.0.0.1:14567 [TCP:169.254.97.160:14371][3735579353] Requesting
RegID: 0
saving RegID info to "UME-example-rcv-RegID" - 127.0.0.1:14567:3735579353:3735579354
Received 15 bytes on topic UME Example (sequence number 0) 'UME Message 01'
Received 15 bytes on topic UME Example (sequence number 1) 'UME Message 02'
Received 15 bytes on topic UME Example (sequence number 2) 'UME Message 03'
```

3. Stop the receiver (Ctrl/C) and leave the source running. Wait a few more seconds so that the source sends some messages while the receiver was down.
4. Restart the Receiver and let it run to completion. `$ ume-example-rcv-3.exe`

```
read in saved RegID info from "UME-example-rcv-RegID" - 127.0.0.1:14567 RegIDs
source 3735579353, receiver 3735579354
UME Store 0: 127.0.0.1:14567 [TCP:169.254.97.160:14371][3735579353]
Requesting RegID: 3735579354
Received 15 bytes on topic UME Example (sequence number 3) 'UME Message 04'
Received 15 bytes on topic UME Example (sequence number 4) 'UME Message 05'
Received 15 bytes on topic UME Example (sequence number 5) 'UME Message 06'
Received 15 bytes on topic UME Example (sequence number 6) 'UME Message 07'
Received 15 bytes on topic UME Example (sequence number 7) 'UME Message 08'
Received 15 bytes on topic UME Example (sequence number 8) 'UME Message 09'
Received 15 bytes on topic UME Example (sequence number 9) 'UME Message 10'
Received 15 bytes on topic UME Example (sequence number 10) 'UME Message 11'
```

Notice that the receiver picked up the message stream right where it had left off - after message 3. The first few messages (which the source had sent while the receiver was down) appear to come in much faster than the source's normal rate of one per second. That's because they are being served to the receiver from the store. The remaining messages continue to come in at the normal one-per-second rate because they're being received from the source's live message stream. This is *durable subscription* at work.

Single Source Fails and Recovers

Perform the following procedure with the store running to see what happens when a source fails and recovers.

1. Start the Receiver. `$ ume-example-rcv-3.exe`
2. Start the source. `$ ume-example-src-3.exe` Let it run for a few seconds so the receiver gets a few messages.
3. Stop the Source (Ctrl/C).
4. Restart the Source and let it run to completion. `$ ume-example-rcv-3.exe`

Source

You should see output similar to the following on the second run of the source.

```
read in saved RegID info from "UME-example-src-RegID" - 127.0.0.1:14567:2118965523
will start with message number 5
removing saved RegID file "UME-example-src-RegID"
```

Receiver

The receiver's output looks like the following.

```
UME Store 0: 127.0.0.1:14567 [TCP:169.254.97.160:14371][2118965523] Requesting RegID: 0
saving RegID info to "UME-example-rcv-RegID" - 127.0.0.1:14567:2118965523:2118965524
Received 15 bytes on topic UME Example (sequence number 0) 'UME Message 01'
Received 15 bytes on topic UME Example (sequence number 1) 'UME Message 02'
Received 15 bytes on topic UME Example (sequence number 2) 'UME Message 03'
Received 15 bytes on topic UME Example (sequence number 3) 'UME Message 04'
UME Store 0: 127.0.0.1:14567 [TCP:169.254.97.160:14371][2118965523] Requesting RegID:
2118965524
saving RegID info to "UME-example-rcv-RegID" - 127.0.0.1:14567:2118965523:2118965524
Received 15 bytes on topic UME Example (sequence number 4) 'UME Message 05'
Received 15 bytes on topic UME Example (sequence number 5) 'UME Message 06'
Received 15 bytes on topic UME Example (sequence number 6) 'UME Message 07'
Received 15 bytes on topic UME Example (sequence number 7) 'UME Message 08'
...
```

When the source was restarted, it read in its previously saved RegID and requested the same ID when registering with the store. The store informed the source that it had left off at sequence number 3 (UME Message 04), and the next sequence number it should send is 4 (UME Message 05). Bringing the source back up also caused the receiver to re-register with the store. Receivers can *only* find out about stores from sources they are listening to. Once the receiver re-registered with the store, it continued receiving messages from the source where it had left off.

Single Store Fails

Perform the following procedure with the store running to see what happens when the store itself fails.

1. Start the Receiver. `$ ume-example-rcv-3.exe`
2. Start the source. `$ ume-example-src-3.exe` Let it run for a few seconds so the receiver gets a few messages.
3. Stop the Store (Ctrl/C).

Notice that with this simple example program, the source simply prints the following and exits.

```
saving RegID info to "UME-example-src-RegID" - 127.0.0.1:14567:4095035673
Store unresponsive: store 0 [127.0.0.1:14567] unresponsive
Store unresponsive: store 0 [127.0.0.1:14567] unresponsive - no registration response.
line 318: not currently registered with enough UMP stores
```

When a source application tries to send a message without being registered with a store, the send call returns an error. Messages sent while not registered with a store cannot be persisted. See “ UMP Stores” on page 69 for information about using multiple stores.

Your source application(s) should assume an unresponsive store is a temporary problem and wait before sending the message again. See *`umesrc.c`* , *`umesrc.java`* , or *`umesrc.cs`* for examples of this behavior.

CHAPTER 7

Designing Persistence Applications

This chapter includes the following topics:

Overview

This section discusses considerations and methods for utilizing UMP persistence in your applications.

- “Pieces of a Persistence Solution” on page 45
- “Fault Recovery” on page 72

Pieces of a Persistence Solution

In UMP , a persistent system is composed of sources, receivers, and stores managed by one or more applications. Sources and receivers are the endpoints of communication and the store(s) provide fault recovery and persistence of state information. Your application can leverage UMP 's flexible methods of persistence to add an unprecedented level of fault tolerance. With this flexibility your applications assume new responsibilities not normally required in other persistent messaging systems. This section identifies the important considerations for your messaging applications when implementing the following UMP features.

- “Registration Identifiers” on page 46
- “ UMP Sources” on page 48
- “ UMP Receivers” on page 62
- “ UMP Stores” on page 69

Registration Identifiers

As mentioned in “Registration Identifier” on page 3 and “Adding Fault Recovery with Registration IDs” on page 37, stores use RegIDs to identify sources and receivers. UMP offers three main methods for managing RegIDs.

- Your applications assign static RegIDs and ensure that the same RegID is not assigned to multiple sources and/or receivers. See “Use Static RegIDs” on page 46.
- You can allow UMP stores to assign RegIDs and then save the assigned RegIDs. See “Save Assigned RegIDs” on page 47
- Use Session IDs to enable the UMP store to both assign and manage RegIDs. See “Managing RegIDs with Session IDs” on page 47

Your applications can manage RegIDs for the lifetime of a source or receiver as long as multiple applications do not reuse RegIDs simultaneously on the same store. RegIDs only need to be unique on the same store and may be reused between stores as desired. You can use a static mapping of RegIDs to applications or use some simple service to assign them.

Use Static RegIDs

The simplest method uses static RegIDs for individual applications. This method works best if:

- Applications use separate stores
- Multiple instances of an application also use separate stores

In the latter case, the same static source RegID can be used in every instance of the application because receivers will identify every Store/Source RegID tuple as unique.

The following source code examples assign a static RegID to a source by adding the RegID, `1000`, to the `ume_store` attribute. (See also ume-example-src-2.c.)

C API

```
lbm_src_topic_attr_t * sattr;

if (lbm_src_topic_attr_create(&sattr) == LBM_FAILURE) {
        fprintf(stderr, "lbm_src_topic_attr_create: %s\n", lbm_errmsg());
        exit(1);
}
if (lbm_src_topic_attr_str_setopt(sattr, "ume_store", "127.0.0.1:14567:1000")
== LBM_FAILURE) {
        fprintf(stderr, "lbm_src_topic_attr_str_setopt: %s\n", lbm_errmsg());
        exit(1);
}
```

JAVA API

```
LBMSourceAttributes sattr = null;
try {
sattr = new LBMSourceAttributes();
sattr.setValue("ume_store", "127.0.0.1:14567:1000");
}
catch (LBMException ex) {
System.err.println("Error creating source attribute: " + ex.toString());
System.exit(1);
}
```

.NET API

```
LBMSourceAttributes sattr = null;
```

```
try {
sattr = new LBMSourceAttributes();
sattr.setValue("ume_store", "127.0.0.1:14567:1000");
}
catch (LBMException ex) {
System.Console.Error.WriteLine ("Error creating source attribute: " + ex.toString());
System.Environment.Exit(1);
       }
```

Save Assigned RegIDs

Your application can save the RegID assigned to a source or receiver from the store because the UMP API informs your application of the RegID used for each registration. This method of managing RegIDs is perhaps the most flexible, but also requires some work by the application to save RegIDs and retrieve them in some way.

The following source code examples save the RegID assigned to a source to a file. (See also ume-example-src-3.c.)

C API

```
typedef struct src_info_t_stct {
    int existing_regid;
    int message_num;
} src_info_t;

#define SRC_REGID_SAVE_FILENAME "UME-example-src-RegID"

int save_src_regid_to_file(const char *filename, lbm_src_event_ume_registration_ex_t
*reg)
{
    FILE *fp;

    if ((fp = fopen(filename, "w")) == NULL)
        return -1;
    fprintf(fp, "%s:%u", reg->store, reg->registration_id);
    printf("saving RegID info to \"%s\" - %s:%u\n", filename, reg->store, reg-
>registration_id);
    fflush(fp);
    fclose(fp);
    return 0;
}
```

Managing RegIDs with Session IDs

The RegIDs used by stores to identify sources and receivers must be unique. Rather than maintaining RegIDs (either statically or dynamically), applications can use a Session ID, which is simply a 64-bit value that uniquely identifies any set of sources with unique topics and receivers with unique topics. A single Session ID allows UMP stores to correctly identify all the sources and receivers for a particular application.

Combinations of sources and receivers that make up a single valid session include the following.

- Sources for topics A, B, and C
- Receivers for topics A, B, and C
- Sources for topics A, B, and C, and receivers for topics X, Y and Z
- Sources for topics A, B, and C, and receivers for topics A, B, and C

Note: Note that any topic can be used for a source and a receiver at the same time, but not for more than one of each. Two sources using topic A, for example, would need to be split into two different contexts.

The UMP configuration option, *`ume_session_id`*, specifies a Session ID for a source, receiver or a context. If you want all sources and receivers for a particular context to use the same Session ID, use *`(context) ume_session_id`*. Any source or receiver that does not specify its own Session ID inherits the context's session ID. If a source or receiver specifies its own Session ID, it overrides the context Session ID for that individual source or receiver.

Of the two mutually exclusive methods for managing RegIDs, ...

1. Enable your application to assign and manage every RegID, ensuring no two objects registered with an individual store share the same RegID.
2. Allow the store to assign every RegID and enable your application to persist the RegIDs.

... using Session IDs simplifies the second management method. Since you cannot combine these two strategies at any single store, you also cannot combine the first method with the use of Session IDs at a single store.

How Stores Associate Session IDs and RegIDs

Session IDs do not replace the use of RegIDs by UMP but rather simplify RegID management. Using Session IDs equates to your application specifying a 0 (zero) RegID for all sources and receivers. However, instead of your application persisting the RegID assigned by the store, the store maintains the RegID for you.

When a store receives a registration request from a source or receiver with a particular Session ID, it checks to see if it already has a source or receiver for that topic/Session ID. If it does, then it responds with that source's or receiver's RegID.

If it does not find a source or receiver for that topic/Session ID pair, the store ...

1. Assigns a new RegID.
2. Associates the topic/Session ID with the new RegID.
3. Responds to the source or receiver with the new RegID.

The source can then advertise with the RegID supplied by the store. Receivers include the source's RegID in their registration request.

UMP Sources

The major concerns of sources revolve around RegID management and message retention. This section discusses the following topics.

- "New or Re-Registration" on page 49
- " Sources Must Be Able to Resume Sending" on page 49
- " Source Message Retention and Release" on page 50
- "Forced Reclaims" on page 51
- "Source Release Policy Options" on page 52
- "Confirmed Delivery" on page 52
- "Source Event Handler" on page 53
- " Source Event Handler - Stability, Confirmation and Release" on page 56
- " Mapping Your Message Numbers to UMS/UMP Sequence Numbers" on page 59
- "Receiver Liveness Detection" on page 62

New or Re-Registration

Any source needs to know at start-up if it is a new registration or a re-registration. The answer determines how a source registers with the store. UMP can not answer this question. Therefore, it is essential that the developer consider what identifies the lifetime of a source and how a source determines the appropriate value to use as the RegID when it is ready to register. RegIDs are per source per topic per store, thus a single RegID per store is needed.

The following source code examples look for an existing RegID from a file and uses a new RegID assigned from the store if it finds no existing RegID. (See also ume-example-src-3.c.)

C API

```
        err = lbm_context_create(&ctx, NULL, NULL, NULL);
        if (err) {printf("line %d: %s\n", __LINE__, lbm_errmsg()); exit(1);}

        srcinfo.message_num = 1;
        srcinfo.existing_regid = 0;

        err = read_src_regid_from_file(SRC_REGID_SAVE_FILENAME, store_info,
    sizeof(store_info));
        if (!err) { srcinfo.existing_regid = 1; }

            err = lbm_src_topic_attr_create(&attr);
            if (err) {printf("line %d: %s\n", __LINE__, lbm_errmsg()); exit(1);}

            err = lbm_src_topic_attr_str_setopt(attr, "ume_store", store_info);
            if (err) {printf("line %d: %s\n", __LINE__, lbm_errmsg()); exit(1);}
```

The use of Session IDs allows UMP , as opposed to your application, to accomplish the same RegID management. See “Managing RegIDs with Session IDs” on page 47.

Sources Must Be Able to Resume Sending

A source sends messages unless UMP prevents it, in which case, the send function returns an error. A source may lose the ability to send messages temporarily if the store(s) in use become unresponsive, e.g. the store(s) die or become disconnected from the source. Once the store(s) are responsive again, sending can continue. Thus source applications need to take into account that sending may fail temporarily under specific failure cases and be able to resume sending when the failure is removed.

The following source code examples demonstrate how a failed send function can sleep for a second and try again.

C API

```
        while (lbm_src_send(src, message, len, 0) == LBM_FAILURE) {
            If (lbm_errnum() == LBM_EUMENOREG) {
                printf("Send unsuccessful. Waiting...\n");
                sleep(1);
                continue;
            }
            fprintf(stderr, "lbm_src_send: %s\n", lbm_errmsg());
                            exit(1);
                    }
```

JAVA API

```
        for (;;) {
            try {
                src.send(message, len, 0);
            }
            catch (UMENoRegException ex) {
                System.out.println("Send unsuccessful. Waiting...");
                try {
                    Thread.sleep(1000);
                }
                catch (InterruptedException e) { }
                continue;
            }
            catch (LBMException ex) {
                System.err.println("Error sending message: " + ex.toString());
    System.exit(1);
            }
            break;
        }
```

.NET API

```
        for (;;) {
            try {
                src.send(message, len, 0);
            }
            catch (UMENoRegException ex) {
                System.Console.Out.WriteLine("Send unsuccessful. Waiting...");
                System.Threading.Thread.Sleep(1000);
                continue;
            }
            catch (LBMException ex) {
                System.Console.Out.WriteLine ("Error sending message: " + ex.toString());
    System.exit(1);
            }
            break;
        }
```

Source Message Retention and Release

UMP allows streaming of messages from a source without regard to message stability at a store, which is one reason for UMP's performance advantage over other persistent messaging systems. Sources retain all messages until notified by the active store(s) that they are stable. This provides a method for stores to be brought up to date when restarted or started anew.

Note: Source message retention is separate from the persistence of messages in the store.

When messages are considered stable at the store, the source can release them which frees up source retention memory for new messages. Generally, the source releases older stable messages first. To release the oldest retained message, all the following conditions must be met:

- message must meet stability requirements of the source, which can range from a single stability notice from the active store to stability notices from a group of stores (See “ Sources Using Quorum/Consensus Store Configuration” on page 71)

  and

- message must have been confirmed as delivered by a configured number of receivers (`ume_retention_unique_confirmations`),

  and

- the aggregate amount of buffered messages exceeds `retransmit_retention_size_threshold` bytes in payload and headers.

Some things to note:

- If the `retransmit_retention_size_threshold` is not met, no messages will be released regardless of stability.

- If the source registered with a "no-cache" store (See [“ UMP Stores” on page 69]) or `ume_message_stability_notification` is turned off, `ume_retention_unique_confirmations` is the only way to allow the source to release messages before retention size options come into play.
- With a quorum/consensus store configuration, when a quorum of stores report stability for a message, remaining stores may or may not send additional stability acks for that message.

Forced Reclaims

If the aggregate amount of buffered messages exceeds `retransmit_retention_size_limit` bytes in payload and headers, then UMP forcibly releases the oldest retained message even if it does not meet one or more of the conditions stated in “ Source Message Retention and Release” on page 50. This condition should be avoided and Informatica suggests increasing the `retransmit_retention_size_limit`.

A second condition that produces a forced reclaim is when a message remains unstabilized when the *`ume_message_stability_lifetime`* expires.

Whenever UMP performs a Forced Reclaim, it notifies the application in the following two ways.

1. The source event callback's RECLAIMED_EX event (see “Source Events” on page 32) includes a "FORCED" flag on the event. (UMP uses the same RECLAIMED_EX event, without the FORCED flag, for normal reclaims.)
2. Through the separate forced reclaim callback, if registered. You set this separate forced reclaim callback with the *`ume_force_reclaim_function`* configuration option.

Note: UMP retains the separate callback for backwards compatibility purposes and may be deprecated in future releases. The source event FORCED flag is the recommended method of tracking forced reclaims.

The following sample code (`umesrc`) implements the extended reclaim source event with the 'Forced' flag set if the reclamation is a forced reclaim.

C API

