Ultra Messaging (Version 6.14)

Configuration Guide

Contents

Chapter 1

Introduction

This document describes how Ultra Messaging-based user applications are configured.

For information on configuring other UM components, see:

- **Lbmrd (Unicast Resolver Daemon) Configuration**
- **SRS (TCP-based Resolver Service) Configuration**
- **DRO Configuration**
- **Persistent Store Configuration**

For policies and procedures related to Ultra Messaging Technical Support, see UM Support.

This document assumes familiarity with the UM Concepts Guide.

See **UM Glossary** for Ultra Messaging terminology, abbreviations, and acronyms.

1.1 Configuration Overview

There are different kinds of configuration.

- Applications create UM objects (contexts, sources, receivers) using the UM library. Those objects must be configured to control their operation and behavior using **"LBM configuration options"**. An application typically uses an **"LBM configuration file"** in either XML or flat format.

- Informatica daemons (e.g. SRS, Store, DRO) are configured using program-specific configuration files in XML format.

- Informatica daemons (e.g. SRS, Store, DRO) also internally create UM objects (contexts, sources, receivers) using the UM library. Those objects must also be configured using one or more LBM configuration files.

This document describes the options available for LBM configuration.

For Ultra Messaging applications, you can set a variety of operational options to customize the application's behavior or performance. You assign values to these options in configuration files or by using API calls. You can assign option values to objects upon or after object creation. Within an object, the implemented option values are referred to as attributes.

Ultra Messaging uses reasonable default values for configuration options, enabling applications to run "out of the box." However, expect to customize Ultra Messaging options to optimize your operating environment. You can use different ways to configure option default and customized value assignments.

1.1.1 Assignment Methods

You can use the following ways to set attributes with configuration options:

plain text configuration file

The simplest way to configure an application, a plain text configuration file (sometimes called a "flat" file) allows you to re-define UM's default berhaviors. These new defaults are read into a process-global configuration buffer, and are used as UM objects are created. Note that after reading a plain text configuration file, an application can still override the defaults on a per-object basis using UM's API.

XML configuration file

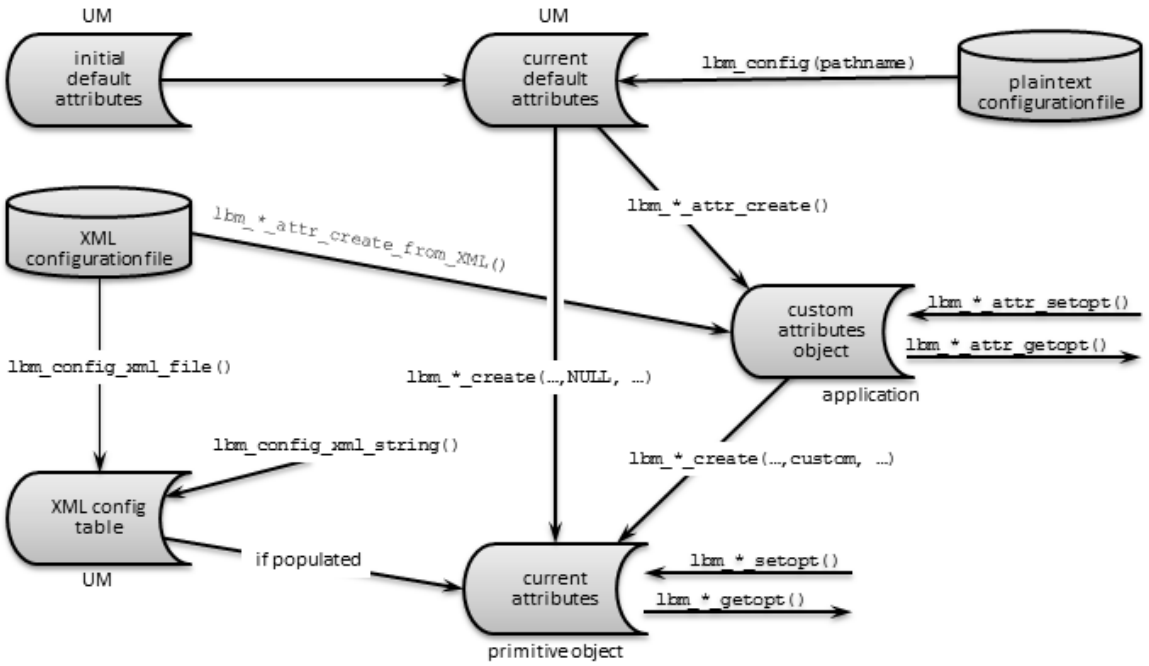
An XML configuration file provides a more sophisticated way to set UM's default behavior, allowing users to customize UM's default behavior on a per-application and/or per-object basis. And while an application can still override the defaults, restrictions can be imposed, constraining applications to only certain options and certain values of those options.

attributes objects API

The application program can call API functions to create UM attributes objects and set configuration options in those objects. The attributes objects are then used to create other UM messaging objects to set those options.

Alternatively, there are API functions which allow you to modify a subset of configurable options on already-created UM messaging objects.

The following image shows the different ways Ultra Messaging stores and assigns option values before, during, and after primitive object creation. Primitive objects are sources, receivers, wildcard receivers, event queues, contexts, or HFX objects. The ultimate result is a primitive object with the assigned values residing in current attributes.



The *initial default attributes* is the set of factory defaults in Ultra Messaging. At process initialization time, these factory defaults are copied into a set of internal process-global attribute structures (*current default attributes*).

An application can modify desired options by reading a *plaintext configuration file*. UM will store these values in *current default attributes*, overwriting the factory defaults.

An instantiated primitive object uses values from current default attributes, the *XML config table*, and the *custom attributes object*, and then holds the results in *current attributes*.

An *XML configuration file* can pass its setting to an object being created either by directly populating the XML config table, or by creating a custom attributes object.

1.1.2 Assignment Flow

The above diagram implies, but does not fully explain, the flow of attribute value assignment that UM performs when an application creates a primitive object. This flow is described below, and is important in understanding how and when default values are overridden:

1. If applicable, copy plain text configuration file values to current process-global default attributes.
2. Start creating object.
3. Custom attributes object(s) created/populated (if applicable).
4. If lbm_∗_create() has a NULL attr, copy current default attributes into current attributes. Otherwise, copy custom attributes object values into current attributes.
5. Read applicable options from the XML config table into the current attributes. Do not overwrite options set with **lbm_config()**, or lbm_∗_attr_setopt(), which were tagged when modified.
6. Finish object creation.
7. current attributes can be changed further (only certain options) via lbm_∗_setopt().

1.1.3 Definitions

Before discussing how UM options can be set, some terminology is in order.

Option

A single configuration item that controls some aspect of UM operation. An option typically resides in a configuration file, but can also be assigned a value via API call. We use options to assign values to an object's attributes.

Attribute

An operational characteristic of an object. An attribute's value is set by an option, hence, there is a one-to-one correspondence between options and attributes. (Note: This use of the term "attribute" is unrelated to, and not to be confused with, "attribute" in XML syntax. In this document, we refer to the latter as "XML attribute".)

XML attribute

See above. In XML syntax, XML attributes are parameters for XML elements.

Custom attributes object

A UM object that contains custom attribute values (set by options) for a specific UM object. Separate (and multiple) sets of attributes can exist for each application, though only one can be used when creating a primitive object.

Initial default attributes

The default attributes values built into UM. UM and your applications use these if you have not set any options for the attributes.

Primitive object

Specifically, an object that is a source, receiver, wildcard receiver, event queue, context, or HFX object.

Configuration file

This comes in two types: XML and plain text. Configuration files contain assigned values for options, but the different types are read/copied at different times during the creation of an object.

XML config table

Contains option values that are read from the XML configuration file.

Current default attributes

The attributes values used to create an object in the absence of custom attributes values.

Current attributes

The attribute values for an instantiated UM object that control the current operation of that object.

Scope

The type of object to which an option can apply. Possible scopes are context, source, receiver, wildcard_↩ receiver, event_queue, and hfx.

1.1.4 Which Method Should I Use?

For the four basic assignment methods listed above, following are some scenarios where specific methods are selected.

- To change a default option value and apply it to all objects you create, call **lbm_config()** for one or more configuration files. For example, to use LBT-RM rather than TCP for all sources, create a plain text configuration file containing

  ```
  source transport LBTRM
  ```

  and pass its file name to **lbm_config()**.

  Note

  > The C API offers APIs lbm_∗_attr_create_default() to create an attribute object with initial factory default values. This is unaffected by default overrides from configuration files. No such corresponding method exists for the Java or .NET APIs.

- To customize specific options before an object is created for a specific object instance, use a custom attributes object. Also, you can assign XML data to the XML config table directly from your application via **lbm_config_xml_string()**.
- To create sets of custom values to be used when creating primitive objects, call **lbm_config_xml_file()** and specify an XML configuration file. This is useful for setting specific default options on a per-topic or per-context basis, which cannot be done with a plain text configuration file. For an example where a sending application uses specific options and values, create an XML configuration file with the application's name (optional) that specifies those options and values. Then pass the XML file name and application name to **lbm_config_xml_file()**.
- To change an option after an object is created, modify the current attributes for the object. (Note that many options cannot be changed after an object has been created.)

These methods can be used in combination. See Assignment Methods to see the relationships between attributes and the various UM API function calls that affect them.

1.1.5 Configuration Error Handling

Prior to UM version 6.13, an error in a configuration file typically resulted in the remainder of the configuration file not being processed.

As of UM version 6.13, UM will attempt to process the entire configuration file, even if there are errors. Any lines which cannot be properly parsed will generate an error to the logger, but subsequent lines will still be processed.

Note that the API function will return a bad status to indicate that one or more errors were encountered, and the application can decide what to do. For example:

```
err = lbm_config("test.cfg");
if (err == LBM_FAILURE) {
  fprintf(stderr, "Warning, ignoring lbm_config error: %s\n", lbm_errmsg());
}
```

The last error encountered will be returned by the **lbm_errmsg()** call. Note that all errors are also reported via the UM logger callback; see **lbm_log()**. Thus, if the "test.cfg" file has two errors in it, both will be logged to the logger, and the last one is returned by **lbm_errmsg()**.

Why Ignore Errors?

Normally an application would want to exit if **lbm_config()** returned an error. However, here is an example use case for considering it a warning and continuing with the application execution.

Let's imagine that future UM version 42.0 has a new configuration option, "receiver predict_next_message 1", which allows receivers to receive messages before they are sent, achieving the elusive goal of negative latency. The user could include this option in the master configuration file, and all applications at version 42.0 and beyond will benefit from negative latencies. Applications at version 6.13 through 41 will log an error when they encounter that option, but will continue to load the rest of the configuration file, and will run normally.

Thus, new configuration options can be included in a master configuration file, and older versions will log warnings about but will still run.

Note however that any pre-6.13 applications will return an error and *not* process the rest of the configuration file.

Also note that the user must be very careful to examine the error messages logged to ensure that all errors are expected (due to earlier versions not understanding options included for later versions). If lbm_config errors are ignored, it would be easy to overlook a mistyped option and have it not take effect.

XML Errors

If an XML configuration file is being used, the basic XML structure must be valid for the parser to read the whole file.

For example:

```
<?xml version="1.0" ?>
<um-configuration version="1.0">
  <mistake />
  <applications>
    <application>
      <contexts>
        <context>
          <options type="context">
            <option name="request_tcp_port_low" default-value="13000">
            <option name="request_tcp_port_high" default-value="13010">
            </option>
          </options>
        </context>
      </contexts>
    </application>
  </applications>
</um-configuration>
```

This XML file will be rejected *without* applying the two request port options because of the "<mistake />" element.

However, if the XML elements are properly coded, invalid option names or values will follow the same pattern as flat configuration files: errors will be reported, and parsing will continue.

For example:

```
<?xml version="1.0" ?>
<um-configuration version="1.0">
  <applications>
    <application>
      <contexts>
        <context>
          <options type="context">
            <option name="mistake" default-value="1">
            <option name="request_tcp_port_low" default-value="13000">
            <option name="request_tcp_port_high" default-value="13010">
            </option>
          </options>
        </context>
      </contexts>
    </application>
  </applications>
</um-configuration>
```

This XML file will be accepted and the request TCP port range will be applied. The "mistake" option will log an error. And the API will return LBM_FAILURE, which the application can decide to ignore.

Daemon Config Files

This behavior of continuing execution in the event of errors is extended to the UMP Store and the DRO for UM configuration configuration files. That is, an error found in a UM configuration file will be logged, and the daemon will continue to run. Users should again be very careful to examine log files to prevent mistyped options from leading to undesired behavior.

However, for the Store and DRO configuration XML files, this "log and continue" error handling is *not* used. Errors in the Store or DRO configuration files will prevent them from running.

1.1.6 Host Name Resolution

Many of UM's configuration options specify an IP address. Prior to UM version 6.10 these needed to be specified in dotted numeric format. For example, **10.23.19.210**. Starting in version 6.10, any configuration option that accepts an IP address can also accept a DNS host name (the few exceptions are noted in the documentation). For example, **myhost.mydomain.com**. Note that the DNS name system is not necessarily used when host names are specified; for example, most Unix systems will first look up the name in /**etc**/**hosts**.

When host names are specified, the name is resolved to an IP address when the configuration option is parsed. If you change the IP address associated with a name, that change will not take effect until the configuration file is re-read, typically by restarting the application.

1.1.7 Configuration Files

There are two types of UM Configuration files:

- Plain Text Configuration Files
- XML Configuration Files

You can read Configuration files either by API call, or automatically upon application launch by specifying a file name in an environment variable. See Assignment Methods and Assignment Flow for details on how these options replace or override default values.

There are some UM configuration options which cannot be set via configuration files. These are options whose values are function pointers or data structures. These options can only be set via API functions ∗**_setopt**. For example, **context resolver_source_notification_function** has a function pointer as its value.

1.2 Plain Text Configuration Files

The plain text configuration file (sometimes called a "flat" file), when invoked, writes option values into UM's current default attributes. These are then read and used in the creation of all objects.

See Example Configuration Scenarios for example configuration files.

1.2.1 Reading Plain Text Configuration Files

There are two ways to read a plain text configuration file to set values in current default attributes.

API function lbm_config()

You can call the API multiple times with different file names to set configuration options in phases.

When you create UM objects (such as a context or receiver), UM sets attributes for that object using the current default attributes. Hence, you must call **lbm_config()** before creating objects (lbm_∗_create()).

Environment variable LBM_DEFAULT_CONFIG_FILE

Reads configuration file when your application is started. You can set this variable to a full pathname or a URL; for example:

```
export LBM_DEFAULT_CONFIG_FILE=/home/lbm/lbtrm.cfg
```

(You can still use the **lbm_config()** API on a different file to make additional changes.)

1.3 Plain Text Configuration File Format

A plain text configuration file contains lines that each take the form:

scope_keyword option_name option_value

where:

scope_keyword - the scope to which the option applies,

option_name - the predefined name for the option, and

option_value - the new value to be assigned to that option.

Allowable values for these parameters are given throughout the rest of this document. Any text following a hash character # (also known as a pound sign, number sign, or octothorp) is interpreted as comment text and is ignored.

For example:

```
# Set transport_tcp_port_low to 4901
context transport_tcp_port_low 4901
# And set transport_tcp_port_high to 4920
context transport_tcp_port_high 4920
```

Note

For plain text configuration files, do not enclose any fields in double quotation marks (").

Chapter 2

XML Configuration Files

XML configuration files let you address many different applications and operating requirements, removing the need to programmatically set and reset options for them. A single XML file can contain options for multiple applications. Moreover, for a single application, you can configure multiple named contexts, event queues, etc., with different values for the same options.

See Example Configuration Scenarios for example configuration files.

2.1 XML Configuration Concepts

The primary motivation for using XML-based configuration is to be able to configure different instances of the same object in different ways. For example, with a single XML configuration file, you can specify different options for different applications. Or, within a given application, you can specify different options for different context objects. Or within a given context, you can specify different options for different topic-based objects (sources and/or receivers). You can do this without the necessity of writing special application code to do it.

Users often have large sets of configuration options that apply to multiple applications or objects. Rather than having to reproduce the entire set for each application or object, *templates* can be used to give common sets a name that can be referenced in applications or objects in the XML file.

Applications may also override configuration options specified in an XML configuration file, either by using a plain text configuration file, or by using UM's API.

However, it is also possible for the XML configuration file to impose restrictions on the application's ability to override options. Using <allow> and <deny> elements, and the `order` attribute, XML files can constrain application to using specific values for desired options.

2.2 XML Reference Names

Given that XML configuration files are intended to allow different objects to be configured differently, there needs to be a way to identify which object should have which configuration. This is done with application and object names, which the XML file references.

Context and Event Queue names are case-sensitive and can consist of only alpha-numeric ASCII characters, dash (-), and underscore (_). They must be 127 characters or less.

Valid examples:

- **abc**
- **123-abc**
- **XYZ_xyz**

Invalid examples:

- **abc xyz** *(no spaces allowed)*
- **123.abc** *(no period allowed)*
- **XYZ,xyz** *(no comma allowed)*

Template and application names are case-sensitive and can consist of any printable ASCII characters. They must be 99 characters or less.

2.2.1 XML Object Names

The simplest apps create UM objects without using attribute objects. For example:

```
err = lbm_context_create(&ctx, NULL, NULL, NULL);
```

Passing NULL for the context attribute object causes UM to simply use the defaults (as possibly modified by a configuration file).

However, if there is a chance that you will want to be able to configure the objects differently, you should create an attribute object using the appropriate "*_attr_create_from_xml()" API, giving it a descriptive name. For example:

```
lbm_context_attr_t *ctx_attr;
...
err = lbm_context_attr_create_from_xml(&ctx_attr, "main_ctx");
err = lbm_context_create(&ctx, ctx_attr, NULL, NULL);
```

You can do this even if you do not yet make use of an XML file. If no XML file has been read, that **lbm_context_attr_create_from_xml()** does the same thing as **lbm_context_attr_create()**.

However, if an XML file *is* supplied and it specifies a configuration for a context named "main_ctx", the "*_attr_create_from_xml()" API will first load the attribute object with the default values and will then apply the proper XML defaults to the attribute object. Desired options can then be overridden using the appropriate "*_attr_setopt()" or "*_attr_str_setopt()" APIs.

The full set of XML-enabled attribute creation APIs are: **lbm_context_attr_create_from_xml()**, **lbm_src_topic_attr_create_from_xml()**, **lbm_rcv_topic_attr_create_from_xml()**, **lbm_event_queue_attr_create_from_xml()**, **lbm_wildcard_rcv_attr_create_from_xml()**, **lbm_hfx_attr_create_from_xml()**.

Note

It is also possible to call **lbm_context_attr_create_from_xml()** passing NULL as the context name. This matches a XML <context> element that has no `name` attribute. An unnamed "<context>" element only matches unnamed contexts.

2.2.2 XML Application Names

An XML configuration file groups configurations into one or more <application> elements. Normally, each <application> is given a unique name, using the "name" attribute. It is also permissible to include an <application> element without a name attribute (the "unnamed" application element).

When an application starts and attempts to use an XML configuration file, the application normally gives UM its name, which UM will match to one of the XML file's <application> elements. It is also permissible that the application does not give itself a name, in which case UM will match the unnamed <application> element.

Unlike most XML elements, the <applications> element does not have a way to allow applications with names that have no matching application element. For example, let's say the XML file contains:

```
<?xml version="1.0" ?>
<um-configuration version="1.0">
  <applications>
    <application name="app1">
      ...
    </application>
    <application name="app2">
      ...
    </application>
    <application>
      ...
    </application>
  </applications>
</um-configuration>
```

If an application starts up and is named "app3", it will fail to initialize. The above file only allows applications named "app1", "app2", and applications that do not set a name. Note that if the unnamed application element is removed from that XML file, then an unnamed application will fail to initialize.

There are several ways to give an application the name that can be referenced by an XML configuration file:

- Set the environment variable LBM_XML_CONFIG_APPNAME. (But be aware that it is ignored if the **lbm_config()** API is used to read the XML configuration file.)
- Read the XML configuration file using the **lbm_config_xml_file()** API.
- Invoke the UMM feature using the **lbm_set_umm_info()** API or using the LBM_UMM_INFO environment variable. See **UM Manager Overview** for more information on UMM.

Note that the application name is *not* related to the executable file name of the program, or the operating system process name. The application name is assigned via one of the above methods only.

2.3 Order and Rule Specifications

An XML configuration file can constrain how an application may override the values supplied in the XML configuration file. It can also restrict which topics the application may publish and subscribe to. These two use cases are handled slightly differently.

2.3.1 Constraining Configuration Values

The way to think of the order attribute in the <option> element is as follows:

- For order "allow,deny" the XML should contain zero or more values that are allowed. If a user-supplied value doesn't match any of them, it is denied.

- For order "deny,allow" the XML should contain zero or more values that are denied. If a user-supplied value doesn't match any of them, it is allowed.

Consider the following fragment of XML:

```
<receivers>
  <topic>
    <options type="receiver">
      <option name="ordered_delivery" order="deny,allow">
        <deny>0</deny>
      </option>
    </options>
  </topic>
</receivers>
```

This prevents the user from setting ordered_delivery (receiver) to 0, but allows values 1 and -1. But the values 1 and -1 are not explicitly allowed. The order attribute is set to "deny,allow", has "allow" as the default behavior if the user-supplied value doesn't match one of "<allow>" or "<deny>" values.

Contrast with this fragment:

```
<receivers>
  <topic>
    <options type="receiver">
      <option name="ordered_delivery" order="allow,deny">
        <allow>1</allow>
      </option>
    </options>
  </topic>
</receivers>
```

This allows the value 1 but denies all others. The order attribute is set to "allow,deny", has "deny" as the default behavior if the user-supplied value doesn't match one of "<allow>" or "<deny>" values.

2.3.2 Restricting Topics

Consider the following fragment of XML:

```
<receivers order="allow,deny">
  <topic topicname="general_info" rule="allow"/>
  <topic topicname="alerts" rule="allow"/>
</receivers>
```

This allows the application to create receivers for topics "general_info" and "alerts" and disallows all others. The order attribute is set to "allow,deny", has "deny" as the default behavior if the user-supplied topic doesn't match one of "<allow>" or "<deny>" values.

Contrast with this fragment:

```
<receivers order="deny,allow">
  <topic topicname="authorize" rule="deny"/>
</receivers>
```

This allows the application to subscribe to any topic except "authorize". The order attribute is set to "deny,allow", has "allow" as the default behavior if the user-supplied value doesn't match one of "<allow>" or "<deny>" values.

Warning

With the above <topic> elements, an application can bypass the intended restrictions by using a wildcard receiver, perhaps with the pattern ".*". This allows the application to effectively subscribe to all topics. The <topic> elements do not restrict wildcard receivers.

Since wildcard patterns can be complex, users who wish to restrict applications should either disallow wildcard receivers, or carefully constrain them. For example:

```
<receivers order="deny,allow">
  <topic topicname="authorize" rule="deny"/>
</receivers>
<wildcard-receivers order="allow,deny">
  <wildcard-receiver pattern="^abc.*$" rule="allow"/>
  <wildcard-receiver pattern="^xyz.*$" rule="allow"/>
</wildcard-receivers>
```

This allows the application to create any single-topic receiver except for the topic "authorize", and it allows two patterns for wildcard receivers (neither of which will match the topic "authorize").

Another, more-restrictive example:

```
<receivers order="allow,deny">
  <topic topicname="general_info" rule="allow"/>
  <topic topicname="alerts" rule="allow"/>
</receivers>
<wildcard-receivers order="allow,deny"/>
```

This only allows the application to subscribe to the two topics "general_info" and "alerts", and it completely disallows any wildcard receivers.

2.3.3 Overlapping Topics

There are some use cases where a special property of the `order` attribute is useful: the order in which `allow` and `deny` rules are applied. When multiple <topic> elements match a given topic name due to overlapping wildcard patterns, the order of applying the rules can be important to obtain the desired behavior.

Consider this example:

```
<receivers order="deny,allow">
  <topic pattern="^trade" rule="deny"/>
  <topic pattern="^trade\.NASD" rule="allow"/>
</receivers>
```

Let's assume that the application subscribes to "trade.NASD.xyz". This matches both patterns. By ordering the patterns as deny first, followed by allow, the *last* match is allow, which allows the topic to be created. The last rule to match determines the permission.

Whereas subscribing to "trade.abc.xyz" will only match the deny, and will be prevented.

Also note that subscribing to "quote", which does not match either topic, follows the default rule, which is allow.

So the above XML allows all non-trade subscriptions, but only allows NASD trade subscriptions.

Contrast with this example:

```
<receivers order="allow,deny">
  <topic pattern="^trade" rule="allow"/>
  <topic pattern="^trade\.NASD" rule="deny"/>
</receivers>
```

Let's again assume that the application subscribes to "trade.NASD.xyz". This also matches both patterns, but in this case the allows are first and the denies are last. Thus, "trade.NASD.xyz" is prevented.

So while the previous example only allowed NASD trades, this example allows any trades *except* NASD.

Also note that subscribing to "quote", which does not match either topic, follows the default rule, which is deny.

2.4 UM Default Values

The following examples will help to illustrate how UM defaults work. In the code fragments shown, the UM API calls shown are assumed to be the first UM API calls made since the process started.

No Attribute Object, No Config File

test.c:

```
err = lbm_context_create(&ctx, NULL, NULL, NULL);
/* The context has request_tcp_port_low = 14393 (factory default) */
```

1. When UM initializes, the "factory defaults" are copied to the process-global internal attribute objects.
2. In the call to **lbm_context_create()**, setting the `"attr"` parameter to NULL causes UM to use the process-global internal context attribute object to create the context.

No Attribute Object, Plain Text Configuration File

test.cfg:

```
context request_tcp_port_low 12000
```

test.c:

```
err = lbm_config("test.cfg");
err = lbm_context_create(&ctx, NULL, NULL, NULL);
/* The context has request_tcp_port_low = 12000 */
```

1. When UM initializes, the "factory defaults" are copied to the process-global internal attribute objects.
2. The call to **lbm_config()** reads the options in the file "test.cfg" and applies them to the process-global internal attribute objects. This overrides the factory default for request_tcp_port_low (context).
3. In the call to **lbm_context_create()**, setting the `"attr"` parameter to NULL causes UM to use the process-global internal context attribute object to create the context.

