Informatica Ultra Messaging Persistence Edition (Version 6.7.1)

Quick Start Guide

Informatica Ultra Messaging Persistence Edition Quick Start Guide

Version 6.7.1
August 2014

Part Number: UMP-QSG-67100-0001

Table of Contents

Preface

The *Ultra Messaging Getting Started Guide* is written for the developers and software engineers who are responsible for Ultra Messaging installation, administration, and application development. It provides a tutorial to help first-time users learn how to use Ultra Messaging. This document assumes you have knowledge of your platform, your network, and messaging 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 topic:

-

Overview

This document provides step-by-step instructions on how to rapidly get started using Ultra Messaging, a fast, efficient, and lightweight messaging system targeted specifically at high performance systems with demanding applications. This document provides step-by-step instructions on how to rapidly get started using Ultra Messaging high performance message streaming. For information about Parallel Persistence, see *The Ultra Messaging Guide for Persistence*.

This document gets you started using our pre-compiled evaluation binaries to test performance, and then walks you through the steps of writing minimal source (sender) and receiver applications.

C H A P T E R 2

Ultra Messaging Binary Quick Start

This chapter includes the following topics:

Overview

A Ultra Messaging software consists of the documentation package (which includes this document) and one or more binary packages. These instructions will help get you started evaluating Ultra Messaging quickly. You can run pre-compiled commands from a Ultra Messaging binary distribution to get some quick performance numbers in your environment. (A subsequent section guides you through creation of simple programs for local compilation.)

Important: If you plan to test multiple machines communicating with each other, the machines will need multicast connectivity between them. Contact your network administrator for information on multicast connectivity. If multicast connectivity is not possible, the test applications can all be run on the same machine. Alternatively, it is possible to configure Ultra Messaging to avoid multicast, see Chapter 4, "Unicast-Only Operation" on page 11.

A binary package consists of:

- Binary versions of source code examples (see the `bin` directory)
- Header files (see the `include` directory)
- Link libraries (see the `lib` directory)
- Shared libraries (see the `lib` directory on Unix for `.so` files or the `bin` directory on Windows for `.dll` files)

The Ultra Messaging code and documentation are copyrighted and confidential information owned by Informatica and are covered under the terms of our Software License Agreement or Non-Disclosure Agreement as appropriate. Use of this code and documentation without a valid Non-Disclosure Agreement or Software License Agreement with Informatica Corporation is strictly prohibited. If this code and documentation is being supplied under the terms of a Non Disclosure Agreement, all copies, in any form, must be returned or destroyed at the end of the evaluation period or as requested by Informatica.

Binary Quick Start on Microsoft Windows

The following steps assume that the windows package is installed on all test machines in the standard place:

`C:\Program Files\Informatica\` *rel-id* `\Win2k-i386`

where *rel-id* is the release identifier. It is also assumed that the `\bin` directory is included in the windows `PATH` environment variable. This is needed so that the `.EXE` and `.DLL` files can be found.

1. Open a command prompt window on the machine you want to use for receiving messages and enter the command lbmrcv topic to start a receiver. Note that `topic` can be any string. You should see output that looks something like this:

   ```
   1.006 secs.   0.0 Kmsgs/sec.   0.0 Kbps
   1.019 secs.   0.0 Kmsgs/sec.   0.0 Kbps
   1.010 secs.   0.0 Kmsgs/sec.   0.0 Kbps
   ```

   A new line will be printed about once per second showing the elapsed time, messages received, and data received. As long as there are no sources yet running on topic, the number of messages received will continue to be zero.

2. Open a command prompt window on the machine you want to use for sending messages and enter the command lbmsrc topic to start sending messages. The receiver will automatically discover the source, at which time its output will change to something like this:

   ```
   1.010 secs. 451.9 Kmsgs/sec.  90.4 Mbps
   1.010 secs. 451.4 Kmsgs/sec.  90.3 Mbps
   1.010 secs. 445.1 Kmsgs/sec.  89.0 Mbps
   ```

3. With no options given, the source will send 10,000,000 small (25 byte) messages. If you would like to test different size packets or number of packets sent, you can set options. Enter lbmsrc -h at the command line, you will get a list of options you can control.