```
    case LBM_SRC_EVENT_UME_MESSAGE_RECLAIMED_EX:
      {
           lbm_src_event_ume_ack_ex_info_t *ackinfo = (lbm_src_event_ume_ack_ex_info_t
*)ed;

                    if (opts->verbose) {
                             printf("UME message reclaimed (ex) - sequence number %x
(cd %p). Flags 0x%x ",
                                       ackinfo->sequence_number, (char*)(ackinfo-
>msg_clientd) - 1, ackinfo->flags);
                             if (ackinfo->flags &
LBM_SRC_EVENT_UME_MESSAGE_RECLAIMED_EX_FLAG_FORCED) {
                        printf("FORCED");
                             }
                             printf("\n");
           }
            }
 break;
```

Java API

```
case LBM.SRC_EVENT_UME_MESSAGE_RECLAIMED_EX:
    UMESourceEventAckInfo reclaiminfo = sourceEvent.ackInfo();

        if (_verbose > 0) {
                if (reclaiminfo.clientObject() != null) {
                        System.out.print("UME message reclaimed (ex) - sequence number "
                               + Long.toHexString(reclaiminfo.sequenceNumber())
                               + " (cd "
                               +
Long.toHexString(((Long)reclaiminfo.clientObject()).longValue())
                               + "). Flags 0x"
                               + reclaiminfo.flags());
                } else {
                        System.out.print("UME message reclaimed (ex) - sequence number "
                               + Long.toHexString(reclaiminfo.sequenceNumber())
                               + " Flags 0x"
                               + reclaiminfo.flags());
                }
                if ((reclaiminfo.flags() &
LBM.SRC_EVENT_UME_MESSAGE_RECLAIMED_EX_FLAG_FORCED) != 0) {
                        System.out.print(" FORCED");
                }
                System.out.println();
        }
        break;
```

.NET API

```
case LBM.SRC_EVENT_UME_MESSAGE_RECLAIMED_EX:
             UMESourceEventAckInfo reclaiminfo = sourceEvent.ackInfo();
             if (_verbose > 0) {
                       System.Console.Out.Write("UME message reclaimed (ex) - sequence
number "
                                     + reclaiminfo.sequenceNumber()
                                     + " (cd "
                                     + ((uint)reclaiminfo.clientObject()).ToString("x")
                                     + "). Flags "
                                     + reclaiminfo.flags());
                       if ((reclaiminfo.flags() &
LBM.SRC_EVENT_UME_MESSAGE_RECLAIMED_EX_FLAG_FORCED) != 0) {
                              System.Console.Out.Write(" FORCED");
                       }
                       System.Console.Out.WriteLine();
             }
             break;
```

Source Release Policy Options

Sources use a set of configuration options to release messages that, in effect, specify the source's release policy. The following configuration options directly impact when the source may release retained messages.

- *ume_message_stability_notification*
- *ume_retention_unique_confirmations*
- *retransmit_retention_size_threshold*
- *retransmit_retention_size_limit*

Confirmed Delivery

As mentioned earlier, `ume_retention_unique_confirmations` requires a message to have a minimum number of unique confirmations from different receivers before the message may be released. This retains messages that have not been confirmed as being received and processed and keeps them available to fulfill any retransmission requests.

The following code samples show how to require a message to have 10 unique receiver confirmations

C API

```
lbm_src_topic_attr_t * sattr;

if (lbm_src_topic_attr_create(&sattr) == LBM_FAILURE) {
        fprintf(stderr, "lbm_src_topic_attr_create: %s\n", lbm_errmsg());
        exit(1);
}
if (lbm_src_topic_attr_str_setopt(sattr, "ume_retention_unique_confirmations",
"10")
== LBM_FAILURE) {
        fprintf(stderr, "lbm_src_topic_attr_str_setopt: %s\n", lbm_errmsg());
        exit(1);
}
```

JAVA API

```
LBMSourceAttributes sattr = null;
try {
sattr = new LBMSourceAttributes();
sattr.setValue("ume_retention_unique_confirmations", "10");
}
catch (LBMException ex) {
System.err.println("Error creating source attribute: " + ex.toString());
System.exit(1);
}
```

.NET API

```
LBMSourceAttributes sattr = null;
try {
sattr = new LBMSourceAttributes();
sattr.setValue("ume_retention_unique_confirmations", "10");
}
catch (LBMException ex) {
System.Console.Error.WriteLine ("Error creating source attribute: " + ex.toString());
System.Environment.Exit(1);
       }
```

Source Event Handler

The Source Event Handler is a function callback initialized at source creation to provide source events to your application related to the operation of the source. The following source code examples illustrate the use of a source event handler for registration events. To accept other source events, additional case statements would be required, one for each additional source event. See also “ UMP Events” on page 31.

C API

```
int handle_src_event(lbm_src_t *src, int event, void *ed, void *cd)
{
    switch (event) {
    case LBM_SRC_EVENT_UME_REGISTRATION_ERROR:
{
const char *errstr = (const char *)ed;
         printf("Error registering source with UME store: %s\n", errstr);
}
break;
        case LBM_SRC_EVENT_UME_REGISTRATION_SUCCESS_EX:
            {
                 lbm_src_event_ume_registration_ex_t *reg =
                        (lbm_src_event_ume_registration_ex_t *)ed;

                    printf("UME store %u: %s registration success. RegID %u. Flags %x
",
                                reg->store_index, reg->store, reg->registration_id,
                                reg->flags);
                    if (reg->flags & LBM_SRC_EVENT_UME_REGISTRATION_SUCCESS_EX_FLAG_OLD)
                        printf("OLD[SQN %x] ", reg->sequence_number);
                    if (reg->flags &
LBM_SRC_EVENT_UME_REGISTRATION_SUCCESS_EX_FLAG_NOACKS)
```

```
                        printf("NOACKS ");
                    printf("\n");
            }
            break;
        case LBM_SRC_EVENT_UME_REGISTRATION_COMPLETE_EX:
            {
                    lbm_src_event_ume_registration_complete_ex_t *reg;

                        reg  = (lbm_src_event_ume__complete_ex_t *)ed;
                    printf("UME registration complete. SQN %x. Flags %x ", reg-
>sequence_number,
                        reg->flags);
                    if (reg->flags &
LBM_SRC_EVENT_UME_REGISTRATION_COMPLETE_EX_FLAG_QUORUM)
                        printf("QUORUM ");
                    printf("\n");
            }
            break;
        case LBM_SRC_EVENT_UME_STORE_UNRESPONSIVE:
        {
            const char *infostr = (const char *)ed;
            printf("UME store: %s\n", infostr);
        }
        break;
    default:
            printf("Unknown source event %d\n", event);
            break;
    }
   return 0;
}
```

JAVA API

```
public int onSourceEvent(Object arg, LBMSourceEvent sourceEvent)
{
    switch (sourceEvent.type()) {
    case LBM.SRC_EVENT_UME_REGISTRATION_ERROR:
System.out.println("Error registering source with UME store: "
+ sourceEvent.dataString());
break;
    case LBM.SRC_EVENT_UME_REGISTRATION_SUCCESS_EX:
 UMESourceEventRegistrationSuccessInfo reg = sourceEvent.registrationSuccessInfo();
 System.out.print("UME store " + reg.storeIndex() + ": " + reg.store()
+ " registration success. RegID " + reg.registrationId() + ". Flags "
+ reg.flags() + " ");
                if (((reg.flags() &
LBM.SRC_EVENT_UME_REGISTRATION_SUCCESS_EX_FLAG_OLD))
                != 0) {
 System.out.print("OLD[SQN " + reg.sequenceNumber() + "] ");
                }
                if (((reg.flags() &
LBM.SRC_EVENT_UME_REGISTRATION_SUCCESS_EX_FLAG_NOACKS))
                != 0) {
System.out.print("NOACKS ");
                }
                System.out.println();
                break;
    case LBM.SRC_EVENT_UME_REGISTRATION_COMPLETE_EX:
                UMESourceEventRegistrationCompleteInfo regcomp =
                sourceEvent.registrationCompleteInfo();
                System.out.print("UME registration complete. SQN " +
regcomp.sequenceNumber()
+ ". Flags " + regcomp.flags() + " ");
                if ((regcomp.flags() &
                LBM.SRC_EVENT_UME_REGISTRATION_COMPLETE_EX_FLAG_QUORUM) != 0) {
System.out.print("QUORUM ");
                }
                System.out.println();
                break;
    case LBM.SRC_EVENT_UME_STORE_UNRESPONSIVE:
                System.out.println("UME store: "
                    + sourceEvent.dataString());
                break;
    ...
    default:
                System.out.println("Unknown source event "
+ sourceEvent.type());
                break;
        }
        return 0;
}
```

.NET API

```
public int onSourceEvent(Object arg, LBMSourceEvent sourceEvent)
{
    switch (sourceEvent.type()) {
    case LBM.SRC_EVENT_UME_REGISTRATION_ERROR:
System.Console.Out.WriteLine("Error registering source with UME store: "
+ sourceEvent.dataString());
break;
    case LBM.SRC_EVENT_UME_REGISTRATION_SUCCESS_EX:
 UMESourceEventRegistrationSuccessInfo reg = sourceEvent.registrationSuccessInfo();
 System.Console.Out.Write("UME store " + reg.storeIndex() + ": " + reg.store()
+ " registration success. RegID " + reg.registrationId() + ". Flags "
+ reg.flags() + " ");
                if (((reg.flags() &
LBM.SRC_EVENT_UME_REGISTRATION_SUCCESS_EX_FLAG_OLD))
                != 0) {
 System.Console.Out.Write("OLD[SQN " + reg.sequenceNumber() + "] ");
                }
                if (((reg.flags() &
LBM.SRC_EVENT_UME_REGISTRATION_SUCCESS_EX_FLAG_NOACKS))
                != 0) {
System.Console.Out.Write("NOACKS ");
                }
                System.Console.Out.WriteLine();
                break;
    case LBM.SRC_EVENT_UME_REGISTRATION_COMPLETE_EX:
                UMESourceEventRegistrationCompleteInfo regcomp =
                sourceEvent.registrationCompleteInfo();
                System.Console.Out.Write("UME registration complete. SQN " +
                regcomp.sequenceNumber()
+ ". Flags " + regcomp.flags() + " ");
                if ((regcomp.flags() &
                LBM.SRC_EVENT_UME_REGISTRATION_COMPLETE_EX_FLAG_QUORUM) != 0) {
System.Console.Out.Write("QUORUM ");
                }
                System.Console.Out.WriteLine();
                break;
    case LBM.SRC_EVENT_UME_STORE_UNRESPONSIVE:
                System.Console.Out.WriteLine("UME store: "
                      + sourceEvent.dataString());
                break;
    ...
    default:
                System.Console.Out.WriteLine("Unknown source event "
+ sourceEvent.type());
                break;
        }
        return 0;
}
```

Source Event Handler - Stability, Confirmation and Release

As shown in “Source Event Handler” on page 53 above, the Source Event Handler can be expanded to handle more source events by adding additional case statements. The following source code examples show case statements to handle message stability events, delivery confirmation events and message release (reclaim) events. See also “ UMP Events” on page 31.

C API

```
case LBM_SRC_EVENT_UME_MESSAGE_STABLE_EX:
/* requires that source ume_message_stability_notification attribute is enabled */
        {
            lbm_src_event_ume_ack_ex_info_t *info = (lbm_src_event_ume_ack_ex_info_t
*)ed;

            printf("UME store %u: %s message stable. SQN %x (msgno %d). Flags %x ",
            info->store_index, info->store,
                    info->sequence_number, (int)info->msg_clientd - 1, info->flags);
            if (info->flags &
LBM_SRC_EVENT_UME_MESSAGE_STABLE_EX_FLAG_INTRAGROUP_STABLE)
                    printf("IA "); /* Stable within store group */
            if (info->flags &
LBM_SRC_EVENT_UME_MESSAGE_STABLE_EX_FLAG_INTERGROUP_STABLE)
                    printf("IR "); /* Stable amongst all stores */
            if (info->flags & LBM_SRC_EVENT_UME_MESSAGE_STABLE_EX_FLAG_STABLE)
                    printf("STABLE ");  /* Just plain stable */
            if (info->flags & LBM_SRC_EVENT_UME_MESSAGE_STABLE_EX_FLAG_STORE)
                    printf("STORE ");   /* Stability reported by UME Store */
            printf("\n");
        }
        break;

case LBM_SRC_EVENT_UME_DELIVERY_CONFIRMATION_EX:
/* requires that source ume_confirmed_delivery_notification attribute is enabled */
        {
            lbm_src_event_ume_ack_ex_info_t *info = (lbm_src_event_ume_ack_ex_info_t
*)ed;

                printf("UME delivery confirmation. SQN %x, Receiver RegID %u (msgno
%d). Flags %x ",
                       info->sequence_number, info->rcv_registration_id,
                       (int)info->msg_clientd - 1, info->flags);
                if (info->flags &
LBM_SRC_EVENT_UME_DELIVERY_CONFIRMATION_EX_FLAG_UNIQUEACKS)
                    printf("UNIQUEACKS ");
                    /* Satisfied number of unique ACKs requirement */
                if (info->flags &
LBM_SRC_EVENT_UME_DELIVERY_CONFIRMATION_EX_FLAG_UREGID)
                    printf("UREGID ");
                    /* Confirmation contains receiver application registration ID */
                if (info->flags & LBM_SRC_EVENT_UME_DELIVERY_CONFIRMATION_EX_FLAG_OOD)
                    printf("OOD ");
                    /* Confirmation received from arrival order receiver */
                if (info->flags & LBM_SRC_EVENT_UME_DELIVERY_CONFIRMATION_EX_FLAG_EXACK)
                    printf("EXACK ");
                    /* Confirmation explicitly sent by receiver */
                printf("\n");
        }
        break;

case LBM_SRC_EVENT_UME_MESSAGE_RECLAIMED:
/* requires that source ume_confirmed_delivery_notification or
ume_message_stability_notification
attributes are enabled */
        {
                lbm_src_event_ume_ack_info_t *ackinfo = (lbm_src_event_ume_ack_info_t
*)ed;

                printf("UME message released - sequence number %x (msgno %d)\n",
                       ackinfo->sequence_number, (int)ackinfo->msg_clientd - 1);
        }
        break;
```

JAVA API

```
case LBM.SRC_EVENT_UME_MESSAGE_STABLE_EX:
// requires that source ume_message_stability_notification attribute is enabled
        UMESourceEventAckInfo staInfo = sourceEvent.ackInfo();
        System.out.print("UME store " + staInfo.storeIndex() + ": "
                    + staInfo.store() + " message stable. SQN " +
staInfo.sequenceNumber()
                    + " (msgno " + staInfo.clientObject() + "). Flags "
                    + staInfo.flags() + " ");
        if ((staInfo.flags() &
LBM.SRC_EVENT_UME_MESSAGE_STABLE_EX_FLAG_INTRAGROUP_STABLE)
        != 0) {
                System.out.print("IA "); // Stable within store group
        }
        if ((staInfo.flags() &
LBM.SRC_EVENT_UME_MESSAGE_STABLE_EX_FLAG_INTERGROUP_STABLE)
        != 0) {
                System.out.print("IR ");  // Stable amongst all stores
        }
        if ((staInfo.flags() & LBM.SRC_EVENT_UME_MESSAGE_STABLE_EX_FLAG_STABLE) != 0) {
                System.out.print("STABLE ");  // Just plain stable
        }
        if ((staInfo.flags() & LBM.SRC_EVENT_UME_MESSAGE_STABLE_EX_FLAG_STORE) != 0) {
                System.out.print("STORE ");   // Stability reported by UME Store
        }
        System.out.println();
        break;

case LBM.SRC_EVENT_UME_DELIVERY_CONFIRMATION_EX:
// requires that source ume_confirmed_delivery_notification attribute is enabled
        UMESourceEventAckInfo cdelvinfo = sourceEvent.ackInfo();
        System.out.print("UME delivery confirmation. SQN " + cdelvinfo.sequenceNumber()
                    + ", RcvRegID " + cdelvinfo.receiverRegistrationId() + " (msgno "
                    + cdelvinfo.clientObject() + "). Flags " + cdelvinfo.flags() + " ");
        if ((cdelvinfo.flags() &
LBM.SRC_EVENT_UME_DELIVERY_CONFIRMATION_EX_FLAG_UNIQUEACKS)
        != 0) {
                System.out.print("UNIQUEACKS "); // Satisfied number of unique ACKs
requirement
        }
        if ((cdelvinfo.flags() &
LBM.SRC_EVENT_UME_DELIVERY_CONFIRMATION_EX_FLAG_UREGID)
        != 0) {
                System.out.print("UREGID ");    // Confirmation contains receiver
application
                registration ID
        }
        if ((cdelvinfo.flags() & LBM.SRC_EVENT_UME_DELIVERY_CONFIRMATION_EX_FLAG_OOD)
        != 0) {
                System.out.print("OOD ");      // Confirmation received from arrival
order
                receiver
        }
        if ((cdelvinfo.flags() & LBM.SRC_EVENT_UME_DELIVERY_CONFIRMATION_EX_FLAG_EXACK)
        != 0) {
                System.out.print("EXACK ");    // Confirmation explicitly sent by
receiver
        }
        System.out.println();
        break;

case LBM.SRC_EVENT_UME_MESSAGE_RECLAIMED:
// requires that source ume_confirmed_delivery_notification or
// ume_message_stability_notification attributes are enabled
        System.out.println("UME message released - sequence number "
                + Long.toHexString(sourceEvent.sequenceNumber())
                + " (msgno "
                + Long.toHexString(((Integer)sourceEvent.clientObject()).longValue())
                + ")");
        break;
```

.NET API

```
case LBM.SRC_EVENT_UME_MESSAGE_STABLE_EX:
// requires that source ume_message_stability_notification attribute is enabled
            UMESourceEventAckInfo staInfo = sourceEvent.ackInfo();
            System.Console.Out.Write("UME store " + staInfo.storeIndex() + ": "
                        + staInfo.store() + " message stable. SQN " +
staInfo.sequenceNumber()
                        + " (msgno " + ((int)staInfo.clientObject()).ToString("x") +
").
                        Flags " + staInfo.flags() + " ");
            if ((staInfo.flags() &
LBM.SRC_EVENT_UME_MESSAGE_STABLE_EX_FLAG_INTRAGROUP_STABLE)
            != 0)
            {
                    System.Console.Out.Write("IA ");  // Stable within store group
            }
            if ((staInfo.flags() &
LBM.SRC_EVENT_UME_MESSAGE_STABLE_EX_FLAG_INTERGROUP_STABLE)
            != 0)
            {
                    System.Console.Out.Write("IR ");  // Stable amongst all stores
            }
            if ((staInfo.flags() & LBM.SRC_EVENT_UME_MESSAGE_STABLE_EX_FLAG_STABLE) !=
0)
            {
                    System.Console.Out.Write("STABLE ");  // Just plain stable
            }
            if ((staInfo.flags() & LBM.SRC_EVENT_UME_MESSAGE_STABLE_EX_FLAG_STORE) != 0)
            {
                    System.Console.Out.Write("STORE ");  // Stability reported by UME
Store
            }
            System.Console.Out.WriteLine();
            break;

case LBM.SRC_EVENT_UME_DELIVERY_CONFIRMATION_EX:
// requires that source ume_confirmed_delivery_notification attribute is enabled

