Ultra Messaging® Quick Start Guide

Copyright © 2005 - 2014 Informatica

Informatica Ultra Messaging
Version 5.3
March 2014

Table of Contents

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®** and **UMQ** operations see The Ultra Messaging Guide for Persistence and Queuing (../UME/index.html).

1. Introduction

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.

2. Ultra Messaging Binary Quick Start

A **Ultra Messaging** software evaluation kit 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 *Unicast-Only Operation*.

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)

2.1. 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\29West\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.

2.2. 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.

2.3. Binary Evaluation Notes

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

2. For general information on measuring performance, see the **UM** Knowledgebase article Performance Testing (https://communities.informatica.com/infakb/faq/5/Pages/80054.aspx).

3. For information on testing the LBT-RM (multicast) transport, see the following **UM** Knowledgebase (https://communities.informatica.com/infakb/kbexternal/default.aspx) articles:

   - Testing with LBT-RM (https://communities.informatica.com/infakb/faq/5/Pages/80071.aspx)
   - Interpreting LBT-RM Source Statistics (https://communities.informatica.com/infakb/faq/5/Pages/80072.aspx)
   - Tuning LBT-RM (https://communities.informatica.com/infakb/faq/5/Pages/80074.aspx)

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.

3. Ultra Messaging Programming Quick Start

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 (../Config/index.html).

The Ultra Messaging Examples page (../example/index.html) 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\29West\<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\29West\`*`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.

3.1. Minimal Ultra Messaging Source Implementation

This is a source code listing of a minimal source (sender) program. You may find it helpful to download the source code (minsrc.c) (most browsers let you right-click on the link and use the *save link target* function, or some variation). We also have equivalent Java and C# programs available in source form. See MinSrc.java and MinSrc.cs.

```
/*file: minsrc.c - minimal source (sender) program.
 *
 * Copyright (c) 2005-2014 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);   ❶
    if (err)
    {
        printf("line %d: %s\n", __LINE__, lbm_errmsg());
        exit(1);
    }

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

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

    SLEEP(3);   ❹

    err = lbm_src_send(src, "Hello!", 6, LBM_MSG_FLUSH | LBM_SRC_BLOCK);   ❺
    if (err)
    {
        printf("line %d: %s\n", __LINE__, lbm_errmsg());
        exit(1);
    }

    /* Finished all sending to this topic, delete the source object. */
    SLEEP(2);   ❻
    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:

❶ 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.

❷ 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.

❸ 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).

❹ 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.

❺ 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.

❻ 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.

3.2. Minimal Ultra Messaging Receiver Implementation

This is a source code listing of a minimal receiver program. You may find it helpful to download the source code (minrcv.c) (most browsers let you right-click on the link and use the *save link target* function, or some variation). We also have 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) 2005-2014 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)    ❶
{
    /* 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",    ❷
               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);   ❸
    if (err)
    {
        printf("line %d: %s\n", __LINE__, lbm_errmsg());
        exit(1);
    }

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

    err = lbm_rcv_create(&rcv, ctx, topic, app_rcv_callback, NULL, NULL);   ❺
    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:

❶ 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.

❷ 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.

❸ 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.

❹ 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.

❺ 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).

4. Unicast-Only Operation

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
   ```

   where

   - *`Iface`* is the interface to use (previously set with `resolver_unicast_interface` (../Config/unicastresolvernetworkoptions.html#CONTEXTRESOLVERUNICASTINTERFACE)).
   - *`Src_Port`* is the source port to use (previously set with `resolver_unicast_port` (../Config/deprecatedoptions.html#CONTEXTRESOLVERUNICASTPORT)).

- `IP` is the resolver daemon's IP address (previously set with `resolver_unicast_address` (../Config/deprecatedoptions.html#CONTEXTRESOLVERUNICASTADDRESS)).
- `Dest_Port` is the resolver daemon's UDP port (previously set with `resolver_unicast_destination_port` (../Config/deprecatedoptions.html#CONTEXTRESOLVERUNICASTDESTINATIONPORT)).