Attribute Object, Plain Text Configuration File

test.cfg:

```
context request_tcp_port_low 12000
```

test.c:

```
err = lbm_config("test.cfg");
err = lbm_context_attr_create(&ctx_attr);
err = lbm_context_create(&ctx, ctx_attr, NULL, NULL);
/* The context has request_tcp_port_low = 12000 */
```

1. When UM initializes, the "factory defaults" are copied to the process-global internal attribute objects.
2. The call to **lbm_config()** reads the options in the file "test.cfg" and applies them to the process-global internal attribute objects. This overrides the factory default for request_tcp_port_low (context).
3. The **lbm_context_attr_create()** API copies the process-global internal context attribute object, with the overridden request_tcp_port_low (context).
4. The call to **lbm_context_create()** passes the attribute object with the overridden request_tcp_port_low (context).

Note

The use of **lbm_context_attr_create()** is not recommended. See next example.

Attribute Object, Plain Text and XML Configuration Files

In this example, the user intends the default for request_tcp_port_low (context) to be overridden to 13000, but there's a problem.

test.xml:

```
<?xml version="1.0" ?>
<um-configuration version="1.0">
  <applications>
    <application>
      <contexts>
        <context>
          <options type="context">
            <option name="request_tcp_port_low" default-value="13000">
            </option>
          </options>
        </context>
      </contexts>
    </application>
  </applications>
</um-configuration>
```

test.cfg:

```
  context request_tcp_port_low 12000
```

test.c:

```
err = lbm_config_xml_file("test.xml", NULL);
err = lbm_config("test.cfg");
err = lbm_context_attr_create(&ctx_attr);
err = lbm_context_create(&ctx, ctx_attr, NULL, NULL);
/* The context has request_tcp_port_low = 12000!! */
```

1. When UM initializes, the "factory defaults" are copied to the process-global internal attribute objects.
2. The call to **lbm_config_xml_file()** reads the elements in the file "test.xml" and stores them internally. Note that **lbm_config_xml_file()** *does not modify* the process-global internal attribute objects.
3. The call to **lbm_config()** reads the options in the file "test.cfg" and applies them to the process-global internal attribute objects. This overrides the factory default for request_tcp_port_low (context) with 12000.
4. The **lbm_context_attr_create()** API copies the process-global internal context attribute object, with the overridden request_tcp_port_low (context). Note that the XML default is *not applied* when the attribute object is created using **lbm_context_attr_create()**.
5. The call to **lbm_context_create()** passes the attribute object with the overridden request_tcp_port_low (context) of 12000.

In this example, the use of **lbm_context_attr_create()** resulted in the *XML file's default being ignored*. However, see the next example.

Attribute Object, Plain Text and XML Configuration Files, Plus Restriction

In this example, the application is constrained to only allow 12000.

test.xml:

```
<?xml version="1.0" ?>
<um-configuration version="1.0">
  <applications>
    <application>
      <contexts>
        <context>
          <options type="context">
            <option name="request_tcp_port_low" default-value="13000"
                    order="allow,deny">
```

```
            <allow>13000</allow>
          </option>
        </options>
      </context>
    </contexts>
  </application>
 </applications>
</um-configuration>
```

test.cfg:

```
context request_tcp_port_low 12000
```

test.c:

```
err = lbm_config_xml_file("test.xml", NULL);
err = lbm_config("test.cfg");
err = lbm_context_attr_create(&ctx_attr);
err = lbm_context_create(&ctx, ctx_attr, NULL, NULL);
/* ERROR RETURNED! */
```

As with the previous example, the default value supplied in the XML configuration file is ignored due to the use of **lbm_context_attr_create()**. However, the XML file's restrictions applied by `order="allow,deny"` and `<allow>13000</allow>` *are* applied at object creation time. Since only 13000 is allowed, but 12000 was attempted, the **lbm_context_create()** API fails.

Attribute Object From XML, Plain Text and XML Configuration Files, Plus Restriction

test.xml:

```
<?xml version="1.0" ?>
<um-configuration version="1.0">
  <applications>
    <application>
      <contexts>
        <context>
          <options type="context">
            <option name="request_tcp_port_low" default-value="13000"
                    order="allow,deny">
              <allow>13000</allow>
            </option>
          </options>
        </context>
      </contexts>
    </application>
  </applications>
</um-configuration>
```

test.cfg:

```
  context request_tcp_port_low 12000
```

test.c:

```
err = lbm_config_xml_file("test.xml", NULL);
err = lbm_config("test.cfg");
err = lbm_context_attr_create_from_xml(&ctx_attr, NULL);
err = lbm_context_create(&ctx, ctx_attr, NULL, NULL);
/* The context has request_tcp_port_low = 13000 */
```

In this example, the attribute object is created using the **lbm_context_attr_create_from_xml()** API. It is created without a name, and therefore matches the "<context>" option that has no `name` attribute. This allows the `default-value` attribute to override the default present in the internal process-global context attribute object. So the call to **lbm_context_create()** succeeds.

Note that if the plain text configuration file "test.cfg" had other UM options set, those overridden defaults would have appeared in the attribute object created by **lbm_context_attr_create_from_xml()**.

Named Attribute Object, XML Configuration File

In this example, the context is named.

test.xml:

```
<?xml version="1.0" ?>
<um-configuration version="1.0">
  <applications>
    <application name="App1">
      <contexts>
        <context name="MainCtx">
          <options type="context">
            <option name="request_tcp_port_low" default-value="13000"
                    order="allow,deny">
              <allow>13000</allow>
            </option>
          </options>
        </context>
      </contexts>
    </application>
  </applications>
</um-configuration>
```

test.c:

```
err = lbm_config_xml_file("test.xml", "App1");
err = lbm_context_attr_create_from_xml(&ctx_attr, "MainCtx");
err = lbm_context_create(&ctx, ctx_attr, NULL, NULL);
```

In this example, the application and context names are specified and matched in the XML file. This is the recommended way of using UM. In fact, even if no XML file is used at all, the `*_attr_create_from_xml()` APIs are recommended to be used, and descriptive names supplied. This "future-proofs" your code so that flexible XML configurations can be added later on without needing to change your source code.

2.5 Reading XML Configuration Files

There are multiple ways to read an XML configuration file to assign values while creating a primitive object.

API function lbm_config_xml_file()

Reads an XML configuration file into XML config table. Call this before the primitive create API. This does not change the current default attributes. Use a file path, or a URL beginning with `http://` or `ftp://`.

API function lbm_config_xml_string()

Populates the XML config table directly from your application. Call this before the primitive create API. This does not change the current default attributes.

API function lbm_*_attr_create_from_XML()

Creates a custom attributes object containing the values from an XML configuration file. The values can then be applied to a primitive object being created by calling API "lbm_*_create()" and specifying this custom attributes object in the second parameter.

Environment variable LBM_XML_CONFIG_FILENAME

Reads the file into the XML config table. These settings are then available to all applications when they start. Use a file path, or a URL beginning with `http://` or `ftp://`.

Environment variable LBM_XML_CONFIG_APPNAME

Reads options for a specific application from the LBM_XML_CONFIG_FILENAME variable's filename. This initiates the specified application's configuration; set this environment variable for every application. Note that this variable is ignored if the XML configuration file is read using the **lbm_config()** API.

API function lbm_set_umm_info()

Initiates the application to read options for an application and user from the UMM daemon. The Java API and .NET API is **com::latencybusters::lbm::LBM::setUmmInfo()**.

Environment variable LBM_UMM_INFO

Initiates the application to read options for an application and user from the UMM daemon. Set this variable for every application/user combination, in the following format:

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

2.6 Using XML Configuration Files With a UM Application

The following procedure describes a general approach to implementing XML configuration files.

1. Create an XML configuration file using an XML editor or text editor. Just for this example, name the file, UM_CONFIG.XML.
2. Insert desired templates in the <**templates**> element. Each template holds configuration options shared by multiple applications or primitive UM objects. You can apply multiple templates to an application and its primitive UM objects, however if the same option appears in multiple templates, the option value in the *last* template overrides the option value in any previous templates. See <templates>.
3. Insert an <**application**> element for your UM application in the <**applications**> element and reference any relevant templates created in the previous step. Just for this example, name the application, SENDAPP. See <applications>.
4. Within the <**contexts**> element, configure the application's <**context**> element and context options. And since our example application, SENDAPP is a sending application, also configure its Source options. (If this was a receiving application, you would configure Receiver or Wildcard Receiver options. Note that most real-world applications both send and receive messages, and would therefore have both.) If your application creates multiple Contexts, enter multiple <**context**> elements within the <**contexts**> element, inserting the appropriate source, receiver or wildcard receiver options. See <contexts>.
5. Configure the applications Event Queue options. See <event-queues>.
6. Save the XML configuration file, UM_CONFIG.XML, and load it onto the machine where the application (SENDAPP) runs.
7. Have the application (SENDAPP) read the XML file. The preferred way to do this is the **lbm_config_xml_file()** API. However, if it is not possible to modify the application's code, set the following environment variables:
   - Set LBM_XML_CONFIG_FILENAME to UM_CONFIG.XML.
   - Set LBM_XML_CONFIG_APPNAME to SENDAPP.
8. Start SENDAPP.

2.7 XML Configuration File Format

A UM XML Configuration File follows standard XML conventions. The first line should be:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

followed by UM elements.

An XML configuration file generally comprises two primary elements: templates and applications. Organized and contained within these are option value assignments. Applications containers let you set options for specific applications. To provide more global control over applications, or to simply reduce repetition, you can create templates to hold option settings that are to be used in one or more different applications.

XML configuration files use the high-level structure shown in the following example. This example includes only some container elements, and no options.

```
<?xml version="1.0" encoding="UTF-8" ?>
<um-configuration version="1.0">
  <templates>
    <template name="Sending">
      <options type="source">
      </options>
      <options type="context">
      </options>
    </template>
  </templates>
  <applications>
    <application name="Sending-Topic1">
      <contexts>
        <context name="Sending-LBTRM">
          <sources>
            <topic topicname="Topic1">
              <options type="source">
              </options>
            </topic>
          </sources>
        </context>
      </contexts>
      <event-queues>
        <event-queue/>
        <event-queue name="EQ-1"/>
      </event-queues>
    </application>
  </applications>
</um-configuration>
```

2.8 Merging Multiple XML Configuration Files

For UM XML configuration files and UMP store daemon XML configuration files, you can use the XInclude mechanism to merge multiple configuration files.

To include an external file, use the following syntax:

```
<xi:include xmlns:xi=http://www.w3.org/2003/XInclude" href="filename.xml" />
```

Files to be included must be formatted such that all elements are enclosed in a single container element, as shown in the following examples:

Example 1

```
<ume-attributes>
  <option type="store" name="allow-proxy-source" value="1"/>
  <option type="lbm-context" name="resolver_multicast_address" value="239.255.38.0" />
  <option type="lbm-context" name="resolver_multicast_port" value="19999" />
</ume-attributes>
```

Example 2

```
<topics>
  <topic pattern="." type="PCRE">
    <ume-attributes>
      <option type="store" name="repository-type" value="disk"/>
      <option type="store" name="retransmission-request-forwarding" value="0"/>
    </ume-attributes>
  </topic>
</topics>
```

2.9 XML Configuration File Elements

Here's a "cheat sheet" showing all of the XML elements.

```
<um-configuration>
  <license format="...">...</license>

  <templates>
    <template name="...">
      <options type="...">
        <option name="..." default-value="..." order="...">
          <allow>...</allow>
          <deny>...</deny>
        </option>
      </options>
    </template>
  </templates>

  <applications>
    <application name="..." template="...">
      <contexts order="..." template="...">
        <context name="..." template="..." rule="...">
          <options ...>...</options>    (see templates for expansion)

          <sources template="..." order="...">
            <topic template="..." rule="..." topicname="..." pattern="...">
              <options ...>...</options>    (see templates for expansion)
            </topic>
          </sources>

          <receivers order="..." template="...">
            <topic template="..." rule="..." topicname="..." pattern="...">
              <options ...>...</options>    (see templates for expansion)
            </topic>
          </receivers>

          <wildcard-receivers template="..." order="...">
            <wildcard-receiver template="..." rule="..." pattern="..." pattern-type="...">
              <options ...>...</options>    (see templates for expansion)
            </wildcard-receiver>
          </wildcard-receivers>
      </context> </contexts>

      <hfxs template="..." order="...">
        <topic template="..." rule="..." topicname="..." pattern="...">
          <options ...>...</options>    (see templates for expansion)
        </topic>
      </hfxs>

      <event-queues template="..." order="...">
        <event-queue name="..." template="..." rule="...">
          <options ...>...</options>    (see templates for expansion)
        </event-queue>
      </event-queues>

    </application>
  </applications>
</um-configuration>
```

2.9.1 UM Element "<um-configuration>"

Container element that holds the UM configuration. Also defines the version of the configuration format used by the file.

- **Children:** <license>, <templates>, <applications>

XML Attributes:

Attribute	Description	Valid Values	Default Value
version	Version number of user's configuration file.	string	**1.0**

Example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<um-configuration version="1.0">
  ...
</um-configuration>
```

2.9.2 UM Element "<applications>"

Container element that holds the configurations for different applications.

- **Parent:** <um-configuration>
- **Children:** <application>

Example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<um-configuration version="1.0">
  <applications>
    ...
  </applications>
  ...
</um-configuration>
```

2.9.3 UM Element "<application>"

Container element that holds the configuration for a specific application.

Note

Applications that set a name which is not included by any <application> element will fail. There is no "default" <application> element that allows and configures applications with non-matching names.

- **Cardinality:** 0 .. unbounded
- **Parent:** <applications>
- **Children:** <contexts>, <event-queues>, <hfxs>, <application-data>

XML Attributes:

Attribute	Description	Valid Values	Default Value
name	A case-sensitive label which UM matches to an application's assigned name. An application is typically assigned a name via API, e.g. **lbm_config_xml_file()**, or by environment variable, LBM_XML_CONFIG_APP_NAME. See XML Reference Names for more information. Names are case-sensitive and can consist of any printable ASCII characters. They must be 99 characters or less.	string	(If omitted, matches applications that don't set a name.)
template	A case-sensitive label which UM matches to a template's assigned name.	string	(If omitted, no template is applied.)

Example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<um-configuration version="1.0">
  <templates>
    <template name="FIX_Config.Prod">
      ...
    </template>
    ...
  </templates>
  <applications>
    <application name="FIX.01" template="FIX_Config.Prod">
      ...
    </application>
    ...
  </applications>
  ...
</um-configuration>
```

2.9.4 UM Element "<application-data>"

Free-form text comment field. **Deprecated; do not use.**

- **Parent:** <options>, <application>
- **Default Value:** Deprecated; do not use.

XML Attributes:

Attribute	Description	Valid Values	Default Value
xml:space	Specifies how whitespace (tabs, spaces, linefeeds) are handled in the element content. See **xml:space Attribute**.	**"default"** - Trim whitespace. **"preserve"** - Retain whitespace exactly as entered.	**default**

Deprecated; do not use.

2.9.5 UM Element "<hfxs>"

Container element that holds the configuration for HFX objects. The contained <topic> elements are matched by topic to the HFX objects created by the application. The `order="..."` attribute is used to constrain the application's access to topics. See Order and Rule Specifications for details.

See **UM Hot Failover Across Contexts Objects** for more information on HFX.

- **Parent:** <application>
- **Children:** <topic>

XML Attributes:

Attribute	Description	Valid Values	Default Value
template	A case-sensitive label which UM matches to a template's assigned name.	string	(If omitted, no template is applied.)
order	Valid values are "deny,allow" and "allow,deny". Used to control how HFX usage is restricted. See Order and Rule Specifications and Overlapping Topics.	string	"deny,allow"

Example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<um-configuration version="1.0">
  <templates>
    <template name="FIX_Config.Prod">
      ...
    </template>
    ...
  </templates>
  <applications>
    <application name="FIX.01" template="FIX_Config.Prod">
      <hfxs template="FIX_Config.Prod" order="deny,allow">
        ...
      </hfxs>
      ...
    </application>
    ...
  </applications>
  ...
</um-configuration>
```

2.9.6 UM Element "<topic>"

Used to match UM objects (sources, receivers, HFX receivers) by their topic names, and control their use and configuration. The attributes `topicname` and `pattern` are mutually exclusive; you may not supply both.

Warning

If the `rule` attribute is being used to restrict the application's receivers, remember that wildcard receivers must also be restricted. For example, if the application must be prevented from subscribing to the "authorize" topic, it is not enough to use:

```
<topic topicname="authorize" rule="deny"/>
```

Wildcards must also be limited or forbidden. For example, to forbid all wildcard receivers:

```
<wildcard-receivers order="allow,deny"/>
```

- **Cardinality:** 0 .. unbounded
- **Parent:** <hfxs>, <sources>, <receivers>
- **Children:** <options>

XML Attributes:

Attribute	Description	Valid Values	Default Value
template	A case-sensitive label which UM matches to a template's assigned name.	string	(If omitted, no template is applied.)
rule	Used to restrict the usage of topics. See Order and Rule Specifications. Note that a particular object might match more than one `<topic>` element due to overlapping pattern matching. See Overlapping Topics.	**`"allow"`** - Permit the matching topic. **`"deny"`** - Prevent the matching topic.	`allow`
pattern	Regular expression to match against the topic name of the application object being created.	string	(no default; either `topicname` or `pattern` must be specified)
topicname	Name of topic to match against the application object being created. Requires exact match.	string	(no default; either `topicname` or `pattern` must be specified)

Example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<um-configuration version="1.0">
  <templates>
    <template name="FIX_Config.Prod">
      ...
    </template>
    ...
  </templates>
  <applications>
    <application name="FIX.01" template="FIX_Config.Prod">
      <hfxs order="deny,allow">
        <topic template="FIX_Config.Prod" topicname="Orders" rule="deny"\>
        ...
      </hfxs>
      ...
    </application>
    ...
  </applications>
    ...
  </hfxs>
  ...
</um-configuration>
```

2.9.7 UM Element "<options>"

Container element that holds a set of UM options of a specific scope (context, source, etc.).

- **Cardinality:** 0 .. unbounded
- **Parent:** <template>, <event-queue>, <context>, <topic>, <wildcard-receiver>

- **Children:** <option>, <application-data>

XML Attributes:

Attribute	Description	Valid Values	Default Value
type	UM configuration scope of the <option> elements contained within this <options> element.	**"event-queue"** - Event queue scope options. **"context"** - Context scope options. **"source"** - Source scope options. **"receiver"** - Receiver scope options. **"wildcard-receiver"** - Wildcard receiver scope options. **"hfx"** - HFX scope options.	(no default; must be specified)

Example:

The <options> element can be contained within many other elements. This example only shows it used within the <template> element, but its syntax and usage is the same when used elsewhere.

```
<?xml version="1.0" encoding="UTF-8" ?>
<um-configuration version="1.0">
  <templates>
    <template name="FIX_Config.Prod">
      <options type="context">
        ...
      </options>
      ...
    </template>
    ...
  </templates>
  ...
</um-configuration>
```

2.9.8 UM Element "<option>"

Configure a specific configuration option. The contained <allow> and <deny> elements are used to allow the XML file to constrain how the application may override the option's value. See Order and Rule Specifications for details.

- **Parent:** <options>
- **Children:** <allow>, <deny>

XML Attributes:

Attribute	Description	Valid Values	Default Value
name	Option name.	string	(no default; must be specified)
default-value	Value to set the option.	string	(if omitted, the option's default value is not changed.)
order	Valid values are "deny,allow" and "allow,deny". Used to control how option values are restricted. See Order and Rule Specifications.	string	"deny,allow"

Example:

The <options> element can be contained within many other elements. This example only shows it used within the <template> element, but its syntax and usage is the same when used elsewhere.

```
<?xml version="1.0" encoding="UTF-8" ?>
<um-configuration version="1.0">
  <templates>
    <template name="FIX_Config.Prod">
      <options type="context">
        <option name="default_interface" default-value="10.1.2.3" order="deny,allow">
          ...
        </option>
        ...
      </options>
      ...
    </template>
    ...
  </templates>
  ...
</um-configuration>
```

2.9.9 UM Element "<deny>"

Contains an option value that the application is explicitly prevented from using. See Order and Rule Specifications.

- **Parent:** <option>

XML Attributes:

Attribute	Description	Valid Values	Default Value
xml:space	Specifies how whitespace (tabs, spaces, linefeeds) are handled in the element content. See **xml:space Attribute**.	**"default"** - Trim whitespace. **"preserve"** - Retain whitespace exactly as entered.	**default**

Example:

The <options> element can be contained within many other elements. This example only shows it used within the <template> element, but its syntax and usage is the same when used elsewhere.

In this example, the application may configure any interface except loopback (127.0.0.1).

```
<?xml version="1.0" encoding="UTF-8" ?>
<um-configuration version="1.0">
  <templates>
    <template name="FIX_Config.Prod">
      <options type="context">
        <option name="default_interface">
          <deny>127.0.0.1</deny>
          ...
        </option>
        ...
      </options>
      ...
    </template>
    ...
  </templates>
  ...
</um-configuration>
```

2.9.10 UM Element "<allow>"

Contains an option value that the application is explicitly allowed to use. See Order and Rule Specifications.

- **Parent:** <option>

XML Attributes:

Attribute	Description	Valid Values	Default Value
xml:space	Specifies how whitespace (tabs, spaces, linefeeds) are handled in the element content. See **xml:space Attribute**.	**"default"** - Trim whitespace. **"preserve"** - Retain whitespace exactly as entered.	**default**

Example:

The <options> element can be contained within many other elements. This example only shows it used within the <template> element, but its syntax and usage is the same when used elsewhere.

This example also demonstrates a specific case where the <option> element has order="allow,deny" which sets the default behavior for overriding the option to deny. This effectively constrains the application's ability to override the default value to only those values explicitly allowed. Importantly, the value specified in default-value="..." must be explicitly allowed, as shown in this example.

```
<?xml version="1.0" encoding="UTF-8" ?>
<um-configuration version="1.0">
  <templates>
    <template name="FIX_Config.Prod">
      <options type="context">
        <option name="default_interface" default-value="10.1.2.3" order="allow,deny">
          <allow>10.1.2.3</allow>
          ...
        </option>
        ...
      </options>
      ...
    </template>
    ...
  </templates>
  ...
</um-configuration>
```

2.9.11 UM Element "<event-queues>"

Container element that holds the configuration for event queues. The <event-queue> elements contained within <event-queues> are matched to the event queue objects created by the application. The order="..." attribute is used to constrain the application's use of event queues. See Order and Rule Specifications for details.

- **Parent:** <application>
- **Children:** <event-queue>

XML Attributes:

Attribute	Description	Valid Values	Default Value
template	A case-sensitive label which UM matches to a template's assigned name.	string	(If omitted, no template is applied.)
order	Valid values are "deny,allow" and "allow,deny". Used to control how event queue usage is restricted. See Order and Rule Specifications.	string	**"deny,allow"**

Example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<um-configuration version="1.0">
  <templates>
    <template name="EVQ_FIX_Config.Prod">
      ...
    </template>
    ...
  </templates>
  <applications>
    <application name="FIX.01">
      <event-queues template="EVQ_FIX_Config.Prod" order="deny,allow">
        ...
      </event-queues>
      ...
    </application>
    ...
  </applications>
  ...
</um-configuration>
```

2.9.12 UM Element "<event-queue>"

Container of configuration for a single event queue.

- **Cardinality:** 0 .. unbounded
- **Parent:** <event-queues>
- **Children:** <options>

XML Attributes:

Attribute	Description	Valid Values	Default Value
name	Name of the event queue. Supplied as a parameter to **lbm_event_queue_attr_create_from_xml()** and **lbm_event_queue_attr_set_from_xml()**. Names are case-sensitive and can consist of only alpha-numeric ASCII characters, dash (-), and underscore (_). They must be 127 characters or less.	string	(If omitted, matches applications that don't set an event queue name.)
template	A case-sensitive label which UM matches to a template's assigned name.	string	(If omitted, matches applications that don't set a name.)
rule	Used to restrict the usage of event queues. See Order and Rule Specifications.	**"allow"** - Permit the matching object. **"deny"** - Prevent the matching object.	allow

Example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<um-configuration version="1.0">
  <templates>
    <template name="EVQ_FIX_Config.Prod">
      ...
```

```
    </template>
    ...
  </templates>
  <applications>
    <application name="FIX.01">
      <event-queues template="EVQ_FIX_Config.Prod" order="deny,allow">
        <event-queue name="EVQ_FIX" rule="allow">
          ...
        </event-queue>
        ...
      </event-queues>
      ...
    </application>
    ...
  </applications>
  ...
</um-configuration>
```

2.9.13 UM Element "<contexts>"

Container element that holds the configurations for context objects. The <context> elements contained within <contexts> are matched to the context objects created by the application. For contexts that do not match any of the contained <context> elements, the default permission is determined by the order="..." attribute.

- **Parent:** <application>
- **Children:** <context>

XML Attributes:

Attribute	Description	Valid Values	Default Value
template	A case-sensitive label which UM matches to a template's assigned name.	string	(If omitted, no template is applied.)
order	Valid values are "deny,allow" and "allow,deny". Used to control how context usage is restricted. See Order and Rule Specifications.	string	**"deny,allow"**

Example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<um-configuration version="1.0">
  <templates>
    <template name="CTX_FIX_Config.Prod">
      ...
    </template>
    ...
  </templates>
  <applications>
    <application name="FIX.01">
      <contexts template="CTX_FIX_Config.Prod" order="deny,allow">
        ...
      </contexts>
      ...
    </application>
    ...
  </applications>
  ...
</um-configuration>
```

2.9.14 UM Element "<context>"

Container of configuration for a single context.