            UMESourceEventAckInfo cdelvinfo = sourceEvent.ackInfo();

            System.Console.Out.Write("UME delivery confirmation. SQN " +
            cdelvinfo.sequenceNumber()
                        + ", RcvRegID " + cdelvinfo.receiverRegistrationId() + " (msgno
"
                        + ((int)cdelvinfo.clientObject()).ToString("x") + "). Flags " +
                        cdelvinfo.flags() + " ");
            if ((cdelvinfo.flags() &
            LBM.SRC_EVENT_UME_DELIVERY_CONFIRMATION_EX_FLAG_UNIQUEACKS) != 0)
            {
                    System.Console.Out.Write("UNIQUEACKS ");  // Satisfied number of
unique
                    ACKs requirement
            }
            if ((cdelvinfo.flags() &
LBM.SRC_EVENT_UME_DELIVERY_CONFIRMATION_EX_FLAG_UREGID)
            != 0)
            {
                    System.Console.Out.Write("UREGID ");  // Confirmation contains
receiver
                    application registration ID
            }
            if ((cdelvinfo.flags() &
LBM.SRC_EVENT_UME_DELIVERY_CONFIRMATION_EX_FLAG_OOD)
            != 0)
            {
                    System.Console.Out.Write("OOD ");  // Confirmation received from
arrival
                    order receiver
            }
            if ((cdelvinfo.flags() &
LBM.SRC_EVENT_UME_DELIVERY_CONFIRMATION_EX_FLAG_EXACK)
            != 0)
            {
                    System.Console.Out.Write("EXACK ");  // Confirmation explicitly
sent by
                    receiver
            }
            System.Console.Out.WriteLine();
            break;

case LBM.SRC_EVENT_UME_MESSAGE_RECLAIMED:
// requires that source ume_confirmed_delivery_notification or
// ume_message_stability_notification attributes are enabled

            System.Console.Out.WriteLine("UME message released - sequence number "
                               + sourceEvent.sequenceNumber().ToString("x")
                               + " (msgno "
                               + ((int)sourceEvent.clientObject()).ToString("x")
                               + ")");
            break;
```

Mapping Your Message Numbers to UMS/UMP Sequence Numbers

lbm_src_sendv_ex() allows you to create a pointer to an object or structure. This pointer will be returned to your application along with all source events. You can then update the object or structure with source event information. For example, if your messages exceed 8K - which requires fragmentation your application's message into more than one UM message - receiving sequence number events with this pointer allows you to determine all the UM sequence numbers for the message and, therefore, how many release (reclaim) events to expect. The following two source code examples show how to:

- Enable message sequence number information
- Handle sequence number source events to determine the application message number in the Source Event Handler

C API - Enable Message Information

```
lbm_src_send_ex_info_t exinfo;

/* Enable message sequence number info to be returned */
exinfo.flags = LBM_SRC_SEND_EX_FLAG_UME_CLIENTD |
LBM_SRC_SEND_EX_FLAG_SEQUENCE_NUMBER_INFO;
exinfo.ume_msg_clientd = (void *)(msgno + 1);
/* msgno set to application message number (can't evaluate to NULL) */
while (lbm_src_send_ex(src, message, msglen, 0, &exinfo) == LBM_FAILURE)
{
    if (lbm_errnum() == LBM_EUMENOREG)
        {
        printf("Send unsuccessful. Waiting...\n");
                SLEEP_MSEC(1000);    /* Sleep for 1 second */
        }
    else
    {
         fprintf(stderr, "lbm_src_send: %s\n", lbm_errmsg());
        break;
    }
}
```

C API - Sequence Number Event Handler

```
int handle_src_event(lbm_src_t *src, int event, void *ed, void *cd)
{
      switch (event) {
      case LBM_SRC_EVENT_SEQUENCE_NUMBER_INFO:
            {
                  lbm_src_event_sequence_number_info_t *info =
                  (lbm_src_event_sequence_number_info_t *)ed;

                  if (info->first_sequence_number != info->last_sequence_number) {
                        printf("SQN [%x,%x] (msgno %d)\n", info->first_sequence_number,
                        info->last_sequence_number, (int)info->msg_clientd - 1);
                  } else {
                        printf("SQN %x (msgno %d)\n", info->last_sequence_number,
                        (int)info->msg_clientd - 1);
                 }
            }
            break;
      ...
      }
      return 0;
}
```

JAVA API - Enable Message Information

```
LBMSourceSendExInfo exinfo = new LBMSourceSendExInfo();
exinfo.setClientObject(new Integer(msgno));  // msgno set to application message number
exinfo.setFlags(LBM.SRC_SEND_EX_FLAG_SEQUENCE_NUMBER_INFO);
// Enable message sequence number info to be returned
for (;;)
{
    try
    {
        src.send(message, msglen, 0, exinfo);
    }
    catch(UMENoRegException ex)
    {
            try
            {
                Thread.sleep(1000);
            }
            catch (InterruptedException e) { }
        continue;

    }
    catch (LBMException ex)
    {
            System.err.println("Error sending message: " + ex.toString());
    }
    break;
}
```

JAVA API - Sequence Number Event Handler

```
public int onSourceEvent(Object arg, LBMSourceEvent sourceEvent)
{
      switch (sourceEvent.type())
      {
            case LBM.SRC_EVENT_SEQUENCE_NUMBER_INFO:
                  LBMSourceEventSequenceNumberInfo info =
sourceEvent.sequenceNumberInfo();
                  if (info.firstSequenceNumber() != info.lastSequenceNumber()) {
                        System.out.println("SQN [" + info.firstSequenceNumber()
                            + "," + info.lastSequenceNumber() + "] (msgno "
                            + info.clientObject() + ")");
                  }
                  else {
                        System.out.println("SQN " + info.lastSequenceNumber()
                            + " (msgno " + info.clientObject() + ")");
                  }
                  break;
          ...
      }
      return 0;
}
```

.NET API - Enable Message Information

```
LBMSourceSendExInfo exinfo = new LBMSourceSendExInfo();
exinfo.setClientObject(msgno);  // msgno set to application message number
exinfo.setFlags(LBM.SRC_SEND_EX_FLAG_SEQUENCE_NUMBER_INFO);
// Enable message sequence number info to be returned
for (;;)
{
    try
    {
        src.send(message, msglen, 0, exinfo);
    }
    catch(UMENoRegException ex)
    {
            System.Threading.Thread.Sleep(100);
        continue;

    }
    catch (LBMException ex)
    {
            System.Console.Out.WriteLine("Error sending message: " + ex.Message());
    }
    break;
}
```

.NET API - Sequence Number Event Handler

```
public void onSourceEvent(Object arg, LBMSourceEvent sourceEvent)
{
      switch (sourceEvent.type())
      {
            case LBM.SRC_EVENT_SEQUENCE_NUMBER_INFO:
                  LBMSourceEventSequenceNumberInfo info =
sourceEvent.sequenceNumberInfo();
                  if (info.firstSequenceNumber() != info.lastSequenceNumber())
                  {
                        System.Console.Out.WriteLine("SQN [" +
info.firstSequenceNumber()
                              + "," + info.lastSequenceNumber() + "] (cd "
                              + ((int)info.clientObject()).ToString("x") + ")");
                  }
                  else
                  {
                        System.Console.Out.WriteLine("SQN " + info.lastSequenceNumber()
                              + " (msgno " + ((int)info.clientObject()).ToString("x") +
")");
                  }
                  break;
          ...

      }
      return 0;
}
```

Receiver Liveness Detection

As an extension to "Confirmed Delivery" on page 52, you can set receivers to send a keepalive to a source during a measured absence of delivery confirmations (due to traffic lapse). In the event that neither message reaches the source within a designated interval, or if the delivery confirmation TCP connection breaks down, the receiver is assumed to have "died". UM then notifies the publishing application via context event callback. This lets the publisher assign a new subscriber.

To use this feature, set these five configuration options:

- *`ume_source_liveness_timeout`*
- *`ume_receiver_liveness_interval`*
- *`ume_confirmed_delivery_notification`*
- *`ume_user_receiver_registration_id`*
- *`ume_session_id`*

Note: You must set the `ume_source_liveness_timeout` option to 5 times the value of `ume_receiver_liveness_interval`.

This specialized feature is not recommended for general use. If you are considering it, please note the following caveats:

- Do not use in conjunction with a UM Router.
- There is a variety of potential network occurrences that can break or reset the TCP connection and falsely indicate the death of a receiver.
- In cases where a receiver object is deleted while its context is not, the publisher may still falsely assume the receiver to be alive. Other false receiver-alive assumptions could be caused by the following:
- TCP connections can enter a half-open or otherwise corrupted state.
- Failed TCP connections sometimes do not fully close, or experience objectionable delays before fully closing.
- A switch or router failure along the path does not affect the TCP connection state.

UMP Receivers

Receivers are predominantly interested in RegID management and recovery management. This section discusses the following topics.

- "Receiver RegID Management" on page 62
- "Receiver Message and Event Handler" on page 63
- "Recovery Management" on page 65
- " Setting Callback Function to Set Recovery Sequence Number" on page 66
- "Message Consumption" on page 68

Receiver RegID Management

RegIDs are slightly more involved for receivers than for sources. Since RegIDs are per source per topic per store and a topic may have several sources, a receiver may have to manage several RegIDs per store in use. Fortunately, receivers in UMP can leverage the RegID of the source with the use of a callback as discussed in "Adding Fault Recovery with Registration IDs" on page 37 and shown in ume-example-rcv-2.c. Your application can determine the correct RegID to use and return it to UMP . You can also use Session IDs to enable UMP to manage receiver RegIDs. See "Managing RegIDs with Session IDs" on page 47.

Much like sources, receivers typically have a lifetime based on an amount of work, perhaps an infinite amount. And just like sources, it may be helpful to consider that a RegID is "assigned" at the start of that work and is out of use at the end. In between, the RegID is in use by the instance of the receiver application. However, the nature of RegIDs being per source means that the expected lifetime of a source should play a role in how RegIDs on the receiver are managed. Thus, it may be helpful for the application developer to consider the source application lifetime when deciding how best to handle RegIDs on the receiver.

Receiver Message and Event Handler

The Receiver Message and Event Handler is a function callback started at receiver initialization to provide Receiver messages to your application on behalf of the receiver. The following source code examples illustrate the use of a receiver message and event handler for registration messages. To accept other receiver events, additional case statements would be required, one for each additional event. See also “ UMP Events” on page 31.

C API

```
int rcv_handle_msg(lbm_rcv_t *rcv, lbm_msg_t *msg, void *clientd)
{
    switch (msg->type) {
    ...
    case LBM_MSG_UME_REGISTRATION_ERROR:
            printf("[%s][%s] UME registration error: %s\n", msg->topic_name, msg-
>source,
                msg->data);
            exit(0);
       break;
    case LBM_MSG_UME_REGISTRATION_SUCCESS:
            {
                    lbm_msg_ume_registration_t *reg = (lbm_msg_ume_registration_t *)
                        (msg->data);

                    printf("[%s][%s] UME registration successful. SrcRegID %u RcvRegID
%u\n",
                       msg->topic_name, msg->source, reg->src_registration_id,
                         reg->rcv_registration_id);
            }
            break;
    case LBM_MSG_UME_REGISTRATION_SUCCESS_EX:
            {
                    lbm_msg_ume_registration_ex_t *reg = (lbm_msg_ume_registration_ex_t
*)
                        (msg->data);

                    printf("[%s][%s] store %u: %s UME registration successful. SrcRegID
%u
                        RcvRegID %u. Flags %x ",
                   msg->topic_name, msg->source, reg->store_index, reg->store,
                   reg->src_registration_id, reg->rcv_registration_id, reg->flags);
                    if (reg->flags & LBM_MSG_UME_REGISTRATION_SUCCESS_EX_FLAG_OLD)
                        printf("OLD[SQN %x] ", reg->sequence_number);
                    if (reg->flags & LBM_MSG_UME_REGISTRATION_SUCCESS_EX_FLAG_NOCACHE)
                        printf("NOCACHE ");
                    printf("\n");
            }
            break;
    case LBM_MSG_UME_REGISTRATION_COMPLETE_EX:
            {
                    lbm_msg_ume_registration_complete_ex_t *reg;

reg  = (lbm_msg_ume_registration_complete_ex_t *)(msg->data);
printf("[%s][%s] UME registration complete. SQN %x. Flags %x ",
                       msg->topic_name, msg->source, reg->sequence_number, reg->flags);
                    if (reg->flags & LBM_MSG_UME_REGISTRATION_COMPLETE_EX_FLAG_QUORUM)
                        printf("QUORUM ");
                    if (reg->flags & LBM_MSG_UME_REGISTRATION_COMPLETE_EX_FLAG_RXREQMAX)
```

```
                        printf("RXREQMAX ");
                    printf("\n");
                }
               break;
    case LBM_MSG_UME_REGISTRATION_CHANGE:
            printf("[%s][%s] UME registration change: %s\n", msg->topic_name, msg-
>source,
                msg->data);
            break;
    ...
    default:
            printf("Unknown lbm_msg_t type %x [%s][%s]\n", msg->type, msg->topic_name,
                msg->source);
            break;
    }
    return 0;
}
```

JAVA API

```
public int onReceive(Object cbArg, LBMMessage msg)
{
    case LBM.MSG_UME_REGISTRATION_ERROR:
            System.out.println("[" + msg.topicName() + "][" + msg.source()
                    + "] UME registration error: " + msg.dataString());
            break;
    case LBM.MSG_UME_REGISTRATION_SUCCESS_EX:
            UMERegistrationSuccessInfo reg = msg.registrationSuccessInfo();
            System.out.print("[" + msg.topicName() + "][" + msg.source()
                    + "] store " + reg.storeIndex() + ": "
                    + reg.store() + " UME registration successful. SrcRegID "
                    + reg.sourceRegistrationId() + " RcvRegID " +
reg.receiverRegistrationId()
                    + ". Flags " + reg.flags() + " ");
            if ((reg.flags() & LBM.MSG_UME_REGISTRATION_SUCCESS_EX_FLAG_OLD) != 0)
                System.out.print("OLD[SQN " + reg.sequenceNumber() + "] ");
            if ((reg.flags() & LBM.MSG_UME_REGISTRATION_SUCCESS_EX_FLAG_NOCACHE) != 0)
                System.out.print("NOCACHE ");
            System.out.println();
            break;
    case LBM.MSG_UME_REGISTRATION_COMPLETE_EX:
            UMERegistrationCompleteInfo regcomplete = msg.registrationCompleteInfo();
            System.out.print("[" + msg.topicName() + "][" + msg.source()
                    + "] UME registration complete. SQN " + regcomplete.sequenceNumber()
                    + ". Flags " + regcomplete.flags() + " ");
            if ((regcomplete.flags() &
LBM.MSG_UME_REGISTRATION_COMPLETE_EX_FLAG_QUORUM)
            != 0) {
                System.out.print("QUORUM ");
            }
            if ((regcomplete.flags() &
LBM.MSG_UME_REGISTRATION_COMPLETE_EX_FLAG_RXREQMAX)
            != 0) {
                System.out.print("RXREQMAX ");
            }
            System.out.println();
            break;
    case LBM.MSG_UME_REGISTRATION_CHANGE:
            System.out.println("[" + msg.topicName() + "][" + msg.source()
                    + "] UME registration change: " + msg.dataString());
            break;
    ...
    default:
            System.err.println("Unknown lbm_msg_t type " + msg.type() + " ["
                    + msg.topicName() + "][" + msg.source() + "]");
            break;
        }
    return 0;
    }
```

.NET API

```
    public int onReceive(Object cbArg, LBMMessage msg)
    {
        case LBM.MSG_UME_REGISTRATION_ERROR:
                System. Console.Out.WriteLine("[" + msg.topicName() + "][" + msg.source()
                        + "] UME registration error: " + msg.dataString());
                break;
        case LBM.MSG_UME_REGISTRATION_SUCCESS_EX:
                UMERegistrationSuccessInfo reg = msg.registrationSuccessInfo();
                System.Console.Out.Write("[" + msg.topicName() + "][" + msg.source()
                        + "] store " + reg.storeIndex() + ": "
                        + reg.store() + " UME registration successful. SrcRegID "
                        + reg.sourceRegistrationId() + " RcvRegID " +
    reg.receiverRegistrationId()
                        + ". Flags " + reg.flags() + " ");
                if ((reg.flags() & LBM.MSG_UME_REGISTRATION_SUCCESS_EX_FLAG_OLD) != 0)
                        System.Console.Out.Write ("OLD[SQN " + reg.sequenceNumber() + "] ");
                if ((reg.flags() & LBM.MSG_UME_REGISTRATION_SUCCESS_EX_FLAG_NOCACHE) != 0)
                        System.Console.Out.Write ("NOCACHE ");
                System.Console.Out.WriteLine();
                break;
        case LBM.MSG_UME_REGISTRATION_COMPLETE_EX:
                UMERegistrationCompleteInfo regcomplete = msg.registrationCompleteInfo();
                System.Console.Out.Write("[" + msg.topicName() + "][" + msg.source()
                        + "] UME registration complete. SQN " + regcomplete.sequenceNumber()
                        + ". Flags " + regcomplete.flags() + " ");
                if ((regcomplete.flags() &
    LBM.MSG_UME_REGISTRATION_COMPLETE_EX_FLAG_QUORUM)
                != 0) {
                    System.Console.Out.Write("QUORUM ");
                }
                if ((regcomplete.flags() &
    LBM.MSG_UME_REGISTRATION_COMPLETE_EX_FLAG_RXREQMAX)
                != 0) {
                    System.Console.Out.Write("RXREQMAX ");
                }
    System.Console.Out.WriteLine();
                break;
        case LBM.MSG_UME_REGISTRATION_CHANGE:
                System.Console.Out.WriteLine("[" + msg.topicName() + "][" + msg.source()
                        + "] UME registration change: " + msg.dataString());
                break;
        ...
        default:
                System.Console.Out.WriteLine("Unknown lbm_msg_t type " + msg.type() + " ["
                        + msg.topicName() + "][" + msg.source() + "]");
                break;
            }
        return 0;
    }
```

Recovery Management

Recovery management for receivers is fairly simple because UMP requests any missing messages from the store(s) and delivers them as they are retransmitted. However, your application can specify a different message to begin retransmission with using either the `retransmit_request_maximum` configuration option or **lbm_ume_rcv_recovery_info_ex_func_t**.

For example, assume a source sends 7 messages with sequence numbers 0-6 which are stabilized at the store. The receiver, configured with the *`retransmit_request_maximum`* set to 2, consumes message 0, goes down, then comes back at message 6. **lbm_ume_rcv_recovery_info_ex_func_t** returns the following:

```
high_sequence_number = 6
low_rxreq_max_sequence_number = 4
low_sequence_number = 1

NOTE: low_rxreq_max_sequence_number = high_sequence_number - retransmit_request_maximum
```

- UMP obeys the *`retransmit_request_maximum`* configuration option and restarts with message 4. This is the default.
- If you modify the `low_sequence_number` to satisfy some other requirements, you can override the configuration option and restart at message 0, 2, 3, 5 or 6. See “ Setting Callback Function to Set Recovery Sequence Number” on page 66 below.
- The only way to restart at message 1 in this case, is to set the *`retransmit_request_maximum`* configuration option to its default value of 0. If your application changes the `low_sequence_number` and for whatever reason, the calculation results in the same value as the `low_sequence_number`, UMP ignores the calculation and restarts with message 4.

All messages retransmitted to a receiver are marked as retransmissions via a flag in the message structure. Thus it is easy for an application to determine if a message is a new message from the source or a retransmission, which may or may not have been processed before the failure. The presence or absence of the retransmit flag gives the application a hint of how best to handle the message with regard to it being processed previously or not.

Informatica recommends that you always check the data or other message properties of messages with the retransmit flag set to be sure the message has not been already processed. Relying on UMP sequence numbers is not a reliable method for detecting duplicate messages.

Setting Callback Function to Set Recovery Sequence Number

The sample source code below demonstrates how to use the recovery sequence number info function to determine the stored message with which to restart a receiver. This method retrieves the low sequence number from the recovery sequence number structure and adds an offset to determine the beginning sequence number. The offset is a value completely under the control of your application. For example, if a receiver was down for a "long" period and you only want the receiver to receive the last 10 messages, use an offset to start the receiver with the 10th most recent message. If you wish not to receive any messages, set the `low_sequence_number` to the `high_sequence_number` plus one.

C API

```
lbm_ume_rcv_recovery_info_ex_func_t cb;

cb.func = ume_rcv_seqnum_ex;
cb.clientd = NULL;
if (lbm_rcv_topic_attr_setopt(&rcv_attr, "ume_recovery_sequence_number_info_function",
&cb, sizeof(cb)) == LBM_FAILURE) {
            fprintf(stderr,
            "lbm_rcv_topic_attr_setopt:ume_recovery_sequence_number_info_function: %s
\n",
            lbm_errmsg());
            exit(1);
}
printf("Will use seqnum info with low offset %u.\n", seqnum_offset);

int ume_rcv_seqnum_ex(lbm_ume_rcv_recovery_info_ex_func_info_t *info, void *clientd)
{
        lbm_uint_t new_lo = info->low_sequence_number + seqnum_offset;
        printf("[%s] SQNs Low %x (will set to %x), Low rxreqmax %x, High %x (CD %p)\n",
info->source, info->low_sequence_number,
                new_lo, info->low_rxreq_max_sequence_number, info-
>high_sequence_number,
info->source_clientd);
        info->low_sequence_number = new_lo;
        return 0;
}
```

JAVA API

```
UMERcvRecInfo umerecinfocb = new UMERcvRecInfo(seqnum_offset);
rcv_attr.setRecoverySequenceNumberCallback(umerecinfocb, null);
System.out.println("Will use seqnum info with low offset " + seqnum_offset + ".");

class UMERcvRecInfo implements UMERecoverySequenceNumberCallback {
    private long _seqnum_offset = 0;

    public UMERcvRecInfo(long seqnum_offset) {
        _seqnum_offset = seqnum_offset;
    }

    public int setRecoverySequenceNumberInfo(Object cbArg,
    UMERecoverySequenceNumberCallbackInfo cbInfo)
    {
long new_low = cbInfo.lowSequenceNumber() + _seqnum_offset;
if (new_low < 0) {
System.out.println("New low sequence number would be negative.
Leaving low SQN unchanged.");
new_low = cbInfo.lowSequenceNumber();
            }
            System.out.println("SQNs Low " + cbInfo.lowSequenceNumber() + " (will set
to "
                    + new_low + "), Low rxreqmax " + cbInfo.lowRxReqMaxSequenceNumber()
                    + ", High " + cbInfo.highSequenceNumber());
            try {
                    cbInfo.setLowSequenceNumber(new_low);
            }
            catch (LBMEInvalException e) {
                    System.err.println(e.getMessage());
           }
            return 0;
 }
```

.NET API

```
UMERcvRecInfo umerecinfocb = new UMERcvRecInfo(seqnum_offset);
rcv_attr.setRecoverySequenceNumberCallback(umerecinfocb, null);
System.Console.Out.WriteLine("Will use seqnum info with low offset " + seqnum_offset +
".");

class UMERcvRecInfo implements UMERecoverySequenceNumberCallback {
    private long _seqnum_offset = 0;

    public UMERcvRecInfo(long seqnum_offset) {
        _seqnum_offset = seqnum_offset;
    }

    public int setRecoverySequenceNumberInfo(Object cbArg,
    UMERecoverySequenceNumberCallbackInfo cbInfo)
    {
long new_low = cbInfo.lowSequenceNumber() + _seqnum_offset;
if (new_low < 0) {
System.Console.Out.WriteLine ("New low sequence number would be negative.
Leaving low SQN unchanged.");
new_low = cbInfo.lowSequenceNumber();
            }
```

```
                System.Console.Out.WriteLine ("SQNs Low " + cbInfo.lowSequenceNumber() + "
                    (will set to "
                        + new_low + "), Low rxreqmax " + cbInfo.lowRxReqMaxSequenceNumber()
                        + ", High " + cbInfo.highSequenceNumber());
                try {
                        cbInfo.setLowSequenceNumber(new_low);
                }
                catch (LBMEInvalException e) {
                        System.Console.Out.WriteLine (e.getMessage());
               }
                return 0;
     }
```

Message Consumption

Receivers use message consumption, defined as message deletion, to indicate that UMP should notify the store(s) that the application consumed the message. This notification takes the form of an acknowledgement, or ACK, to the store(s) in use as well as to the source if you configured the source for *delivery confirmation*.

- In the *C API*, message deletion happens by default when the receive callback returns, unless the callback uses **lbm_msg_retain()**. If the callback uses **lbm_msg_retain()** then the application has responsibility to use **lbm_msg_delete()** when it has finished processing the message.
- In the *Java API* and *.NET API*, message deletion must be triggered explicitly by the application by using the `dispose()` method of the message object. Without explicit usage of `dispose()`, UMP does not know when the application has finished processing the message.

Batching Acknowledgments

You can configure UMP to acknowledge message consumption to a store(s) for a series of messages independent of when the receiving application consumed the messages. This option works well if multiple threads process messages off of an event queue, which may result in messages being consumed out of order. This feature is not compatible with Explicit Acknowledgments.

If you set *`ume_use_ack_batching`* to 1, UMP does not acknowledge individual messages as the application consumes them. Instead, UMP checks the consumed, but unacknowledged messages at the interval configured with *`ume_ack_batching_interval`* . When UMP discovers a contiguous series of consumed message sequence numbers (`sqn`), it sends acknowledgments to the store(s) for all the contiguous messages.

For example, assume your application consumes and acknowledges messages 1 and 2, then consumes subsequent messages in the following order: 4, 5, 7, 8, 6, 10, 3. At the next *`ume_ack_batching_interval`* , UMP sends consumption acknowledgments to the store(s) for messages 3 - 8.

Explicit Acknowledgments

In addition, UMP supports Explicit ACKs (*`ume_explicit_ack_only`*), which silences UMP's acknowledgement behavior, allowing your application control of message consumption notification. See also **lbm_msg_ume_send_explicit_ack()** in the *C API* and the `LBMMessage` class method **sendExplicitAck()** in the *Java API* and *.NET API*.

The explicit ACK sending function/method automatically supplies additional ACKs for missing messages in sequence number gaps. This can be a useful efficiency feature, but note that to acknowledge each message consumption individually, you must issue their ACKs in ascending sequence-number order.

Object-free Explicit Acknowledgments

When using explicit ACKs in your *.NET* application, you can extract ACK information from messages and then send acknowledgements to the store(s) for any sequence number. You can also extract ACK information from a message when using the *C API* with **lbm_msg_extract_ume_ack()**.

The following source code examples show how to extract ACK information and send an explicit ACK.

C API

```
int rcv_handle_msg(lbm_rcv_t *rcv, lbm_msg_t *msg, void *clientd)
{
    lbm_ume_rcv_ack_t *ack = NULL;
...

    ack = lbm_msg_extract_ume_ack(msg);
    lbm_ume_ack_send_explicit_ack(ack, msg->sequence_number);
    lbm_ume_ack_delete(ack);
...

}
```

JAVA API or .NET API

```
public int onReceive(Object cbArg, LBMMessage msg)
{
    UMEMessageAck ack;
...

    ack = msg.extractUMEAck();
    ack.sendExplicitAck(msg.sequenceNumber());
    ack.dispose();

...

}
```

UMP Stores

As mentioned in “Persistent Store” on page 3, the UMP persistent stores, also just called stores, actually persist the source and receiver state and use RegIDs to identify sources and receivers. Each source to which a store provides persistence may have zero or more receivers. The store maintains each receiver's state along with the source's state and the messages the source has sent.

The store can be configured with its own set of *options* to persist this state information on disk or simply in memory. The term *disk store* is used to signify a store that persists state to disk, and the term *memory store* is used to signify a store that persists state only in memory. A store may also be configured not to cache the source's data, but to simply persist the source and receiver state in memory. This is called a *no-cache store*.

Unlike many persistent systems, the persistent store in UMP is not in the message path. In other words, a source does not send data to the store and then have the store forward it to the receivers. In UMP, the source sends to receivers and the stores in parallel. See “Normal Operation” on page 17. Thus, UMP can provide extremely low latency to receiving applications.

The store(s) that a source uses are part of the source's configuration settings. Sources must be configured to use specific store(s) and use one of two different types of store failover behaviors to match expected failure scenarios:

- Round-Robin Store Usage, which is deprecated
- Quorum/Consensus Store Usage

Receivers, on the other hand, do not need to be configured with store information a priori. The source provides store information to receivers via a Source Registration Information (SRI) message after the source registers with a store. Thus the receivers learn about stores from the source, without needing to be configured themselves. Because receivers learn about the store or stores with which they must register via a SRI record, the source must be available to receivers. However, the source does not have to be actively sending data to do this.

Round-Robin Store Usage

Stores can be used in a round-robin fashion by a source during failover. A source provides UMP with a list of stores to use. The first is the primary, the second is the secondary, the third is the tertiary, etc. The source uses a single store at any one time. If the currently active store becomes unresponsive due to a crash or network disconnect, UMP tries other stores in the list one by one until it finds a responsive store.

With round-robin store usage, inactive stores do not receive data from the source. Thus, a store that becomes the active store will not have any data from the source. In this case, the source may be configured to retain messages and stream those messages to the new store using Late Join. Cascading failures of sources, stores and receivers may require using stores in a Quorum/Consensus fashion.

Note: The Round-Robin Store failover configuration is deprecated.

Sources Using Round-Robin Store Configuration

The source retains messages until they are considered stable at the active store(s). For Round-Robin store behavior, this means the current active store notifies the source that it has stabilized the message via a message stability notification. The following configuration file statements implement Round-Robin behavior among 3 stores.

```
source ume_store 10.29.3.77:15313:150000:0
source ume_store 10.29.3.76:16313:160000:0
source ume_store 10.29.3.75:17313:170000:0
source ume_message_stability_notification 1
source ume_store_behavior rr
```

Quorum/Consensus Store Usage

To provide the highest degree of resiliency in the face of failures, UMP provides the Quorum/Consensus failover strategy which allows a source to provide UMP with a number of stores to be used at the same time. Multiple stores can fail and UMP can continue operation unhindered. Moreover, Late Join is not needed as in Round-Robin.

Quorum/Consensus, also called QC, allows a source and the associated receivers to have their persisted state maintained at several stores at the same time. Central to QC is the concept of a group of stores, which is a logical grouping of stores that are intended to signify a single entity of resilience. Within the group, individual stores may fail but for the group as a whole to be viable and provide resiliency, a quorum must be available. In UMP , a quorum is a simple majority. For example, in a group of five stores, three stores are required to maintain a quorum. One or two stores may fail and the group continues to provide resiliency. UMP requires a source to have a quorum of stores available in the group in order to send messages. A group can consist of a single store.

QC also provides the ability to use multiple groups. As long as a single group maintains quorum, then UMP allows a source to proceed. Groups are logical in nature and can be combined in any way imaginable, such as by store location, store type, etc. In addition, QC provides the ability to specify backup stores within groups. Backups may be used if or when a store in the group becomes unresponsive to the source. Quorum/Consensus allows a source many different failure scenarios simply not available in other persistent messaging systems.

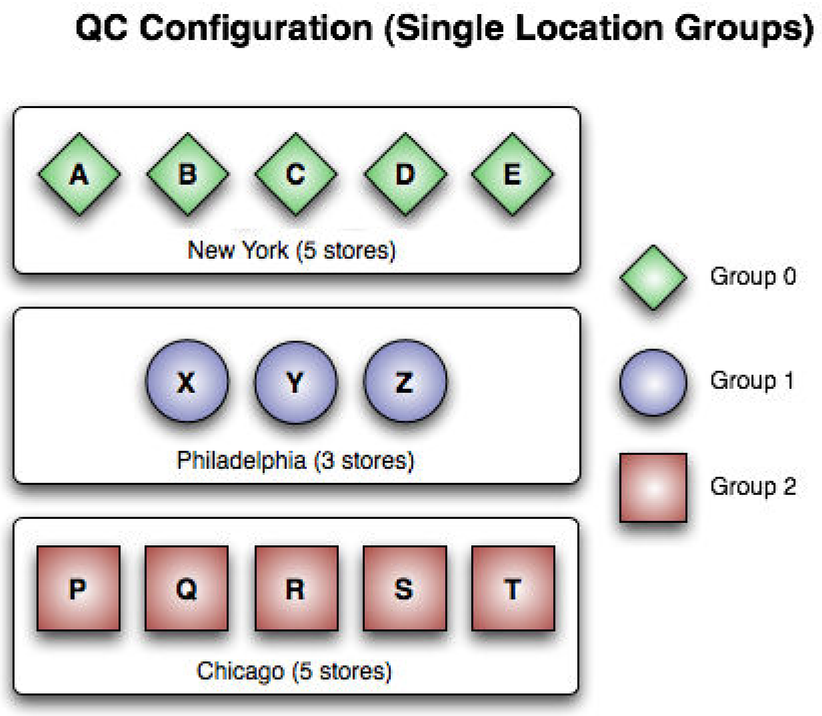
Sources Using Quorum/Consensus Store Configuration

In the case of Quorum/Consensus store behavior, a message is considered stable after it has been successfully stored within a group of stores or among groups of stores according to the two settings, intergroup behavior and intragroup behavior, described below.

- The *intragroup behavior* specifies the requirements needed to stabilize a message among the stores within a group. A message is stable for the group once it is successfully stored at a quorum (majority) of the group's stores or successfully stored in all the stores in the group.
- The *intergroup behavior* specifies the requirements needed to stabilize a message among groups of stores. A message is stable among the groups if it is successfully stored at any group, a majority of groups, all groups, or all active groups.

Notice that a message needs to meet intragroup stability requirements before it can meet intergroup stability requirements. These options provide a number of possibilities for retention of messages for the source.

The following figure displays a 3-group Quorum/Consensus configuration with each group in a different location. A message is considered stable when it has been successfully stored at a quorum of stores in all the active groups. The UM configuration file statements appear below the figure.