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

See also `resolver_unicast_daemon` (../Config/unicastresolvernetworkoptions.html#CONTEXTRESOLVERUNICASTDAEMON).

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 Section 2) which which *do* allow unicast-only configuration.

5. Ultra Messaging JMS Quick Start

UMQ also includes a Ultra Messaging JMS evaluation package. You can run pre-compiled commands from an Ultra Messaging JMS distribution to get some quick performance numbers in your environment. (A subsequent section guides you through creation of simple programs for local compilation.)

Important: The application demonstrations described below assume that you will use a single machine. 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.

An Ultra Messaging JMS evaluation package includes the following.

- Ultra Messaging JMS jar file that contains the Ultra Messaging JMS and compiled example applications. (See the `jmsclient/lib` directory.)
- JMS jar file and other 3rd party jar files (See the `jmsclient/lib` directory.)
- Example source code (see the `doc/jms_example/*.java` directory)
- The Ultra Messaging JMS Guide (../jms_concepts/index.html)
- JMS Application Programmer's Interface (../JMSAPI/html/index.html)

The Ultra Messaging JMS code and documentation are copyrighted and confidential information owned by Informatica Corporation 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, Inc. 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 Corporation.

Note: For Windows installations, you must also install the Microsoft Visual C++ 2005 Service Pack 1 Redistributable Package MFC Security Update, which is available on the download page and contains runtime components of the Visual C++ Libraries.

Note: The Ultra Messaging JMS JAR installation is not compatible with Microsoft® Windows® due to a new dependency on `libeay32.dll`. As a workaround, set `use_native_loader=false` and load the dependencies from the PATH.

5.1. Configure Your Environment to Run Ultra Messaging JMS Examples

5.1.1. Configure Ultra Messaging JMS for Unix

Perform the following steps to configure Ultra Messaging JMS for a Unix environment.

1. Edit the `bin/env.sh` file to configure your environment. You must set the following two variables.

   ```
   export JAVA_HOME="/usr/local/jdk1.6.0_16"
   export LBM_LICENSE_FILENAME=/home/user/lbm.lic
   ```

2. Run `startJMS.sh`. You can now run any of the example applications with the appropriate script. See *Microsoft Windows or Unix*.

   Note: If you make any changes to the config.xml file to test different aspects of Ultra Messaging JMS, you must re-run the "config.sh" script. This re-creates a `.bindings` file from the config.xml (../jms_example/config.xml) file that all example programs use.

5.1.2. Configure Ultra Messaging JMS for Microsoft Windows

The following steps assume that Microsoft Windows is installed in the standard place.

Note: It is critical that you set the Microsoft Windows `PATH` environment variable to include the location of the **UMQ** product installation bin directory. In particular, `iconv.dll` from the `bin` directory must be in the executable's `dll` search path at runtime.

1. Set the following three environment variables.

   ```
   set JAVA_HOME=C:\Program Files\Java\jdk1.6
   set JAVA_BIN=%JAVA_HOME%\bin
   ```

```
set LBM_LICENSE_FILENAME=C:\lbm.lic
```

> **Note:** Your Java directory may be different, such as `jre6`. Also, your license file path or name may vary.

2. Run `startJMS.bat`. You can now run any of the example applications with the appropriate script.

   > **Note:** If you make any changes to the config.xml file to test different aspects of Ultra Messaging JMS, you must re-run the "config.bat" script. This re-creates a `.bindings` file from the config.xml (../jms_example/config.xml) file that all example programs use.

5.2. Microsoft Windows or Unix

All source for the following example applications is Java.

- If you are running on Microsoft Windows you will execute a `.bat` file.
- If you are running a Unix system, you will execute a `.sh` file.