4. Press `Ctrl-C` to kill the source or receiver.

Binary Quick Start on Unix

The following steps assume that the Unix package is installed on all test machines under a normal user account:

`/home/` *user-id* `/lbmeval/` *rel-id* `/` *platform-id*

where *rel-id* is the release identifier and *platform-id* describes the ABI (Application Binary Interface - e.g. `Linux-2.4-glibc-2.2-i686`). It is also assumed that the `/bin` directory is included in the `PATH` environment variable and the `/lib` directory is in the appropriate loader library search path environment variable (e.g. `LD_LIBRARY_PATH` for Linux; see http://bhami.com/rosetta.html for equivalences in other flavors of Unix).

1. Open a command prompt window on the machine you want to use for receiving messages and enter the command lbmrcv topic to start a receiver. Note that `topic` can be any string. You should see output that looks something like this:

   ```
   1.006 secs.   0.0 Kmsgs/sec.   0.0 Kbps
   1.019 secs.   0.0 Kmsgs/sec.   0.0 Kbps
   1.010 secs.   0.0 Kmsgs/sec.   0.0 Kbps
   ```

   A new line will be printed about once per second showing the elapsed time, messages received, and data received. As long as there are no sources yet running on topic, the number of messages received will continue to be zero.

2. Open a command prompt window on the machine you want to use for sending messages and enter the command lbmsrc topic to start sending messages. The receiver will automatically discover the source, at which time its output will change to something like this:

```
1.010 secs. 451.9 Kmsgs/sec.  90.4 Mbps
1.010 secs. 451.4 Kmsgs/sec.  90.3 Mbps
1.010 secs. 445.1 Kmsgs/sec.  89.0 Mbps
```

3. With no options given, the source will send 10,000,000 small (25 byte) messages. If you would like to test different size packets or number of packets sent, you can set options. Enter lbmsrc -h at the command line, you will get a list of options you can control.

4. Press `Ctrl-C` to kill the source or receiver.

Binary Evaluation Notes

1. There are many other example programs available. See *doc/example/index.html* for documentation on all of them.

2. For general information on measuring performance, see the
   UM

   Knowledgebase article Performance Testing.

3. For information on testing the LBT-RM (multicast) transport, see the following UM Knowledgebase articles:

   - Testing with LBT-RM

   - Interpreting LBT-RM Source Statistics

   - Tuning LBT-RM

4. Microsoft has batching issues in its networking stack, so the performance results can vary. We have a work around if this is an issue for you.

5. The Linux version was compiled on Fedora Core 1.

6. The Windows version was compiled on Windows XP with Microsoft Visual C++ version 7.

Ultra Messaging Programming Quick Start

This chapter includes the following topics:

Overview

The C programs below contain the minimum code and supporting material. Their purpose is to verify that the user's build and run-time environments are set up correctly. They also give a basic introduction to the Ultra Messaging API.

We also have equivalent Java and C# programs available in source form. See *MinSrc.java* and *MinRcv.java* or *MinSrc.cs* and *MinRcv.cs* We also have an example of how application callbacks are coded in C++ programs. See *minrcv.cpp* (Most browsers let you right-click on the link and use the *save link target* function, or some variation.)

Note that these programs do not allow the user to override any of the default configuration values. As a result, operation is fixed according to the normal LBM defaults; for example TCP is the transport protocol, topic resolution is performed using multicast, etc. See the *Ultra Messaging Configuration Guide*.

The *Source Code Examples* tab on the left panel provides a much richer set of source files that use a wide variety of features. However, those programs double as performance testing tools, so they tend to be more complex than just demonstrating the features. We recommend to first build and run these minimal examples.

This source code example is provided by Informatica for educational and evaluation purposes only.

Error handling in these programs is primitive. A production program would want to have better error handling, but for the purposes of a minimal example, it would just be a distraction. Also, a production program would want to support a configuration file to override default values on options.

When building on Windows, the following notes are applicable.