```
source ume_store 10.29.3.77:10313:101000:0
source ume_store 10.29.3.77:11313:110000:0
source ume_store 10.29.3.77:12313:120000:0
source ume_store 10.29.3.77:13313:130000:0
source ume_store 10.29.3.77:14313:140000:0
source ume_store 10.29.3.78:15313:150000:1
source ume_store 10.29.3.78:16313:160000:1
source ume_store 10.29.3.78:17313:170000:1
source ume_store 10.29.3.79:18313:180000:2
source ume_store 10.29.3.79:19313:190000:2
source ume_store 10.29.3.79:29313:290000:2
source ume_store 10.29.3.79:39313:390000:2
source ume_store 10.29.3.79:49313:490000:2

source ume_message_stability_notification 1
source ume_store_behavior qc

source ume_store_group 0:5
source ume_store_group 1:3
source ume_store_group 2:5

source ume_retention_intragroup_stability_behavior quorum
source ume_retention_intergroup_stability_behavior all-active
```

Fault Recovery

Recovery from source and receiver failure is the real heart of UMP operation. For a source, this means continuing operation from where it stopped. For a receiver, this means essentially the same thing, but with the retransmission of missed messages. Application developers can easily leverage the information in UMP to make their applications recover from failure in graceful ways.

Late Join is the mechanism of UMP recovery as well as an UM streaming feature. If Late Join is turned off on a source (`late_join`) or receiver (`use_late_join`), it also turns off UMP recovery. In order to control Late Join behavior, UMP provides a mechanism for a receiver to control the low sequence number. See “Recovery Management” on page 65.

Not all failures are recoverable. For application developers it usually pays in the long run to identify what types of errors are non-recoverable and how best to handle them when possible. Such an exercise establishes the precise boundaries of expected versus abnormal operating conditions.

Note: UMP does not acknowledge messages that are lost. If the store is unable to recover a lost message, any receivers attempting to recover this message from the store will experience unrecoverable loss as well. Sources can pay attention to any gaps in stability or confirmed delivery acknowledgements as these most likely represent unrecoverable loss at the store or receivers, respectively.

This section discussed the following recovery topics.

- “Source Recovery” on page 72
- “Receiver Recovery” on page 73

Source Recovery

The following shows the basic steps of source recovery.

1. Re-register with the store.
2. Determine the highest sequence number that the store has from the source.
3. Resume sending with the next sequence number.

Because UMP allows you to stream messages and not wait until a message is stable at the persistent store before sending the next message, the main task of source recovery is to determine what messages the persistent store(s) have and what they don't. Therefore, when a source re-registers with a store during recovery, the store tells the source what sequence number it has as the most recent from the source. The registration event informs the application of this sequence number. See “Source Event Handler” on page 53.

In addition, a mechanism exists (`LBM_SRC_EVENT_SEQUENCE_NUMBER_INFO`) that allows the application to know the sequence number assigned to every piece of data it sends. The combination of registration and sequence number information allows an application to know exactly what a store does have and what it does not and where it should pick up sending. An application designed to stream data in this way should consider how best to maintain this information.

When QC is in use, UMP uses the consensus of the group(s) to determine what sequence number to use in the first message it will send. This is necessary as not all stores can be expected to be in total agreement about what was sent in a distributed system. The application can configure the source with the *ume_consensus_sequence_number_behavior* to use the lowest sequence number of the latest group of sequence numbers seen from any store, the highest, or the majority. In most cases, the majority, which is the default, makes the most sense as the consensus. The lowest is a very conservative setting. And the highest is somewhat optimistic. Your application has the flexibility to handle this in any way needed.

If streaming is not what an application desires due to complexity, then it is very simple to use the UMP events (“ UMP Events” on page 31) delivered to the application to mimic the behavior of restricting a source to having only one unstable message at a time.

Receiver Recovery

The following shows the basic steps of receiver recovery.

1. Re-register with the store.
2. Determine the low sequence number.
3. Request retransmission of messages starting with the low sequence number.

UMP provides extensive options for controlling how receivers handle recovery. By default, receivers want to restart after the last piece of data that was consumed prior to failure or graceful suspension. Since UMP persists receiver state at the store, receivers request this state from the store as part of re-registration and recovery. Receiving applications experiencing unrecoverable loss can potentially retrieve missed messages from the stores by deleting and recreating the receiver object.

The actual sequence number that a receiver uses as the first topic level message to resume reception with is called the "low sequence number". UMP provides a means of modifying this sequence number if desired. An application can decide to use the sequence number as is, to use an even older sequence number, to use a more recent sequence number, or to simply use the most recent sequence number from the source. See “Recovery Management” on page 65 and “ Setting Callback Function to Set Recovery Sequence Number” on page 66. This allows receivers great flexibility on a per source basis when recovering. New receivers, receivers with no pre-existing registration, also have the same flexibility in determining the sequence number to begin data reception.

Like sources, when QC is in use, UMP uses the consensus of the group(s) to determine the low sequence number. And as with sources, this is necessary as not all stores can be expected to be in total agreement about what was acknowledged. The application can configure the receiver with *ume_consensus_sequence_number_behavior* to use the lowest sequence number of the latest group of sequence numbers seen from any store, the highest, or the majority. In most cases, the majority, which is the default, makes the most sense as the consensus. The lowest is a very conservative setting. And the highest is somewhat optimistic. In addition, this sequence number may be modified by the application after the consensus is determined.

For QC, UMP load balances receiver retransmission requests among the available stores. In addition, if requests are unanswered, retransmissions of the actual requests will use different stores. This means that as long as a single store has a message, then it is possible for that message to be retransmitted to a requesting receiver.

Note: Receivers need to consider if the use of arrival order delivery is appropriate. See *ordered_delivery* . UMP stores save the highest sequence number acknowledged by a receiver. When receivers using arrival order delivery receive - and thereby acknowledge - messages out of order, recovery problems may arise because stores will not have earlier messages not acknowledged by the receiver.

CHAPTER 8

Fault Tolerance

This chapter includes the following topics:

UMP Message Loss Recovery

UMP offers the following message recovery mechanisms.

Table 1. Message Loss Recovery Mechanisms

Method	Product	Transport	Description
Negative Acknowledgments (NAKs)	UMS, UMP, UMQ	LBT-RM, LBT-RU	Recovers lost transport datagrams from the source which may contain many small topic messages or fragments of a large message. Receivers send unicast NAKs to the source for missed transport datagrams. Source retransmits datagrams over the configured UM transport.
Late Join	UMS, UMP, UMQ	All	Retransmits messages via unicast to receivers joining the stream after the messages were originally sent. See *UM Concepts Guide, Using Late Join*.
Durable Receiver Recovery	UMP, UMQ	All	Recovers messages persisted while a durable receiver was offline. UMP initiates recovery when a durable receiver joins a persistent stream. The receiver then requests retransmission from the store starting with the low sequence number, defined as the last message it acknowledged to the store plus one. The store unicasts retransmissions. See "Receiver Recovery" on page 73

Method	Product	Transport	Description
Off Transport Recovery	UMS, UMP, UMQ	All	Recovers lost topic messages. Receiver detects lost sequence number and requests retransmission from the source or persistent stores (if applicable). UM unicasts retransmissions. See *UM Concepts Guide, Off Transport Recovery*.
Proactive Retransmissions	UMP, UMQ	All	Recovers lost messages never received by the store/queue or never acknowledged by the store/queue. Operates independently of any receivers. Source unicasts retransmissions. See “ Proactive Retransmissions” on page 82.

Configuring for Persistence and Recovery

Deployment decisions play a huge role in the success of any persistent system. Configuration in UMP has a number of options that aid in performance, fault recovery, and overall system stability. It is not possible, or at least not wise, to totally divorce configuration from application development for high performance systems. This is true not only for persistent systems, but for practically all distributed systems. When designing systems, deployment considerations need to be taken into account for the following:

- Source Considerations
- Receiver Considerations
- Store Configuration Considerations

Source Considerations

Performance of sources is heavily impacted by:

- the release policy that the source uses
- streaming methods of the source
- the throughput and latency requirements of the data

Source release settings have a direct impact on memory usage. As messages are retained, they consume memory. You reclaim memory when you release messages. Message stability, delivery confirmation and retention size all interact to create your release policies. UMP provides a hard limit on the memory usage. When exceeded, UMP delivers a Forced Reclamation event. Thus applications that anticipate forced reclamations can handle them appropriately. See also “ Source Message Retention and Release” on page 50.

How the source streams data has a direct impact on latency and throughput. One streaming method sets a maximum, outstanding count of messages. Once reached, the source does not send any more until message stability notifications come in to reduce the number of outstanding messages. The `umesrc` example program uses this mechanism to limit the speed of a source to something a store can handle comfortably. This also provides a maximum bound on recovery that can simplify handling of streaming source recovery.

The throughput and latency requirements of the data are normal UM concerns. See *Ultra Messaging Concepts* .

Receiver Considerations

In addition to the following, receiver performance shares the same considerations as receivers during normal operation.

Acknowledgement Generation

Receivers in a persistence implementation of UMP send an a message consumption acknowledgement to stores and the message source. Some applications may want to control this acknowledgement explicitly themselves. In this case, *`ume_explicit_ack_only`* can be used.

Controlling Retransmission

Receivers in UMP during fault recovery are another matter entirely. Receivers send retransmission requests and receive and process retransmissions. Control over this process is crucial when handling very long recoveries, such as hundreds of thousands or millions of messages. A receiver only sends a certain number of retransmission requests at a time.

This means that a receiver will not, unless configured to with *`retransmit_request_outstanding_maximum`* , request everything at once. The value of the low sequence number ("Receiver Recovery" on page 73) has a direct impact on how many requests need to be handled. A receiving application can decide to only handle the last X number of messages instead of recovering them all using the option, *`retransmit_request_maximum`* . The timeout used between requests, if the retransmission does not arrive, is totally controllable with *`retransmit_request_interval`* . And the total time given to recover all messages is also controllable.

Recovery Process

Theoretically, receivers can handle up to roughly 2 billion messages during recovery. This limit is implied from the sequence number arithmetic and not from any other limitation. For recovery, the crucial limiting factor is how a receiver processes and handles retransmissions which come in as fast as UMP can request them and a store can retransmit them. This is perhaps much faster than an application can handle them. In this case, it is crucial to realize that as recovery progresses, the source may still be transmitting new data. This data will be buffered until recovery is complete and then handed to the application. It is prudent to understand application processing load when planning on how much recovery is going to be needed and how it may need to be configured within UMP .

Store Configuration Considerations

UMP stores have numerous configuration options. See Chapter 10, "Configuration Reference for Umestored" on page 86. This section presents issues relating to these options.

Configuring Store Usage per Source

A store handles persisted state on a per topic per source basis. Based on the load of topics and sources, it may be prudent to spread the topic space, or just source space, across stores as a way to handle large loads. As configuration of store usage is per source, this is extremely easy to do. It is easy to spread CPU load via multi-threading as well as hard disk usage across stores. A single store process can have a set of virtual stores within it, each with their own thread.

Disk vs. Memory

As mentioned previously in “ UMP Stores” on page 69, stores can be memory based or disk based. Disk stores also have the ability to spread hard disk usage across multiple physical disks by using multiple virtual stores within a single store process. This gives great flexibility on a per source basis for spreading data reception and persistent data load.

UMP stores provide settings for controlling memory usage and for caching messages for retransmission in memory as well as on disk. All messages in a store, whether in memory or on disk, have some small memory state. This is roughly about 72 bytes per message. For very large caches of messages, this can become non-trivial in size.

Activity Timeouts

UMP stores are NOT archives and are not designed for archival. Stores persist source and receiver state with the aim of providing fault recovery. Central to this is the concept that a source or receiver has an activity timeout attached to it. Once a source or receiver suspends operation or has a failure, it has a set time before the store will forget about it. This activity timeout needs to be long enough to handle the recovery demands of sources and receivers. However, it can not and should not be infinite. Each source takes up memory and disk space, therefore an appropriate timeout should be chosen that meets the requirements of recovery, but is not excessively long so that the limited resources of the store are exhausted.

Recommendations for Store Configuration

The following conditions allow sources to continue to send messages.

- Quorum - Completed registration of a quorum of stores within at least one group. This is affected by group definitions, plus intragroup and intergroup stability settings. See also “ UMP Stores” on page 69
- Flight Size - Maximum number of messages sent but not stable which is determined by store group definitions, intragroup and intergroup stability settings and delivery confirmation setting. See also “ UMP Flight Size” on page 18

Configure your stores to address the failure cases you believe are more probable and from which you want to recover. For example, if a particular store group persists topics of higher importance, you may want to increase the number of stores in that group to maintain quorum in the face of a store failure. Or if a particular location has a higher incidence of failures than other locations, you may want to add additional stores in other locations.

Although many different conditions and requirements can apply to the configuration of persistent stores, Informatica recommends the following best practices.

1. **Minimum of 3 stores** - Requiring a minimum of 3 stores needed for quorum in a single store group is optimal. Using 5 stores, for example, in a group allows sources to keep sending in the face of the loss of up to 2 stores.
2. **Multiple store groups** - When using multiple store groups, Informatica recommends using at least 3 stores in each group.

3. **Set Intergroup Stability to** `all-active`. This setting for *`ume_retention_intergroup_stability_behavior`* provides a more immediate evaluation of your store configuration. Active groups must have at least a quorum of active stores, registered with the source and sending stability acknowledgements for persisted messages. By default, if a store becomes unresponsive, a store group could lose quorum and therefore messages in-flight cannot be stabilized by the unresponsive store's group until the store's *`ume_store_activity_timeout`* expires (default of 3 seconds) and the store restarts. However, with `all-active`, the source does not wait for the unresponsive store's *`ume_store_activity_timeout`* to expire. The source removes the unresponsive store's group from the list of stores from which the source uses to determine that messages in-flight are stable. An inactive store with a running activity timeout does not impede message stabilization.

Store Configuration Practices to Avoid

Informatica does not support the following store configuration practices.

- Do not use **multiple store groups of one store each**. Recovery does not work well in this configuration because it allows sources to resume sending as soon as it has registered with a single store, and if that store is not fully up-to-date, this can lead to message loss for receivers.
- Do not use **backup stores**. The configuration option, *`ume_store_group`* allows you to identify a store group and its size in number of stores. Setting the group size in this option to a number of stores less than the number of stores configured with the *`ume_store`* option can lead to messages that were reported stable to the source being unavailable for receivers to recover, in the event that multiple stores became unresponsive and were replaced by backup stores. For example, setting *`ume_store_group`* with a size of 3 stores, but configuring *`ume_store`* with 5 stores is not supported.

Proxy Sources

The Proxy Source capability allows you to configure stores to automatically continue sending the source's topic advertisements which allow new receivers to join the source's transport session and request Source Registration Information (SRI) to register with the store and request retransmissions. After the source returns, the store automatically stops acting as a proxy source. Stores can be located across a UM Router or within the same LAN as the failed source.

Some other features of Proxy Sources include:

- Requires a Quorum/Consensus store configuration.
- Normal store failover operation also initiates a new proxy source.
- A store can be running more than one proxy source if more than one source has failed.
- A store can be running multiple proxy sources for the same topic.

How Proxy Sources Operate

The following sequence illustrates the life of a proxy source.

1. A source configured for Proxy Source sends to receivers and a group of Quorum/Consensus stores.
2. The source fails.
3. The source's *`ume_activity_timeout`* or the store's `source-activity-timeout` expires.
4. The Quorum/Consensus stores elect a single store to run the proxy source.
5. The elected store creates a proxy source and sends topic advertisements.

6. The failed source reappears.
7. The store deletes the proxy source and the original source resumes activity.

If the store running the proxy source fails, the other stores in the Quorum/Consensus group detect a source failure again and elect a new store to initiate a proxy source.

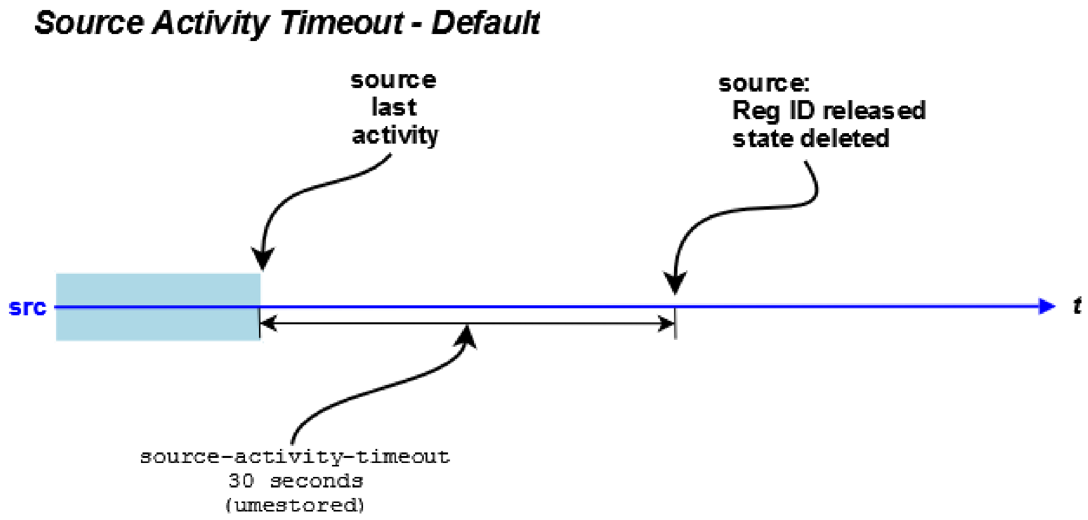
If a loss of quorum occurs, the proxy source can continue to send advertisements, but cannot send messages until a quorum is re-established.

Activity Timeout and State Lifetime Options

UMP provides activity and state lifetime timers for sources and receivers that operate in conjunction with the proxy source option or independently. This section explains how these timers work together and how they work with proxy sources.

The *`ume_activity_timeout`* options determine how long a source or receiver must be inactive before a store allows another source or receiver to register using that RegID. This prevents a second source or receiver from *stealing* a RegID from an existing source or receiver. An activity timeout can be configured for the source/receiver with the UM Configuration Option cited above or with a topic's `ume-attribute` configured in the `umestored` XML configuration file. The following diagram illustrates the default activity timeout behavior, which uses `source-state-lifetime` in the `umestored` XML configuration file.

Figure 12. Source Activity Timeout Default

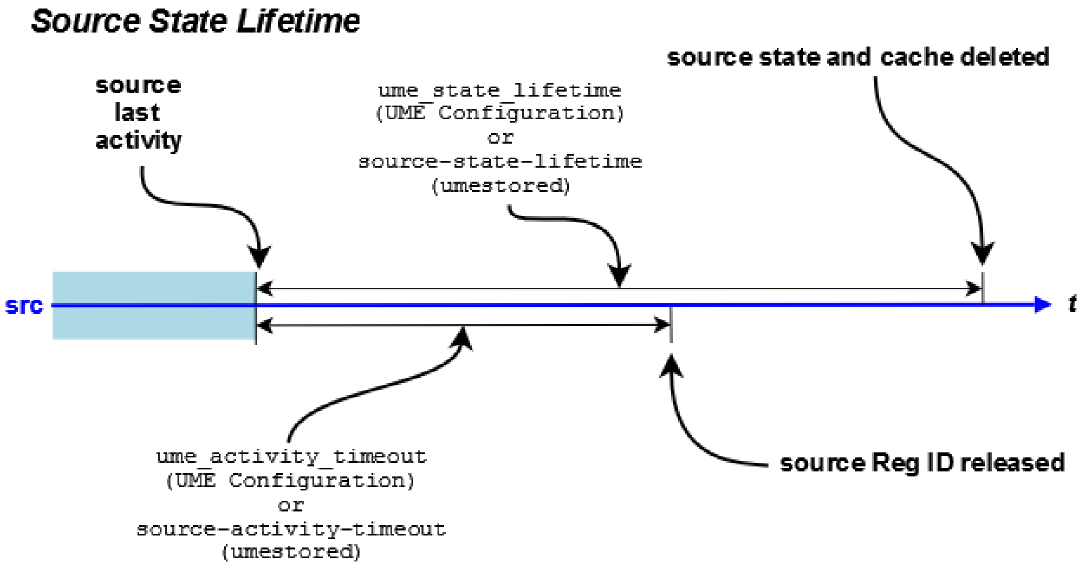


In addition to the activity timeout, you can also configure sources and receivers with a state lifetime timer using the following options.

- *`(source) ume_state_lifetime`*
- *`(receiver) ume_state_lifetime`*
- The topic's ume-attributes options, `source-state-lifetime` and `receiver-state-lifetime`.

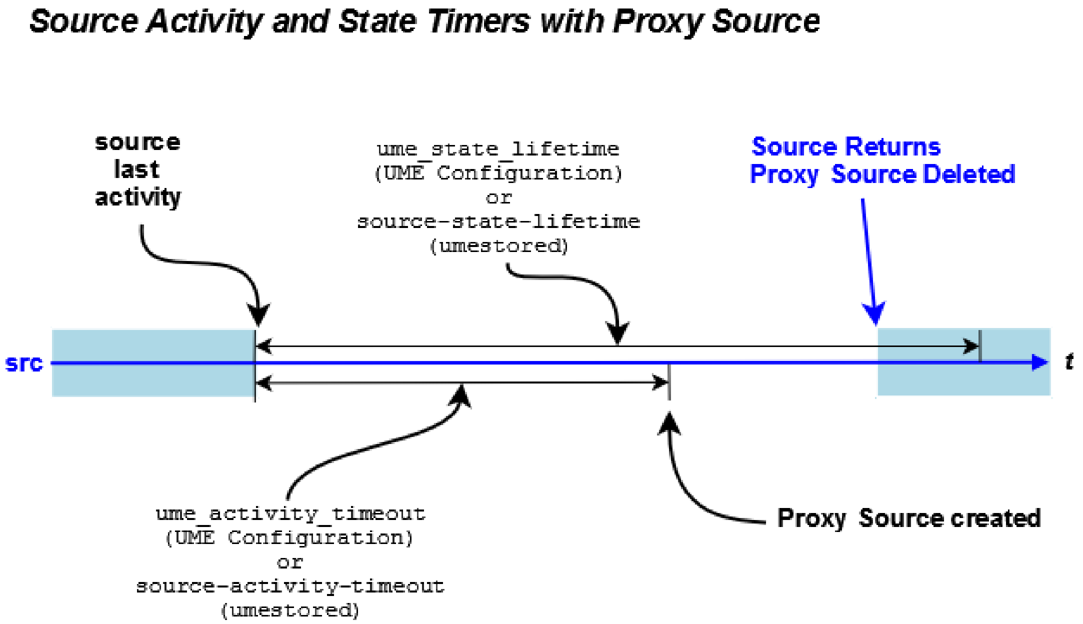
The `ume_state_lifetime` , when used in conjunction with the `ume_activity_timeout` options, determines at what point UMP removes the source or receiver state. UMP does not check the `ume_state_lifetime` until `ume_activity_timeout` expires. The following diagram illustrates this behavior.

Figure 13. Source or Receiver State Lifetime


Source State Lifetime
source state and cache deleted
source last activity
ume_state_lifetime (UME Configuration) or source-state-lifetime (umestored)
src
t
ume_activity_timeout (UME Configuration) or source-activity-timeout (umestored)
source Reg ID released

If you have enabled the Proxy Source option, the Activity Timeout triggers the creation of the proxy source. The following diagram illustrates this behavior.

Figure 14. Source Activity and State Timers with the Proxy Source Option


Source Activity and State Timers with Proxy Source
source last activity
ume_state_lifetime (UME Configuration) or source-state-lifetime (umestored)
Source Returns
Proxy Source Deleted
src
t
ume_activity_timeout (UME Configuration) or source-activity-timeout (umestored)
Proxy Source created

Enabling the Proxy Source Option

You must configure both the source and the stores to enable the Proxy Source option.

- Configure the source in a UM Configuration File with the source configuration option, *`ume_proxy_source`* .
- Configure the stores in the `umestored` XML configuration file with the Store Element Option, `allow-proxy-source`. See “Options for a Store's ume-attributes Element” on page 89 for more information.

Note: Proxy sources operate with Session IDs as well as Reg IDs. See “Managing RegIDs with Session IDs” on page 47

Proxy Source Elections

When multiple stores in a Quorum/Consensus configuration notice the loss of a registered source (expiration of the source's *`ume_activity_timeout`*) configured for proxy sources, only one of the stores needs to create a proxy source to continue sending topic advertisements.

The proxy source election process determines which store creates the proxy source. Each store starts by waiting a randomized amount of time based on its `proxy-election-interval` setting. The store creates a proxy source if it has not received a persistent registration request (PREG) from a proxy on a different store. The proxy source then sends a PREG containing a unique random value to the other stores. This value determines which store deletes it's proxy source in the case that any two stores independently determine they should create a proxy source. The nature of the random values ensures that only one store within the Q/C group or configuration of groups keeps its proxy source.

Proactive Retransmissions

Proactive Retransmissions, which is enabled by default, address two types of loss.

- loss of message data between the source and a store
- loss of stability acknowledgments (ACK) between the store and the source

The store sends message stability acknowledgments to the source after the store persists the message data.

With Proactive Retransmissions, the source maintains an unstable message queue for those messages sent but not acknowledged by the store. The source checks this queue at the *`ume_message_stability_timeout`* . If a message in this queue exceeds its *`ume_message_stability_timeout`* , the source retransmits the message and puts it back on the unstabilized message queue, restarting the message's *`ume_message_stability_timeout`* .

The source continues to retransmit and check the message's stability timeout until the *`ume_message_stability_lifetime`* expires or it receives a stability acknowledgment from the store. If the source has not received a stability acknowledgment when the *`ume_message_stability_lifetime`* expires, the source sends a `Store Message Not Stable` source event notification to the application. When the store discards the message because it has not met stability requirements, the store sends a `Store Forced Reclaim` source event notification to the application.

To disable Proactive Retransmissions, set *`ume_message_stability_timeout`* to 0 (zero). As a result, sources do not create an unstable message queue.

The following applies whether you enable or disable Proactive Retransmissions.

- The store does not discard duplicate messages, but rather always responds to duplicate, retransmitted messages by sending stability acknowledgments even if the message is already stable.
- If the store has marked the message unrecoverably lost and receives a duplicate message from the source, the store sends the source a negative stability acknowledgment (NACK), which induces the source to remove the message from its unstabilized message queue. A stability NACK is identical to a stability ACKs except that it has a NACK flag set.

CHAPTER 9

Man Pages

This chapter includes the following topics:

- umestored, 83
- umestoreds, 84
- umqsltool, 84

umestored

{ `umestored` | { `-d` } | { `--dump-dtd` } | { `-f` } | { `--detach` } | { `-h` } | { `--help` } | { `-v` } | { `--validate` } | { *configfile* } }

Description

Persistent Store services are provided by umestored. A store configuration file is required.

The DTD used to validate a configuration file will be dumped to standard output with the -d or --dump-dtd. After dumping the DTD, umestored exits instead of providing persistence services as usual.

The configuration file will be validated against the DTD if either the -v or --validate options are given. After attempting validation, umestored exits instead of providing persistence services as usual. The exit status will be 0 for a configuration file validated by the DTD and non-zero otherwise.

Umestored normally remains attached to the controlling terminal and runs until interrupted. If the -f or --detach options are given, umestored instead forks, detaches the child from the controlling terminal, and the parent exits immediately.

Command line help is available with -h.

Usage Notes

When shutting down the UM Persistent Store daemon, use a SIGINT to trigger a clean shutdown, which attempts to cleanly finish outstanding IO requests before shutting down. Two successive SIGINTs force an immediate shutdown.

Exit Status

The exit status from umestored is 0 for success and some non-zero value for failure.

umestoreds

{ `umestoreds` | { `-d` } | { `--dump-dtd` } | { `-h` } | { `--help` } | { `-s action` } | { `--service=action` } | { `-v` } | { `--validate` } | { `configfile` } }

Description

Persistent Store services are provided by the umestoreds Windows Service. A store configuration file is optional. If not present, the Registry will be consulted.

The DTD used to validate a configuration file will be dumped to standard output with the -d or --dump-dtd. After dumping the DTD, umestoreds exits instead of providing persistence services as usual.

The configuration file will be validated against the DTD if either the -v or --validate options are given. After attempting validation, umestoreds exits instead of providing persistence services as usual. The exit status will be 0 for a configuration file validated by the DTD and non-zero otherwise.

The -s install or --service=install options will install the service using the given configuration file. Once installed, umestoreds exits. Once installed, the service may be stopped or started via the Windows Service Control Panel.

The -s remove or --service=remove options will remove the service. Once removed, umestoreds exits.

The -s config or --service=config options will update the configuration file used with the service to be the given configuration file. Once updated, umestoreds exits.

Command line help is available with -h.

Usage Notes

When installing the UM Persistent Store as a Microsoft Windows service, use only local disk devices and fully qualified path names for all filenames. This is because Windows services run by default under a Local System account, which has reduced privileges and is not allowed access to network devices.

Stopping the UM Persistent Store service triggers a clean shutdown, which attempts to cleanly finish outstanding IO requests before shutting down.

Exit Status

The exit status from umestored is 0 for success and some non-zero value for failure.

umqsltool

{ `umqsltool` | { `options` } | { *sinclogfile1* } | { `sinclogfile2` } }

Description

This command provides SINC log file tools that let you dump to text, compare two files, or trim events to reduce file size (all without modifying the original log files).

The -t *toolname* or --tool= *toolname* invokes the desired tool. *Toolname* can be:

`dump`

> Dumps the events from a SINC log file in a human-readable text format. Operates on a single SINC log file.

`diff`

Compares events in two SINC log files, noting any differences. Requires two SINC log files to be specified.

`analyze`

Analyzes a SINC log file for events that look suspicious. Operates on a single SINC log file.

`prune`

Trims a SINC log file down to the minimum number of events needed to preserve correct queue state. This can sometimes dramatically reduce SINC log file size without any loss of state. SINC log files are not pruned during normal operation due to performance considerations.

This tool operates on a single SINC log file, and outputs a new SINC log file in the same location and with the same name as the original plus an added ".pruned.n" suffix (where n is a counter starting at 1). The original SINC log file remains untouched. When pruning, we recommend to also specify the --config option, otherwise some state information may be lost in the pruned output.

The -c *configfile* or --config= *configfile* uses the given umestored XML config file. Specifying the umestored XML configuration file is optional, but can improve the accuracy of various tools, so its use is recommended.

The -h or --help displays this help and exits.

CHAPTER 10

Configuration Reference for Umestored

This chapter includes the following topics:

Overview

The operating parameters for umestored come from an XML configuration file that must be supplied on the Chapter 9, “Man Pages” on page 83. umestored contains a UM context and receivers that may be configured with default values through a UM configuration file referenced in the XML configuration file. Default UM options my be overridden for each configured store using the XML configuration file.

You configure `umestored` to instantiate stores or queues with the `umestored` XML configuration file which UM reads at start up.

Note: Although `umestored` configures both stores and queues, this guide only discusses configuring stores. See the *Ultra Messaging Guide for Queuing*for information about configuring queues.

The `umestored` XML configuration file for persistence has the following sections.

- Daemon section - holds administrative parameters for such things as the location of log files, the UM Configuration File, etc.
- Stores section - holds parameters for any persistent stores and also the topics to be persisted.

High Level Store Configuration File.