- **Cardinality:** 0 .. unbounded
- **Parent:** <contexts>
- **Children:** <sources>, <receivers>, <wildcard-receivers>, <options>

XML Attributes:

Attribute	Description	Valid Values	Default Value
name	Name of the context. Supplied as a parameter to **lbm_context_attr_create_from_xml()** and **lbm_context_attr_set_from_xml()**. Names are case-sensitive and can consist of only alpha-numeric ASCII characters, dash (-), and underscore (_). They must be 127 characters or less.	string	(If omitted, matches applications that don't set a context name.)
template	A case-sensitive label which UM matches to a template's assigned name.	string	(If omitted, no template is applied.)
rule	Used to restrict the usage of contexts. See Order and Rule Specifications.	**"allow"** - Permit the matching object. **"deny"** - Prevent the matching object.	allow

Example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<um-configuration version="1.0">
  <templates>
    <template name="CTX_FIX_Config.Prod">
      ...
    </template>
    ...
  </templates>
  <applications>
    <application name="FIX.01">
      <contexts template="CTX_FIX_Config.Prod" order="deny,allow">
        <context name="CTX_FIX" rule="allow">
          ...
        </context>
        ...
      </contexts>
      ...
    </application>
    ...
  </applications>
  ...
</um-configuration>
```

2.9.15 UM Element "<wildcard-receivers>"

Container element that holds the configurations for wildcard receiver objects. The <wildcard-receiver> elements contained within <wildcard-receivers> are matched to the wildcard receiver objects created by the application.

Note

If the user desires to constrain the use of wildcard receivers, it should be done with `order="allow,deny"` and `rule="allow"` attributes (which denies all wildcards except those specifically allowed). The use of `order="deny,allow"` and `rule="deny"` to allow any wildcard except those specifically denied will not work as desired. For example, denying the pattern ".*" will still permit the use of "^.*", which will match the same topics (i.e. all of them).

- **Parent:** <context>
- **Children:** <wildcard-receiver>

XML Attributes:

Attribute	Description	Valid Values	Default Value
template	A case-sensitive label which UM matches to a template's assigned name.	string	(If omitted, no template is applied.)
order	Valid values are "deny,allow" and "allow,deny". Used to control how wildcard receiver usage is restricted. See Order and Rule Specifications.	string	**"deny,allow"** (But note that this value is not useful for restrictingwildcard receivers. "allow,deny" should always be used.)

Example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<um-configuration version="1.0">
  <templates>
    <template name="CTX_FIX_Config.Prod">
      ...
    </template>
    ...
  </templates>
  <applications>
    <application name="FIX.01">
      <contexts template="CTX_FIX_Config.Prod" order="deny,allow">
        <context name="CTX_FIX" rule="allow">
          <wildcard-receivers template="WC_FIX_Config.Prod" order="deny,allow">
            ...
          </wildcard-receivers>
          ...
        </context>
        ...
      </contexts>
      ...
    </application>
    ...
  </applications>
  ...
</um-configuration>
```

2.9.16 UM Element "<wildcard-receiver>"

Container of configuration for a single wildcard receiver.

- **Cardinality:** 0 .. unbounded
- **Parent:** <wildcard-receivers>
- **Children:** <options>

XML Attributes:

Attribute	Description	Valid Values	Default Value
template	A case-sensitive label which UM matches to a template's assigned name.	string	(If omitted, no template is applied.)
rule	Used to restrict the usage of wildcard receivers. See Order and Rule Specifications.	**"allow"** - Permit the matching object. **"deny"** - Prevent the matching object. (Note that this is not a useful setting for restricting wildcard receivers since the application can choose a different pattern that matches the forbidden topic.)	allow
pattern	Match wildcard receivers with this pattern. Note that this string is matched exactly to the pattern supplied to the wildcard receiver. This pattern is not intended to match more than one wildcard receiver.	string	(no default; must be specified)
pattern-type	Type of wildcard receiver pattern matching engine the wildcard receiver is using. Only "pcre" is supported.	**"pcre"** - Perl regular expression. This is the only supported selection. **"regex"** - Posix regular expression. **Deprecated; do not use.** **"application-callback"** - Application-supplied pattern matcher. **Deprecated; do not use.**	**"pcre"**

Example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<um-configuration version="1.0">
  <templates>
    <template name="CTX_FIX_Config.Prod">
      ...
    </template>
    ...
  </templates>
  <applications>
    <application name="FIX.01">
      <contexts template="CTX_FIX_Config.Prod" order="deny,allow">
        <context name="CTX_FIX" rule="allow">
          <wildcard-receivers template="WC_FIX_Config.Prod" order="deny,allow">
            <wildcard-receiver pattern="WC_FIX" rule="allow">
              ...
            </wildcard-receiver>
            ...
          </wildcard-receivers>
          ...
        </context>
        ...
      </contexts>
      ...
    </application>
    ...
  </applications>
  ...
</um-configuration>
```

2.9.17 UM Element "<receivers>"

Container element that holds the configurations for receiver objects. The <topic> elements contained within <receivers> are matched to the receiver objects created by the application. For receivers that do not match any of the contained <topic> elements, the default permission is determined by the order="..." attribute.

- **Parent:** <context>
- **Children:** <topic>

XML Attributes:

Attribute	Description	Valid Values	Default Value
template	A case-sensitive label which UM matches to a template's assigned name.	string	(If omitted, no template is applied.)
order	Valid values are "deny,allow" and "allow,deny". Used to control how receiver usage is restricted. See Order and Rule Specifications.	string	**"deny,allow"**

Example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<um-configuration version="1.0">
  <templates>
    <template name="CTX_FIX_Config.Prod">
      ...
    </template>
    ...
  </templates>
  <applications>
    <application name="FIX.01">
      <contexts template="CTX_FIX_Config.Prod" order="deny,allow">
        <context name="CTX_FIX" rule="allow">
          <receivers template="RCV_FIX_Config.Prod" order="deny,allow">
            ...
          </receivers>
          ...
        </context>
        ...
      </contexts>
      ...
    </application>
    ...
  </applications>
  ...
</um-configuration>
```

2.9.18 UM Element "<sources>"

Container element that holds the configurations for source objects. The <topic> elements contained within <sources> are matched to the source objects created by the application. For sources that do not match any of the contained <topic> elements, the default permission is determined by the order="..." attribute.

- **Parent:** <context>
- **Children:** <topic>

XML Attributes:

Attribute	Description	Valid Values	Default Value
template	A case-sensitive label which UM matches to a template's assigned name.	string	(If omitted, no template is applied.)
order	Valid values are "deny,allow" and "allow,deny". Used to control how source usage is restricted. See Order and Rule Specifications.	string	**"deny,allow"**

Example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<um-configuration version="1.0">
  <templates>
    <template name="CTX_FIX_Config.Prod">
      ...
    </template>
    ...
  </templates>
  <applications>
    <application name="FIX.01">
      <contexts template="CTX_FIX_Config.Prod" order="deny,allow">
        <context name="CTX_FIX" rule="allow">
          <sources template="SRC_FIX_Config.Prod" order="deny,allow">
            ...
          </sources>
          ...
        </context>
        ...
      </contexts>
      ...
    </application>
    ...
  </applications>
  ...
</um-configuration>
```

2.9.19 UM Element "<templates>"

Container element that holds one or more configuration template definitions. A configuration template holds a set of UM configuration options. See XML Configuration File Format for information on templates.

- **Parent:** <um-configuration>
- **Children:** <template>

Example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<um-configuration version="1.0">
  <templates>
    ...
  </templates>
  ...
</um-configuration>
```

2.9.20 UM Element "<template>"

Container element that holds a collection of UM configuration options which can be referenced by other elements. See XML Configuration File Format for information on templates.

- **Cardinality:** 0 .. unbounded
- **Parent:** <templates>
- **Children:** <options>

XML Attributes:

Attribute	Description	Valid Values	Default Value
name	A case-sensitive label assigned to the template, which can be referenced by most other elements via their "template" attribute. Names are case-sensitive and can consist of any printable ASCII characters. They must be 99 characters or less.	string	(no default; must be specified)

Example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<um-configuration version="1.0">
  <templates>
    <template name="FIX_Config.Prod">
      ...
    </template>
    ...
  </templates>
  ...
</um-configuration>
```

2.9.21 UM Element "<license>"

Identifies the UM product license, either as the license key or as a pointer to a license file, as an alternative to setting it in an environment variable. The content within the `<license>...</license>` is either a file name or a license string, depending on the value supplied for the `format` attribute.

- **Parent:** <um-configuration>

XML Attributes:

Attribute	Description	Valid Values	Default Value
format	Specifies how the content within the `<license>...</license>` is interpreted.	**`"filename"`** - The license element contains the name of a file that contains the license key. **`"string"`** - The license element contains the actual license key.	**string**
xml:space	Specifies how whitespace (tabs, spaces, linefeeds) are handled in the element content. See **xml:space Attribute**.	**`"default"`** - Trim whitespace. **`"preserve"`** - Retain whitespace exactly as entered.	**default**

Example 1:

```
<?xml version="1.0" encoding="UTF-8" ?>
<um-configuration version="1.0">
  <license format="filename">um_license.txt</license>
  ...
</um-configuration>
```

Example 2:

```
<?xml version="1.0" encoding="UTF-8" ?>
<um-configuration version="1.0">
  <license format="string">
Product=LBM,UME,UMQ,UMDRO:Organization=User or Org:Expiration-Date=never:License-Key=1234 5678 9ABC DEF0
  </license>
  ...
</um-configuration>
```

2.10 XML Configuration File DTD

The XML configuration file DTD is integrated into UM and appears below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT um-configuration (license | templates | applications)*>
<!ATTLIST um-configuration version CDATA #REQUIRED>
<!ELEMENT license ( #PCDATA )>
<!ATTLIST license format (filename | string) "string">
<!ATTLIST license xml:space (default | preserve) "default">
<!ELEMENT templates (template*)>
<!ELEMENT template (options+)>
<!ATTLIST template name CDATA #REQUIRED>
<!ELEMENT options (option | application-data)*>
<!ATTLIST options type (event-queue | context | source | receiver | wildcard-receiver | hfx) #IMPLIED>
<!ELEMENT option (allow | deny)*>
<!ATTLIST option name CDATA #REQUIRED>
<!ATTLIST option default-value CDATA #IMPLIED>
<!ATTLIST option order CDATA #IMPLIED>
<!ELEMENT application-data ( #PCDATA )>
<!ATTLIST application-data xml:space (default | preserve) "default">
<!ELEMENT allow ( #PCDATA )>
<!ATTLIST allow xml:space (default | preserve) "default">
<!ELEMENT deny ( #PCDATA )>
<!ATTLIST deny xml:space (default | preserve) "default">
<!ELEMENT applications (application*)>
<!ELEMENT application (contexts | event-queues | hfxs | application-data)+>
<!ATTLIST application name CDATA #IMPLIED>
<!ATTLIST application template CDATA #IMPLIED>
<!ELEMENT contexts (context*)>
<!ATTLIST contexts template CDATA #IMPLIED>
<!ATTLIST contexts order CDATA #IMPLIED>
<!ELEMENT event-queues (event-queue*)>
<!ATTLIST event-queues template CDATA #IMPLIED>
<!ATTLIST event-queues order CDATA #IMPLIED>
<!ELEMENT hfxs (topic*)>
<!ATTLIST hfxs template CDATA #IMPLIED>
<!ATTLIST hfxs order CDATA #IMPLIED>
<!ELEMENT event-queue (options*)>
<!ATTLIST event-queue name CDATA #IMPLIED>
<!ATTLIST event-queue template CDATA #IMPLIED>
<!ATTLIST event-queue rule (allow | deny) "allow">
<!ELEMENT context (sources | receivers | wildcard-receivers | options)+>
<!ATTLIST context name CDATA #IMPLIED>
<!ATTLIST context template CDATA #IMPLIED>
<!ATTLIST context rule (allow | deny) "allow">
<!ELEMENT sources (topic*)>
<!ATTLIST sources template CDATA #IMPLIED>
<!ATTLIST sources order CDATA #IMPLIED>
<!ELEMENT receivers (topic*)>
<!ATTLIST receivers template CDATA #IMPLIED>
<!ATTLIST receivers order CDATA #IMPLIED>
<!ELEMENT wildcard-receivers (wildcard-receiver*)>
<!ATTLIST wildcard-receivers template CDATA #IMPLIED>
<!ATTLIST wildcard-receivers order CDATA #IMPLIED>
<!ELEMENT topic (options*)>
<!ATTLIST topic template CDATA #IMPLIED>
<!ATTLIST topic rule (allow | deny) "allow">
<!ATTLIST topic pattern CDATA #IMPLIED>
<!ATTLIST topic topicname CDATA #IMPLIED>
<!ELEMENT wildcard-receiver (options*)>
<!ATTLIST wildcard-receiver template CDATA #IMPLIED>
<!ATTLIST wildcard-receiver rule (allow | deny) "allow">
<!ATTLIST wildcard-receiver pattern CDATA #IMPLIED>
<!ATTLIST wildcard-receiver pattern-type (pcre | regex | application-callback) #IMPLIED>
```

2.11 Sample XML Configuration File

A sample XML configuration file appears below and has the following notable aspects.

- Contains object attributes for a UM context and source.
- Application name is **Sending**.
- Uses a template of attributes also called **Sending-LBTRM**.
- The template, **Sending-LBTRM**, uses the **order** attribute for the **fd_management_type** to allow all file descriptor types except DEVPOLL. However the **Sending-LBTRM** application further restricts the file descriptor types to exclude EPOLL in addition to DEVPOLL.

```
<?xml version="1.0" encoding="UTF-8" ?>
<um-configuration version="1.0">
 <templates>
  <template name="Sending-LBTRM">
   <options type="source">
    <option default-value="0" name="late_join"/>
    <option default-value="500" name="resolver_advertisement_maximum_initial_interval"/>
    <option default-value="5000" name="resolver_advertisement_minimum_initial_duration"/>
    <option default-value="10" name="resolver_advertisement_minimum_initial_interval"/>
    <option default-value="60" name="resolver_advertisement_minimum_sustain_duration"/>
    <option default-value="1000" name="resolver_advertisement_sustain_interval"/>
    <option default-value="lbtrm" name="transport"/>
    <option default-value="14400" name="transport_lbtrm_destination_port"/>
    <option default-value="0.0.0.0" name="transport_lbtrm_multicast_address"/>
   </options>
   <options type="context">
    <option default-value="wsaeventselect" name="fd_management_type" order="deny,allow">
     <deny>wincompport</deny>
    </option>
    <option default-value="5000" name="mim_delivery_control_activity_check_interval"/>
    <option default-value="60000" name="mim_delivery_control_activity_timeout"/>
    <option default-value="2000000" name="resolver_initial_advertisement_bps"/>
    <option default-value="2000" name="resolver_initial_advertisements_per_second"/>
    <option default-value="2000" name="resolver_initial_queries_per_second"/>
    <option default-value="2000000" name="resolver_initial_query_bps"/>
   </options>
  </template>
 </templates>
 <applications>
  <application name="Sending">
   <contexts order="deny,allow">
    <context rule="allow" template="Sending-LBTRM">
     <sources order="deny,allow">
      <topic rule="allow" topicname="IXCM">
       <options type="source">
        <option default-value="1" name="late_join"/>
        <option default-value="lbtrm" name="transport"/>
        <option default-value="14488" name="transport_lbtrm_destination_port"/>
        <option default-value="224.12.5.101" name="transport_lbtrm_multicast_address"/>
       </options>
      </topic>
     </sources>
     <receivers order="deny,allow"/>
     <wildcard-receivers order="deny,allow"/>
     <options type="context">
      <option default-value="224.9.10.11" name="resolver_multicast_address"/>
      <option default-value="224.9.10.11" name="resolver_multicast_incoming_address"/>
      <option default-value="12965" name="resolver_multicast_incoming_port"/>
      <option default-value="224.9.10.11" name="resolver_multicast_outgoing_address"/>
      <option default-value="12965" name="resolver_multicast_outgoing_port"/>
      <option default-value="12965" name="resolver_multicast_port"/>
      <option default-value="224.9.10.12" name="resolver_multicast_interface"/>
      <option default-value="0" name="resolver_multicast_receiver_socket_buffer"/>
      <option default-value="wsaeventselect" name="fd_management_type" order="deny,allow">
       <deny>wincompport</deny>
      </option>
     </options>
    </context>
   </contexts>
   <event-queues order="deny,allow">
    <event-queue rule="allow">
```

```
      <options type="event-queue">
        <option default-value="lbm" name="monitor_transport"/>
        <option default-value="" name="monitor_appid"/>
      </options>
    </event-queue>
   </event-queues>
  </application>
 </applications>
</um-configuration>
```

Chapter 3

Attributes Objects

Many UM primitive objects have a corresponding attributes object, which contains the configuration information specific to that UM object type. You can set configuration options in an attributes object, and supply the attributes when creating the UM object. This allows assignment of different options for different instances of UM objects. The following table lists the UM primitive objects and corresponding attributes objects.

UM object	Corresponding Attributes Object(s)
lbm_context_t	**lbm_context_attr_t**
lbm_topic_t	**lbm_src_topic_attr_t**, **lbm_rcv_topic_attr_t**
lbm_wildcard_rcv_t	**lbm_wildcard_rcv_attr_t**
lbm_event_queue_t	**lbm_event_queue_attr_t**
lbm_hfx_t	**lbm_hfx_attr_t**

You call API functions to create attributes objects and set, retrieve, or delete their values. These API names are based on the attributes object name and are shown in the following table, using the context object as an example. See the C API for all attributes APIs.

Action	UM API function
Create Attributes Object	**lbm_context_attr_create_from_xml()**
Set Option from Binary Value	**lbm_context_attr_setopt()**
Set Option from String Value	**lbm_context_attr_str_setopt()**
Get Option as Binary Value	**lbm_context_attr_getopt()**
Get Option as String Value	**lbm_context_attr_str_getopt()**
Delete Attributes Object	**lbm_context_attr_delete()**

For other object types, replace **context** with **src_topic**, **rcv_topic**, **wildcard_rcv**, **event_queue**, or **hfx**.

The following sections describe in detail the use of these UM API functions. The APIs related to **lbm_context_attr_t** objects are used for the purpose of illustration, but the instructions (if not the specifics) apply to all UM attributes objects.

3.1 Creating An Attributes Object

In the following example, the call to **lbm_context_attr_create_from_xml()** creates the custom attributes object, and initializes each option from the current default values. Subsequent calls to **lbm_context_attr_setopt()** or **lbm_context_attr_str_setopt()** modify only the option values in the attributes object.

```
lbm_context_attr_t * attrib;
int rc;

rc = lbm_context_attr_create_from_xml(&attrib, "MyCtx");
if (rc != 0)
{
  /* Immediately after UM returns error, capture error details. */
  int errnum = lbm_errnum();
  const char * errmsg = lbm_errmsg();
  fprintf(stderr, "Error %d returned from lbm_context_attr_create_from_xml(), %s\n",
          errnum, errmsg);
}
```

This example also illustrates the proper way to determine the success or failure of an UM API call. Most UM API calls return 0 to indicate success, and -1 to indicate failure. To retrieve the specific UM error code for the failure, call **lbm_errnum()**. To retrieve a text string describing the error code, call **lbm_errmsg()**.

3.2 Setting an Option from a Binary Value

For an option of type other than "string", call **lbm_context_attr_setopt()** to set its value. (See the C API reference for details on this API.) The final two parameters in the API are a pointer to a variable containing the option value, and a variable of type size_t that contains the correct length of the option value variable.

The example code below sets three options. First, we set operational_mode (context) to **sequential**. Then we set the transport_tcp_port_low (context) and transport_tcp_port_high (context) values to **4901** and **4920**, respectively.

```
lbm_context_attr_t * attrib; /* Must have already been created */
int rc;
unsigned short int optval;
size_t optlen;

/* Set the operational_mode */
optlen = sizeof(optval);
optval = LBM_CTX_ATTR_OP_SEQUENTIAL;
rc = lbm_context_attr_setopt(attrib, "operational_mode", &optval, optlen);
if (rc != 0) {
  /* Handle error */
}

/* Set transport_tcp_port_low */
optlen = sizeof(optval);
optval = 4901;
rc = lbm_context_attr_setopt(attrib, "transport_tcp_port_low", &optval, optlen);
if (rc != 0) {
  /* Handle error */
}

/* Set transport_tcp_port_high */
optlen = sizeof(optval);
optval = 4920;
rc = lbm_context_attr_setopt(attrib, "transport_tcp_port_high", &optval, optlen);
if (rc != 0) {
  /* Handle error */
}
```

3.2.1 Setting an Option from Arrays of Binary Values

There are some configuration options which expect an array of a particular type. The *_setopt() API uses its "optlen" parameter to determine the number of valid elements in the array.

For example, when using umq_ulb_application_set (source) to configure a ULB source's application sets, the **lbm_umq_ulb_receiver_type_entry_t** structure is used to define one mapping between receiver type ID and application set index. It is common to have more than one receiver type and/or more than one application set, so the application code must pass an array of lbm_umq_ulb_receiver_type_entry_t structures. Note how **lbm_src_topic_attr_setopt()**'s "optlen" is calculated in the following code:

```
lbm_umq_ulb_receiver_type_entry_t appsets[32];  /* This application's worst case need. */
int optlen, num_valid_elements;
...
/* We need three entries, the equiv of "source umq_ulb_application_set 0:10,20;1:100". */
appsets[0].application_set_index = 0;
appsets[0].id = 10;  /* Receiver type ID. */
appsets[1].application_set_index = 0;
appsets[1].id = 20;  /* Receiver type ID. */
appsets[2].application_set_index = 1;
appsets[2].id = 100;  /* Receiver type ID. */
num_valid_elements = 3;

optlen = num_valid_elements * sizeof(lbm_umq_ulb_receiver_type_entry_t);
rc = lbm_src_topic_attr_setopt(tattr, "umq_ulb_application_set", appsets, optlen);
if (rc != 0) {
  /* Handle error */
}
```

3.3 Setting an Option from a String Value

Setting an option from a string value effectively does the same thing that setting an option from a binary value does. However, the option value is passed as a null-terminated string, rather than as value and length pointers. UM uses this mechanism to process options in a configuration file. Thus, the format used for option values must match the format you would use in a configuration file.

In the following example, as before, we set the operational mode to sequential. Then we set the transport TCP port low and high values to **4901** and **4920**, respectively.

```
lbm_context_attr_t * attrib; /* Must have already been created */
int rc;

/* Set the operational_mode */
rc = lbm_context_attr_str_setopt(attrib, "operational_mode", "sequential");
if (rc != 0) {
  /* Handle error */
}

/* Set transport_tcp_port_low */
rc = lbm_context_attr_str_setopt(attrib, "transport_tcp_port_low", "4901");
if (rc != 0) {
  /* Handle error */
}

/* Set transport_tcp_port_high */
rc = lbm_context_attr_str_setopt(attrib, "transport_tcp_port_high", "4920");
if (rc != 0) {
  /* Handle error */
}
```

3.4 Getting an Option as a Binary Value

Getting an option as a binary value is very similar to setting an option from a binary value: it requires knowledge of not only the option name, but its type as well. The final two parameters in the call to **lbm_context_attr_getopt()** are a pointer to a variable to receive the current option value, and a pointer to a variable of type **size_t** which contains the length of the option value variable. This length must be correct for the specified option.

In the example code below, we get the option values for operational mode and the transport TCP port low and high values.

```
lbm_context_attr_t * attrib; /* Must have already been created */
int rc;
unsigned short int optval;
size_t optlen;

/* Get the operational_mode */
optlen = sizeof(optval);
rc = lbm_context_attr_getopt(attrib, "operational_mode", &optval, &optlen);
if (rc != 0) {
  /* Handle error */
}
/* optval now contains LBM_CTX_ATTR_OP_EMBEDDED or LBM_CTX_ATTR_OP_SEQUENTIAL */

/* Get transport_tcp_port_low */ optlen = sizeof(optval);
rc = lbm_context_attr_getopt(attrib, "transport_tcp_port_low", &optval, &optlen);
if (rc != 0) {
  /* Handle error */
}
/* optval now contains the value of transport_tcp_port_low, which should be 4901 */

/* Get transport_tcp_port_high */ optlen = sizeof(optval);
rc = lbm_context_attr_getopt(attrib, "transport_tcp_port_high", &optval, &optlen);
if (rc != 0) {
  /* Handle error */
}
/* optval now contains the value of transport_tcp_port_high, which should be 4920 */
```

3.5 Getting an Option as a String Value

Getting an option as a string value effectively does the same thing that getting an option as a binary value does. However, the option value is returned as a null-terminated string, just as you would specify the option value in a configuration file. The final two parameters in the call to **lbm_context_attr_str_getopt()** are a pointer to a string variable to receive the current option value, and a pointer to a variable of type **size_t** which contains the maximum size of the option value string variable.

In the example code below, we get the option values for operational mode and the transport TCP port low and high values.

```
lbm_context_attr_t * attrib; /* Must have already been created */
int rc;
char optval_string[256];

/* Get the operational_mode */
optlen = sizeof(optval_string);
rc = lbm_context_attr_str_getopt(attrib, "operational_mode", optval_string, &optlen);
if (rc != 0) {
  /* Handle error */
}
/* optval_string now contains either "embedded" or "sequential" */

/* Get transport_tcp_port_low */
optlen = sizeof(optval_string);
rc = lbm_context_attr_str_getopt(attrib, "transport_tcp_port_low",
                                 optval_string, &optlen);
if (rc != 0) {
  /* Handle error */
}
/* optval_string now contains the string value of transport_tcp_port_low,
   which should be "4901" */
```

```
/* Get transport_tcp_port_high */ optlen = sizeof(optval_string);
rc = lbm_context_attr_str_getopt(attrib, "transport_tcp_port_high",
                                 optval_string, &optlen);
if (rc != 0) {
  /* Handle error */
}
/* optval_string now contains the string value of transport_tcp_port_high,
   which should be "4920" */
```

3.6 Deleting an Attributes Object

Once the attributes object is no longer needed, it should be deleted.

```
lbm_context_attr_t * attrib; /* Must have already been created */
int rc;

rc = lbm_context_attr_delete(attrib);
if (rc != 0) {
  /* Handle error */
}
```

Chapter 4

Access to Current Operating Options

After a UM object is created, the current operating option values can be retrieved, and a small subset of its current operating options can be modified. UM API functions supporting such actions operate on the object itself, rather than on an attributes object.

4.1 Retrieving Current Option Values

Almost all UM objects allow their current attributes' option values to be retrieved during operation. UM API functions supporting such actions operate on the object itself.

The UM objects which support these actions are **lbm_src_t**, **lbm_rcv_t**, **lbm_context_t**, and **lbm_event_queue_t**. For each such object, there are corresponding API functions to get an option as a binary value, and get an option as a string value. These API names are based on the object name, suffixed with **_getopt()**, and **_str_getopt()**. As an illustration of this convention, the API functions for working with **lbm_event_queue_t** objects are shown in the following table.

Action	UM API function
Get Option from Binary Value	**lbm_event_queue_getopt()**
Get Option from String Value	**lbm_event_queue_str_getopt()**

For other object types, replace **event_queue** with **context**, **src_topic**, **rcv_topic**, **wildcard_rcv**, or **hfx**.