5.3. Demonstrating Request/Reply

Follow the steps below to demonstrate Ultra Messaging JMS request/reply. This demonstration uses Reply.java (../jms_example/Reply.java) and Request.java (../jms_example/Request.java) source files and requires a UME server.

1. Open a command prompt window and enter the command `reply.sh` or `reply.bat` for Microsoft Windows.
2. Open a second command prompt window and enter the command `request.sh` or `request.bat` for Microsoft Windows.
3. You will see output that indicates the message has been sent back and forth. Press **Ctrl-C** to kill either application.

5.4. Demonstrating Synch and Async Reads

Follow the steps below to demonstrate Ultra Messaging JMS synchronous and asynchronous message delivery. This demonstration uses AsyncConsumer.java (../jms_example/AsyncConsumer.java), Producer.java (../jms_example/Producer.java) and SyncConsumer.java (../jms_example/SyncConsumer.java) source files.

1. Open a command prompt window and enter the command `AsyncConsumer.sh` or `AsyncConsumer.bat` for Microsoft Windows.

2. Open a second command prompt window and enter the command `Producer.sh` or `Producer.bat` for Microsoft Windows.
3. Open a third command prompt window and enter the command `SyncConsumer.sh` or `SyncConsumer.bat` for Microsoft Windows.

For the `Producer`, you will see output similar to the following.

```
Sent ********* This is a test ***********
Sent ********* This is a test ***********
Sent ********* This is a test ***********
Sent ********* This is a test ***********
Sent ********* This is a test ***********
Sent ********* This is a test ***********
Sent ********* This is a test ***********
Sent ********* This is a test ***********
Sent ********* This is a test ***********
Sent ********* This is a test ***********
```

For the `AsyncConsumer` and the `SyncConsumer`, you will see output similar to the following.

```
Received message: ********* This is a test ***********
Received message: ********* This is a test ***********
Received message: ********* This is a test ***********
Received message: ********* This is a test ***********
Received message: ********* This is a test ***********
Received message: ********* This is a test ***********
Received message: ********* This is a test ***********
Received message: ********* This is a test ***********
Received message: ********* This is a test ***********
Received message: ********* This is a test ***********
```

Press **Ctrl-C** to kill any application.

5.5. Demonstrating A Durable Consumer

Follow the steps below to demonstrate Ultra Messaging JMS durable consumer messaging. This demonstration uses DurableConsumer.java (../jms_example/DurableConsumer.java) and DurableProducer.java (../jms_example/DurableProducer.java) source files and requires a UME server.

1. Open a command prompt window and enter the command `DurableConsumer.sh` or `DurableConsumer.bat` for Microsoft Windows.
2. Open a second command prompt window and enter the command `DurableProducer.sh` or `DurableProducer.bat` for Microsoft Windows. The `DurableProducer` sends 20 messages.
3. In the command prompt window for the `DurableConsumer`, you should see output similar to the following.

   ```
   Reading message: This is message 1
   Reading message: This is message 2
   Reading message: This is message 3
   Reading message: This is message 4
   Reading message: This is message 5
   ```

```
Reading message: This is message 6
Reading message: This is message 7
Reading message: This is message 8
```

4. After message 8, press **Ctrl-C** to kill the `DurableConsumer`. Then restart the consumer. The remaining 12 messages sent by the `DurableProducer` appear in the window.

```
Reading message: This is message 9
Reading message: This is message 10
Reading message: This is message 11
Reading message: This is message 12
Reading message: This is message 13
Reading message: This is message 14
Reading message: This is message 15
Reading message: This is message 16
Reading message: This is message 17
Reading message: This is message 18
Reading message: This is message 19
Reading message: This is message 20
```

5.6. Demonstrating UMQ

Follow the steps below to demonstrate Ultra Messaging JMS queuing. This demonstration uses QueueReceiver.java (../jms_example/QueueReceiver.java) and QueueSender.java (../jms_example/QueueSender.java).java source files and requires a UMQ server.