```
<ume-store version="1.2">
  <daemon>
    Daemon configuration options
  </daemon>
  <stores>
    <store attributes>
      <topics>
        <topic attributes>
```

```
          <ume-attributes>
            <option attributes/>
          </ume-attributes>
        </topic>
      </topics>
    </store>
  </stores>
</ume-store>
```

This section discusses the following topics.

- “Daemon Element” on page 87
- “Stores Element” on page 88
- “Topics Element” on page 90
- “Option Types for ume-attributes Elements” on page 94
- “umestored Configuration DTD” on page 95
- “Store Configuration Example” on page 96

Daemon Element

The following table presents child elements allowed in the daemon configuration section.

Tag	Description	Default Value
`log`	Required. Pathname for log file.	None--this is a required element.
`pidfile`	Pathname for daemon process ID (pid) file	No pidfile
`uid`	User ID (uid) for daemon process (if started as root)	Daemon retains starting uid
`gid`	Group ID (gid) for daemon process (if started as root)	Daemon retains starting gid
`lbm-config`	Pathname for UM configuration file	No config file; use UM defaults
`lbm-license-file`	Pathname for UM license file	License read from environment
`web-monitor`	Address:port where web monitor listens. Address of * listens on all interfaces. Also has a single attribute, permission , allowable values are *read-only* and *read-write*. Using *read-only* disables the text fields and buttons on a Web Monitor "debug page" that can only be enabled by Informatica Support. Example: `*:15304`	No web monitor
`lbm-password-file`	Pathname for Queue Browser authentication file.	*rel-id/platform-id*/bin/password.xml

Stores Element

The Stores Element is a container for individual store elements which define specific store instances. The below is an example of a Stores Element.

```
<stores>
  <store name="test-store-1" port="14567">
    <ume-attributes> ... </ume-attributes>
    <topics>
      <topic pattern="quote*" type="PCRE">
        <ume-attributes> ... </ume-attributes>
      </topic>
      <topic pattern="subject*" type="PCRE">
        <ume-attributes> ... </ume-attributes>
      </topic>
    </topics>
  </store>
  <store name="test-store-2" port="14568">
    <ume-attributes> ... </ume-attributes>
    <topics>
      <topic pattern="issue*" type="PCRE">
        <ume-attributes> ... </ume-attributes>
      </topic>
      <topic pattern="topic*" type="PCRE">
        <ume-attributes> ... </ume-attributes>
      </topic>
    </topics>
  </store>
</stores>
```

Store Element

The Store Element contains information about an individual UMP store and has attributes, options and topics. See the example below.

```
<store name="test-store-1" port="14567">
  <ume-attributes> ... </ume-attributes>
  <topics>
    <topic pattern="quote*" type="PCRE">
      <ume-attributes> ... </ume-attributes>
    </topic>
    <topic pattern="subject*" type="PCRE">
      <ume-attributes> ... </ume-attributes>
    </topic>
  </topics>
</store>
```

The following table gives attributes for store elements.

Attribute	Description	Default Value
`name`	Name that identifies log messages for this store in the umestored log file.	None—this is a required attribute
`port`	TCP port where umestored should listen for connection requests to this store.	None—this is a required attribute
`interface`	Specifies the IP address over which umestored accepts connection requests for this store. You can specify a single IP address, such as 10.29.3.16, or a range of addresses, 10.29.3.16/25. See also "Identifying Persistent Stores" on page 10	0.0.0.0 (**INADDR_ANY**)

Child Elements of the Store Element

The following table gives the child elements allowed in the store configuration section.

Child Element	Description	Default Value
`topics`	A container for `topic` elements. See "Topics Element" on page 90 for more information.	None
`ume-attributes`	A container for `option` elements. See "Options for a Store's ume-attributes Element" on page 89 for more information.	None

Options for a Store's ume-attributes Element

You can configure context (scope) options with a type attribute of lbm-context. UM passes such options through the normal receiver and context configuration option setting mechanisms. See the *UM Configuration Guide* for details. Store options without a type attribute or those explicitly given a type attribute of store simply configure the store itself.

The following table gives options allowed for a store element. Use the `store` Option Type for these options. A Store's ume-attributes Element can also accept the `lbm-context` Option Type. See "Option Types for ume-attributes Elements" on page 94 for more information.

Option	Description	Default Value
`disk-cache-directory`	Pathname for disk store message cache directory	umestored-cache
`disk-state-directory`	Pathname for disk store state directory	umestored-state
`allow-proxy-source`	Allows the store to act as a proxy source in case a registered source terminates.	0 (Disable)

Option	Description	Default Value
`context-name`	Name of the store that can be used by sources to refer to the store instead of the address:request port. Restricted to 128 characters in length, and may contain only alphanumeric characters, hyphens, and underscores. A store runs in its own context, so the store's context name can be used to identify the store. UM automatically resolves store names, which can facilitate UMP operation across the UM Router. A context name must be unique across the entire network and not be the same as any context names used in a UMM XML configuration. See also "Identifying Persistent Stores" on page 10	None.
`retransmission-request-processing-rate`	Specifies the number of retransmission requests processed by a store per second across all topics. The store drops all retransmission requests that exceed this value.	262144

Topics Element

The Topics element is a container element for all the topics persisted by the UMP store. It is one of the two child elements of the Store Element. See the example below.