1. Make sure the preprocessor variable "WIN32" is defined. From Visual Studio 6, navigate: `"Project"`->`"Settings..."`, `"C/C++"`, `"Category:General"`, `"Preprocessor definitions:"`.

2. Add `C:\Program Files\Informatica\<VERS>\Win2k-i386\include\lbm` as an additional include directory (where "<VERS>" is the appropriate version identifier). From Visual Studio 6, navigate: `"Project"`->`"Settings..."`, `"C/C++"`, `"Category:Preprocessor"`, `"Additional include directories:"`

3. Add `lbm.lib` and `wsock32.lib` as Object/library modules. From Visual Studio 6, navigate: "`Project`"->"`Settings...`", "`Link`", "`Category:General`", "`Object/library modules:`"

4. Add `C:\Program Files\Informatica\`*VERS*`\Win2k-i386\lib` as an additional library path (where *VERS* is the appropriate version identifier). From Visual Studio 6, navigate: "`Project`"->"`Settings...`", "`Link`", "`Category:Input`", "`Additional library path:`"

5. The install procedure should already have added the LBM `bin` directory to the Windows PATH. This is necessary so that `lbm.dll` can be found when a program is run. To modify the Windows PATH from Windows XP, navigate: "`right-click My Computer`"->"`properties`", "`Advanced`", "`Environment variables`", "`System variables:Path`", "`Edit`"

When building on Unix, the following notes are applicable.

1. Sample build command:

```
cc -I$HOME/lbm/<VERS>/<PLATFORM>/include
    -L$HOME/lbm/<VERS>/<PLATFORM>/lib -llbm -lm -o min_src min_src.c
```

(This should be all one line.)

2. The appropriate library search path should be updated to include the Ultra Messaging `lib/` directory. For example, on Linux:

```
LD_LIBRARY_PATH="$HOME/lbm/<VERS>/<PLATFORM>/lib:$LD_LIBRARY_PATH"
export LD_LIBRARY_PATH
```

(For other flavors of Unix, see http://bhami.com/rosetta.html.) Alternatively, the shared library can be copied from the LBM `lib/` directory to a directory which is already in the library search path.

Minimal Ultra Messaging Source Implementation

This is a source code listing of a minimal source (sender) program. Examples also include equivalent Java and C# programs available in source form. See *MinSrc.java* and *MinSrc.cs*.

```c
/*file: minsrc.c - minimal source (sender) program.
 *
 * Copyright (c) 2005-2012 Informatica Corporation. All Rights Reserved.
 * Permission is granted to licensees to use
 * or alter this software for any purpose, including commercial applications,
 * according to the terms laid out in the Software License Agreement.
 */

#include <stdio.h>

#if defined(_MSC_VER)
/* Windows-only includes */
#include <winsock2.h>
#define SLEEP(s) Sleep((s)*1000)
#else
/* Unix-only includes */
#include <stdlib.h>
#include <unistd.h>
#define SLEEP(s) sleep(s)
#endif

#include <lbm/lbm.h>

main()
{
    lbm_context_t *ctx;     /* pointer to context object */
    lbm_topic_t *topic;     /* pointer to topic object */
    lbm_src_t *src;         /* pointer to source (sender) object */
```

```
        int err;                    /* return status of lbm functions (true=error) */

#if defined(_MSC_VER)
    /* windows-specific code */
    WSADATA wsadata;
    int wsStat = WSAStartup(MAKEWORD(2,2), &wsadata);
    if (wsStat != 0)
    {
        printf("line %d: wsStat=%d\n",__LINE__,wsStat);
        exit(1);
    }
#endif

    err = lbm_context_create(&ctx, NULL, NULL, NULL);   /* See Note #1 below */
    if (err)
    {
        printf("line %d: %s\n", __LINE__, lbm_errmsg());
        exit(1);
    }

    err = lbm_src_topic_alloc(&topic, ctx, "Greeting", NULL);   /* See Note #2 below */
    if (err)
    {
        printf("line %d: %s\n", __LINE__, lbm_errmsg());
        exit(1);
    }

    err = lbm_src_create(&src, ctx, topic, NULL, NULL, NULL);   /* See Note #3 below */
    if (err)
    {
        printf("line %d: %s\n", __LINE__, lbm_errmsg());
        exit(1);
    }

    SLEEP(3);   /* See Note #4 below */

    err = lbm_src_send(src, "Hello!", 6, LBM_MSG_FLUSH | LBM_SRC_BLOCK);   /* See Note
#5 below */
    if (err)
    {
        printf("line %d: %s\n", __LINE__, lbm_errmsg());
        exit(1);
    }


    /* Finished all sending to this topic, delete the source object. */
    SLEEP(2);   /* See Note #6 below */
    lbm_src_delete(src);

    /* Do not need to delete the topic object - LBM keeps track of topic
     * objects and deletes them as-needed.  */

    /* Finished with all LBM functions, delete the context object. */
    lbm_context_delete(ctx);

#if defined(_MSC_VER)
    WSACleanup();
#endif
}  /* main */
```