1. Open two command prompt windows and in each window, enter the command `QueueReceiver.sh` or `QueueReceiver.bat` for Microsoft Windows.
2. Open a third command prompt window and enter the command `QueueSender.sh` or `QueueSender.bat` for Microsoft Windows. The `QueueSender` sends 15 messages; odd numbered messages go to the `QueueReceiver` in Window 1 and even numbered messages to `QueueReceiver` in Window 2.

   Output for `QueueReceiver` in Window 1.

```
Reading message: This is message 1
Reading message: This is message 3
Reading message: This is message 5
Reading message: This is message 7
Reading message: This is message 9
Reading message: This is message 11
Reading message: This is message 13
Reading message: This is message 15
```

   Output for `QueueReceiver` in Window 2.

```
Reading message: This is message 2
Reading message: This is message 4
Reading message: This is message 6
Reading message: This is message 8
Reading message: This is message 10
Reading message: This is message 12
Reading message: This is message 14
```

Press **Ctrl-C** to kill any application.

5.7. Other Example Applications

1. MyExceptionListener.java (../jms_example/MyExceptionListener.java)
2. MyListener.java (../jms_example/MyListener.java)
3. Ping.java (../jms_example/Ping.java)

   To achieve lower latency for this example application, you should change the value of `implicit_batching_minimum_length` from 2000 to 1 in the config.xml (../jms_example/config.xml) file.
4. Pong.java (../jms_example/Pong.java)

   To achieve lower latency for this example application, you should change the value of `implicit_batching_minimum_length` from 2000 to 1 in the config.xml (../jms_example/config.xml) file.
5. SimpleBytePublisher.java (../jms_example/SimpleBytePublisher.java)
6. SimpleByteSubscriber.java (../jms_example/SimpleByteSubscriber.java)
7. SimpleDestinationPublisher.java (../jms_example/SimpleDestinationPublisher.java)
8. SimpleDestinationSubscriber.java (../jms_example/SimpleDestinationSubscriber.java)
9. SimpleMapPublisher.java (../jms_example/SimpleMapPublisher.java)
10. SimpleMapSubscriber.java (../jms_example/SimpleMapSubscriber.java)
11. SimpleObjectPublisher.java (../jms_example/SimpleObjectPublisher.java)
12. SimpleObjectSubscriber.java (../jms_example/SimpleObjectSubscriber.java)
13. SimpleTopicPublisher.java (../jms_example/SimpleTopicPublisher.java)
14. SimpleTopicSubscriber.java (../jms_example/SimpleTopicSubscriber.java)
15. Version.java (../jms_example/Version.java)

6. Ultra Messaging JMS Programming Quick Start

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, use the following compile line. (`QueueReceiver.java` is the application being compiled.)

```
$JAVA_HOME/bin/javac QueueReceiver.java -cp $JMS_HOME/lib/uJMS_1.0.jar;$JMS_HOME/lib/jms.jar
```

When building on Unix, use the following compile line. (`QueueReceiver.java` is the application being compiled.)

```
$JAVA_HOME/bin/javac QueueReceiver.java -cp $JMS_HOME/lib/JMS_API.jar:$JMS_HOME/lib/jms.jar
```