4.1.1 Getting Current Option as a Binary Value

Getting an option as a binary value is very similar to setting an option from a binary value: it requires knowledge of not only the option name, but its type as well. The final two parameters in the call to **lbm_event_queue_getopt()** are a pointer to a variable to receive the current option value, and a pointer to a variable of type **size_t** which contains the length of the option value variable. This length must be correct for the specified option.

In the example code below, the option value for the queue size warning is retrieved.

```
unsigned long int optval;
size_t optlen;
```

```
lbm_event_queue_t evq; /* must be previously created */
int rc;

/* Get the queue size warning value */
optlen = sizeof(optval);
rc = lbm_event_queue_getopt(&evq, "queue_size_warning", &optval, &optlen);
if (rc != 0) {
  /* Handle error */
}
/* optval now contains the value of queue_size_warning, which should be 5000 */
```

4.1.2 Getting Current Option as a String Value

Getting an option as a string value effectively does the same thing that getting an option as a binary value does. However, the option value is returned as a null-terminated string, just as you would specify the option value in a configuration file. The final two parameters in the call to **lbm_event_queue_str_getopt()** are a pointer to a string variable to receive the current option value, and a pointer to a variable of type **size_t** which contains the maximum size of the option value string variable.

In the example code below, the option value for the queue size warning is retrieved.

```
char optval_string[256];
size_t optlen;
lbm_event_queue_t evq; /* must be previously created */
int rc;

/* Get the queue size warning value */
optlen = sizeof(optval_string);
rc = lbm_event_queue_str_getopt(&evq, "queue_size_warning", optval_string, &optlen);
if (rc != 0) {
  /* Handle error */
}
/* optval now contains the value of queue_size_warning, which should be "5000" */
```

4.2 Modifying Current Option Values

A small subset of UM object options may be modified after the object is created. See the individual option descriptions to determine if an options value may be changed after the UM object is created.

The UM objects which support these actions are **lbm_src_t**, **lbm_rcv_t**, **lbm_context_t**, and **lbm_event_queue_t**. For each such object, there are corresponding API functions to set an option from a binary value and set an option from a string value. These API names are based on the object name, suffixed with **_setopt()** and **_str_setopt()**.

As an illustration of this convention, the API functions for working with **lbm_event_queue_t** objects are shown in the following table.

Action	UM API function
Set Option from Binary Value	**lbm_event_queue_setopt()**
Set Option from String Value	**lbm_event_queue_str_setopt()**

For other object types, replace **event_queue** with **context**, **src_topic**, **rcv_topic**, **wildcard_rcv**, or **hfx**.

The following sections describe in detail the use of these UM API functions. The APIs related to **lbm_event_↩ queue_t** objects are used for the purpose of illustration, but the instructions (if not the specifics) apply to all such

UM objects.

4.2.1 Setting Current Option from a Binary Value

Setting an option from a binary value requires knowledge of not only the option name, but its type and allowable values as well. The final two parameters in the call to **lbm_event_queue_setopt()** are a pointer to a variable which contains the option value to be set, and a pointer to a variable of type size_t which contains the length of the option value variable. This length must be correct for the specified option.

In the example code below, we set the queue size warning to 5000 events.

```
unsigned long int optval;
size_t optlen;
lbm_event_queue_t evq; /* must be previously created */
int rc;

/* Set the queue size warning */
optlen = sizeof(optval);
optval = 5000;
rc = lbm_event_queue_setopt(&evq, "queue_size_warning", &optval, &optlen);
if (rc != 0) {
  /* Handle error */
}
```

4.2.2 Setting Current Option from a String Value

Setting an option from a string value effectively does the same thing that setting an option from a binary value does. However, the option value is passed as a null-terminated string, rather than as value and length pointers. This is similar to the mechanism used by UM to process options in a configuration file. Thus, the format used for option values must match the format you would use in a configuration file.

As before, we set the queue size warning to 5000 events.

```
lbm_event_queue_t evq; /* must be previously created */
int rc;

/* Set the queue size warning */
rc = lbm_event_queue_setopt(&evq, "queue_size_warning", "5000");
if (rc != 0) {
  /* Handle error */
}
```

Chapter 5

Example Configuration Scenarios

5.1 Highest Throughput

The following configuration option tunes UM for the highest possible throughput.

```
#
# LBM can be configured to make efficient use of CPU time, leading
# to the highest-possible throughput (bytes per second or messages
# per second). This may come at the expense of latency at low
# message rates. The following line configures LBM to accumulate
# 8KB of messages (or for wait implicit_batching_interval) before sending.
#
source implicit_batching_minimum_length 8192
```

5.2 Lowest Latency

This is an example configuration that favors low latency at the expense of higher CPU utilization and potentially lower throughput.

```
#
# Latency can be reduced at the expense of network efficiency and
# system CPU time by adjusting implicit batching parameters. The
# default parameters hold messages for up to 200 milliseconds or until
# 2048 bytes are waiting to go. The lowest possible latency is
# obtained by setting the minimum batching length to 1 byte, which
# effectively disables the implicit batching feature. For example:
#
context mim_implicit_batching_minimum_length 1
source implicit_batching_minimum_length 1
#
# Latency can be kept to a minimum with UM by writing receiving
# applications that can accept messages in the order they arrive.
# See https://communities.informatica.com/infakb/faq/5/Pages/80043.aspx
# for more information. Here's how to use arrival-order delivery:
#
receiver ordered_delivery 0
#
```

```
# Disable Nagel's algorithm (batching) for TCP responses to eliminate
# queuing latency when sending only single responses.
#
context response_tcp_nodelay 1
#
# If you are running a LAN environment with under 100 machines, you can
# drastically improve your recovery related latencies without significant
# additional network overhead by using the following UM loss recovery parameter.
# See https://communities.informatica.com/infakb/faq/5/Pages/80070.aspx
# for additional information about this and other recovery parameters.
#
receiver transport_lbtrm_nak_backoff_interval 10
```

5.3 Creating Multicast Sources

This is an example configuration file that changes the default transport to reliable multicast so all sources created send messages over LBT-RM.

```
#
# UM can be configured to create sources using the LBT-RM reliable
# multicast protocol instead of the default TCP.
#
source transport LBT-RM
#
# Stable and reliable operation with multicast requires careful
# setting of rate control limits.
#
# It's generally best to start with small limits and gradually
# increase them after testing indicates that they can be safely
# sustained on your network.
#
# The following example limits (new) data to 10 Mbps and retransmissions
# to 1 Mbps (10%).
#
context transport_lbtrm_data_rate_limit 10000000
context transport_lbtrm_retransmit_rate_limit 1000000
```

5.4 Disabling Aspects of Topic Resolution

If you need to reduce the amount of Topic Resolution traffic on your network, use the following Configuration options and values in a Ultra Messaging Configuration file.

Note

Ultra Messaging does not recommend disabling both advertisements and queries because topics may not resolve at all.

5.4.1 Disabling Topic Advertisements

You can disable topic advertisements in the Initial Phase, Sustaining Phase or both phases of topic resolution.

Disabling Initial Phase Advertisements

Use the following options to disable topic advertisements in only the Initial Phase.

```
source resolver_advertisement_minimum_initial_interval 0
source resolver_advertisement_maximum_initial_interval 0
```

Disabling Sustaining Phase Advertisements

Use the following option to disable topic advertisements in only the Sustaining Phase.

```
source resolver_advertisement_sustain_interval 0
```

5.4.2 Disabling Receiver Topic Queries

You can disable the querying of topics by receivers in the Initial Phase, Sustaining Phase or both phases of topic resolution.

Disabling Initial Phase Queries

Use the following options to disable topic queries in only the Initial Phase.

```
receiver resolver_query_minimum_initial_interval 0
receiver resolver_query_maximum_initial_interval 0
```

Disabling Sustaining Phase Queries

Use the following options to disable topic queries in only the Sustaining Phase.

```
receiver resolver_query_sustain_interval 0
receiver resolution_number_of_sources_query_threshold 0
```

5.4.3 Disabling Wildcard Topic Queries

Use the following options to disable topic queries by wildcard receivers.

```
wildcard_receiver resolver_query_minimum_interval 0
wildcard_receiver resolver_query_maximum_interval 0
```

5.4.4 Disabling Store (Context) Name Queries

When using Persistence, use the following options to disable context name queries by sources.

```
resolver_context_name_query_maximum_interval 0
resolver_context_name_query_minimum_interval 0
```

5.4.5 All But the Minimum Topic Resolution Traffic

A minimalist approach to topic resolution can take different forms based on you requirements. One approach is to disable all traffic except for queries in the sustaining phase. Add the following settings to your Ultra Messaging configuration file to implement this approach.

```
source resolver_advertisement_minimum_initial_interval 0
source resolver_advertisement_sustain_interval 0
receiver resolver_query_minimum_initial_interval 0
receiver resolution_number_of_sources_query_threshold 1
wildcard_receiver resolver_query_minimum_interval 0
```

5.5 Unicast Resolver

To use the unicast resolver, use a configuration file like the following example:

```
#
# Topic resolution can be configured to use unicast traffic with an
# LBM resolver daemon (lbmrd) instead of the default which uses multicast.
# Be sure to insert the IP address of your lbmrd below.
#
context resolver_unicast_daemon 127.0.0.1:15380
```

5.6 Re-establish Pre-4.0 Topic Resolution

Ultra Messaging topic resolution prior to LBM Version 4.0 did not have resolution phases. To implement pre-4.0 topic resolution, include the following configuration option changes in your Ultra Messaging configuration file.

```
# ----- Disable Advertisements in 4.0 Initial Phase
source resolver_advertisement_minimum_initial_interval 0

# ----- Re-establish pre-4.0 Advertisement Behavior
source resolver_advertisement_minimum_sustain_duration 0
context resolver_sustain_advertisement_bps 0

# ----- Disable Queries in 4.0 Initial Phase
receiver resolver_query_minimum_initial_interval 0

# ----- Re-establish pre-4.0 Query Behavior
receiver resolver_query_sustain_interval 100
receiver resolver_query_minimum_sustain_duration 0
context resolver_sustain_query_bps 0
receiver resolution_number_of_sources_query_threshold 1

# ----- Re-establish pre-4.0 Wildcard Query Behavior
wildcard_receiver resolver_query_minimum_interval 0
```

5.7 Re-establish Pre-LBM 3.3 (Pre-UME 2.0) Port Defaults

To use the early default ports (prior to LBM 3.3 and UME 2.0), the following configuration file may be used.

```
context mim_destination_port 4401
context mim_incoming_destination_port 4401
context mim_outgoing_destination_port 4401
context resolver_multicast_port 2965
context resolver_multicast_incoming_port 2965
context resolver_multicast_outgoing_port 2965
context resolver_unicast_destination_port 5380
context resolver_unicast_port_high 4406
context resolver_unicast_port_low  4402
source  transport_lbtrm_destination_port 4400
context transport_lbtrm_source_port_high 4399
context transport_lbtrm_source_port_low  4390
context transport_lbtru_port_high 4389
context transport_lbtru_port_high 4380
receiver transport_lbtru_port_high 4379
receiver transport_lbtru_port_low  4360
context request_tcp_port_high 4395
context request_tcp_port_low 4391
context transport_tcp_port_high 4390
context transport_tcp_port_low  4371
source  ume_primary_store_port 4567
source  ume_secondary_store_port 4567
source  ume_tertiary_store_port 4567
```

Note

Alternatively, UM will use the early port settings when the environment variable **LBM_USE_ORIG_DEFAU↩ LT_PORTS** is set to **1**.

5.8 Configure New Port Defaults

In the unusual case that you must run older versions of Ultra Messaging (less than LBM 3.3 / UME 2.0) on certain machine(s) and need these older version to work with the machines running the current versions of UMS and UMP, you can use the following configuration file for the older versions to synchronize port usage between old and current versions.

```
context mim_destination_port 14401
context mim_incoming_destination_port 14401
context mim_outgoing_destination_port 14401
context resolver_multicast_port 12965
context resolver_multicast_incoming_port 12965
context resolver_multicast_outgoing_port 12965
context resolver_unicast_destination_port 15380
context resolver_unicast_port_high 14406
context resolver_unicast_port_low  14402
source  transport_lbtrm_destination_port 14400
context transport_lbtrm_source_port_high 14399
context transport_lbtrm_source_port_low  14390
context transport_lbtru_port_high 14389
context transport_lbtru_port_low 14380
receiver transport_lbtru_port_high 14379
receiver transport_lbtru_port_low  14360
context request_tcp_port_high 14395
```

```
context request_tcp_port_low 14391
context transport_tcp_port_high 14390
context transport_tcp_port_low  14371
source  ume_primary_store_port 14567
source  ume_secondary_store_port 14567
source  ume_tertiary_store_port 14567
```

Chapter 6

Interrelated Configuration Options

Some Ultra Messaging configuration options are related in ways that might not be immediately apparent. Changing the value for one option without adjusting its related option can cause problems such as NAK storms, tail loss, etc. This section identifies these relationships and recommends a best practice for setting the interrelated options.

The following sections discuss configuration option relationships.

6.1 Preventing NAK Storms with NAK Intervals

The NAK generation interval should be sufficiently longer than the NAK backoff interval so that the source, after receiving the first NAK from a receiver, has time to retransmit the missing datagram and prevent a NAK storm from all receivers. LBTRM, LBTRU, and MIM all use NAK generation and backoff intervals. The NAK behavior for all transports is the same.

Interrelated Options:

- transport_lbtrm_nak_backoff_interval (receiver)
- transport_lbtrm_nak_generation_interval (receiver)
- transport_lbtru_nak_backoff_interval (receiver)
- transport_lbtru_nak_generation_interval (receiver)
- mim_nak_backoff_interval (context)
- mim_nak_generation_interval (context)

Recommendation:

Set the NAK generation interval to at least 2x the NAK backoff interval.

Example:

```
#
# To avoid NAK storms, set NAK generation interval to at least 2x the
# NAK backoff interval.
#
receiver transport_lbtrm_nak_backoff_interval 200       # .2 seconds
receiver transport_lbtrm_nak_generation_interval 10000  # 10 seconds
```

See also:

Transport LBT-RM Reliability Options
Transport LBT-RU Reliability Options
Multicast Immediate Messaging Reliability Options

6.2 Preventing Tail Loss With TSNI and NAK Interval Options

Tail Loss refers to the situation where a receiver (subscriber) does not receive the last few (tail) messages sent by a source (publisher). When unrecoverable loss occurs on a transport, due to the possibility of multiple topic-level messages being contained in a single transport-level sequence number (due to implicit batching), a receiver does not know which particular messages were unrecoverable until the arrival of later messages (revealing earlier gaps in topic-level sequence number) or until the arrival of Topic Sequence Number Information (TSNI) records sent periodically by a publisher. Specific topic-level knowledge of sequence gaps is a prerequisite for the receiver to deliver event callbacks to the application indicating that unrecoverable loss has occurred, because those event callbacks are per-receiver (topic-level). A TSNI active threshold that is too small relative to the TSNI and/or NAK generation interval may prevent the reporting of tail loss to the application, especially with ordered delivery.

Interrelated Options:

- transport_topic_sequence_number_info_active_threshold (source)
- transport_topic_sequence_number_info_interval (source)
- transport_lbtrm_nak_generation_interval (receiver)
- transport_lbtru_nak_generation_interval (receiver)

Recommendation:

Set the source's transport_topic_sequence_number_info_active_threshold (source) to at least 4x the transport_topic_sequence_number_info_interval (source) plus the receiver's transport_lbtru_nak_generation↩ _interval (receiver), all divided by 1000 to get seconds..

Example:

```
#
# NOTE: transport_topic_sequence_number_info_active_threshold is in seconds.
#
source   transport_topic_sequence_number_info_interval 5000
receiver transport_lbtrm_nak_generation_interval       10000
# (5000*4 + 10000)/1000 = 30
source   transport_topic_sequence_number_info_active_threshold 30
```

See also:

Preventing Undetected Unrecoverable Loss
Transport LBT-RM Reliability Options
Transport LBT-RU Reliability Options

6.3 Preventing Undetected Unrecoverable Loss

The UM UDP-based protocols are generally able to successfully recover packet loss. However, there can be cases where UM is not able to recover the lost packets, leading to **Unrecoverable Loss**.

With the default settings, there is a type of unrecoverable loss which can remain unreported to the application for an unbounded period of time.

For example:

1. A sudden burst of data from a source overloads a receiver, resulting in the last few packets being lost.
2. The source sends one more data message and then exits.
3. The receiver's **Delivery Controller** gets the last message and sees the sequence number gap. So it buffers the last message and waits for the transport layer to recover the missing messages. But since the source no longer exists, there is no recovery.
4. The NAK generation interval lapses. Thus, the gapped messages are considered unrecoverable. However, due to UM's design, a receive event is needed to deliver the unrecoverable loss event and the buffered message. But since the source is deleted, no more receive events will happen. The delivery controller is an a "stale loss" state.
5. Finally, the transport session times out and the delivery controller is deleted, delivering EOS to the application, but not the unrecoverable loss event or the buffered message.

In this scenario, not only is the unrecoverable loss not delivered, but the buffered message which was successfully received is also never delivered. Note that this kind of **Tail Loss** is rare, but can happen.

This result can be avoided by enabling the "loss check interval" feature on the delivery controller. For example:

```
receiver delivery_control_loss_check_interval 2500
```

This starts a timer that wakes up every 2.5 seconds and scans UM's internal list of all topic receivers, looking for delivery controllers in the "stale loss" state. For each one it finds, it generates the unrecoverable loss event to the application's receiver callback, and also delivers the subsequently buffered message.

However, for applications that have large numbers of receivers, the cost of scanning every receiver can become significant, introducing regular latency outliers. For latency-sensitive applications, an alternate method to avoid the unreported loss is to make sure transport_topic_sequence_number_info_interval (source) is non-zero, and have the publisher delays two of those intervals plus the NAK generation interval (default: 2∗5+60=70 seconds) before deleting a source that isn't needed any more. The TSNI messages will serve as receiver events to force delivery. See Preventing Tail Loss With TSNI and NAK Interval Options.

Be aware that the delivery_control_loss_check_interval (receiver) can interact with other interval configurations.

Interrelated Options:

- delivery_control_loss_check_interval (receiver)
- transport_lbtrm_activity_timeout (receiver)
- transport_lbtrm_nak_generation_interval (receiver)
- transport_lbtru_activity_timeout (receiver)

Recommendation, if using loss check interval:

For LBT-RM, set the transport activity timeout to value greater than the sum of the delivery control loss check interval and the NAK generation interval. Also, set the NAK generation interval to at least 4x the delivery control loss check interval.
For LBT-RU, set the transport activity timeout to value greater than the delivery control loss check interval
For UMP, always enable and set accordingly the delivery control loss check interval when configuring a store

Example:

```
#
# To avoid undetected or unreported loss, set NAK generation to 4x the delivery
# control check interval, and ensure that these two combined are less than the
# transport activity timeout
#
receiver delivery_control_loss_check_interval 2500
receiver transport_lbtrm_activity_timeout 60000
receiver transport_lbtrm_nak_generation_interval 10000
```

See also:

Delivery Control Options

6.4 Preventing Undetected Late Join Loss

If during a Late Join operation, a transport times out while a receiver is requesting retransmission of missing messages, this can cause lost messages to go undetected and likely become unrecoverable.

Interrelated Options:

- retransmit_request_generation_interval (receiver)
- transport_tcp_activity_timeout (receiver)
- transport_lbtrm_activity_timeout (receiver)
- transport_lbtru_activity_timeout (receiver)
- transport_lbtipc_activity_timeout (receiver)

Recommendations:

Set the Late Join retransmit request interval to a value less than its transport's activity timeout value

Example:

```
#
# To avoid a transport inactivity timeout while requesting Late Join
# retransmissions, set the Late Join retransmit request interval to a value
# less than its transport's activity timeout.
#
receiver retransmit_request_generation_interval 10000
receiver transport_lbtrm_activity_timeout 60000
```

See also:

Late Join Options

6.5 Preventing IPC Receiver Deafness With Keepalive Options

With an LBT-IPC transport, an activity timeout that is too small relative to the session message interval may cause receiver deafness. If a timeout is too short, the keepalive messages might not be received in time to prevent the receiver from being deleted or disconnecting because the source appears to be gone.

Interrelated Options:

- transport_lbtipc_activity_timeout (receiver)
- transport_lbtipc_sm_interval (source)

Recommendations:

Set the activity timeout to at least 2x the session message interval

Example:

```
#
# To avoid receiver deafness:
# - set client activity timeout to at least 2x the acknowledgement interval.
# - set activity timeout to at least 2x the session message interval.
#
receiver transport_lbtipc_activity_timeout 60000
source   transport_lbtipc_sm_interval      10000
```

See also:

Transport LBT-IPC Operation Options

6.6 Preventing Erroneous LBT-RM/LBT-RU Session Timeouts

An LBT-RM or LBT-RU receiver-side quiescent timeout may delete a transport session that a source is still active on. This can happen if the timeout is too short relative to the source's interval between session messages (which serve as a session keepalive).

Interrelated Options:

- transport_lbtrm_activity_timeout (receiver)
- transport_lbtrm_sm_maximum_interval (source)
- transport_lbtru_activity_timeout (receiver)
- transport_lbtru_sm_maximum_interval (source)

Recommendations:

Set the receiver LBT-RM or LBT-RU activity timeout to at least 3x the source session message maximum interval.

Example:

```
#
# To avoid erroneous session timeouts, set receiver transport activity
# timeout to at least 3x the source session message maximum interval.
#
receiver  transport_lbtrm_activity_timeout    60000
source    transport_lbtrm_sm_maximum_interval 10000
receiver  transport_lbtru_activity_timeout    60000
source    transport_lbtru_sm_maximum_interval 10000
```

See also:

Transport LBT-RM Operation Options
Transport LBT-RU Operation Options

6.7 Preventing Errors Due to Bad Multicast Address Ranges

Sometimes it is easy to accidentally reverse the low and high values for LBT-RM multicast addresses, which actually creates a very large range. Aside from excluding intended addresses, this can cause error conditions.

Interrelated Options:

- transport_lbtrm_multicast_address_low (context)
- transport_lbtrm_multicast_address_high (context)

Recommendations:

Ensure that the intended low and high values for LBT-RM multicast addresses are not reversed

Example:

```
#
# To avoid incorrect LBT-RM multicast address ranges, ensure that you have not
# reversed the low and high values.
#
context transport_lbtrm_multicast_address_low 224.10.10.10
context transport_lbtrm_multicast_address_high 224.10.10.14
```

See also:

Transport LBT-RM Network Options

6.8 Preventing Store Timeouts

When using Persistence, a store may be erroneously declared unresponsive if its activity timeout expires before it has had adequate opportunity to verify it is still active via activity check intervals.

Interrelated Options:

- ume_store_activity_timeout (source)
- ume_store_check_interval (source)

Recommendations:

Set the store activity timeout to at least 5x the activity check interval

Example:

```
#
# To avoid erroneous store activity timeouts, set the activity
# timeout to at least 5x the activity check interval.
#
source ume_store_activity_timeout 3000
source ume_store_check_interval 500
```

6.9 Preventing ULB Timeouts

When using ULB queuing, ULB source or receiver may be erroneously declared unresponsive if its activity timeout expires before it has had adequate opportunities to attempt to re-register via activity check intervals if the source appears to be inactive. It is also possible for sources to attempt to reassign messages that have already been processed.

Interrelated Options:

- umq_ulb_source_activity_timeout (receiver)
- umq_ulb_source_check_interval (receiver)
- umq_ulb_application_set_message_reassignment_timeout (source)
- umq_ulb_application_set_receiver_activity_timeout (source)
- umq_ulb_check_interval (source)

Recommendations:

Set the ULB source activity timeout to at least 5x the ULB source activity check interval.
Set the ULB application set message reassignment timeout to at least 5x the ULB check interval.
Set the ULB receiver activity timeout to at least 5x the ULB check interval.

Example:

```
#
# To avoid erroneous ULB source, receiver or application set message activity
# timeouts, set the activity timeout to at least 5x the activity check interval.
#
receiver umq_ulb_source_activity_timeout 10000
receiver umq_ulb_source_check_interval   1000
source   umq_ulb_application_set_message_reassignment_timeout 50000
source   umq_ulb_application_set_receiver_activity_timeout    10000
source   umq_ulb_check_interval 1000
```

See also:

Ultra Messaging Queuing Options]]])

6.10 Preventing Unicast Resolver Daemon Timeouts

A unicast resolver daemon may be erroneously declared inactive if its activity timeout expires before it has had adequate opportunity to verify that it is still alive.

Interrelated Options:

- resolver_unicast_activity_timeout (context)
- resolver_unicast_check_interval (context)

Recommendations:

Set the unicast resolver daemon activity timeout to at least 5x the activity check interval. Or, if activity notification is not desired, set both options to 0.

Example:

```
#
# To avoid erroneous unicast resolver daemon timeouts, set the activity
# timeout to at least 5x the activity check interval.
#
context resolver_unicast_activity_timeout 1000
context resolver_unicast_check_interval 200
```

See also:

UDP-Based Resolver Operation Options

6.11 Preventing Store Registration Hangs

The following configuration options come into play when sources register with stores in a lossy environment:

Interrelated Options:

- ume_sri_request_interval (receiver)
- ume_sri_request_maximum (receiver)
- transport_topic_sequence_number_info_request_interval (receiver)
- transport_topic_sequence_number_info_request_maximum (receiver)
- transport_tcp_activity_timeout (receiver)
- transport_lbtrm_activity_timeout (receiver)
- transport_lbtru_activity_timeout (receiver)
- transport_lbtipc_activity_timeout (receiver)

The sri_request "interval" and "maximum" options multiply to define a duration over which the receiver requests Store Information Records (SRI) messages from the source. Similarly, the transport_topic_sequence_number_↩ info_request "interval" and "maximum" options multiply to define a duration over which the receiver requests Transport Topic Sequence Number Info (TSNI) messages from the source.

Recommendations:

The two request durations should be twice the value of the appropriate transport activity timer.

Example:

```
#
# To avoid hung store registration, set the durations of the SRI and TSNI
# requests to 2x the transport activity timeout.
#
receiver transport_lbtrm_activity_timeout 60000
receiver ume_sri_request_maximum 120
receiver ume_sri_request_interval 1000
receiver transport_topic_sequence_number_info_request_maximum 120
receiver transport_topic_sequence_number_info_request_interval 1000
```

Warning

As of this version of UM, the default values for these options do not satisfy this recommendation. Users are advised to double the values for ume_sri_request_maximum (receiver) and transport_topic_sequence_↩ number_info_request_maximum (receiver).