```
<topics>
  <topic pattern="issue*" type="PCRE">
    <ume-attributes> ... </ume-attributes>
  </topic>
  <topic pattern="topic*" type="PCRE">
    <ume-attributes> ... </ume-attributes>
  </topic>
</topics>
```

Topic Element

The Topic Element defines an individual topic persisted on the UMP store. The following table gives attributes for the topic element.

Attribute	Description	Default Value
`pattern`	Specifies a pattern used to select topics for which a store provides persistence services.	None—this is a required attribute
`type`	Specifies the type of matching to be performed on the `pattern` attribute. A value of `direct` selects an exact string match. A value of `PCRE` selects a Perl Compatible Regular Expression match. A value of `regexp` selects a POSIX extended regular expression. `PCRE`, or `regexp`.	`direct`

The Topic Element has one child element, ume-attributes, the options for which appear in Options for a Topic's ume-attributes Element.

Options for a Topic's ume-attributes Element

The following table gives options allowed for a topic element. Use the `store` Option Type for these options. You can also configure receiver (scope) options and source (scope) options in a Topic's ume-attributes

Element by using the Option Types `lbm-receiver` and `lbm-source`, respectively. See "Option Types for ume-attributes Elements" on page 94 for more information.

Option	Description	Default Value
`retransmission-request-forwarding`	If enabled (value = 1), the store forwards retransmission requests to sources if and only if the store does not have the data. If disabled (value = 0), the store services retransmission requests for data it has, and does not forward requests to sources for data it does not have.	0 (store services retransmission requests and does not forward requests)
`repository-type`	Specifies how messages should be retained by the store. A value of no-cache does not retain messages, only state information. A value of memory retains messages only in the (presumably volatile) main memory of the store. A value of disk retains messages to (presumably non-volatile) disk storage as quickly as possible. In addition, messages are cached in main memory for a time as well. A value of reduced-fd retains messages in disk storage using significantly fewer File Descriptors. Use of this repository type may impact performance. (See "Persistent Store Architecture" on page 6.) The reduced-fd disk storage option is not available on Microsoft Windows.	no-cache
`repository-size-threshold`	For topics with a repository-type of memory, disk or reduced-fd, specifies the minimum number of message bytes (includes payload, headers, and store structure overhead) retained for a topic before the repository starts to delete old messages. Pertains to a memory store or the memory cache of a disk or reduced-fd repository. For RPP repositories, this value only includes message payload. (units: bytes)	25165824 (24 MB)
`repository-size-limit`	For topics with a repository-type of memory, disk or reduced-fd, specifies the maximum number of message bytes (includes payload, headers, and store structure overhead) retained for each source. Pertains to a memory store or the memory cache of a disk or reduced-fd repository. For RPP repositories, this value only includes message payload. (units: bytes)	50331648 (48 MB)
`repository-age-threshold`	For topics with a repository-type of memory, disk or reduced-fd, specifies how long the repository keeps a message available. Pertains to a memory store or the memory cache of a disk or reduced-fd repository. The repository reclaims space used to store messages that exceed this threshold. A value of 0 means message age is not considered in retention decisions. (units: seconds)	0
`repository-disk-max-async-cbs`	For topics with a repository-type of disk or reduced-fd, specifies the maximum number of outstanding async I/O callbacks for reading and writing messages to disk. (units: async callbacks)	16 callbacks
`repository-disk-max-write-async-cbs`	For topics with a repository-type of disk or reduced-fd, specifies the maximum number of outstanding async I/O callbacks for writing messages to disk. Reducing this option can improve throughput by batching more fragments into a single write. (units: async callbacks) This option is deprecated.	16 callbacks

Option	Description	Default Value
`repository-disk-max-read-async-cbs`	For topics with a repository-type of disk or reduced-fd, specifies the maximum number of outstanding async I/O callbacks for reading messages from disk. Raising this value can improve recovery rates. For topics with a repository-type of reduced-fd, Informatica recommends a value of 200 times the number of expected receivers per topic. (units: async callbacks)	16 callbacks
`repository-disk-file-size-limit`	For topics with a repository-type of disk or reduced-fd, specifies the maximum amount of disk space that will be used to store retained messages. A minimum value of 196992 is enforced. (units: bytes)	104857600 (100 MB)
`repository-disk-file-preallocate`	For topics with a repository-type of disk, If set to 1, UMP pre-allocates a store's cache files to match their maximum size on disk (as configured by `repository-disk-file-size-limit`) upon creation, as opposed to growing to that size as the store receives new messages. For ext3/4 and NTFS file systems, this options creates a sparse file, which does not allocate all of the underlying data blocks. Advantages of pre-allocation include better performance on rotating disks due to less file fragmentation, and knowing that enough disk space exists for any new source that registers. Disadvantage is the time to create the cache files, especially if many sources register at once.	0 (zero) - do not pre-allocate
`repository-disk-async-buffer-length`	For topics with a repository-type of disk or reduced-fd, specifies the size of the buffers that will be used in async I/O operations for reading and writing messages to disk. A minimum value of 65664 is enforced. (units: bytes)	65664 (64 KB + 128)
`repository-disk-message-checksum`	For topics with a repository-type of disk or reduced-fd, specifies whether the messages saved to disk should include a checksum field or not for validation if the store is restarted. (units: flag)	0 (disabled)
`source-activity-timeout`	Establishes the period of time from a source's last activity to the release of the source's RegID. Stores return an error to any new source requesting the source's RegID during this period. If proxy sources are enabled (*`ume_proxy_source`*) the store does not release the source's RegID and UMP elects a proxy source. If neither proxy sources nor *`ume_state_lifetime`* are configured, the store also deletes the source's state and cache. Can be overridden by *`ume_activity_timeout`* . See also "Proxy Sources" on page 78. (units: milliseconds)	30000 (30 seconds)
`source-state-lifetime`	Establishes the period of time from a source's last activity to the deletion of the source's state and cache by the store, regardless of whether a proxy source has been created or not. You can also configure *`ume_state_lifetime`* for the source. The store uses whichever is shorter. See also "Proxy Sources" on page 78. (units: milliseconds)	0 (zero)
`receiver-activity-timeout`	Establishes the period of time from a receiver's last activity to the release of the receiver's RegID. Stores return an error to any new request for the receiver's RegID during this period. Can be overridden by *`ume_activity_timeout`* . See also "Proxy Sources" on page 78. (units: milliseconds)	30000 (30 seconds)

Option	Description	Default Value
`receiver-state-lifetime`	Establishes the period of time from a receiver's last activity to the deletion of the receiver's state and cache by the store. You can also configure *`ume_state_lifetime`* for the receiver. The store uses whichever is shorter. See also "Proxy Sources" on page 78. (units: milliseconds)	0 (zero)
`source-check-interval`	Specifies how often a store will check for activity of sources and receivers. (units: milliseconds)	100 (100 milliseconds)
`keepalive-interval`	Specifies how often a store will generate keepalive traffic to sources and receivers if there has been no traffic required in the normal course of operation. (units: milliseconds)	500 (500 milliseconds)
`receiver-new-registration-rollback`	Specifies the number of stabilized messages that a newly registered receiver should consume. For example, setting this to 10, "rolls back" the new receiver's starting message to the 10th most recent message. This value must be positive and less than 2147483648. The recommended value of 2147483647 indicates that the rollback should begin at the start of the stream. A value of 0 indicates the store should instruct the receivers to start with the next new message from the source known by the store. (units: messages)	0 (no recovery requested)
`proxy-election-interval`	Specifies the interval, in milliseconds, used when electing a proxy source. When a source, which requested that a proxy source be provided for it, has been detected as no longer active, each store eligible to provide a proxy source for it waits for an amount of time which is randomized in the range [0.5*`proxy-election-interval` .. 1.5*`proxy-election-interval`]. If no other store has been elected to serve as the proxy source, the store declares itself as the proxy source. (units: milliseconds)	5000 (5 seconds)
`stability-ack-interval`	Specifies the maximum amount of time that stability acknowledgments will be batched before being sent to a source. Batching stability ACKs can increase throughput of UMP stores (especially memory stores) significantly, but introduces a delay between when a message is actually stable in the UMP store and when the source is notified of message stability. (units: milliseconds)	200 (200 milliseconds)
`stability-ack-minimum-number`	Specifies the minimum number of message stability acknowledgments that must accumulate before a stability ACK is sent to a source. With the default value of 1, stability ACKs are sent immediately as soon as messages are stable. Increasing this value causes stability ACKs to be batched, which can increase throughput of UMP stores (especially memory stores) significantly, but introduces a delay between when a message is actually stable in the UMP store and when the source is notified of message stability. If using a stability ACK-based flight size on a UMP source in combination with this option, it is advisable to make sure `stability-ack-minimum-number` is set less than or equal to the source's flight size. Otherwise, stability ACKs will only be sent upon expiration of the `stability-ack-interval` timer, resulting in bursty stop-and-go sending. (units: number of message fragments)	1 (1 fragment)

Option	Description	Default Value
`repository-allow-receiver-paced-persistence`	Specifies if the repository allows receiver-paced persistence (1) or a source-paced persistence (0). The source must enable or disable this option with *ume_receiver_paced_persistence*, but cannot enable it if this value is set to 0.	0 (source-paced persistence)
`repository-allow-ack-on-reception`	Specifies if the repository should acknowledge to the source that a message is persisted as soon as the repository receives it instead of when it writes the message to disk. Set this option to 1 to allow this behavior. The source must also set this option with `ume_repository_ack_on_reception` to enable this behavior, but cannot enable it if this value is set to 0. For memory stores, this option has no effect. This option only applies to RPP repositories (`repository-allow-receiver-paced-persistence = 1`).	0 (store acknowledges persistence when the store writes the message to disk)
`repository-disk-write-delay`	For topics with a repository-type of disk or reduced-fd, specifies the maximum delay in milliseconds before the repository persists a message to disk. If the repository sets this option a value other than the default, the source can reconfigure this option with `ume_write_delay` to a lower or equal value, but cannot increase it. (units: milliseconds)	0 milliseconds
`source-flight-size-bytes-maximum`	Specifies the maximum message payload in bytes allowed to be inflight (un-stabilized at a store and without delivery confirmation) before a new message send either blocks or triggers a notification source event. UMP monitors both this option and `ume_flight_size` . If either threshold is met, the configured blocking or notification behavior executes. See `ume_flight_size_behavior` . A source can reconfigure this option to a value of less than or equal to this value. This option only applies to RPP repositories (`repository-allow-receiver-paced-persistence = 1`). (units: bytes)	4194304 bytes (4MB)

Option Types for ume-attributes Elements

All options configured for <ume-attributes> require an <option> type, as shown in the following example:

```
<?xml version="1.0"?>
  <stores>
    <store name="test-store" port="14567">
      <ume-attributes>
        <option type="store" name="disk-cache-directory" value="cache"/>
            <option type="lbm-context" name="transport_lbtrm_rate_interval" value="100">
      </ume-attributes>
      <topics>
        <topic pattern="test*" type="PCRE">
          <ume-attributes>
            <option type="lbm-receiver" name="transport_lbtrm_send_naks" value="0">
            <option type="lbm-source" name="transport" value="lbtru">
            <option type="store" name="repository-size-limit" value="209715200"/>
          </ume-attributes>
        </topic>
      </topics>
    </store>
  </stores>
```

The following table describes the Option Types.

Option Type	**Description**	**Default Value**
`lbm-receiver`	Allows you to configure receiver-scope options that you usually specify in an Ultra Messaging configuration file or set using `lbm_*_attr_setopt()`. For example, you could turn off delivery of NAKs for a particular topic by including the following within the topic's <ume-attributes> element: `<option type="lbm-receiver" name="transport_lbtrm_send_naks" value="0">` . <option> is a child of <ume-attributes>, but you can use option type `lbm-receiver` within only a <topic> element, not a <store> element.	None - this is a required attribute.
`lbm-context`	Allows you to configure context-scope options that you usually specify in an Ultra Messaging configuration file or set using `lbm_*_attr_setopt()`. For example, you could increase the receiver socket buffer by including the following within the <ume-attributes> element: `<option type="lbm-context" name="transport_lbtrm_receiver_socket_buffer" value="1048576">` . <option> is a child of <ume-attributes>, but you can use option type `lbm-receiver` within only a <store> element, not a <topic> element.	None - this is a required attribute.
`lbm-source`	Allows you to configure source-scope options that you usually specify in an Ultra Messaging configuration file or set using `lbm_*_attr_setopt()`. For example, you could change the transport by including the following within the <ume-attributes> element: `<option type="lbm-source" name="transport" value="lbtru">` . <option> is a child of <ume-attributes>, but you can use option type `lbm-source` within only a <topic> element, not a <store> element.	None - this is a required attribute.
`store`	Option type used for all ume-attributes configured for the store element and its topic element.	None - this is a required attribute.

umestored Configuration DTD

The DTD for version 1.2 of `umestored` appears below. See also the DTD revision table below.

DTD Version	**Release Date**	**Product Version**	**Supported Features**
1.0	Feb. 2007	UME 1.0	Persistent Stores
1.1	April 2010	UME 3.0.1 / UMQ 1.0	Persistent Stores, Queues and Ultra Load Balancing (ULB)
1.2	March 2011	UME 3.2 / UMQ 2.1	Persistent Stores, Queues, Ultra Load Balancing (ULB), Dead Letter Queue, Indexed Queuing and Indexed ULB

```
<!ELEMENT ume-store (daemon, stores?, queues?)>
<!ATTLIST ume-store version CDATA #REQUIRED>
<!ELEMENT daemon (log | uid | pidfile | gid | lbm-config | lbm-license-file | web-
monitor)*>
<!ELEMENT log ( #PCDATA )>
<!ATTLIST log type CDATA #IMPLIED>
<!ATTLIST log xml:space (default | preserve) \"default\">
<!ELEMENT pidfile ( #PCDATA )>
<!ATTLIST pidfile xml:space (default | preserve) \"default\">
<!ELEMENT uid ( #PCDATA )>
<!ATTLIST uid xml:space (default | preserve) \"default\">
<!ELEMENT gid ( #PCDATA )>
<!ATTLIST gid xml:space (default | preserve) \"default\">
<!ELEMENT lbm-config ( #PCDATA )>
<!ATTLIST lbm-config xml:space (default | preserve) \"default\">
<!ELEMENT lbm-license-file ( #PCDATA )>
<!ATTLIST lbm-license-file xml:space (default | preserve) \"default\">
<!ELEMENT web-monitor ( #PCDATA )>
<!ATTLIST web-monitor xml:space (default | preserve) \"default\">
<!ATTLIST web-monitor permission CDATA #IMPLIED>
<!ELEMENT stores (store*)>
<!ELEMENT store (ume-attributes | topics)+>
<!ATTLIST store name CDATA #REQUIRED>
<!ATTLIST store interface CDATA #IMPLIED>
<!ATTLIST store port CDATA #REQUIRED>
<!ELEMENT topics (topic+)>
<!ELEMENT topic (ume-attributes | application-sets)*>
<!ATTLIST topic pattern CDATA #REQUIRED>
<!ATTLIST topic type (direct | PCRE | regexp) #IMPLIED>
<!ELEMENT ume-attributes (option+)>
<!ELEMENT option EMPTY>
<!ATTLIST option type (lbm-receiver | lbm-context | lbm-source | store | queue)
#IMPLIED>
<!ATTLIST option name CDATA #REQUIRED>
<!ATTLIST option value CDATA #REQUIRED>
```

Store Configuration Example

```
<?xml version="1.0"?>
<ume-store version="1.2">
  <daemon>
    <log>stored.log</log>
    <pidfile>stored.pid</pidfile>
    <web-monitor>*:15304</web-monitor>
  </daemon>

  <stores>
    <store name="test-store" port="14567">
      <ume-attributes>
        <option type="store" name="disk-cache-directory" value="cache"/>
        <option type="store" name="disk-state-directory" value="state"/>
        <option type="store" name="context-name" value="remote-store"/>
      </ume-attributes>
      <topics>
        <topic pattern="test*" type="PCRE">
          <ume-attributes>
            <option type="store" name="repository-type" value="disk"/>
            <option type="store" name="repository-size-threshold" value="104857600"/>
            <option type="store" name="repository-size-limit" value="209715200"/>
            <option type="store" name="repository-disk-file-size-limit"
value="1073741824"/>
            <option type="store" name="source-activity-timeout" value="120000"/>
            <option type="store" name="receiver-activity-timeout" value="120000"/>
            <option type="store" name="retransmission-request-forwarding" value="0"/>
          </ume-attributes>
        </topic>
      </topics>
    </store>
  </stores>
</ume-store>
```

CHAPTER 11

Ultra Messaging Web Monitor

This chapter includes the following topics:

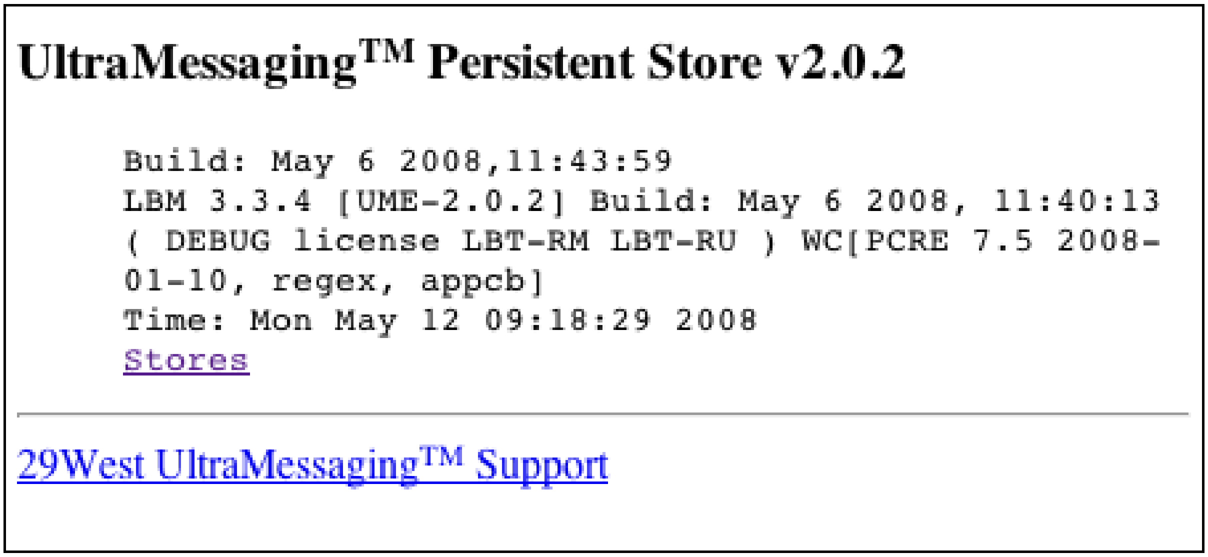
Overview

The built-in web monitor (configured in the umestored XML configuration file) is a rich source of information about the health of a UM stores. This section contains a page-by-page guide to reading and interpreting the output of a UM web monitor, with just a couple example sources and one receiver using a single store.

Ultra Messaging Web Monitor Index Page

The web monitor's index page tells what build of UM is running.

Figure 15. Web Monitor Index Page



UltraMessaging™ Persistent Store v2.0.2

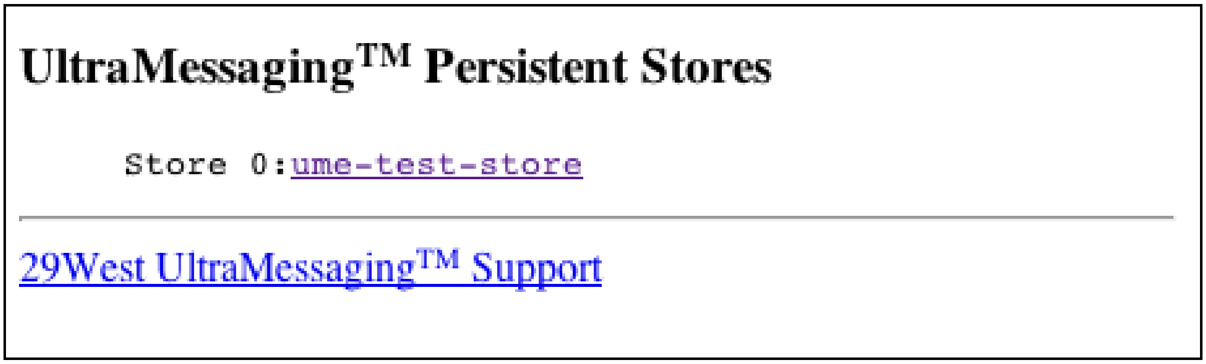
```
Build: May 6 2008,11:43:59
LBM 3.3.4 [UME-2.0.2] Build: May 6 2008, 11:40:13
( DEBUG license LBT-RM LBT-RU ) WC[PCRE 7.5 2008-
01-10, regex, appcb]
Time: Mon May 12 09:18:29 2008
Stores
```

29West UltraMessaging™ Support

Click on the link, Stores, to see the "Persistent Stores Page" on page 99.

Persistent Stores Page

Figure 16. Persistent Stores Page



UltraMessaging™ Persistent Stores

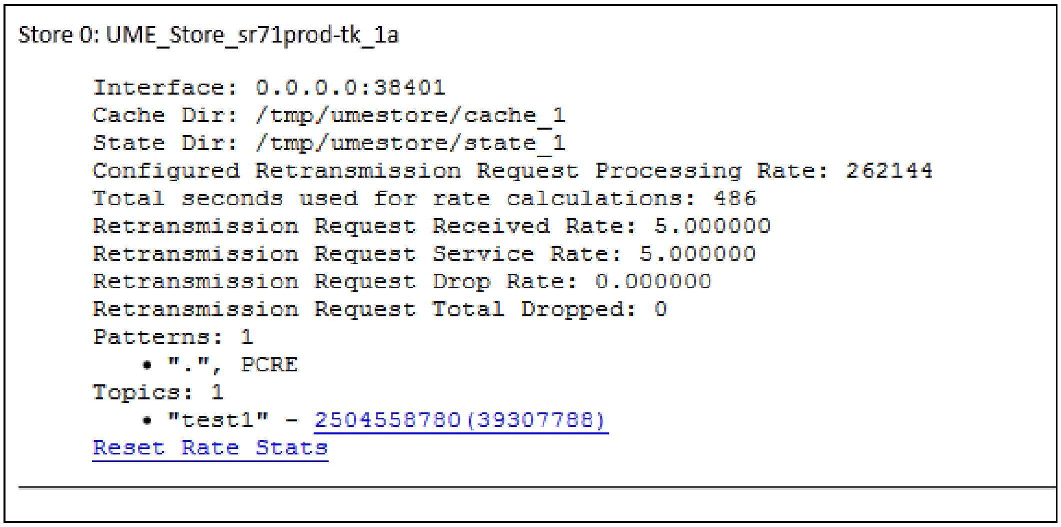
```
Store 0:ume-test-store
```

29West UltraMessaging™ Support

This page shows all the stores configured under the `umestored` process. If you had 5 stores configured, they would be numbered Store 0 through Store 4. Our example has only one store configured, `ume-test-store`. Click on the link, ume-test-store, to see the "Store Page" on page 100.

Store Page

Figure 17. Persistent Store Page



Store 0: UME_Store_sr71prod-tk_1a