Notes:

1. Create a context object. A context is an environment in which LBM functions. Note that the first parameter is a pointer to a pointer variable; lbm_context_create writes the pointer to the context object into "ctx". Also, by passing NULL to the context attribute parameter, the default option values are used. For most applications only a single context is required regardless of how many sources and receivers are created.

2. Allocate a topic object. A topic object is little more than a string (the topic name). During operation, LBM keeps some state information in the topic object as well. The topic is bound to the containing context, and will also be bound to a source object. Note that the first parameter is a pointer to a pointer variable; lbm_src_topic_alloc writes the pointer to the topic object into "topic". Also, by passing NULL to the source topic attribute, the default option values are used. The string "Greeting" is the topic string.

3. Create the source object. A source object is used to send messages. It must be bound to a topic. Note that the first parameter is a pointer to a pointer variable; lbm_src_create writes the pointer to the source object to into "src". Use of the third and fourth parameters is optional but recommended in a production program - some source events can be important to the application. The last parameter is an optional event queue (not used in this example).

4. Need to wait for receivers to find us before first send. There are other ways to accomplish this, but sleep is easy. See https://communities.informatica.com/infakb/faq/5/Pages/80061.aspx for details.

5. Send a message to the "Greeting" topic. The flags make sure the call to lbm_src_send doesn't return until the message is sent.

6. For some transport types (mostly UDP-based), a short delay before deleting the source is advisable. Even though the message is sent, there may have been packet loss, and some transports need a bit of time to request re-transmission. Also, if the above lbm_src_send call didn't include the flush, some time might also be needed to empty the batching buffer.