Chapter 7

General Configuration Guidelines

7.1 Case Sensitivity

All Ultra Messaging scope, option, and value strings are case-insensitive. Thus, the following are identical:

```
context fd_management_type wincompport
Context Fd_Management_Type WinCompPort
CONTEXT FD_MANAGEMENT_TYPE WINCOMPPORT
```

7.2 Specifying Interfaces

The ∗**_interface** options require a network interface, usually supplied as a string (from a configuration file or in source code via ∗**_attr_str_setopt()**), the syntax used for network interface specifications is CIDR notation:

a.b.c.d/num

where '/***num***' is the optional "prefix length", the number of leading 1 bits in the netmask. If the prefix length is omitted, it defaults to 32 (netmask 255.255.255.255), which means that it must be an exact match for the interface's IP address. However, if the prefix length '/num' is supplied, it tells Ultra Messaging to select the first interface that starts with that network number. This makes it easier to share a configuration file between many (possibly multi-homed) machines on the same network.

For example:

```
context resolver_unicast_interface 192.168.0.0/24
```

specifies a netmask of 255.255.255.0 and would match the interface 192.168.0.3 on one host, and 192.168.0.251 on another host. But would not match 192.168.1.3.

The prefix length ("/num") does not need to match the actual network mask used by the host. For example, you have many hosts on different internal IP networks, but if they all start with "10", you can specify the interface as "10.0.0.0/8". UM will scan the list of interfaces and select the first one it finds that starts with 10. This is useful for selecting any network-connected interface, omitting the loopback 127.0.0.1.

You can also set network interfaces by device name. When setting a configuration option's interface by device name, you must use double quotes, as illustrated below.

```
context resolver_unicast_interface "en0"
```

Finally, you can also set network interfaces by DNS host name. When setting a configuration option's interface by DNS name, simply replace the dotted IP address with the host name, as illustrated below.

```
context resolver_unicast_interface myhost.mydomain.com/24
```

Notice the use of the optional netmask even though the host name will typically resolve to a specific host IP address. In this case, UM will zero out the host bits of **myhost**'s address and find any interface within that network. If the netmask is omitted, an exact match to **myhost**'s address is needed.

7.2.1 Interface Device Names and XML

As mentioned above, when a device name is supplied as an interface specification, the device name must be enclosed in double quotes. This presents a problem when the configuration option is specified within an XML file. In XML files, the values for all options must be enclosed in double quotes, but those quotes are only used by the XML parser to delimit the value. The quote characters themselves are not passed to the UM configuration parser. But the UM configuration parser needs the double quotes to indicate that the device name is being used.

The solution is to use the """ escape when specifying device names for interfaces within an XML file. The XML parser will convert those to actual double quote characters as part of the value passed to UM.

For example:

```
<options type="context">
  <option name="resolver_multicast_interface" default-value="&quot;en0&quot;">
  </option>
</options>
```

Another example:

```
<options type="context">
  <option name="monitor_transport_opts"
    default-value="context|resolver_multicast_interface=&quot;en0&quot;;source|transport=lbt-rm">
  </option>
</options>
```

(The repeated semicolon looks strange; the first one closes the """, and the second one separates the resolver_↩ multicast_interface option from the transport option.)

7.3 Socket Buffer Sizes

When specifying send or receive socket buffer sizes, keep the following platform-specific information in mind.

Linux

The kernel value net.core.rmem_max dictates the highest value allowed for a receive socket. The kernel value net.core.wmem_max dictates the highest value allowed for a sending socket. Increase these values to increase the amount of buffering allowed.

Windows

Windows should allow socket buffer sizes to be set very high if needed without requiring registry changes.

See our whitepaper Topics in High Performance Messaging for background and guidelines on UDP buffer sizing.

7.4 Port Assignments

There are a large number of configuration options which are network port numbers. In many cases, ranges of ports are specified so that multiple instances of UM-based programs can be run on the same machine without interference. Each instance will find a free port in the configured range. However, if the range is not large enough, an instance of UM can fail to initialize due to ports not being available.

Port range exhaustion can also happen if other software packages assign to ports in the range configured for UM. Users should be careful to configure all their networking packages to use non-overlapping port numbers.

7.4.1 Ephemeral Ports

The operating system allocates a range of ports for *ephemeral* ports. These ports are allocated dynamically as-needed by networking packages, including UM, for sockets that don't need a well-known, predictable port number. See Wikipedia's article Ephemeral port for ephemeral port ranges used by popular operating systems.

UM port configurations should avoid the host's ephemeral port range. Since these ports are allocated dynamically by the operating system, these allocations can interfere with UM by exhausting UM port ranges.

7.4.2 Network VS Host Order

When the UM C API is used to set configuration options programmatically, port numbers can be specified as a string or as a binary value. For example, here is an option being set by binary value:

```
unsigned short int optval = 4901;  /* host byte order required */
size_t optlen = sizeof(optval);
rc = lbm_context_attr_setopt(attrib, "transport_tcp_port_low", &optval, optlen);
```

See Setting an Option from a Binary Value.

There are some port options whose binary values must be supplied in network order. For example:

```
unsigned short int optval = htons(4901);  /* network byte order required */
size_t optlen = sizeof(optval);
rc = lbm_source_attr_setopt(attrib, "transport_tcp_port", &optval, optlen);
```

It is generally the case where setting a port to a specific value (i.e. not setting up a range) requires network order. Whereas setting the high and low port values of a range are done in host order.

The reference documentation for each port option specifies the byte order required when binary values are being specified. For example, transport_tcp_port (source) has a table row that says:

Byte order:	Network

7.5 Reference Entry Format

This section describes the format of each option reference entry.

Each entry begins with a brief description of the option. Following the description is a series of items that defines permissible usage and describes the values for the option.

Scope

Defines the scope to which the option applies.

Type

Defines the data type of the option. The type is required for calls to the ∗_setopt() and ∗_getopt() API functions.

Units

Defines the units in which the option value is expressed. This item is optional.

Default value

For range-valued options, indicates the base default value for the option.

Byte order

For options whose value is an IP address or port, defines the byte ordering (Host or Network) expected by the API for ∗_setopt() calls, and returned by the API for ∗_getopt() calls.

May be set during operation

If an option may be set after the UM object is created, it is so indicated here.

Next, for enumerated-valued options with limited specific choices, a table details the permissible String Value (configuration file), Integer Value (programmatic attribute setting), and a Description of each choice that includes default value designations.

Alternately, for switch-valued options (0 or 1), a table describes the meaning of each of the two possible values. The default value is noted within the description.

Chapter 8

Special Notes

8.1 Configuring Multi-Homed Hosts

By default, UM will select the first multicast-capable, non-loopback interface for multicast topic resolution. If you are fortunate, on a multi-homed host, the correct interface will be selected. However, this fortuitous selection should not be relied upon. Moving the interface card to a different slot, a change in the operating system kernel, and numerous other factors can lead to a different ordering of interfaces as reported by the operating system. This in turn can lead UM to a select a different interface after the change.

It is strongly recommended that the actual interface be specified. The resolver_multicast_interface (context) option allows you to explicitly specify the multicast interface. Note that this also changes the interface for LBT-RM and multicast immediate messaging.

Other interface options:

resolver_unicast_interface (context) when using the unicast resolver
request_tcp_interface (context) when using the request/response messaging
transport_lbtru_interface (receiver)
transport_lbtru_interface (source)
transport_tcp_interface (receiver)
transport_tcp_interface (source)

TCP transport:

transport_tcp_port_low (context)
transport_tcp_port_high (context)
transport_tcp_port (source)

LBT-RM transport:

transport_lbtrm_source_port_low (context)
transport_lbtrm_source_port_high (context)
transport_lbtrm_destination_port (source)

LBT-RU transport:

transport_lbtru_port_low (context)
transport_lbtru_port_high (context)
transport_lbtru_port (source)
transport_lbtru_port_low (receiver)
transport_lbtru_port_high (receiver)

Multicast immediate messaging:

mim_destination_port (context)
mim_incoming_destination_port (context)
mim_outgoing_destination_port (context)

Requests:

request_tcp_port (context)
request_tcp_port_low (context)
request_tcp_port_high (context)

In addition, since machines acting as a firewall are often multi-homed as well, see Configuring Multi-Homed Hosts for additional considerations.

8.2 Traversing a Firewall

To use UM across a firewall, several port options may need to be changed. The options of interest include:

Multicast resolver:

resolver_multicast_port (context)

Unicast resolver:

resolver_unicast_port (context)
resolver_unicast_port_low (context)
resolver_unicast_port_high (context)
resolver_unicast_destination_port (context)

Chapter 9

Major Options

Options in this group have a major impact on the operation of Ultra Messaging. Most UM application developers will need to be aware of the default values of these options or perhaps override them.

9.1 Reference

9.1.1 broker (context)

Add a broker specification to the list of brokers. Unlike most other UM settings, every time this setting is called, it adds one or more service specifications to the list, and does NOT overwrite previous specifications.

For the configuration file as well as string API method of setting this option, you can specify multiple brokers with a comma or semicolon-separated list on a single line. Each entry contains the broker IP address (or domain name of the IP address) and destination port in the format `IP:Dest_Port[,IP:Dest_Port]`.

An entry or string with the IP address of 0.0.0.0 and port 0 removes all previous broker specifications.

When the binary form of option setting is used, UM does NOT expect an array of structures. Instead, only one broker specification can be supplied for each call to **lbm_context_attr_setopt()**. However, when the binary form of option retrieval **lbm_context_attr_getopt()** is used, the list of brokers is returned as an array, and the **`optlen`** parameter should be set as:

```
optlen = (max_num_brokers * sizeof(lbm_transport_broker_entry_t));
```

Scope:	context
Type:	**lbm_transport_broker_entry_t**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMQ 6.8

9.1.2 compatibility_include_pre_um_6_0_behavior (context)

Enable Ultra Messaging 6.x applications to inter-operate with pre-6.0 applications.

Enabling this option increases overhead data on the wire and slightly changes some operational behaviors of persistent sources.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.7

String value	Integer value	Description
"1"	1	Inter-operate with pre-6.0 applications.
"0"	0	Disable Inter-operation with pre-6.0 applications. **Default for all.**

9.1.3 context_event_function (context)

Callback function (and associated client data pointer) that is called when a context event occurs. This callback may be called inline or from an event queue, if one is given.

If called inline, the callback function used should not block or it will block the context thread processing. See **lbm_context_event_cb_proc**.

Scope:	context
Type:	**lbm_context_event_func_t**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMQ 1.0.

9.1.4 context_name (context)

The name of the context, limited to 128 alphanumeric characters, hyphens or underscores.

This is only used for XML Configuration Files.

Scope:	context
Type:	**string**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.3/UME 3.3/UMQ 2.3.

9.1.5 default_interface (context)

Specifies the network interface to be used as the default setting for all other interface configuration options.

You can specify the full IP address of an interface, or just the network part (see Specifying Interfaces for details).

Default is set to INADDR_ANY, meaning that it will not bind to a specific interface.

Note: if specifying an interface name in an XML-format file, see Interface Device Names and XML.

Scope:	context
Type:	**lbm_ipv4_address_mask_t**
Default value:	0.0.0.0 (INADDR_ANY)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 6.10

9.1.6 dynamic_fragmentation_reduction (context)

Reduce UM and/or IP **fragmentation** by dynamically calculating the header size per message datagram.

Enabling this option makes UM's transport protocols more-fully utilize the configured datagram max size. This option is typically only of interest to users of LBT-RM and/or LBT-RU who need to avoid IP fragmentation, such as users of a **kernel-bypass driver**.

See **Dynamic Fragmentation Reduction** for details.

Enabling this option should be done in conjunction with setting the datagram max size options to 1472.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.14

Value	Description
1	Uses optimal fragmentation lengths.
0	Uses conservative fragmentation lengths. **Default for all.**

9.1.7 fd_management_type (context)

Define the mechanism UM uses for socket file descriptor (FD) management.

For more information, search on "file descriptors" in the Informatica Knowledge Base.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.

String value	Integer value	Description
"poll"	LBM_CTX_ATTR_FDTYPE_POLL	FD management uses poll(). *Unix only.*
"select"	LBM_CTX_ATTR_FDTYPE_SELECT	FD management uses select(). *Unix only.* **Default for Unix.**
"epoll"	LBM_CTX_ATTR_FDTYPE_EPOLL	FD management uses epoll(). *Linux kernel 2.6 or later only.*
"devpoll"	LBM_CTX_ATTR_FDTYPE_DEVPOLL	FD management uses the /dev/poll driver. *Solaris 8 or later only.*
"wsaeventselect"	LBM_CTX_ATTR_FDTYPE_WSAEV	FD management uses WSAEventSelect() and WaitForMultipleObjects(), which imposes a limit of 64 file descriptors. *Windows only.*
"wincompport"	LBM_CTX_ATTR_FDTYPE_WINCPORT	FD management uses Windows completion ports and completion routines. Avoids the 64 file descriptor limit set by WSA↩ EventSelect(). *Windows XP or later only.* **Default for Windows.**

9.1.8 file_descriptor_management_behavior (context)

Set how the context monitors file descriptors (sockets) for events.

The "busy_wait" selection can reduce latency and especially latency outliers (jitter), at the expense of the thread consuming 100% CPU.

Only use "busy_wait" if there are enough cores to allocate a core exclusively to each receive thread. If there are too few cores, enabling "busy_wait" can actually increase latencies due to threads time-sharing CPU resources. Also, pinning threads to cores is highly recommended to prevent thread migration across cores.

See **Receive Thread Busy Waiting** for more information.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.12.1

String value	Integer value	Description
"pend"	LBM_CTX_ATTR_FD_MANAGEMENT_↩ BEHAVIOR_PEND	Causes the context or **XSP thread** to go to sleep waiting for socket events to happen. **Default for all.**
"busy_wait"	LBM_CTX_ATTR_FD_MANAGEMENT_↩ BEHAVIOR_BUSY_WAIT	The context or XSP thread will check repeatedly in a tight loop (busy waiting) for socket events to happen.

9.1.9 message_selector (receiver)

Enables UM to pass a message selector string to any receiver.

The value must be an expression that conforms to JMS message selector syntax as defined in the Oracle JMS specification.

Scope:	receiver
Type:	**string**
Default value:	NULL
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMQ 5.3.

9.1.10 multiple_receive_maximum_datagrams (context)

The number of datagrams requested to be read at a time from a UDP-based transport socket.

Normally, UM reads one datagram at a time from each socket that has data. This option allows the reading of multiple datagrams in a single read (using recvmmsg()), and processing them in a tight loop. This improves efficiency and can reduce average latency.

Value of 0 means do NOT use recvmmsg().

Only supported for LBT-RM and LBT-RU transport types. The multiple_receive_maximum_datagrams option does not apply to **MIM** or Topic Resolution.

Requires glibc 2.12 or later. This option is ignored for non-Linux platforms.

See **Receive Multiple Datagrams** for more information.

Scope:	context
Type:	**lbm_uint32_t**
Units:	datagrams
Default value:	0
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.8. However, it did not work correctly until UM 6.9.

9.1.11 operational_mode (context)

The mode in which UM's context thread operates to process events.

See **Embedded Mode** and **Sequential Mode** for more information.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.

String value	Integer value	Description
"embedded"	LBM_CTX_ATTR_OP_EMBEDDED	A thread is spawned within UM to handle processing of events (timers and socket events). **Default for all.**
"sequential"	LBM_CTX_ATTR_OP_SEQUENTIAL	The application is responsible for calling **lbm_context_process_events()** to process events. Sequential mode does not support Multi-Transport Threads.

9.1.12 operational_mode (xsp)

The mode in which UM operates to process events.

Refer to **Embedded Mode** for additional information.

Scope:	xsp
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.11

String value	Integer value	Description
"embedded"	LBM_CTX_ATTR_OP_EMBEDDED	A thread is spawned within UM to handle processing of events (timers and socket events). **Default for all.**
"sequential"	LBM_CTX_ATTR_OP_SEQUENTIAL	The application is responsible for calling **lbm_↩ xsp_process_events()** to process events.

9.1.13 ordered_delivery (receiver)

Indicates whether or not the topic should have its data delivered in order and reassembled.

For LBT-RM, LBT-RU, TCP-LB or LBT-IPC transport sessions only. (This option also applies to TCP when using Late Join because the Late Join messages are not part of the TCP message stream.)

Changing this option from the default value to a value of 0 (zero) results in message fragments being delivered as soon as they fully arrive. Value -1 allows arrival order delivery after the reassembly of large messages.

Note that ordering only applies to a specific topic from a single publisher. UM does not ensure ordering across topics, or on a single topic across different publishers.

See **Message Ordering** and **Message Fragmentation and Reassembly** for more information.

Scope:	receiver
Type:	**int**
When to Set:	Can only be set during object initialization.

String value	Integer value	Description
"1"	1	UM delivers topic messages to a receiver in-order and reassembles large messages. **Default for all.**
"-1"	-1	UM delivers topic messages to a receiver as they arrive and may be out of order. Duplicate delivery is possible. However, UM reassembles large messages. Your application can use the **`sequence_number`** field of **`lbm_msg_t`** objects to order or discard messages.
"0"	0	UM delivers topic messages to a receiver as they arrive and may be out of order. WARNING: This mode of operation is deprecated and may be removed in a future UM version. The user is advised to use mode -1. UM delivers large messages as individual fragments of less than the maximum datagram size for the transport in use. Duplicate delivery is possible. This mode is incompatible with **Message Properties**.

9.1.14 receiver_callback_service_time_enabled (context)

Indicates if UM collects receiver callback statistics, which provide the maximum, mean and minimum time in microseconds required to complete wildcard, hot-failover, and regular receiver callbacks.

Enabling this function slightly decreases the efficiency of the receive code path, but provides operators with greater visibility of application behavior.

Scope:	context
Type:	**int**
Default value:	0
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.5

Value	Description
1	UM collects receiver callback statistics.
0	UM does NOT collects receiver callback statistics. **Default for all.**

9.1.15 resolver_source_notification_function (context)

Application callback function (and associated client data pointer) that is called when a new source is discovered for any topic, even if the application does not have a matching receiver.

Contrast this with source_notification_function (receiver).

This callback is called by the context thread and can not use an event queue. Therefore the callback function used should not block or it will delay reception of latency-sensitive messages.

If this feature is used with a context that is connected to an SRS for Topic Resolution, resolver_service_interest_mode (context) may need to be set to "flood". See **TCP-Based TR Interest**.

Scope:	context
Type:	**lbm_src_notify_func_t**
Default value:	NULL
When to Set:	Can only be set during object initialization.
Config File:	Cannot be set from an UM configuration file.

9.1.16 source_event_function (context)

Callback function (and associated client data pointer) that is called when a context source event (such as a multicast immediate mode source wakeup event) occurs.

This callback may be called inline or from an event queue, if one is given. If called inline, the callback function used should not block or it will block the context thread processing.

Scope:	context
Type:	**lbm_context_src_event_func_t**
Default value:	NULL
When to Set:	Can only be set during object initialization.

Config File:	Cannot be set from an UM configuration file.
Version:	This option was implemented in LBM 3.4/UME 2.1.

9.1.17 source_includes_topic_index (context)

Determines whether the topic index is included in the source string generated for messages and new source notifications.

Users should not disable this.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.6/UME 3.0.

Value	Description
1	Indicates the topic index should be included in the source string. **Default for all.**
0	Indicates the topic index should not be included.

9.1.18 transport (source)

The transport type to be used for created sources.

Note

With **Smart Sources**, only LBT-RM and LBT-RU are supported.

Note

With **Transport Services Provider (XSP)**, only LBT-RM, LBT-RU, and TCP are supported.

Scope:	source
Type:	**int**
When to Set:	Can only be set during object initialization.

String value	Integer value	Description
"tcp"	LBM_SRC_TOPIC_ATTR_TRANSP↩ ORT_TCP	TCP over IPv4. **Default for all.**
"lbtrm", "lbt-rm"	LBM_SRC_TOPIC_ATTR_TRANSP↩ ORT_LBTRM	UDP-based reliable multicast with unicast NAKs.
"lbtru", "lbt-ru"	LBM_SRC_TOPIC_ATTR_TRANSP↩ ORT_LBTRU	UDP-based reliable unicast with unicast NAKs.
"lbtipc", "lbt-ipc"	LBM_SRC_TOPIC_ATTR_TRANSP↩ ORT_LBTIPC	Inter-Process Communication between processes on the same host using a shared memory area.
"lbtsmx", "lbt-smx"	LBM_SRC_TOPIC_ATTR_TRANSP↩ ORT_LBTSMX	Shared Memory Acceleration. Ultra-low-latency Inter-Process Communication transport between processes on the same host using a shared memory area. Restrictions apply.
"broker"	LBM_SRC_TOPIC_ATTR_TRANSP↩ ORT_BROKER	Sources send messages to a broker, which manages the messages for consumption.
"lbtrdma", "lbt-rdma"	LBM_SRC_TOPIC_ATTR_TRANSP↩ ORT_LBTRDMA	InfiniBand Remote Direct Memory Access transport. Deprecated in UM 6.9.

9.1.19 transport_demux_tablesz (receiver)

Specifies the size of the table used for storing receiver delivery controllers used by UM for message delivery.

Must be a power of two (1, 2, 4, 8, 16, etc.). If not a power of two, UM generates a log warning and uses the next highest power of two. For most use cases with low to moderate numbers of topics per transport session, the default suffices. For large numbers of topics and in cases where the lowest latency is desired, set the option to the next higher power of two than the number of topics expected on the transport session.

See **Transport Demultiplexer Table Size** for more information.

Scope:	receiver
Type:	**size_t**
Default value:	1
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.2/UME 3.2.

9.1.20 transport_mapping_function (context)

Application callback function (and associated client data pointer) that is called when a context is about to join a new transport session.

This callback provides an opportunity for the user to map the transport session in question to an XSP.

This callback is called by the context thread and can not use an event queue. Therefore the callback function used should not block or it will delay reception of latency-sensitive messages.

Scope:	context
Type:	**lbm_transport_mapping_func_t**
Default value:	NULL
When to Set:	Can only be set during object initialization.
Config File:	Cannot be set from an UM configuration file.
Version:	This option was implemented in UM 6.11

9.1.21 transport_session_multiple_sending_threads (context)

Flag that indicates the application intends to use multiple sending threads per transport session.

Disabling this flag can improve send efficiency but renders the send functions thread-unsafe.

For the most-efficient sending method, see **Smart Sources**.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.

Value	Description
1	Indicates the application does intend to use multiple sending threads per transport session and that UM should make that assumption. **Default for all.**
0	Indicates the application does not intend to use multiple sending threads per transport session and that UM should make that assumption.

9.1.22 transport_session_single_receiving_thread (context)

Flag that indicates the application intends to use only a single thread for receiving.

This improves message reception latency and outliers by converting certain thread-safe operations to more-efficient thread-unsafe operations. For example, certain bus-locked operations (e.g. atomic increment) are replaced by non-bus-locked equivalents (e.g. non-atomic increment).

See **Single Receiving Thread** for more information and restrictions.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.12

Value	Description
1	The user intends to ensure that only one thread is used to process LBM transport messages.
0	No assumptions will be made by LBM regarding threading. **Default for all.**

9.1.23 transport_source_side_filtering_behavior (source)

Enable and set the behavior for UM sources to filter out topics prior to sending to clients.

This option is not applicable for multicast-based sources (LBT-RM). These control messages are sent to the TCP UIM port (also known as the "request port") of the senders context and processed internally.

This option affects the transport session underlying the source rather than the source itself. See **Source Object** for additional information.

Scope:	source
Type:	**int**
When to Set:	Can only be set during object initialization.

String value	Integer value	Description
"none"	LBM_SRC_TOPIC_ATTR_SSF_NONE	The source sends all data to all clients regardless of the topics they are listening to. **Default for all.**
"inclusion"	LBM_SRC_TOPIC_ATTR_SSF_INCLUSION	The source sends only that data to a client that the client specifically requests.
"ulb"	LBM_SRC_TOPIC_ATTR_SSF_ULB	The ULB source sends control and data only to the ULB client that the source has specifically assigned for a given message. See **ULB Performance** for more information.

9.1.24 transport_topic_sequence_number_info_active_threshold (source)

Duration in seconds that an inactive source sends contiguous Topic Sequence Number Info (TSNI) messages.

A value of 0 indicates that sources continue sending TSNIs until data messages resume, with no timeout.

TSNIs are sent at an interval defined by transport_topic_sequence_number_info_interval (source).

See also Interrelated Configuration Options.

Scope:	source
Type:	**lbm_ulong_t**
Units:	seconds
Default value:	60
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0

9.1.25 transport_topic_sequence_number_info_interval (source)

The interval between Topic Sequence Number Info (TSNI) messages that a source sends.

TSNI messages are enabled on all transports except LBT-SMX, and they carry the topic sequence number of the latest message sent by the source. The interval is also a source inactivity threshold. In other words, a source does not send TSNIs during normal data transmission, but once the source is inactive for as long as this interval, it starts sending TSNI messages. A value of 0 turns off TSNI messages for the source.

See also Interrelated Configuration Options.

Scope:	source
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	5000 (5 second)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.3

9.1.26 transport_topic_sequence_number_info_request_interval (receiver)

The interval at which the receiver requests a Topic Sequence Number Information Record (TSNI) from the source.

Controlling these requests helps reduce receiver start-up traffic on your network.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	1000 (1 second)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMP 6.0

9.1.27 transport_topic_sequence_number_info_request_maximum (receiver)

The maximum number of requests the receiver issues for a Topic Sequence Number Information Record (TSNI) from the source.

If the receiver has not received an TSNI after this number of requests, it stops requesting.

Scope:	receiver
Type:	**lbm_ulong_t**
Default value:	60
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMP 6.0

9.1.28 use_extended_reclaim_notifications (source)

Specifies which reclaim notification your application receives.

The older **LBM_SRC_EVENT_UME_MESSAGE_RECLAIMED** source event delivered the structure **lbm_src_event_ume_ack_info_t**. The newer **LBM_SRC_EVENT_UME_MESSAGE_RECLAIMED_EX** source event delivers the structure **lbm_src_event_ume_ack_ex_info_t**, which contains additional information.

Scope:	source
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.2.

Value	Description
1	Indicates your application receives the expanded reclaim notification. **Default for all.**
0	Indicates your application receives the older reclaim notification.

9.1.29 zero_transports_function (xsp)

Application callback function (and associated client data pointer) that is called when the number of transports associated with a given XSP falls to zero.