6.1. Ultra Messaging JMS Synchronous Consumer Implementation

This is a source code listing of a simple synchronous consumer program. You may find it helpful to download the source code (SimpleSyncConsumer.java) (most browsers let you right-click on the link and use the *save link target* function, or some variation).

```
/*file: SimpleSyncConsumer.java
 *
 * Copyright (c) 2005-2014 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.
 */
package examples;

   ❶
import javax.jms.*;
import javax.naming.*;

public class SyncConsumer implements ExceptionListener {

    public static void main(String[] args) {
        new SyncConsumer();
    }

    public SyncConsumer() {
        Context jndiContext = null;

        try {    ❷
            jndiContext = new InitialContext();
        } catch (NamingException e) {
            System.out.println("Could not create JNDI API "
                    + "context: " + e.toString());
            e.printStackTrace();
            System.exit(1);
        }

        try {    ❸
            ConnectionFactory factory = (ConnectionFactory) jndiContext.lookup("uJMSConnectionFactor
            Connection connection = factory.createConnection();

            // Create a Session
            Session session = connection.createSession(false,
                    javax.jms.Session.AUTO_ACKNOWLEDGE);

            // set the exception listener callback
            connection.setExceptionListener(this);

            // Create a topic destination
            Destination destination = session.createTopic("TOPIC.1");

            // create the consumer
            MessageConsumer msgConsumer = session.createConsumer(destination);
            connection.start();

            while (true) {
                System.out.println("Received message " + msgConsumer.receive());
            }

        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

// The exception listener
    public void onException(JMSException e) {
        // print the connection exception status
        System.err.println("Exception occurred: " + e.getMessage());
    }
}
```

Notes:

❶ The SimpleSyncConsumer class consists only of a main method, which receives one or more messages from a topic using synchronous message delivery Run this program in conjunction with SimpleProducer.

❷ Create a JNDI API InitialContext object if none exists yet.

❸ Look up connection factory and topic. If either does not exist, exit.

6.2. Ultra Messaging JMS Asynchronous Consumer Implementation

This is a source code listing of a simple asynchronous consumer program. You may find it helpful to download the source code (SimpleAsyncConsumer.java) (most browsers let you right-click on the link and use the *save link target* function, or some variation).

```
/*file: SimpleAsyncConsumer.java
 *
 * Copyright (c) 2005-2014 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.
 */
package examples;   ❶

import javax.jms.*;
import javax.naming.*;

public class AsyncConsumer implements MessageListener, ExceptionListener {

    public static void main(String[] args) {
        new AsyncConsumer();
        while (true) {
            try {
                Thread.sleep(1000000);
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
    }

    public AsyncConsumer() {
        Context jndiContext = null;

        try {   ❷
            jndiContext = new InitialContext();
        } catch (NamingException e) {
            System.out.println("Could not create JNDI API "
                    + "context: " + e.toString());
            e.printStackTrace();
            System.exit(1);
        }

        try {   ❸
            ConnectionFactory factory = (ConnectionFactory) jndiContext.lookup("uJMSConnectionFactor
            Connection connection = factory.createConnection();

            // Create a Session
            Session session = connection.createSession(false,
                    javax.jms.Session.AUTO_ACKNOWLEDGE);

            // set the exception listener callback
            connection.setExceptionListener(this);

            // Create a topic destination
            Destination destination = session.createTopic("TOPIC.1");
            // create the consumer
            MessageConsumer msgConsumer = session.createConsumer(destination);

            // set the message listener callback
            msgConsumer.setMessageListener(this);
            // start the connection
            connection.start();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

// The exception listener
    public void onException(JMSException e) {
        // print the connection exception status
        e.printStackTrace();
    }
```

```
// The message listener callback
    public void onMessage(Message msg) {
        try {
            System.err.println("Received message: " + msg);
        } catch (Exception e) {
            System.err.println("Exception occurred: " + e.getMessage());
            System.exit(-1);
        }
    }
}
```

Notes:

❶ The SimpleAsyncConsumer class consists only of a main method, which receives one or more messages from a topic using asynchronous message delivery. It uses the message listener TextListener. Run this program in conjunction with SimpleProducer. To end the program, enter Q or q on the command line.

❷ Create a JNDI API InitialContext object if none exists yet.

❸ Look up connection factory and topic. If either does not exist, exit.