Minimal Ultra Messaging Receiver Implementation

This is a source code listing of a minimal receiver program. Examples also include equivalent Java, C#, and C++ programs available in source form. See *MinRcv.java MinRcv.cs* and *minrcv.cpp*.

```
/*file: minrcv.c - minimal receiver program.
 *
 * Copyright (c) 2005-2012 Informatica Corporation. All Rights Reserved.
 * Permission is granted to licensees to use
 * or alter this software for any purpose, including commercial applications,
 * according to the terms laid out in the Software License Agreement.
 */

#include <stdio.h>

#if defined(_MSC_VER)
/* Windows-only includes */
#include <winsock2.h>
#define SLEEP(s) Sleep((s)*1000)
#else
/* Unix-only includes */
#include <stdlib.h>
#include <unistd.h>
#define SLEEP(s) sleep(s)
#endif

#include <lbm/lbm.h>

/*
 * A global variable is used to communicate from the receiver callback to
 * the main application thread.
 */
int msgs_rcvd = 0;

int app_rcv_callback(lbm_rcv_t *rcv, lbm_msg_t *msg, void *clientd)   /* See Note #1
below */
{
    /* There are several different events that can cause the receiver callback
```

```
 * to be called.  Decode the event that caused this.  */
    switch (msg->type)
    {
    case LBM_MSG_DATA:     /* a received message */
        printf("Received %d bytes on topic %s: '%.*s'\n",    /* See Note #2 below */
                msg->len, msg->topic_name, msg->len, msg->data);

        /* Tell main thread that we've received our message. */
        ++ msgs_rcvd;
        break;

    case LBM_MSG_BOS:
    printf("[%s][%s], Beginning of Transport Session\n", msg->topic_name, msg->source);
    break;

    case LBM_MSG_EOS:
    printf("[%s][%s], End of Transport Session\n", msg->topic_name, msg->source);
    break;

    default:    /* unexpected receiver event */
        printf("Received lbm_msg_t type %x [%s][%s]\n", msg->type, msg->topic_name, msg-
>source);
        break;
    }  /* switch msg->type */

    return 0;
}  /* app_rcv_callback */


main()
{
    lbm_context_t *ctx;     /* pointer to context object */
    lbm_topic_t *topic;     /* pointer to topic object */
    lbm_rcv_t *rcv;         /* pointer to receiver object */
    int err;                /* return status of lbm functions (true=error) */

#if defined(_MSC_VER)
    /* windows-specific code */
    WSADATA wsadata;
    int wsStat = WSAStartup(MAKEWORD(2,2), &wsadata);
    if (wsStat != 0)
    {
        printf("line %d: wsStat=%d\n",__LINE__,wsStat);
        exit(1);
    }
#endif

    err = lbm_context_create(&ctx, NULL, NULL, NULL);   /* See Note #3 below */
    if (err)
    {
        printf("line %d: %s\n", __LINE__, lbm_errmsg());
        exit(1);
    }

    err = lbm_rcv_topic_lookup(&topic, ctx, "Greeting", NULL);   /* See Note #4 below */
    if (err)
    {
        printf("line %d: %s\n", __LINE__, lbm_errmsg());
        exit(1);
    }

    err = lbm_rcv_create(&rcv, ctx, topic, app_rcv_callback, NULL, NULL);   /* See Note
#5 below */
    if (err)
    {
        printf("line %d: %s\n", __LINE__, lbm_errmsg());
        exit(1);
    }

    while (msgs_rcvd == 0)
        SLEEP(1);
```

```
        /* Finished all receiving from this topic, delete the receiver object. */
        lbm_rcv_delete(rcv);

        /* Do not need to delete the topic object - LBM keeps track of topic
         * objects and deletes them as-needed.  */

        /* Finished with all LBM functions, delete the context object. */
        lbm_context_delete(ctx);

#if defined(_MSC_VER)
    WSACleanup();
#endif
}  /* main */
```

Notes:

1. LBM passes received messages to the application by means of a callback. I.e. the LBM context thread reads the network socket, performs its higher-level protocol functions, and then calls an application-level function that was set up during initialization. This callback function has some severe limitations placed upon it. It must execute very quickly; any potentially blocking calls it might make will interfere with the proper execution of the LBM context thread. One common desire is for the receive function to send an LBM message (via lbm_src_send), however this has the potential to produce a deadlock condition. If it is desired for the receive callback function to call LBM or other potentially blocking functions, it is strongly advised to make use of an event queue, which causes the callback to be executed from an application thread. See the example tool lbmrcvq.c for an example of using a receiver event queue.

2. Note - printf can block, which is normally a bad idea for a callback (unless an event queue is being used). However, for this minimal application, only one message is expected.

3. Create a context object. A context is an environment in which LBM functions. Note that the first parameter is a pointer to a pointer variable; lbm_context_create writes the pointer to the context object into "ctx". Also, by passing NULL to the context attribute parameter, the default option values are used. For most applications only a single context is required regardless of how many sources and receivers are created.

4. Lookup a topic object. A topic object is little more than a string (the topic name). During operation, LBM keeps some state information in the topic object as well. The topic is bound to the containing context, and will also be bound to a receiver object. Note that the first parameter is a pointer to a pointer variable; lbm_rcv_topic_lookup writes the pointer to the topic object into "topic". Also, by passing NULL to the source topic attribute, the default option values are used. The string "Greeting" is the topic string.