This callback provides an opportunity to delete the given XSP.

This callback is called by the context thread and can not use an event queue. Therefore the callback function used should not block or it will delay reception of latency-sensitive messages.

Scope:	xsp
Type:	**lbm_xsp_zero_transports_func_t**
Default value:	NULL
When to Set:	Can only be set during object initialization.
Config File:	Cannot be set from an UM configuration file.
Version:	This option was implemented in UM 6.11

Chapter 10

UDP-Based Resolver Operation Options

This section describes configuration options for UDP-based TR. The options generally apply equally to both Multicast UDP and Unicast UDP Topic Resolution. See **Topic Resolution Overview** for more information.

The following topic resolution options have been deprecated in LBM Version 4.0.

- resolver_active_source_interval (context)
- resolver_active_threshold (context)
- resolver_maximum_advertisements (context)
- resolver_maximum_queries (context)
- resolver_query_interval (context)

See Re-establish Pre-4.0 Topic Resolution for option values that configure the topic resolution used in LBM Version 3.6 and prior versions. You should also comment out or remove from your Ultra Messaging Configuration file the deprecated configuration options shown above.

10.1 Minimum Values for Advertisement and Query Intervals

These intervals have the following effective minimal values.

- 10 ms for Initial Phase Advertisements
- 20 ms for Initial Phase Queries
- 30 ms Wildcard Queries
- 100 ms for Sustaining Phase Advertisements and Queries
- 100 ms for Context Name Queries (mostly for persistence)

These effective minimums exist because the internal timer that schedules advertisements and queries fires at the stated interval, i.e., every 10 ms for Initial Phase Advertisements, every 20 ms for Initial Phase Queries, etc. If you set the option's value below the minimum, after the initial advertisement or query at 0 ms, the resolver schedules the second advertisement or query at the first timer "tick", which is the minimum.

Subsequent advertisements or queries can only be issued at the next timer "tick". If you increase this option from the default to a value that is not a multiple of the minimum, the resolver maintains the rate you establish as an average over subsequent "ticks".

As an example, if you set resolver_advertisement_sustain_interval (source) or resolver_query_sustain_interval (receiver) to 10 ms, the resolver schedules the second advertisement or query after the initial (0 ms) at the first timer "tick", which is 100 ms. Subsequent advertisements or queries can only be issued at the next timer "tick" (every 100 ms). If you increase either option from the default to 1.25 seconds, for example and not a multiple of 100 ms, the resolver maintains the rate you establish as an average over subsequent "ticks". That is, the second advertisement or query goes out at the 1300 ms "tick". The resolver tracks the tardiness of this advertisement (50 ms) and adjusts the next advertisement or query, which goes out at 2500 ms, giving an average of 1250 ms or 1.25 seconds.

10.2 Reference

10.2.1 disable_extended_topic_resolution_message_options (context)

This is a topic resolution compatibility option that, when set to 1, lets LBM 4.0 (or later) installations work with LBM 3.5.3 / UME 2.2.4 (or earlier) installations.

If you do not have early-version installations in the network, leave this option at 0.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0.

Value	Description
1	Enable compatibility with earlier-version installations (and disable some message structure features).
0	Normal current-version compatibility. Strongly recommended. **Default for all.**

10.2.2 resolution_no_source_notification_threshold (receiver)

The threshold for the number of unanswered topic resolution queries before UM delivers a LBM_MSG_NO_↩ SOURCE_NOTIFICATION for the topic.

The receiver does not stop querying after the delivery of this notification. A value of 0 indicates no notifications will be sent.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	Number of queries
Default value:	0 (do not notify)
When to Set:	May be set during operation.

10.2.3 resolution_number_of_sources_query_threshold (receiver)

The threshold for the number of sources a topic must have before topic resolution queries are not sent.

A value of zero results in no topic resolution queries being generated. See also Disabling Aspects of Topic Resolution.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	Number of sources
Default value:	10000000 (10 million)
When to Set:	May be set during operation.

10.2.4 resolver_advertisement_maximum_initial_interval (source)

The longest - and last - interval in the initial phase of topic advertisement.

A value of 0 disables the initial phase of advertisement. See also Disabling Aspects of Topic Resolution.

Scope:	source
Type:	**lbm_ulong_t**
Units:	milliseconds

Default value:	500 (0.5 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0

10.2.5 resolver_advertisement_minimum_initial_duration (source)

The duration of the initial phase of topic advertisement.

A value of 0 guarantees that the initial phase of advertisement never completes.

Scope:	source
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	5000 (5 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0

10.2.6 resolver_advertisement_minimum_initial_interval (source)

Interval between the first topic advertisement sent upon creation of the source and the second advertisement sent by the source.

A value of 0 disables the initial phase of advertisement. This option has an effective minimum of 10 ms. See UDP-Based Resolver Operation Options.

See also Disabling Aspects of Topic Resolution.

Scope:	source
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	10 (0.01 seconds)

When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0

10.2.7 resolver_advertisement_minimum_sustain_duration (source)

The duration of the sustaining phase of topic advertisement.

A value of 0 guarantees that the sustaining phase of advertising never completes.

Scope:	source
Type:	**lbm_ulong_t**
Units:	seconds
Default value:	60 (1 minute)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0

10.2.8 resolver_advertisement_send_immediate_response (source)

Allows you to disable the normal immediate response to queries and wildcard queries.

Sources normally send topic advertisements (TIR) immediately in response to topic queries (TQR) for a local topic or wildcard queries (WC-TQR) with a pattern that matches a local topic. This helps to resolve topics quickly. However, these immediate TIRs are also inefficient; each TIR is sent in a UDP datagram of its own.

In contrast, TIRs sent using the normal, rate-limited phases of advertisement are batched, with multiple TIRs collected into a single UDP datagram. For systems with large numbers of sources and receivers, allowing immediate response to queries can lead to high short-term network loading and packet loss. In these cases, it can be beneficial to disable immediate responses, at the expense of longer times required to resolve new receivers.

Scope:	source
Type:	**lbm_uint_t**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.2/UME 3.2/UMQ 2.1

Value	Description
1	Sources immediately send advertisements (TIR) in response to topic queries (TQR) or wildcard queries (WC-TQR). **Default for all.**
0	Sources delay sending advertisements (TIR) in response to topic queries (TQR) or wildcard queries (WC-TQR).

10.2.9 resolver_advertisement_sustain_interval (source)

Interval between sending topic advertisements in the sustaining phase of topic advertisement.

A value of 0 disables the sustaining phase of advertisement. This option has an effective minimum of 100 ms. See UDP-Based Resolver Operation Options.

See also Disabling Aspects of Topic Resolution.

Scope:	source
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	1000 (1 second)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0

10.2.10 resolver_cache (context)

Whether or not to enable the resolver cache to hold topic resolution information. Disabling the resolver cache uses less memory, but can increase network load. Informatica recommends against disabling the resolver cache.

Warning

The resolver cache must be enabled if wildcard receivers and/or if resolver_service (context) is used.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.

Value	Description
1	Topic resolution information will be cached. **Default for all.**
0	Do not cache topic resolution information.

10.2.11 resolver_context_name_activity_timeout (context)

Period of inactivity before a context name is declared unresolved.

The minimum amount of time without any context name resolution traffic that must pass before UM declares a resolved context name unresolved. Context name resolution traffic is defined as the reception of context name advertisements and/or unicast control traffic from the resolved context.

Scope:	context
Type:	**lbm_uint64_t**
Units:	milliseconds
Default value:	60000 (60 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.0

10.2.12 resolver_context_name_query_duration (context)

Maximum period of time UM sends context name queries.

The maximum duration for which UM sends context name queries for a given context name. UM sends queries until the context name resolves. A value of 0 means queries have no time limit and UM continues to query until the context name resolves.

Scope:	context
Type:	**lbm_uint64_t**
Units:	milliseconds
Default value:	0 (query for as long as unresolved)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.0

10.2.13 resolver_context_name_query_maximum_interval (context)

The longest interval between sending context name queries.

A value of 0 disables context name queries.

This option has an effective minimum of 100 ms. See UDP-Based Resolver Operation Options.

See also Disabling Aspects of Topic Resolution.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	1000 (1 second)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.0

10.2.14 resolver_context_name_query_minimum_interval (context)

Interval between the first context name query sent upon creation of the persistent source using named Stores and the second query sent.

A value of 0 disables context name queries. This option has an effective minimum of 100 ms. See UDP-Based Resolver Operation Options.

See also Disabling Aspects of Topic Resolution.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	100 (0.1 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.0

10.2.15 resolver_datagram_max_size (context)

The maximum UDP datagram payload size that can be generated for topic resolution advertisements and queries.

Note that this does not include UDP, IP, or packet overhead added by the network stack. The default value is 8192, the minimum is 500 bytes, and the maximum is 65535. See **Message Fragmentation and Reassembly** for more information.

All components in the UM network, both applications and UM daemons, should have the same setting for this option.

Scope:	context
Type:	**lbm_uint_t**
Units:	bytes
Default value:	8192
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.6/UME 3.0.

10.2.16 resolver_disable_udp_topic_resolution (context)

Used to disabled UDP-based Topic Resolution.

This is typically only used when TCP-based Topic Resolution is enabled; see resolver_service (context).

When enabled, no UPD based Topic Resolution traffic will be generated nor will any UDP based Topic Resolution traffic be processed.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.14

Value	Description
1	Disables UDP based Topic Resolution.
0	Normal UDP based Topic Resolution is used. **Default for all.**

10.2.17 resolver_domain_id_active_propagation_timeout (context)

Indicates how a context learns the ID of its own Topic Resolution Domain (TRD).

See **Topic Resolution Domain**.

Scope:	context
Type:	**int**
Default value:	0
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.7.1

String value	Integer value	Description
"-1"	-1	Learn TRD ID from other contexts in the same TRD, without restriction. This is the method Ultra Messaging has traditionally used. This method assigns TRD IDs quickly to avoid partial connectivity. However, note that to change a TRD ID, you must reconfigure and restart all DROs, if present. Then you must delete all application contexts, and then re-create all application contexts. Note: With this option value, newly-created contexts can learn from earlier versions of Ultra Messaging software.
"0"	0	Learn TRD ID only from a DRO directly. Do not learn the TRD ID from other contexts in the same TRD. Consider using this option with a TRD that has many contexts and a possible need to change a TRD ID. **Default for all.**
"1" to "2147483647"	1 to 2,147,483,647	Learn TRD ID from other contexts in the same TRD that have heard the domain ID advertised by a DRO within this timeout value in seconds. Use the following formula: `3 * {propagation-interval}/1000 + {maximum expected duration of DRO outage}` where "propagation-interval" is an attribute value of the DRO configuration element <**route-info**>, which defaults to 1000. With a timeout value set, a restarted context does not learn obsolete TRD IDs from un-restarted contexts, but instead, learns from DROs or restarted contexts. You do not need to bring all contexts to a deleted state simultaneously before you re-create the first context. Note: During this timeout period, there is a risk for temporary incomplete connectivity in networks with no DROs.

10.2.18 resolver_initial_advertisement_bps (context)

Maximum advertisement rate during the initial phase of topic advertisement.

A value of 0 means that initial phase advertisements for the topic are not limited to a maximum number of bits per second. Note that the topic's advertisements are still subject to being rate limited by resolver_initial_↩ advertisements_per_second (context).

Refer to **Rate Controls** for additional information about the UM rate limiting algorithm.

Scope:	context
Type:	**lbm_uint64_t**
Units:	bits per second
Default value:	1000000
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0

10.2.19 resolver_initial_advertisements_per_second (context)

Maximum number of advertisements sent within a one second period during the initial phase of topic advertisement.

A value of 0 means that initial phase advertisements for the topic are not limited to a maximum number of advertisements per second. Note that the topic's advertisements are still subject to being rate limited by resolver_initial_advertisement_bps (context).

Refer to **Rate Controls** for additional information about the UM rate limiting algorithm.

Scope:	context
Type:	**lbm_ulong_t**
Units:	advertisements
Default value:	1000
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0

10.2.20 resolver_initial_queries_per_second (context)

Maximum number of queries sent within a one second period during the initial phase of topic querying.

A value of 0 means that initial phase queries for the topic are not limited to a maximum number of queries per second. Note that the topic's queries are still subject to being rate limited by resolver_initial_query_bps (context).

Refer to **Rate Controls** for additional information about the UM rate limiting algorithm.

Scope:	context
Type:	**lbm_ulong_t**
Units:	advertisements
Default value:	1000

When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0

10.2.21 resolver_initial_query_bps (context)

Maximum query rate during the initial phase of topic querying.

A value of 0 means that initial phase queries for the topic are not limited to a maximum number of bits per second. Note that the topic's queries are still subject to being rate limited by resolver_initial_queries_per_second (context).

Refer to **Rate Controls** for additional information about the UM rate limiting algorithm.

Scope:	context
Type:	**lbm_uint64_t**
Units:	bits per second
Default value:	1000000
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0

10.2.22 resolver_query_maximum_initial_interval (receiver)

The longest - and last - interval in the initial phase of topic querying.

A value of 0 disables the initial phase of querying.

See also Disabling Aspects of Topic Resolution.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds

Default value:	200 (0.2 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0

10.2.23 resolver_query_minimum_initial_duration (receiver)

The duration of the initial phase of topic querying.

A value of 0 guarantees that the initial phase of querying never completes.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	5000 (5 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0

10.2.24 resolver_query_minimum_initial_interval (receiver)

Interval between the first topic query sent upon creation of the receiver and the second query sent by the receiver.

A value of 0 disables the initial phase of querying. This option has an effective minimum of 20 ms. See UDP-Based Resolver Operation Options.

See also Disabling Aspects of Topic Resolution.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	20 (0.02 seconds)

When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0

10.2.25 resolver_query_minimum_sustain_duration (receiver)

The duration of the sustaining phase of topic querying.

A value of 0 guarantees that the sustaining phase of querying never completes.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	seconds
Default value:	60 (1 minute)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0

10.2.26 resolver_query_sustain_interval (receiver)

Interval between sending topic queries in the sustaining phase of topic querying.

A value of 0 disables the sustaining phase of querying. This option has an effective minimum of 100 ms. See UDP-Based Resolver Operation Options.

See also Disabling Aspects of Topic Resolution.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	1000 (1 second)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0

10.2.27 resolver_receiver_map_tablesz (context)

The size of the hash table used for storing receiver topic information used for topic resolution. This value should be a prime number.

For UM deployments with very large numbers of topics (more than 100,000), increasing this number can improve efficiency.

Scope:	context
Type:	**size_t**
Units:	map entries
Default value:	131111
When to Set:	Can only be set during object initialization.

10.2.28 resolver_send_final_advertisements (source)

Controls whether or not a source sends "final advertisements" before deletion.

A final advertisement is an announcement that the source object is being deleted. Without final advertisements, receivers are not informed that a source has been deleted until all sources on a transport session are deleted and the transport session is disposed. At that point, any receivers to sources on that transport session will simultaneously be delivered an EOS event.

However, if the source has final advertisements enabled, the source will send the final advertisement and trigger the delivery of the EOS event in a more-timely way. They also give other contexts an opportunity to purge the source from their local topic resolution cache.

Note: the final advertisements are not necessarily sent immediately upon deletion of the source. They are scheduled with other topic resolution traffic and obey the rate limits. As a result, if an application is in the process of cleaning up prior to exit and it deletes the context object too soon after the deletion of its sources, the final advertisements may not be sent at all. Typically this will simply result in a less-timely notification of EOS as transport sessions time out, but there are some circumstances where the time required to deliver EOS is not technically bounded. If timely delivery of EOS is important, it is recommended to add a few seconds of delay after the sources are deleted before deleting the context.

This setting does not affect the topic resolution phases you have configured, which execute as expected. See Disabling Aspects of Topic Resolution.

Scope:	source
Type:	**lbm_uint_t**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 6.10

Value	Description
1	Source sends final advertisements before deletion.
0	Source does not send any final advertisements before deletion. **Default for all.**

10.2.29 resolver_send_initial_advertisement (source)

Controls whether or not a source sends an advertisement upon creation.

Turning off this advertisement speeds source creation and reduces the number of messages on your network through application initialization.

See Disabling Aspects of Topic Resolution.

Scope:	source
Type:	**lbm_uint_t**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0

Value	Description
1	Source sends a topic advertisement immediately upon creation. **Default for all.**
0	Source does not send an advertisement upon creation. This setting does not affect the topic resolution phases you have configured, which execute as expected.

10.2.30 resolver_source_map_tablesz (context)

The size of the hash table used for storing source topic information used by topic resolution. This value should be a prime number.

For UM deployments with very large numbers of topics (more than 100,000), increasing this number can improve efficiency.

Scope:	context
Type:	**size_t**
Units:	map entries
Default value:	131111
When to Set:	Can only be set during object initialization.

10.2.31 resolver_string_hash_function (context)

The hash function to use for hashing topic name strings for source and receiver topics.

The application may choose from a list of defined hash functions or it may define its own hash function, as identified by the string value of this option. When setting a hash function, note that:

- If set through a configuration file or a call to **lbm_context_attr_str_setopt()**, only the string values classic, djb2, sdbm, or murmur2 are valid. (If retrieved by a call to **lbm_context_attr_str_getopt()**, one of these string values is returned.)

- If set through a call to **lbm_context_attr_setopt()**, you must pass a pointer to a hash function. Use this method for hash functions other than the four pre-defined functions.

Informatica's own testing has indicated that the default (murmur2) outperforms all the others, but there may be topic string formats for which a different function is better. Informatica suggests careful testing before changing the hash function.

Scope:	context
Type:	**lbm_str_hash_func_t**
Default value:	NULL
When to Set:	Can only be set during object initialization.

String value	Integer value	Description
"classic"	n.a.	A "classic" good string hash function. Works best when topic names have a constant prefix with a changing suffix.
"djb2"	n.a.	The Dan Bernstein algorithm from comp.lang.c. Works best when topic names have a changing prefix with a constant suffix.
"sdbm"	n.a.	sdbm database library (used in Berkeley DB). A useful alternative to djb2.
"murmur2"	n.a.	Good all-around hash function by Austin Appleby. Best for medium to long topic strings. **Default for all.**

10.2.32 resolver_string_hash_function_ex (context)

The hash function to use for hashing topic name strings for source and receiver topics.

This option is similar to the resolver_string_hash_function (context) except for the following differences: This option can be set via only **lbm_context_attr_setopt()** (not from a configuration file or **lbm_context_attr_str_setopt()**). Hence, this also means you cannot use the string options (classic, etc.). You can pass a string length to the hash function, allowing it to then possibly run faster by operating on multiple-character strings at a time. Note that if -1 is passed in, you must use a strlen to calculate the length. The hash function accepts a clientd pointer, which you can set as needed, and which is passed back in each time the function is called.

This option is the better choice when setting your own custom hash function. Note that both the resolver_string_hash_function and resolver_string_hash_function_ex options set the same attributes, hence, if you use both (not recommended) one will override the other.

Scope:	context
Type:	**lbm_str_hash_func_ex_t**
Default value:	NULL
When to Set:	Can only be set during object initialization.
Config File:	Cannot be set from an UM configuration file.

10.2.33 resolver_sustain_advertisement_bps (context)

Maximum advertisement rate during the sustaining phase of topic advertisement.

A value of 0 means that sustaining phase advertisements for the topic are not limited to a maximum number of bits per second. Note that the topic's advertisements are still subject to being rate limited by resolver_sustain_advertisements_per_second (context).

Refer to **Rate Controls** for additional information about the UM rate limiting algorithm.

Scope:	context
Type:	**lbm_uint64_t**
Units:	bits per second
Default value:	1000000
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0

10.2.34 resolver_sustain_advertisements_per_second (context)

Maximum number of advertisements sent within a one second period during the sustaining phase of topic advertisement.

A value of 0 means that sustaining phase advertisements for the topic are not limited to a maximum number of advertisements per second. Note that the topic's advertisements are still subject to being rate limited by resolver_sustain_advertisement_bps (context).

Refer to **Rate Controls** for additional information about the UM rate limiting algorithm.

Scope:	context
Type:	**lbm_ulong_t**
Units:	advertisements
Default value:	0

When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0

10.2.35 resolver_sustain_queries_per_second (context)

Maximum number of queries sent within a one second period during the sustaining phase of topic querying.

A value of 0 means that sustaining phase queries for the topic are not limited to a maximum number of queries per second. Note that the topic's queries are still subject to being rate limited by resolver_sustain_query_bps (context).

Refer to **Rate Controls** for additional information about the UM rate limiting algorithm.

Scope:	context
Type:	**lbm_ulong_t**
Units:	advertisements
Default value:	0
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0

10.2.36 resolver_sustain_query_bps (context)

Maximum query rate during the sustaining phase of topic querying.

A value of 0 means that sustaining phase queries for the topic are not limited to a maximum number of bits per second. Note that the topic's queries are still subject to being rate limited by resolver_sustain_queries_per_↩ second (context).

Refer to **Rate Controls** for additional information about the UM rate limiting algorithm.

Scope:	context
Type:	**lbm_uint64_t**

Units:	bits per second
Default value:	1000000
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0

10.2.37 resolver_unicast_activity_timeout (context)

Indicates the maximum time between messages from a unicast resolver daemon before UM declares it inactive and stops sending normal topic resolution traffic via that daemon.

UM will still send keepalives to the daemon. A value of 0 will force all resolver daemons to be treated as permanently active.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	1000
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMS 5.0

10.2.38 resolver_unicast_change_interval (context)

Indicates how often UM will change to the next available unicast resolver daemon.

The actual value used is random, and is selected from the range (1/2∗change_interval, 3/2∗change_interval). If all resolver daemons have been marked inactive, UM enters a quick-change mode where it uses a random value from the range (1/4∗change_interval, 3/4∗change_interval) in order to more quickly locate an active daemon.

See resolver_unicast_daemon (context) option.

Scope:	context
Type:	**lbm_ulong_t**

Units:	milliseconds
Default value:	200
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMS 5.0

10.2.39 resolver_unicast_check_interval (context)

Indicates how often a UM checks for resolver activity in order to determine liveness.

A value of 0 will disable activity checks. This setting only applies to the unicast resolver.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	200
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMS 5.0

10.2.40 resolver_unicast_force_alive (context)

Controls whether a context with no sources or receivers should register with and send keepalive messages to a configured Unicast Topic Resolver.

By default, at least one source or receiver must exist in a context before it registers with a configured Unicast Topic Resolver.

However, some receiving application designs use resolver_source_notification_function (context) to notify them of discovered sources, and do not create a receiver until sources are discovered. If these designs use unicast topic resolution, they should set this option to "1".

Scope:	context
Type:	**lbm_uint16_t**
When to Set:	Can only be set during object initialization.

Value	Description
1	Contexts send keepalive messages to the Unicast Resolver at the resolver_unicast_keepalive_interval (context) regardless of whether the context has any sources or receivers that require topic resolution.
0	Contexts do not send keepalive messages to the Unicast Resolver until sources or receivers have been created. Then Contexts send keepalives at the resolver_unicast_keepalive_interval (context). **Default for all.**

10.2.41 resolver_unicast_ignore_unknown_source (context)

Indicates whether contexts using unicast topic resolution accept topic resolution udp datagrams that originate from any lbmrd or only the specific lbmrd configured for use.

Note: Do not modify this setting without guidance from Informatica.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.7.1

String value	Integer value	Description
"0"	0	A context using unicast topic resolution accepts traffic from lbmrd resolver daemons not configured for use by the context.
"1"	1	Contexts using unicast topic resolution accept topic resolution udp datagrams that originate from only the specific lbmrd configured for use. The context discards topic resolution datagrams from unrecognized origins and logs a message. This prevents applications at the same IP address, but in different topic resolution domains that might share resolver unicast port ranges, from processing unintended topic resolution traffic while lbmrd resolver daemons time out stale client entries. **Default for all.**

String value	Integer value	Description

10.2.42 resolver_unicast_keepalive_interval (context)

Indicates how often keepalive messages should be sent to a resolver daemon.

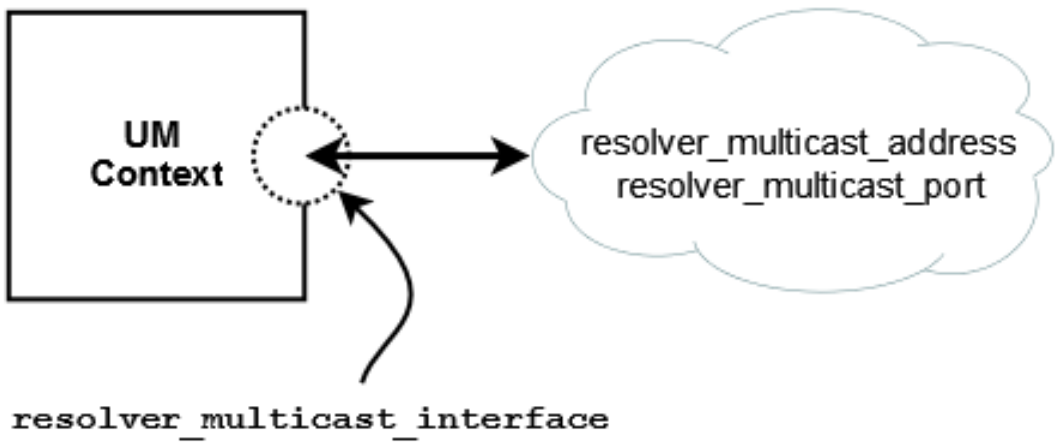
Keepalives are only sent if no other traffic has been sent since the last keepalive interval expired.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	500
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMS 5.0

Chapter 11

Multicast Resolver Network Options

The image below shows a simplified relationship between the primary multicast resolver network options.



See **Topic Resolution Overview** for general information on Topic Resolution.

11.1 Reference

11.1.1 resolver_multicast_address (context)

Multicast address (or domain name of the multicast address) used for Topic Resolution.

This option automatically sets the values for resolver_multicast_incoming_address (context) and resolver_multicast_outgoing_address (context) as evidenced by the default values for all three options, which are the same.

See also **UDP-Based Topic Resolution Details**.

Scope:	context
Type:	**struct in_addr**
Default value:	224.9.10.11
When to Set:	Can only be set during object initialization.

11.1.2 resolver_multicast_incoming_address (context)

Incoming multicast address (or domain name of the multicast address) used for finer control of Topic Resolution.

This value may be set to 0.0.0.0 (INADDR_ANY), to switch off listening to topic resolution messages. This means that queries from receivers or advertisements from sources will not be handled.

See also resolver_multicast_outgoing_address (context).