```
Interface: 0.0.0.0:38401
Cache Dir: /tmp/umestore/cache_1
State Dir: /tmp/umestore/state_1
Configured Retransmission Request Processing Rate: 262144
Total seconds used for rate calculations: 486
Retransmission Request Received Rate: 5.000000
Retransmission Request Service Rate: 5.000000
Retransmission Request Drop Rate: 0.000000
Retransmission Request Total Dropped: 0
Patterns: 1
    • ".", PCRE
Topics: 1
    • "test1" - 2504558780(39307788)
Reset Rate Stats
```

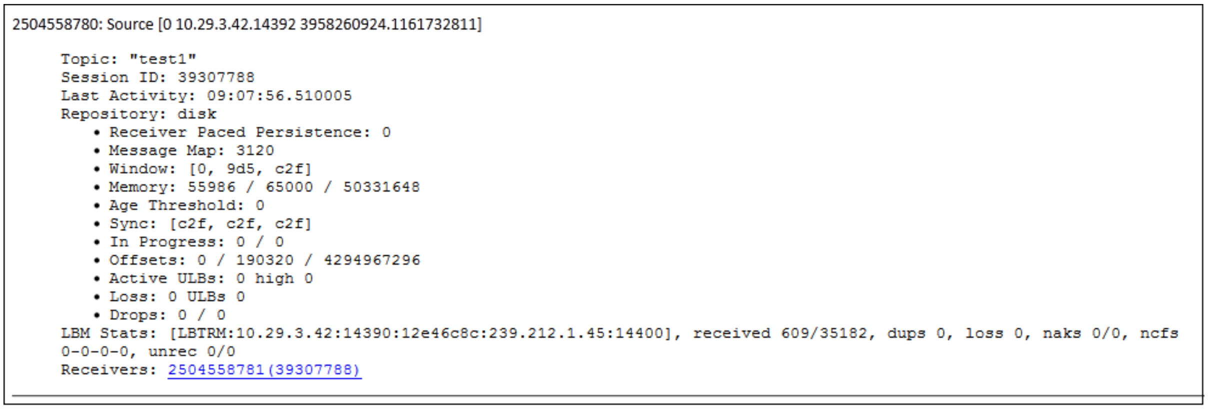
This page shows the following information about the store.

Item	Description
Interface	This store is listening on all interfaces (0.0.0.0) on port 38401.
Cache Dir	Pathname for disk store message cache directory. This would be configured as a store attribute in the store's XML configuration file. `<option type="store" name="disk-cache-directory" value="cache/" />`
State Dir	Pathname for disk store state directory. This would be configured as a store attribute in the store's XML configuration file. `<option type="store" name="disk-state-directory" value="state/" />`
Configured Retransmission Request Processing Rate	Current value for the store's `retransmission-request-processing-rate` setting.
Total Seconds Used for Rate Calculations	Accumulating counter that displays the number of seconds since the last rate reset. The Web Monitor divides the Retransmission Request Received, Retransmission Request Service and Retransmission Request Drop totals by the Total Seconds to calculate the rates displayed. If you click the `Reset Rate Stats`, the Web Monitor resets this value to zero.
Retransmission Request Received Rate	Number of retransmission requests received per second.
Retransmission Request Service Rate	Number of retransmission requests serviced per second.
Retransmission Request Drop Rate	Number of retransmission requests dropped per second. Requests are dropped if the rate of retransmission requests exceeds the configured retransmission request rate.

Item	Description
Retransmission Request Total Dropped	The number of retransmission requests since the time the store was started.
Patterns	Specifies the wildcard pattern used to select topics for which a store will provide persistence services. This would be configured as a topic attribute in the store's XML configuration file. `<topic pattern="test*" type="PCRE">`
Topics	Displays the topic names and `Registration ID (Session ID)` for any sources publishing on the topic. The screen examples display one topic, `test1 - 2504558780(39307788)`. Click on the `Registration ID (Session ID)` link to review information about the sources publishing on the topic. The Source Page appears.
Reset Rate Stats	Click the `Reset Rate Stats` link to reset the retransmission rates. After clicking the link, The Web Monitor rests Total Seconds Used for Rate Calculations to zero and displays a page with the store number and the message, `Rate Statistics have been reset`. Click the Back link to return to the Store Page

Source Page

Figure 18. Web Monitor Source Page



```
2504558780: Source [0 10.29.3.42.14392 3958260924.1161732811]

     Topic: "test1"
     Session ID: 39307788
     Last Activity: 09:07:56.510005
     Repository: disk
        • Receiver Paced Persistence: 0
        • Message Map: 3120
        • Window: [0, 9d5, c2f]
        • Memory: 55986 / 65000 / 50331648
        • Age Threshold: 0
        • Sync: [c2f, c2f, c2f]
        • In Progress: 0 / 0
        • Offsets: 0 / 190320 / 4294967296
        • Active ULBs: 0 high 0
        • Loss: 0 ULBs 0
        • Drops: 0 / 0
     LBM Stats: [LBTRM:10.29.3.42:14390:12e46c8c:239.212.1.45:14400], received 609/35182, dups 0, loss 0, naks 0/0, ncfs
     0-0-0-0, unrec 0/0
     Receivers: 2504558781(39307788)
```

The following table explains the information found in the title of the Source Page.

Source Page Title	Description
2504558780	The source's registration ID.
10.29.3.42.14392	The IP address and port of the source's UM configuration option, `request_tcp_port`.
3958260924	The source's transport session index.
1161732811	The source's topic index within the transport session, 3958260924.

The transport session and topic indices are useful for debugging purposes when combined with a Wireshark capture, but are otherwise not relevant here. The following table provides descriptions of the items in the source page.

Source Page Item	Description
Topic	test is the source's topic string.
Session ID	39307788 is the source's Session ID.
Last Activity	09:19:39.501350 is the timestamp when the store last heard from the source, including keepalives sent by UM
Repository	disk is the type of repository. Possible values are memory, disk or reduced-fd.
Receiver Paced Persistence	Setting for Receiver-paced Persistence (RPP), which is a repository option both the repository and source must enable. A value of 0 means RPP is not enabled and the repository is using the default Source-paced Resistence. A value of 1 means RPP is enabled.
Message Map: 3120	Message Map, 3120, is the total number of message fragments the store has for this source, both on disk and in memory. These are UM-level fragments, not IP-level fragments. UM messages are fragmented into roughly 8 kilobyte chunks for UDP-based protocols (LBT-RM and LBT-RU) and into roughly 64 kilobyte chunks for LBT-TCP. The majority of application messages tend to be well under the fragment boundaries, so the value after "Message Map" could be used as a rough estimate of the number of messages in the store from this particular source. It's at least a strict upper bound.
Window: [0, 9d5, c2f]	Window format is: trail_sqn, mem_trail_sqn, lead_sqn - trail_sqn, 0, is the trailing sequence number, which is the oldest sequence number in the store for this source. In most cases, this starts at 0 and stays there for a while. The trailing sequence number changes if the store reaches a disk file size limit and then deletes the oldest messages. - mem_trail_sqn, 9d5, is the trailing sequence number for messages in memory. It is the oldest sequence number still in memory. Typically, you might have more sequence numbers on disk than you do in memory, or possibly the same number. - lead_sqn, c2f, is the leading sequence number, which is the newest sequence number in the store. **Note:** For a memory store, the first and second values would always be the same. The oldest sequence number in memory is the oldest in the store, so only two values are displayed. The trailing sequence number and the leading sequence number.

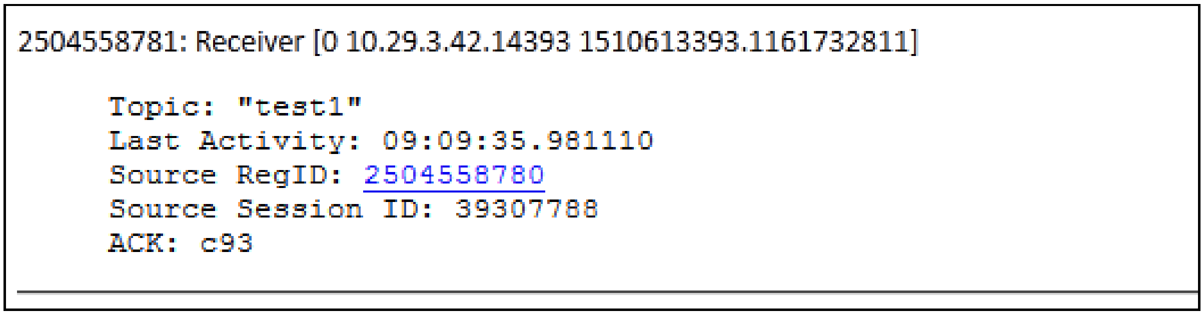
Source Page Item	Description
Memory: 55986 / 65000 / 50331648	Memory format is: repository memory size / repository size threshold / repository size limit - repository memory size, 55986, is the number of bytes of messages in memory, which includes headers and store overhead. - repository size threshold, 65000, is the `repository-size-threshold` topic option found in the store's XML configuration file. - repository size limit, 50331648, is the store's `repository-size-limit` topic option found in the store's XML configuration file. You would expect the number of bytes in memory to be under the threshold most of the time, but it could spike above it before going back down if the store is really busy momentarily. It should never go above the limit.
Age Threshold: 0	Age Threshold, 0, is the store's `repository-age-threshold` topic option found in the store's XML configuration file..
Sync: [c2f, c2f, c2f]	Pertains to disk or reduced-fd. repositories only. Sync format is: sync_complete_sqn, sync_sqn, contig_sqn - sync_complete_sqn, c2f, Most recent sequence number that the Operating System has confirmed persisting to disk. - sync_sqn, c2f, Most recent sequence number for which the store has initiated persisting to disk, but the Operating System has not confirmed completion of persistence. - contig_sqn, c2f, Most recent sequence number that along with the trail_sqn, creates a range of sequence numbers with no sequence number gaps. For example, if trail_sqn = 0 and the store has persisted all eleven messages with sequence numbers 0 through 10, contig_sqn would equal 10. contig_sqn would also be 10 if a receiver declared message sequence number 7 unrecoverably lost. contig_sqn would be 6 if message sequence number 7 was not persisted, but not declared lost.
In progress: 0 / 0	Pertains to disk or reduced-fd. repositories only. In progress format is: num_ios_pending / num_read_ios_pending - num_ios_pending, 0, Number of disk writes the store has submitted to the Operation System. A disk write refers to the store persisting a message to disk. - num_read_ios_pending, 0, Number of disk reads that the store has submitted to the Operating System. A disk read, for example, results from an application retransmission request.

Source Page Item	Description
Offsets: 0 / 190320 / 4294967296	Pertains to disk or reduced-fd. repositories only. Offsets format is: start_offset, offset, max_offset - start_offset, 0, The relative location of the first message, trail_sqn, in the disk. start_offset is 0 for a reduced-fd repository. - offset, 190320, The relative location of where the message , contig_sqn plus one will be written. offset represents the size of the repository on disk for a reduced-fd repository. - max_offset, 4294967296, The maximum size of the cache file. max_offset is the maximum repository size on disk for a reduced-fd repository.
Active ULBs: 0 high 0	ULB stands for Unrecoverable Loss Burst. A little extra work is required to keep cache files consistent when the store gets an unrecoverable loss burst, because unrecoverable loss bursts are delivered all at once for lots of messages, rather than one at a time like normal unrecoverable loss messages.
	Active ULB is the number of unrecoverable loss burst events the store is dealing with at the moment. It'll go to zero after the ULB has been resolved.
	The high number (0) is the highest sequence number reported among any unrecoverable loss burst event, and is not reset after the ULB is handled; it increments throughout the process life of the store.
	WARNING: If you see any number other than 0 here, the store is losing large numbers of messages, and they are likely not being persisted.
Loss: 0 ULBs 0	These values are counters for number of unrecoverable loss messages (Loss) and for number of unrecoverable burst loss messages (ULB). These start at 0 when the store starts up and aren't reset until the store exits. They don't include any loss events that were persisted to disk from a previous run, only new loss events since the store started. There are cases with UME 2.0 where one individual store could legitimately report some unrecoverable loss, or maybe even unrecoverable loss bursts.
	WARNING: If you see any number other than 0 for either of these counters, you should investigate.
Drops: 0 / 0	If the store is nearing the `repository-size-limit` and gets another message, the store will intentionally drop a message. A drop requires a bit of work on the store's part.
	The first 0 is the number of active drops, which are drops that are currently being worked on.

Source Page Item	Description
	The second 0 is the total number of drops that have happened for this store since it was started. Some people want a low `repository-size-limit` and therefore lots of intentional drops can occur. Some don't want to drop any message the whole day - so the interpretation of the values is up to you.
LBM Stats	These represent transport-level statistics for the underlying receivers in the store for the source. The example shown is for a TCP source, so not too many stats are available (stats for a TCP source are less important from a monitoring perspective).
	Statistics for an LBT-RM or LBT-RU source, however, show number of NAKs sent, which is important. Ideally, the number of NAKs sent should be 0. A few NAKs from a store throughout the day is not an emergency. It can be, however, an early warning sign of more severe problems, and should be taken seriously.
	If you see a non-zero number of NAKs here, take a look at the overall network load the store's machine is attempting to handle, particularly in very busy periods and spikes; it may be too much.
Receivers	Registration IDs and accompanying Session ID for the receivers listening on the source's topic. You can review information about the receivers listening on the topic by clicking on the Registration ID(Session ID).

Receiver Page

Figure 19. Web Monitor Receiver Page



2504558781: Receiver [0 10.29.3.42.14393 1510613393.1161732811]

```
    Topic: "test1"
    Last Activity: 09:09:35.981110
    Source RegID: 2504558780
    Source Session ID: 39307788
    ACK: c93
```

The receiver page title shows the following information.

Receiver Page Title Item	Description
2504558781	The receiver's registration ID.
10.29.3.42.14393	The IP address and port of the source's UM configuration option, request_tcp_port.
1510613393	The receiver's transport session index.
1161732811	The source's topic index within the transport session, 1510613393.

The receiver page shows the following information.

Receiver Page Item	Description
Topic	The topic that the receiver is listening on.
Last Activity	09:09:35.981110 is the timestamp of when the store last heard from the receiver, including keepalives sent by UM .
Source RegID	Registration ID of the source publishing on the topic. Click on the Registration ID link to display the Source Page.
Source Session ID	The Session ID of the Source sending messages on the topic.
ACK	c93 is the last message sequence number the receiver acknowledged.

INDEX