6.3. Ultra Messaging JMS Simple Producer Implementation

This is a source code listing of a simple producer program. You may find it helpful to download the source code (SimpleProducer.java) (most browsers let you right-click on the link and use the *save link target* function, or some variation).

```
/*file: SimpleProducer.java
 *
 * Copyright (c) 2005-2014 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.
 */
package examples;

import javax.jms.*;    ❶
import javax.naming.*;

public class Producer {

    public static void main(String[] args) {
        new Producer();
    }

    public Producer() {
        Context jndiContext = null;

        try {    ❷
            jndiContext = new InitialContext();
        } catch (NamingException e) {
            System.out.println("Could not create JNDI API "
                + "context: " + e.toString());
            e.printStackTrace();
            System.exit(1);
        }

        Connection connection = null;    ❸
        try {
            ConnectionFactory factory = (ConnectionFactory) jndiContext.lookup("uJMSConnectionFactor
            connection = factory.createConnection();
            Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
            connection.start();
            Destination destination = session.createTopic("TOPIC.1");
            MessageProducer producer = session.createProducer(destination);
            TextMessage message = session.createTextMessage();
            for (int i = 0; i < 10; i++) {
                ((TextMessage) message).setText("********* This is a test *********** ");
                producer.send(message);
                Thread.sleep(10);
                System.out.println("Sent " + message.getText());
            }
        } catch (Exception e) {
            System.out.println("JNDI API lookup failed: "
                + e.toString());
            e.printStackTrace();
            System.exit(1);

        } finally {
            if (connection != null) {
                try {

                    connection.close();
                    System.exit(0);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Notes:

❶ The SimpleProducer class consists only of a main method, which sends10 messages to a topic. Run this program in conjunction with SimpleAsyncConsumer and SimpleSyncConsumer.

❷ Create a JNDI API InitialContext object if none exists yet.

❸ Look up connection factory and topic. If either does not exist, exit.

7. Starting Ultra Messaging Daemons

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.

This section discusses the following topics.

- *Ultra Messaging Gateway*
- *UMM Daemon*
- *Persistent Store and Queue Daemon*

7.1. Ultra Messaging Gateway

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

1. Know your Topic Resolution Domains. See Topic Resolution Domain (../Gateway/concepts.html#TR-DOMAIN).
2. Consider the size and quality of network paths into and out of your gateways. Obviously, gateways cannot efficiently forward message from a 1 GB path to a 100 MB path.
3. Fully examine any use of Late Join, taking care to configure retransmission options correctly.
4. All remote stores must be named stores.

7.2. UMM Daemon

The UMM Daemon stores **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 (../UMM/umm-daemon.html).

> **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 (../Config/lowestlatency.html) and high throughput (../Config/examples.html#HIGHESTTHROUGHPUT). 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.Serve
   ```
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 (../UMM/umm-gui.html#UMMGUI-USERS)
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
    ```

7.3. Persistent Store and Queue Daemon

The daemon, `umestored`, provides persistent store and queue 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 (see Configuration Reference for Umestored (../UME/umestored-config.html)) or use a sample configuration, ume-example-config.xml (../UME/ume-example-config.xml) or q-example-config.xml (../UME/q-example-config.xml)
3. Start the daemon. `$ umestored config.xml`

   See also Manpage for umestored (../UME/umestored-manpage.html).

8. Next Steps

The Ultra Messaging Concepts (../Design/index.html) introduces the important fundamental concepts you will need to design and code your applications.

The Ultra Messaging Examples page (../example/index.html) contains descriptions and links to 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 and Queuing (../UME/index.html) contains concepts, tutorials and configuration information for **UMP** and **UMQ** , along with discussions about designing persistent or queuing applications. It also contains detailed configuration information for the `umestored` daemon.

The **Ultra Messaging** API documentation (C API (../API/index.html), Java API (../JavaAPI/html/index.html) or .NET API (../DotNetAPI/doc/Index.html)) is a reference guide to the **Ultra Messaging** API. See also JMS API (../JMSAPI/html/index.html), a reference that shows which classes Ultra Messaging JMS supports.

The **UM** Knowledgebase (https://communities.informatica.com/infakb/kbexternal/default.aspx) contains a wealth of articles answering common support questions, error messages, new hardware issues, source code examples and more.