Scope:	context
Type:	**struct in_addr**
Default value:	224.9.10.11
When to Set:	Can only be set during object initialization.

11.1.3 resolver_multicast_incoming_port (context)

Incoming multicast port used for finer control of Topic Resolution.

See also resolver_multicast_outgoing_port (context).

See Port Assignments for more information about configuring ports.

Scope:	context
Type:	**lbm_uint16_t**
Default value:	12965
Byte order:	Network
When to Set:	Can only be set during object initialization.

11.1.4 resolver_multicast_interface (context)

Specifies which network interface UM sends/receives all multicast traffic (Topic Resolution, LBT-RM, Multicast Immediate Messaging).

You can specify full IP address of interface, or just network part (see Specifying Interfaces for details).

Default is set to default_interface (context), if specified. Otherwise, it is set to the default multicast interface as determined by UM (the first multicast-capable, non-loopback interface).

Note: if specifying an interface name in an XML-format file, see Interface Device Names and XML.

Scope:	context
Type:	**lbm_ipv4_address_mask_t**
Default value:	0.0.0.0
When to Set:	Can only be set during object initialization.

11.1.5 resolver_multicast_outgoing_address (context)

Outgoing multicast address (or domain name of the multicast address) used for finer control of Topic Resolution.

See also resolver_multicast_incoming_address (context).

Scope:	context
Type:	**struct in_addr**
Default value:	224.9.10.11
When to Set:	Can only be set during object initialization.

11.1.6 resolver_multicast_outgoing_port (context)

Outgoing multicast port used for finer control of Topic Resolution.

See also resolver_multicast_incoming_port (context).

See Port Assignments for more information about configuring ports.

Scope:	context
Type:	**lbm_uint16_t**
Default value:	12965
Byte order:	Network
When to Set:	Can only be set during object initialization.

11.1.7 resolver_multicast_port (context)

Multicast port used for Topic Resolution.

This option automatically sets the values for resolver_multicast_incoming_port (context) and resolver_↩ multicast_outgoing_port (context) as evidenced by the default values for all three options, which are the same.

See Port Assignments for more information about configuring ports.

Scope:	context
Type:	**lbm_uint16_t**

Default value:	12965
Byte order:	Network
When to Set:	Can only be set during object initialization.

11.1.8 resolver_multicast_receiver_socket_buffer (context)

Value used to set SO_RCVBUF value of the resolver receivers.

In some cases the OS will not allow all of this value to be used. A value of 0 instructs UM to use the default OS values. See Socket Buffer Sizes for platform-dependent information. See also our white paper Topics in High Performance Messaging for background and guidelines on UDP buffer sizing.

Scope:	context
Type:	**lbm_ulong_t**
Units:	bytes
Default value:	8388608 (8MB)
When to Set:	Can only be set during object initialization.

11.1.9 resolver_multicast_ttl (context)

The IP TTL (hop count) to use for a Topic Resolution packet.

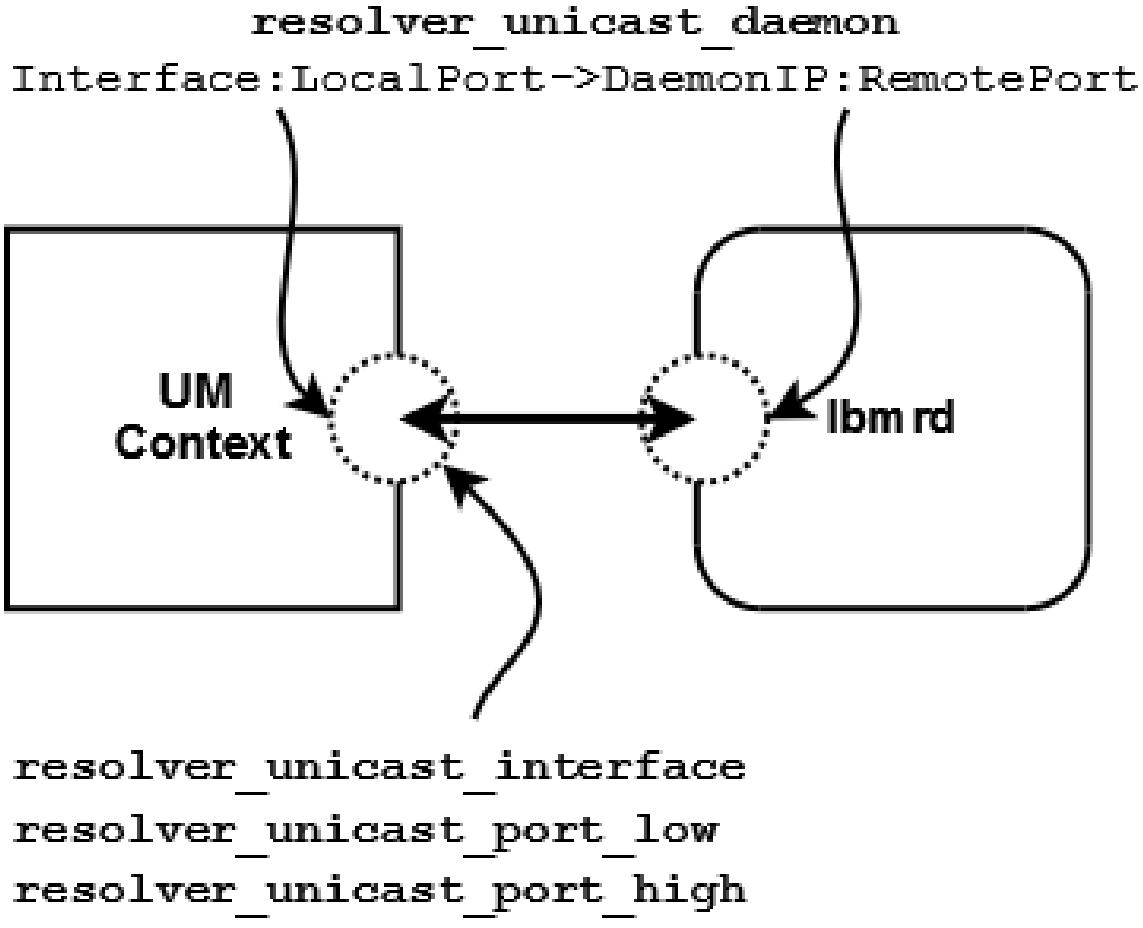
A value of 1 confines the packet to the local network (but may also cause high CPU usage on some routers). Also controls TTL on LBT-RM packets.

Scope:	context
Type:	**lbm_uint8_t**
Default value:	16
When to Set:	Can only be set during object initialization.

Chapter 12

Unicast Resolver Network Options

The image below shows a simplified relationship between the primary unicast resolver network options.



If using multiple lbmrd instances with a single context, you can configure **resolver_unicast_interface** and **resolver_unicast_port_low/high** and omit the **Interface:LocalPort** section of **resolver_unicast_daemon**.

See also **Unicast Topic Resolution** for general information on Unicast Topic Resolution.

12.1 Reference

12.1.1 resolver_unicast_daemon (context)

Enable Unicast UDP-based TR and add one or more unicast resolver daemon (lbmrd) specifications to the current lbmrd list. Unlike most other UM settings, every time this setting is called, it adds one or more daemon specifications to the list, and does NOT overwrite previous specifications.

Setting this option Disables Multicast UDP-based TR, but does not affect whether TCP-based TR is enabled or disabled. See **UDP-Based Topic Resolution Details**.

For the configuration file as well as string API method of setting this option, the string value consists of one or more lbmrd specifications separated by commas or semicolons, formatted as follows:

```
[Iface[:Src_Port]->]IP:Dest_Port[,...]
```

- **`Iface`** is the interface to use (previously set via resolver_unicast_interface (context)).
- **`Src_Port`** is the source port to use (previously resolver_unicast_port (context)). Normally only specified if firewalls require specific source ports be used.
- **`IP`** is the resolver daemon's IP address (previously resolver_unicast_address (context)).
- **`Dest_Port`** is the resolver daemon's UDP port (previously resolver_unicast_destination_port (context)).

You can omit either the **`Src_Port`** or both the **`Iface`** and **`Src_Port`**, in which case the default settings resolver_unicast_interface (context) and resolver_unicast_port (context) are used.

Because each entry adds a new daemon specification and does not overwrite previous values, a special construct must be used to clear a previously-specified list. An entry with the IP address of 0.0.0.0 and port of 0 removes all previous daemon specifications. This can be useful if multiple configuration files are used, and a later file should override the daemon list from an earlier file.

Possible formats of each entry are as follows:

```
Interface:LocalPort->DaemonIP:RemotePort
Interface->DaemonIP:RemotePort
DaemonIP:RemotePort
```

You can specify **`Interface`** in any of the ways described in Specifying Interfaces.

When the binary form of option setting is used, UM does NOT expect an array of structures. Instead, only one lbmrd specification can be supplied for each call to **lbm_context_attr_setopt()**. However, when the binary form of option retrieval **lbm_context_attr_getopt()** is used, the list of lbmrds is returned as an array, and the **`optlen`** parameter should be set as:

```
optlen = (max_num_lbmrds * sizeof(lbm_ucast_resolver_entry_t));
```

Scope:	context
Type:	**lbm_ucast_resolver_entry_t**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMS 5.0

12.1.2 resolver_unicast_interface (context)

Specifies the network interface over which UM receives unicast Topic Resolution messages.

You can specify full IP address of interface, or just network part (see Specifying Interfaces for details).

Default is set to default_interface (context), if specified. Otherwise, it is set to INADDR_ANY, meaning that it will accept unicast Topic Resolution messages on any interface.

Note: if specifying an interface name in an XML-format file, see Interface Device Names and XML.

Scope:	context
Type:	**lbm_ipv4_address_mask_t**
Default value:	0.0.0.0 (INADDR_ANY)
When to Set:	Can only be set during object initialization.

12.1.3 resolver_unicast_port_high (context)

The highest local UDP port in a range of ports used for unicast topic resolution messages.

The UM resolution daemon (lbmrd) sends unicast topic resolution messages to the UDP port range defined by this option and resolver_unicast_port_low (context).

See Port Assignments for more information about configuring ports.

Scope:	context
Type:	**lbm_uint16_t**
Default value:	14406
Byte order:	Host
When to Set:	Can only be set during object initialization.

12.1.4 resolver_unicast_port_low (context)

The lowest local UDP port in a range of ports used for unicast topic resolution messages.

The UM resolution daemon (lbmrd) sends unicast topic resolution messages to the UDP port range defined by this option and resolver_unicast_port_high (context).

See Port Assignments for more information about configuring ports.

Scope:	context
Type:	**lbm_uint16_t**
Default value:	14402
Byte order:	Host
When to Set:	Can only be set during object initialization.

12.1.5 resolver_unicast_receiver_socket_buffer (context)

Value used to set SO_RCVBUF value of the UDP receivers for unicast topic resolution messages.

In some cases the OS will not allow all of this value to be used. A value of 0 instructs UM to use the default OS values. See ref socketbuffersizes for platform-dependent information.

Scope:	context
Type:	**lbm_ulong_t**
Units:	bytes

Default value:	8388608 (8MB)
When to Set:	Can only be set during object initialization.

Chapter 13

TCP-Based Resolver Operation Options

See **TCP-Based Topic Resolution Details** for general information on TCP-based Topic Resolution.

13.1 Reference

13.1.1 resolver_service (context)

Enable TCP-based TR and add one or more (up to 5) Stateful Resolver Service (SRS) specifications to the current SRS list.

The SRS is used to provide TCP-based Topic Resolution services. For general information on Topic Resolution, see **Topic Resolution Description**. For details on TCP-based TR, see **TCP-Based Topic Resolution Details**.

Unlike most other UM settings, every time this setting is called, it adds one or more service specifications to the list, and does NOT overwrite previous specifications. Multiple SRS instances are recommended for fault tolerance. See **TCP-Based TR and Fault Tolerance**.

Setting this option does not affect whether UDP-based TR is enabled or disabled. It is appropriate in many use cases to have both TCP and UDP TR enabled. For example, see **TCP-Based TR Version Interoperability**.

Warning

Do not turn off the resolver cache when TCP-based TR is used. See resolver_cache (context).

For the configuration file as well as string API method of setting this option, the string value consists of one or more (up to 5) SRS specifications separated by commas or semicolons, formatted as follows:

```
[Iface[:Src_Port]->]IP:Dest_Port[,...]
```

- **Iface** is the interface to use.
- **Src_Port** is the source port to use. Normally only specified if firewalls require specific source ports be used.
- **IP** is the SRS's IP address.
- **Dest_Port** is the SRS's TCP listening port.

You can omit either the **Src_Port** or both the **Iface** and **Src_Port**, in which case the interface defaults to default_interface (context), if specified, and the port defaults to 0, which allocates an ephemeral port.

Because each entry adds a new SRS specification and does not overwrite previous values, a special construct must be used to clear a previously-specified list. An entry with the IP address of 0.0.0.0 and port of 0 removes all previous SRS specifications. This can be useful if multiple configuration files are used, and a later file should override the SRS list from an earlier file.

Possible formats of each entry are as follows:

```
Interface:LocalPort->SrsIP:RemotePort
Interface->SrsIP:RemotePort
SrsIP:RemotePort
```

You can specify **Interface** in any of the ways described in Specifying Interfaces.

When the binary form of option setting is used, UM does NOT expect an array of structures. Instead, only one SRS specification can be supplied for each call to **lbm_context_attr_setopt()**. However, when the binary form of option retrieval **lbm_context_attr_getopt()** is used, the list of SRSes is returned as an array, and the **optlen** parameter should be set as:

```
optlen = (max_num_srs * sizeof(lbm_resolver_service_entry_t));
```

Scope:	context
Type:	**lbm_resolver_service_entry_t**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMQ 6.12

13.1.2 resolver_service_interest_mode (context)

Allows an SRS to perform interest-based filtering of source advertisements (SIRs).

With filter mode enabled, the SRS will only send SIRs for sources that the context is interested in (has a receiver for). This reduces the Topic Resolution traffic to the context.

With flood mode, the SRS will send SIRs for all topics to the context. This mode is normally not needed, except when the resolver_source_notification_function (context) or resolver_event_function features are used.

See **TCP-Based TR Interest** for more information.

Scope:	context
Type:	**lbm_uint8_t**
Default value:	1
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.13

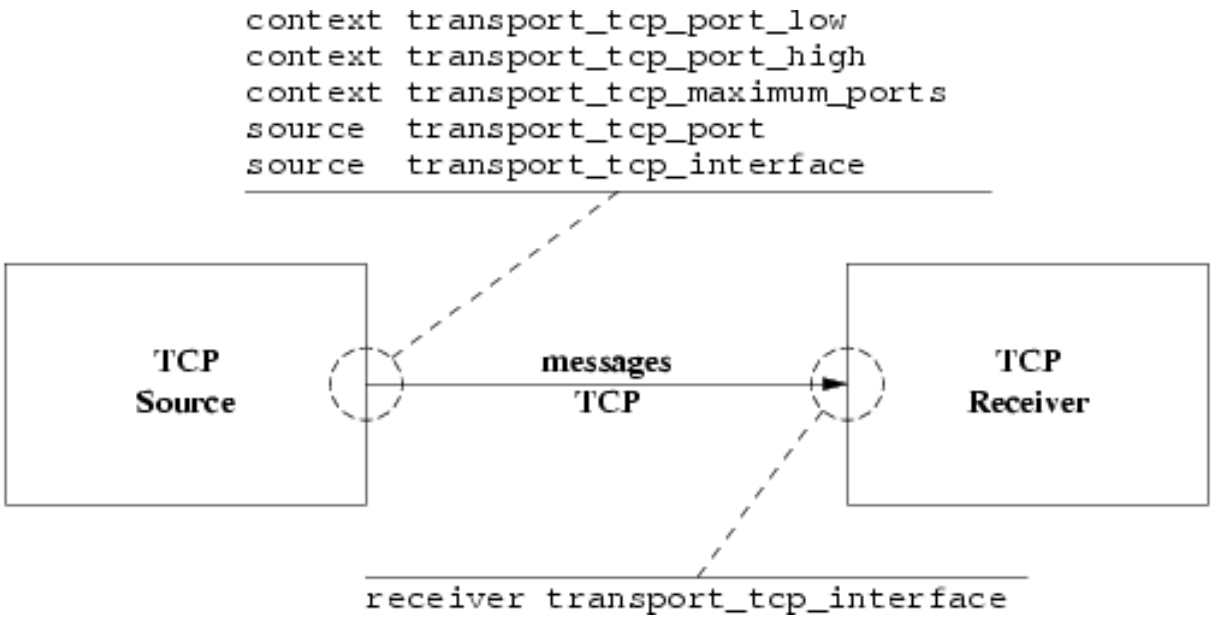
String value	Integer value	Description
"filter"	LBM_CTX_ATTR_INTEREST_MODE_FILTER	Filter Mode. The SRS sends SIRs for sources the context is interested in. **Default for all.**
"flood"	LBM_CTX_ATTR_INTEREST_MODE_FLOOD	Flood Mode. The SRS sends SIRs for all topics.

Chapter 14

Transport TCP Network Options

14.1 TCP Transport Session Management

The image below shows a simplified relationship between the primary TCP transport network options.



When a source is created, the application can explicitly map it to a transport session by setting the transport_tcp_port (source) option. If a previous source was created on the same context with the same port number, this new source will be mapped to the same transport session. However, two different contexts on the same host may not share the same port number. If a source is created with a port number that is already in use, UM will return an error.

Alternatively, if the application does not explicitly specify a source port, UM will implicitly assign the new source to a pool of transport sessions defined when the context was created. The pool is defined as a range of port numbers specified by the options transport_tcp_port_low (context) and transport_tcp_port_high (context). In addition, the option transport_tcp_maximum_ports (context) defines the number of transport sessions in the pool.

When a new source is created and the source port is not explicitly defined, UM will check to see how many transport sessions are currently active from the pool within the context. If it is less than transport_tcp_maximum_ports (context) then UM will attempt to use the next port in the range transport_tcp_port_low (context) to transport_tcp_port_high (context). If that port is already in use, UM continues along the range until it finds an unused port,

then it uses that port to create the transport session. However, if the context already has activated all transport sessions in the pool, then the new topic is mapped to one of the existing transport sessions, in round-robin fashion.

Notice that the default range of ports, 14371 to 14390, is 30 ports. But the default number of transport sessions in the pool is 10. This allows three contexts to be created on the same host and use the same configuration. If more than 3 contexts are intended to co-exist on the same host, the port range and number of transport session per context must be managed to give a unique port number to every transport session.

The option transport_tcp_interface (source) may be used on TCP sources to choose particular interface, overriding the default INADDR_ANY which accepts connections on all interfaces. Similarly, transport_tcp_interface (receiver) may be used on receivers to choose a particular interface.

14.2 Reference

14.2.1 transport_tcp_interface (receiver)

Specifies the network interface to which UM receivers bind before connecting to sources.

You can specify the full IP address of interface, or just the network part (see Specifying Interfaces for details).

Default is set to default_interface (context), if specified.

Note: if specifying an interface name in an XML-format file, see Interface Device Names and XML.

Scope:	receiver
Type:	**lbm_ipv4_address_mask_t**
Default value:	0.0.0.0 (INADDR_ANY)
When to Set:	Can only be set during object initialization.

14.2.2 transport_tcp_interface (source)

Specifies the network interface over which UM accepts connection requests (from topic receivers).

You can specify the full IP address of interface, or just the network part (see Specifying Interfaces for details).

Default is set to default_interface (context), if specified. Otherwise, it is set to INADDR_ANY, meaning that it will not bind to a specific interface. You can also set this option to 0.0.0.0/0 which produces the same result.

Be aware that this option is applied to the transport session when the first topic is created on that session. Thus, setting a different interface for a subsequent topic that maps onto the same transport session will have no effect.

Note: if specifying an interface name in an XML-format file, see Interface Device Names and XML.

Scope:	source
Type:	**lbm_ipv4_address_mask_t**
Default value:	0.0.0.0 (INADDR_ANY)
When to Set:	Can only be set during object initialization.

14.2.3 transport_tcp_maximum_ports (context)

Maximum size of TCP source transport session pool.

See TCP Transport Session Management for how TCP source transport sessions are managed.

Scope:	context
Type:	**lbm_uint16_t**
Units:	number of ports
Default value:	10
When to Set:	Can only be set during object initialization.

14.2.4 transport_tcp_port (source)

The TCP port to be used for the source transport session.

Setting this option to non-zero overrides the use of the pool of TCP source transport sessions.

The UM source listens on this port. Receivers that join the source's transport session connect to this port, and the source sends message data across that connection.

See TCP Transport Session Management for how TCP source transport sessions are managed.

See Port Assignments for more information about configuring ports.

Note that this port is only used by TCP sources. Receiver port numbers are taken from the host's Ephemeral Ports and are not configurable.

Scope:	source
Type:	**lbm_uint16_t**
Default value:	0 (pick open port)
Byte order:	Network
When to Set:	Can only be set during object initialization.

14.2.5 transport_tcp_port_high (context)

High TCP port number of range for pool of TCP source transport sessions.

When transport_tcp_port (source) is not specified, a newly-created transport session will use an unused port from this range. The UM source listens on this port. Receivers that join the source's transport session connect to this port, and the source sends message data across that connection.

See also transport_tcp_port_high (context).

See TCP Transport Session Management for how TCP source transport sessions are managed.

See Port Assignments for more information about configuring ports.

Note that this range of ports is only used by TCP sources. Receiver port numbers are taken from the host's Ephemeral Ports and are not configurable.

Scope:	context
Type:	**lbm_uint16_t**
Default value:	14390

Byte order:	Host
When to Set:	Can only be set during object initialization.

14.2.6 transport_tcp_port_low (context)

Low TCP port number of range for pool of TCP source transport sessions.

When transport_tcp_port (source) is not specified, a newly-created transport session will use an unused port from this range. The UM source listens on this port. Receivers that join the source's transport session connect to this port, and the source sends message data across that connection.

See also transport_tcp_port_high (context).

See TCP Transport Session Management for how TCP source transport sessions are managed.

See Port Assignments for more information about configuring ports.

Note that this range of ports is only used by TCP sources. Receiver port numbers are taken from the host's Ephemeral Ports and are not configurable.

Scope:	context
Type:	**lbm_uint16_t**
Default value:	14371
Byte order:	Host
When to Set:	Can only be set during object initialization.

Chapter 15

Transport TCP Operation Options

15.1 Reference

15.1.1 transport_session_maximum_buffer (source)

Value used to control the maximum amount of data buffered in UM for the transport session used for the topic.

For the normal multiple receiver behavior, this value represents the total buffered by all TCP receivers. For the bounded_latency and source_paced multiple receiver behavior, this value represents the individual receiver buffered amount. This option affects the transport session underlying the source rather than the source itself. The transport session uses the value from the first source created on the session and ignores subsequent sources' configuration.

Refer to **Source Object** for additional information.

Scope:	source
Type:	**lbm_ulong_t**
Units:	bytes
Default value:	65536
When to Set:	Can only be set during object initialization.

15.1.2 transport_tcp_activity_method (receiver)

The type of timeout method to use for TCP receivers to detect "half-open" TCP connections.

For TCP sessions only.

This defines how transport_tcp_activity_timeout (receiver) is interpreted. Note that transport_tcp_activity_timeout (receiver) defaults to 0 (disabled), meaning that half-open TCP connections may not be detected in a timely way.

Scope:	receiver
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.3.8/UME 2.0.6.

String value	Integer value	Description
"timer"	LBM_RCV_TOPIC_ATTR_TCP_ACTIVITY_TIMEOUT_TIMER	Timer method that requires new TCP session data to be sent to determine if the connection is alive. **Default for all.**
"SO_KEEPALIVE"	LBM_RCV_TOPIC_ATTR_TCP_ACTIVITY_TIMEOUT_SO_KEEPALIVE	Set SO_KEEPALIVE on the TCP connection which uses the TCP keepalive support in the operating system to determine if the connection is alive. When you use the SO_KEEPALIVE method, UM uses transport_tcp_activity_timeout (receiver) value to set the idle and probe times for SO_KEEPALIVE. The idle time is 90% of the timeout value at most. The probe time is 10% with 10 seconds as the minimum.

15.1.3 transport_tcp_activity_timeout (receiver)

A timeout used by a receiver to close a TCP transport session that has no activity.

For TCP sessions only.

Normally, when a source transport session is deleted by the sending application, the TCP connection is closed, which the receiver detects within a few milliseconds. However, there are unusual situations where a temporary

network outage prevents the receiver from detecting the closing of the connection, resulting in a "half-open" connection. This situation can prevent the receiver from detecting the closed connection for an unbounded time.

This timeout can be used to detect and close half-open connections.

This option has two different meanings, depending on the setting of transport_tcp_activity_method (receiver). See that option for details.

A value greater than zero turns the timer on.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	0
When to Set:	Can only be set during object initialization.

15.1.4 transport_tcp_activity_timeout (source)

This timeout option enables a source to use SO_KEEPALIVE to detect when a receiver does not cleanly disconnect or is no longer reachable from the source.

For TCP sessions only.

When the timeout expires, a DISCONNECT source event is delivered. This option affects only Linux or Windows platforms. Outstanding TCP retransmit timers must expire before this timer starts. With a default Linux or Windows system configuration, TCP retransmit timers may take minutes or even hours to expire. Therefore the total time taken to detect an unreachable receiver may be significantly higher than the value configured for this timeout. A value of zero (0) defers TCP timeout to OS settings.

Scope:	source
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	0
When to Set:	Can only be set during object initialization.

15.1.5 transport_tcp_coalesce_threshold (source)

UM passes implicitly batched messages to the Operating System sendmsg() as a set unless the size of the set exceeds the coalescing threshold at which point the messages are coalesced and passed to the O/S as one copy.

This option accommodates the different number of iovecs supported by different O/Ss. Tuning this option balances the efficiency of less iovecs handled by the OS vs. the expense of an additional copy operation of the messages before sending. The default values are also the maximum allowable values.

Scope:	source
Type:	**int**
Units:	number of individual messages
Default value:	1024 for Linux, Microsoft Windows, Darwin; 16 for Solaris, AIX, HPUX
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.6/UME 2.3.

15.1.6 transport_tcp_datagram_max_size (context)

The maximum datagram size that can be generated for a TCP transport session. While TCP does not use UDP datagrams, this option limits the size of the UM message which is given to the underlying transport type, including all UM headers and overhead. It does not include TCP, IP, or packet overhead added by the network stack. The default value is 65535, the minimum is 500 bytes, and the maximum is 65535.