5. Create the receiver object and bind it to a topic. Note that the first parameter is a pointer to a pointer variable; lbm_rcv_create writes the pointer to the source object to into "rcv". The second and third parameters are the function and application data pointers. When a message is received, the function is called with the data pointer passed in as its last parameter. The last parameter is an optional event queue (not used in this example).

C H A P T E R   4

Unicast-Only Operation

This chapter includes the following topic:

Overview

One of Ultra Messaging's great strengths is its use of the network hardware and protocols to achieve very high performance and scalability. By default, Ultra Messaging uses multicast for topic resolution so that data sources and receivers can find each other.

In general, we recommend the use of multicast whenever possible because it provides the best performance and scalability. However, we recognize that it is not always possible to provide multicast connectivity between the machines. For those cases, we support unicast-only operation.

There are two parts to unicast-only operation. One is to use a unicast form of transport. Ultra Messaging uses TCP by default for transport, but LBT-RU is also available and has some performance advantages. For initial "quick start" testing, we recommend not changing the default TCP.

The second part of unicast-only operation is to use the lbmrd (UM Resolver Daemon) for topic resolution. (NOTE: this does *not* route message data through a daemon. This is a helper process that lets data sources and receivers find each other.) To enable unicast topic resolution, do the following:

1. Choose one machine to host the resolver daemon and enter the lbmrd command (use lbmrd -h for full instructions).

2. Create a Ultra Messaging configuration file containing:

       context resolver_unicast_daemon *[Iface[:Src_Port]->]IP:Dest_Port*

   - `Iface` is the interface to use (previously set via *resolver_unicast_interface*).
   - `Src_Port` is the source port to use (previously *resolver_unicast_port*)
   - `IP` is the resolver daemon's IP address (previously *resolver_unicast_address*)
   - `Dest_Port` is the resolver daemon's UDP port (previously *resolver_unicast_destination_port*)

   For more on *resolver_unicast_daemon* , see the *UM Configuration Guide*.

3. Run the test applications with the option -c *filename* where *filename* is the configuration file created in step 2.

Due to the fact that the minimal source programs presented in this document (`minsrc.c` and `minrcv.c`) do not allow the use of a configuration file, it is not possible to configure them for unicast-only operation. If multicast operation is not possible on your network, then please use the binary test programs (described in Chapter 2, " Ultra Messaging Binary Quick Start" on page 2) which which *do* allow unicast-only configuration.

CHAPTER 5

Starting Ultra Messaging Daemons

This chapter includes the following topics:

Overview

Starting UM daemons is not really a quick start operation, but the information provided in this should provide a good overview of the type of activities required to start these important UM components.

Ultra Messaging Dynamic Routing Option

The UM Router bridges disjoint topic resolution domains by forwarding multicast and/or unicast topic resolution traffic. Before installing and starting a UM Router daemon (`tnwgd`) clear objectives and proper planning are very important. Approach this planning with the consideration that a UM Router condenses your network into a single process. You must be clear about the traffic you expect to forward through a UM Router. The following highlights some other specifics.

1. Know your Topic Resolution Domains. See *Topic Resolution Domain* in the *UM Dynamic Routing Guide*.

2. Consider the size and quality of network paths into and out of your UM Routers. For example, UM Routers cannot efficiently forward messages from a 1GB path to a 100MB path.

3. Fully examine any use of Late Join, taking care to configure retransmission options correctly.

UMM Daemon

The UMM Daemon manages UM XML configuration information defined in the GUI as well as user and license information. It serves license and configuration information to UM applications. The UMM Daemon requires either a MySQL or Oracle database installation. Informatica does not supply MySQL or Oracle nor any licensing to use them. For additional information about the steps in this process, see *UM Manager Daemon* in the Ultra Messaging Manager Guide.

Note: The UMM GUI requires Java Version 1.6.x.

1. Install MySQL or Oracle according to the user documentation.

2. Create a database. (You provide the UMM Daemon with the database name, username and password at start up. This user name and password should be kept secure.)

3. Log into the database.

4. From the database's command line or home page, execute the `/UMM/install_tables_oracle.sql` or `source /UMM/install_tables_mysql.sql` script. (This script installs the tables required by UMM.)

5. From the database's command line or home page, load `/UMM/oracle_templates.txt` or `/UMM/mysql_templates.txt`. (This script loads the example configuration templates for *low latency* and *high throughput*. These templates appear in the UMM GUI under Templates in the object tree in the left pane.

   - For Oracle, load `/UMM/oracle_templates.txt`.

   - For MySQL, run `LOAD DATA INFILE 'mysql_templates.txt' INTO TABLE TEMPLATE;`.

6. From the database's command line or home page, load `/UMM/oracle_application_jms.txt` or `/UMM/mysql_application_jms.txt`. (This script loads the ConnectionFactories and Destinations required to run the example JMS applications. These templates appear in the UMM GUI under JMS/ConnectionFactory and JMS/Destination in the object tree in the left pane.)

   - For Oracle, load `/UMM/oracle_application_jms.txt` into the `Application` table.

   - For MySQL, run `LOAD DATA INFILE 'mysql_application_jms.txt' INTO TABLE APPLICATION;`.

7. Modify the `/UMM/umm.properties` to indicate the database username, password and driver to reflect your configuration database.

8. Edit the `ummd.bat` or `ummd.sh` file that starts UMM Daemon. The appropriate driver jar file name must be added to the classpath. Specify your certificate file or Java keystore information if using SSL certificates. The `ummd.bat` or `ummd.sh` provided with installation appears below.

   ```
   java -cp .;mysql-connector-java-5.0.8-bin.jar;UMMD_1.6.0_02.jar -Xms512m -Xmx1024m
   umm.ummd.Server
   ```

9. Start the UMM Daemon with `/UMM/ummd.bat` or `/UMM/ummd.sh`.

10. Edit the `umm.bat` or `umm.sh` file that starts the UMM GUI. The appropriate driver jar file name must be added to the classpath. The `umm.bat` or `umm.sh` provided with installation appears below.

    ```
    java -cp .;mysql-connector-java-5.0.8-bin.jar;UMM_1.jar; -Xms256m -Xmx1024m
    umm.gui.MainFrame
    ```

11. Start the UMM GUI with `/UMM/umm.bat` or `/UMM/umm.sh`.

12. Log into the UMM GUI with username, `default` and password `default`. Either change the default user's password or create new administrative user and delete the default user. See *Managing Users* in the in the *Ultra Messaging Manager Guide*.

13. To enable your UM applications to be served configuration information by UMM Daemon, set the following environment variable for every application/user combination.

```
export LBM_UMM_INFO=<application_name>:<user_name>:<password>@<ip>:<port>
```

For example:

```
export LBM_UMM_INFO=lbmrcv:lbmrcv:default:default@10.29.3.95:21273
```

Persistent Store and Queue Daemon

The daemon, `umestored`, provides persistent store services. Follow the steps below to start `umestored`.

1. Create the cache and state directories. `$ mkdir umestored-cache ; mkdir umestored-state`

2. Create a simple `umestored` XML configuration file or use a sample configuration, *ume-example-config.xml* or *q-example-config.xml*. See the *Ultra Messaging Guide for Persistence*.

3. Start the daemon. `$ umestored config.xml`

   See also *Manpage for umestored* in the *Ultra Messaging Guide for Persistence*.

C H A P T E R 6

Next Steps

This chapter includes the following topic:

Overview

The *Ultra Messaging Concepts* introduces the important fundamental concepts you will need to design and code your applications.

See *Examples* for descriptions and source files for the binary tools used for evaluation and performance measurements. They can be a valuable resource for seeing how various features can be implemented.

The Ultra Messaging Guide for Persistence contains concepts, tutorials and configuration information for UMP, along with discussions about designing persistent applications. It also contains detailed configuration information for the `umestored` daemon.

The Ultra Messaging *C API*, *Java API* and *.NET API* documentation provide a reference guide to the Ultra Messaging API libraries.

The Informatica MySupport site contains a wealth of articles answering common support questions, error messages, new hardware issues, source code examples and more.