See **Message Fragmentation and Reassembly** for more information.

Informatica does not recommend setting datagram max size options to the network MTU. See **Datagram Max Size and Network MTU**.

Warning

When the DRO is in use, it is recommended that all UM applications and components (including the DRO and Persistent Store) share the same maximum datagram size setting. See **Protocol Conversion**.

Users of **kernel-bypass drivers** should also see **Dynamic Fragmentation Reduction**.

Scope:	context
Type:	**lbm_uint_t**
Units:	bytes
Default value:	65535
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1

15.1.7 transport_tcp_dro_loss_recovery_timeout (receiver)

For TCP transport sessions originating from a DRO endpoint portal, delay declaring as unrecoverable a lost message.

Message streams traversing a DRO can have the message order changed. If the DRO's outgoing transport session uses the TCP protocol, these out-of-order messages will normally trigger immediate unrecoverable loss. This timeout allows an opportunity for the messages to be re-ordered properly.

The value 0 disables this delay (i.e. receivers immediately declare unrecoverable loss).

See **DRO Reliable Loss** for more information.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	0
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 6.12

15.1.8 transport_tcp_exclusiveaddr (source)

Indicate whether the TCP session should set SO_EXCLUSIVEADDRUSE or not before it binds.

Applicable only to Windows. The default setting in Windows allows multiple binds to the same port. By de-

fault, UM will set SO_EXCLUSIVEADDRUSE to minimize port sharing. Refer to Microsoft's web site for more information on SO_EXCLUSIVEADDRUSE.

Scope:	source
Type:	**int**
When to Set:	Can only be set during object initialization.

Value	Description
1	Set SO_EXCLUSIVEADDRUSE. **Default for Windows.**
0	Do not set SO_EXCLUSIVEADDRUSE.

15.1.9 transport_tcp_listen_backlog (source)

The backlog used in the TCP listen() call to set the queue length for incoming connections.

If 20 or more receivers will be joining this source, it may be beneficial to increase this number.

Scope:	source
Type:	**int**
Units:	number of queued connections
Default value:	5
When to Set:	Can only be set during object initialization.

15.1.10 transport_tcp_multiple_receiver_behavior (source)

This option determines the flow control behavior of a TCP source that is sending to multiple receivers.

In particular, it addresses the scenario where some receivers are consuming messages slower than other receivers (or not at all). In this scenario, pending messages will be buffered by the source up to the configured

limit specified by the transport_session_maximum_buffer (source) option. Once that limit is reached, the source can either wait for slow receivers (resulting in a blocked source) or discard messages from the buffer (resulting in gaps and potentially unrecoverable loss of messages for slow receivers).

This option affects the transport session underlying the source rather than the source itself. The transport session uses the value from the first source created on the session and ignores the value set for subsequent sources.

Refer to **Source Object** for additional information.

Scope:	source
Type:	**int**
When to Set:	Can only be set during object initialization.

String value	Integer value	Description
"normal"	LBM_SRC_TOPIC_ATTR_TCP_MULTI_RECV_NORMAL	The application will wait for the slowest receiver rather than discarding recent messages. This slows down all sources on the TCP session. **Default for all.**
"source_paced"	LBM_SRC_TOPIC_ATTR_TCP_MULTI_RECV_SOURCE_PACED	The application sends as fast as it can even if recent messages headed for any or all receivers must be discarded. Note that for applications requiring source-paced behavior, LBT-RU and LBT-RM are more efficient than source-paced TCP, so this setting is primarily for scenarios where UDP-based protocols are not viable.
"bounded_latency"	LBM_SRC_TOPIC_ATTR_TCP_MULTI_RECV_BOUNDED_LATENCY	The application sends as fast as the fastest receiver can consume data even if recent data headed for slower receivers must be discarded. Deprecated since UM 6.9.

15.1.11 transport_tcp_multiple_receiver_send_order (source)

In the case of multiple receivers, this option determines whether datagrams are sent to each receiver in the established order of receivers, or if receivers are selected in random order for each datagram transmission.

Using random ordering can avoid giving one receiver a consistent latency advantage, at the expense of slightly higher per-message processing (calculating the random number).

Scope:	source
Type:	**int**
When to Set:	Can only be set during object initialization.

String value	Integer value	Description
"serial"	LBM_SRC_TOPIC_ATTR_TCP_MULTI_↩ RECV_SEND_ORDER_SERIAL	Select receivers to receive a datagram based on current established order. **Default for all.**
"random"	LBM_SRC_TOPIC_ATTR_TCP_MULTI_↩ RECV_SEND_ORDER_RANDOM	For each datagram sent, select receivers in random order, for the sake of "fairness". Note that this option adds a small amount of CPU overhead.

15.1.12 transport_tcp_nodelay (source)

Controls whether the context sets TCP_NODELAY before it binds to the source port.

Setting TCP_NODELAY disables Nagle's algorithm, which somewhat decreases the efficiency and throughput of TCP, but decreases the latency of individual messages.

Scope:	source
Type:	**int**
When to Set:	Can only be set during object initialization.

Value	Description
1	TCP transport sockets should set TCP_NODELAY (disable Nagle). **Default for all.**
0	TCP transport sockets should not set TCP_NODELAY (leave Nagle enabled).

15.1.13 transport_tcp_receiver_socket_buffer (context)

Value used to set SO_RCVBUF value of the TCP receivers for topics. In some cases the OS will not allow all of this value to be used.

A value of 0 instructs UM to use the default OS values. See Socket Buffer Sizes for platform-dependent information.

Scope:	context
Type:	**lbm_ulong_t**
Units:	bytes
Default value:	0 (use TCP autotuning)
When to Set:	Can only be set during object initialization.

15.1.14 transport_tcp_reuseaddr (source)

Whether the TCP session should set SO_REUSEADDR or not before it binds.

Warning

This option is not recommended for Microsoft Windows users because the SO_REUSEADDR socket option in Windows allows a socket to forcibly bind to a port in use by another socket. Multiple sockets using the same port results in indeterminate behavior.

Scope:	source
Type:	**int**
When to Set:	Can only be set during object initialization.

Value	Description
1	Set SO_REUSEADDR.
0	Do not set SO_REUSEADDR. **Default for all.**

15.1.15 transport_tcp_sender_socket_buffer (source)

Value used to set the SO_SNDBUF value of the TCP session.

In some cases the OS will not allow all of this value to be used. A value of 0 instructs UM to use the OS defaults. See Socket Buffer Sizes for platform-dependent information.

Scope:	source
Type:	**lbm_ulong_t**
Units:	bytes
Default value:	0 (use TCP autotuning)
When to Set:	Can only be set during object initialization.

15.1.16 transport_tcp_use_session_id (source)

Control whether a session ID is used for TCP Transport sessions.

This option should be set to 0 if a version 6.0 (and beyond) TCP source must interoperate with a version pre-6.0 receiver.

Scope:	source
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.0

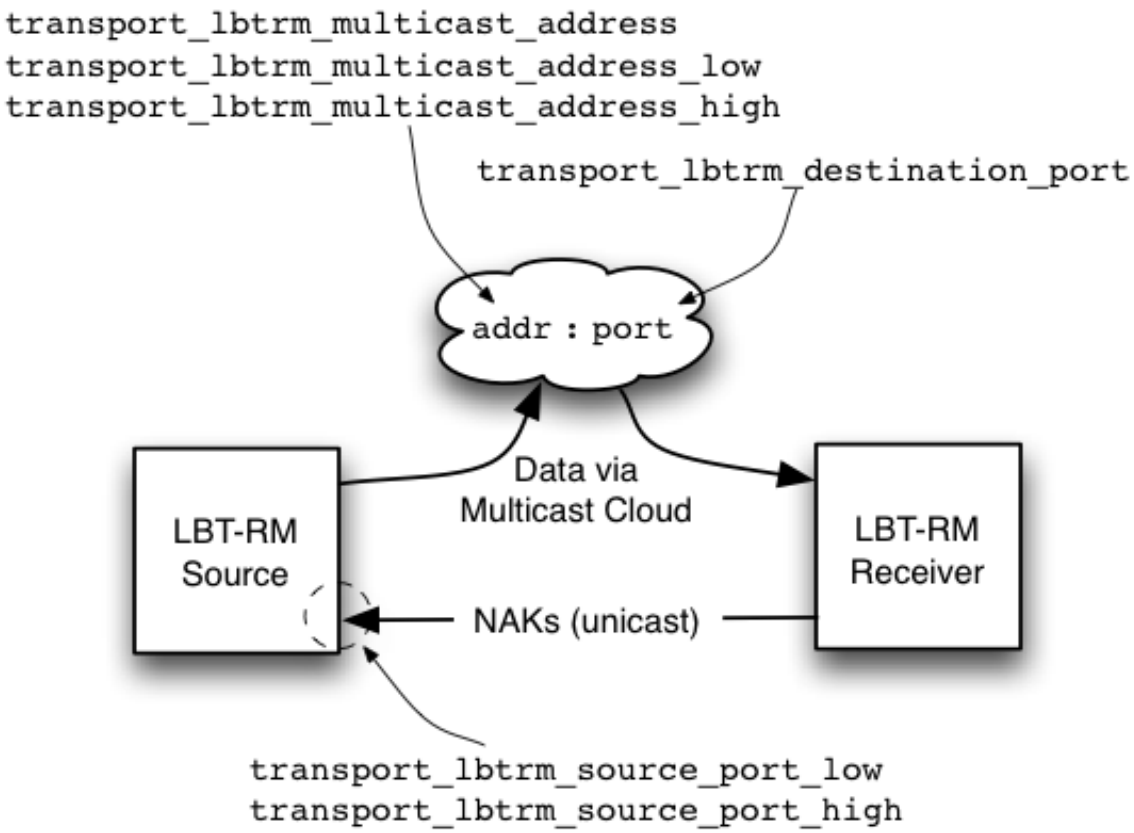
Value	Description
1	Indicates the application desires TCP to use a session ID. **Default for all.**
0	Indicates the application does not desire TCP to use a session ID. For use when version pre-6.0 receivers must be used with TCP sources that are version 6.0 and beyond.

Chapter 16

Transport LBT-RM Network Options

16.1 LBT-RM Transport Session Management

The image below shows a simplified relationship between the primary LBT-RM transport network options.



Note

for a multi-homed LBT-RM source, the interface LBT-RM multicast resolver interface specified with resolver_multicast_interface (context) will be used as the source for LBT-RM.

When a source is created, the application can explicitly map it to a transport session by setting the transport_lbtrm_multicast_address (source) and transport_lbtrm_destination_port (source) options. If a previous source was created on the same context with the same group/port pair, this new source will be mapped to the same transport session. Note that two different contexts on the same host may share the same group/port pair, and the resulting transport sessions will be separate and independent.

Alternatively, if the application does not explicitly specify a multicast group and destination port, UM will implicitly assign the new source to a pool of transport sessions defined when the context was created. The pool is defined as a range of multicast groups specified by the options transport_lbtrm_multicast_address_low (context) and transport_lbtrm_multicast_address_high (context). The number of transport sessions in the pool is the range of the two multicast group IP addresses, as represented by a 32-bit number. For example, the default settings 224.10.10.10 and 224.10.10.14 are represented by the numbers 0xE00A0A0A and 0xE00A0A0E. This represents 5 different groups, so the pool contains 5 transport sessions (all with the same destination port).

When a new source is created and the multicast group is not explicitly defined, UM will check to see how many transport sessions are currently active from the pool within the context. If it is less than the number in the pool, then UM will activate the next transport session in the range. However, if the context already has activated all transport sessions in the pool, then the new topic is mapped to one of the existing transport sessions, in round-robin fashion.

16.2 Reference

16.2.1 transport_lbtrm_destination_port (source)

The UDP destination port used for this Topic when the transport is LBT-RM.

See LBT-RM Transport Session Management for how LBT-RM source transport sessions are managed.

See Port Assignments for more information about configuring ports.

Scope:	source
Type:	**lbm_uint16_t**
Default value:	14400
Byte order:	Network
When to Set:	Can only be set during object initialization.

16.2.2 transport_lbtrm_multicast_address (source)

The preferred multicast address (or domain name of the multicast address) for this Topic when the transport is LBT-RM.

If 0.0.0.0 (INADDR_ANY), the default, the context will use a round-robin method to select an address in the configured multicast multicast address range: transport_lbtrm_multicast_address_high (context) - transport_lbtrm_multicast_address_low (context).

See LBT-RM Transport Session Management for how LBT-RM source transport sessions are managed.

See Port Assignments for more information about configuring ports.

Scope:	source
Type:	**struct in_addr**
Default value:	0.0.0.0 (INADDR_ANY)
When to Set:	Can only be set during object initialization.

16.2.3 transport_lbtrm_multicast_address_high (context)

Multicast address (or domain name of the multicast address) used as the highest value to assign to LBT-RM sessions.

See LBT-RM Transport Session Management for how LBT-RM source transport sessions are managed.

Scope:	context
Type:	**struct in_addr**
Default value:	224.10.10.14
When to Set:	Can only be set during object initialization.

16.2.4 transport_lbtrm_multicast_address_low (context)

Multicast address (or domain name of the multicast address) used as the lowest value to assign to LBT-RM sessions.

See LBT-RM Transport Session Management for how LBT-RM source transport sessions are managed.

Scope:	context
Type:	**struct in_addr**
Default value:	224.10.10.10
When to Set:	Can only be set during object initialization.

16.2.5 transport_lbtrm_source_port_high (context)

Highest port number value used for LBT-RM source session's unicast NAK processing. Receivers send NAKs to this port for to request retransmission. Each LBT-RM session must use a unique port value. Note that this does not control the UDP source port on the outbound LBT-RM stream.

See Port Assignments for more information about configuring ports.

Scope:	context
Type:	**lbm_uint16_t**
Default value:	14399
Byte order:	Host
When to Set:	Can only be set during object initialization.

16.2.6 transport_lbtrm_source_port_low (context)

Lowest port number value used for LBT-RM source session's unicast NAK processing. Receivers send NAKs to this port for to request retransmission. Each LBT-RM session must use a unique port value. Note that this does not control the UDP source port on the outbound LBT-RM stream.

See Port Assignments for more information about configuring ports.

Scope:	context
Type:	**lbm_uint16_t**
Default value:	14390
Byte order:	Host
When to Set:	Can only be set during object initialization.

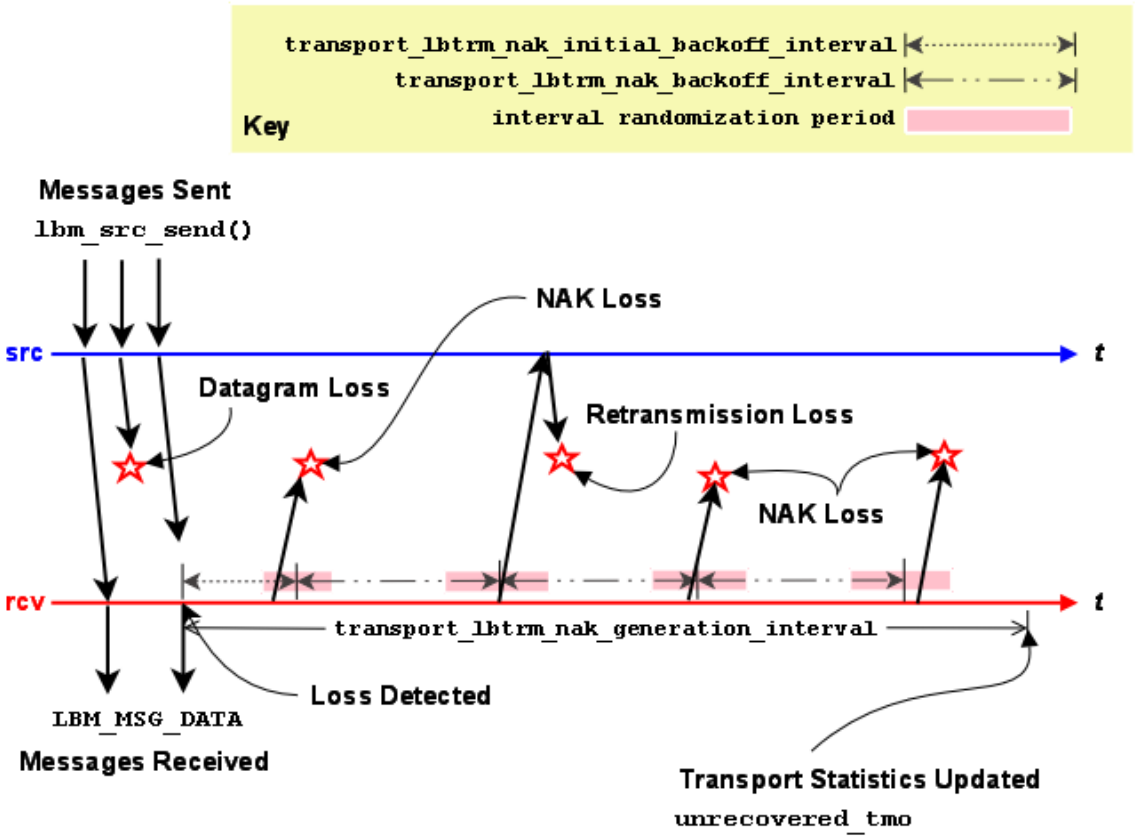
Chapter 17

Transport LBT-RM Reliability Options

17.1 LBT-RM Datagram

Loss Resulting in Unrecovered Message Loss

An LBT-RM receiver will attempt to recover lost datagrams. The options transport_lbtrm_nak_backoff_interval (receiver) and transport_lbtrm_nak_generation_interval (receiver) control the timing of the recovery effort. Timers for both start when loss is detected. The following timeline illustrates a case where a receiver is notified of unrecoverable message loss following repeated datagram loss.



Note

the actual length of the interval randomization periods are between 1/2 and 3/2 of the configured interval value. In the diagram above, time periods are not drawn to scale to simplify the diagram.

Set transport_lbtrm_nak_backoff_interval (receiver) to the NAK service time that could be reasonably expected from the receiver's location in the network plus some cushion for network congestion. Set transport_lbtrm_nak_↩ generation_interval (receiver) to the latency budget established for the transport layer. See our whitepaper Topics in High Performance Messaging for background on latency budgets. See also the KB article Reducing Loss Recovery Latencies for more advice on tuning.

Note

these parameters relate to loss at the transport session (datagram) level, not the topic level. See Delivery Control Options for information on how applications are informed of topic-level unrecoverable loss.

17.2 LBT-RM Source Ignoring NAKs for Efficiency

Bandwidth efficiency of an LBT-RM source may be improved by avoiding useless retransmissions. Consider the case of an LBT-RM source that has received a NAK for a datagram that it has just retransmitted. If the NAK and the retransmission crossed on the network, it is likely that the receiver generating the NAK will receive the retransmission just sent. If so, there's no need for the source to send another retransmission, so the NAK can be safely ignored.

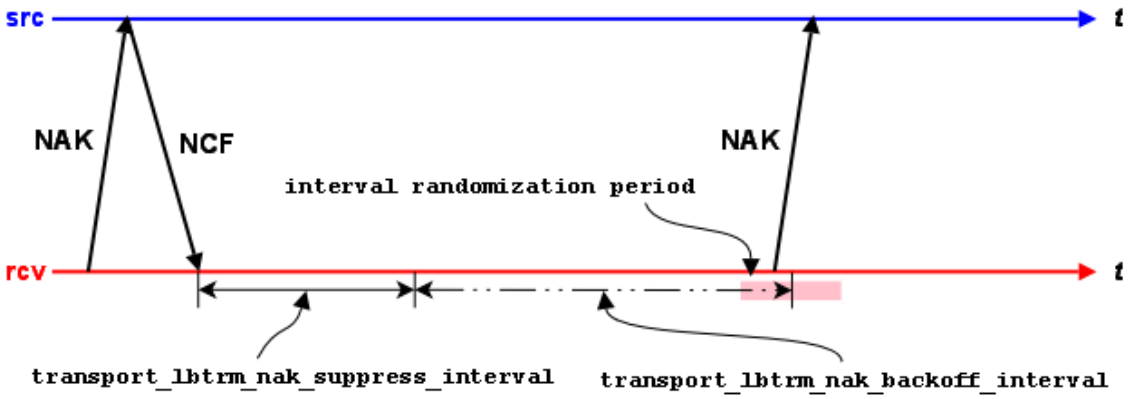
NAKs for a given datagram are ignored for transport_lbtrm_ignore_interval (source) following the retransmission of that datagram. A NAK will be serviced as normal following the passage of the interval.

When ignoring a NAK, the source can send an NCF (NAK ConFirmation) instead of a retransmission. See **NAK Suppression** for more information.

17.3 LBT-RM Receiver Suppressing NAK Generation

LBT-RM sources want receivers to be notified that their NAKs have been heard. Prompt notification via a retransmission or NCF can suppress useless NAK generation. There are a variety of circumstances where the source does not send a retransmission in response to a receiver's NAK. For example, NAKs received during the ignore interval do not generate retransmissions. Another example would be if previous retransmissions have used up all the retransmission bandwidth for the current rate limiter interval.

The image below illustrates a receiver's reaction to an NCF.



Following the receipt of an NCF, a receiver suppresses all NAK generation for that sequence number until transport_lbtrm_nak_suppress_interval (receiver) passes. NAK generation resumes with the usual transport_lbtrm_nak_backoff_interval (receiver) if repair was not received during the suppression interval.

Note

the actual length of the interval randomization period is between 1/2 and 3/2 of the configured interval value. In the diagram above, time periods are not drawn to scale to simplify the diagram.

17.4 Reference

17.4.1 transport_lbtrm_ignore_interval (source)

The interval to ignore NAKs after a retransmission is sent.

This option affects the transport session underlying the source rather than the source itself. The transport session uses the value from the first source created on the session and ignores subsequent sources' configuration.

Refer to **Source Object** for additional information.

Scope:	source
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	500 (0.5 seconds)
When to Set:	Can only be set during object initialization.

17.4.2 transport_lbtrm_nak_backoff_interval (receiver)

The maximum interval between transmissions of LBT-RM NAKs for a given sequence number, after the first NAK.

When an LBT-RM receiver detects a sequence number gap, it delays an initial amount before sending its first NAK (controlled by transport_lbtrm_nak_initial_backoff_interval (receiver)), and then delays an a separately configurable time between sending subsequent NAKs for the same sequence number. This configuration option controls those subsequent delays.

The actual time the receiver will wait to NAK again is random. The algorithm used to determine the time range is (1/2 ∗ backoff_interval - 3/2 ∗ backoff_interval). This will result in a delay longer or shorter than the specified value.

This option affects the transport session underlying the receiver rather than the receiver itself. The transport session uses the value from the first receiver created on the session and ignores subsequent receivers' configuration.

See also transport_lbtrm_nak_initial_backoff_interval (receiver).

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	200 (0.2 seconds)
When to Set:	Can only be set during object initialization.

17.4.3 transport_lbtrm_nak_generation_interval (receiver)

The maximum time that a piece of data may be outstanding before the data is unrecoverably lost.

For LBT-RM transport sessions only. Although the minimum valid value is 5 milliseconds, larger values are advisable. This option affects the transport session underlying the receiver rather than the receiver itself. The transport session uses the value from the first receiver created on the session and ignores subsequent receivers' configuration.

Refer to **Receiver Object** and Interrelated Configuration Options for additional information.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	10000 (10 seconds)
When to Set:	Can only be set during object initialization.

17.4.4 transport_lbtrm_nak_initial_backoff_interval (receiver)

The interval between loss detection and transmission of the first LBT-RM NAK.

When an LBT-RM receiver detects a sequence number gap, it delays an initial amount before sending its first NAK controlled by this option, and then delays an a separately configurable time between sending subsequent NAKs for the same sequence number, controlled by transport_lbtrm_nak_backoff_interval (receiver).

The actual time the receiver will wait to NAK is random. The algorithm used to determine the time range is (1/2 $*$ initial_backoff_interval - 3/2 $*$ initial_backoff_interval). This will result in a delay longer or shorter than the specified value. A value of 0 indicates that the receiver should immediately send a NAK. Note that this is rarely a good idea; see **UM Recovery of Lost Packets**.

This option affects the transport session underlying the receiver rather than the receiver itself. The transport session uses the value from the first receiver created on the session and ignores subsequent receivers' configuration.

See also transport_lbtrm_nak_backoff_interval (receiver).

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	50 (0.05 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.4/UME 2.1.

17.4.5 transport_lbtrm_nak_suppress_interval (receiver)

The time that an LBT-RM receiver will suppress sending a NAK for a missing datagram after an NCF is received from the source.

The source sends an NCF in response to a NAK which the source temporarily cannot retransmit. For example, if the source gets a NAK for a sequence number for which it has recently sent a retransmission, it will send an NCF with reason code "ignored". The receiver responds by suppressing NAKs for that sequence number for the interval configured by this option. See **NAK Suppression** for more information about NCFs.

For LBT-RM transport sessions only. This option affects the transport session underlying the receiver rather than the receiver itself. The transport session uses the value from the first receiver created on the session and ignores subsequent receivers' configuration.

Refer to **Receiver Object** for additional information.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	1000 (1 second)
When to Set:	Can only be set during object initialization.

17.4.6 transport_lbtrm_receiver_socket_buffer (context)

Value used to set SO_RCVBUF value of the LBT-RM receiver multicast socket.

In some cases the OS will not allow all of this value to be used. See Socket Buffer Sizes for platform-dependent information. See also our white paper Topics in High Performance Messaging for background and guidelines on UDP buffer sizing.

Scope:	context
Type:	**lbm_ulong_t**
Units:	bytes
Default value:	8388608 (8MB)
When to Set:	Can only be set during object initialization.

17.4.7 transport_lbtrm_send_naks (receiver)

This flag indicates whether LBT-RM should send negative acknowledgements (NAKs) for missing packets or not.

For LBT-RM transport sessions only. This option affects the transport session underlying the receiver rather than the receiver itself. The transport session uses the value from the first receiver created on the session and ignores subsequent receivers' configuration.

Refer to **Receiver Object** for additional information.

Scope:	receiver
Type:	**int**
When to Set:	Can only be set during object initialization.

Value	Description
1	NAKs are sent for missing packets to request retransmission. **Default for all.**
0	Do not send NAKs for missing packets.

17.4.8 transport_lbtrm_source_socket_buffer (context)

Value used to set SO_SNDBUF value of the LBT-RM send multicast socket.

In some cases the OS will not allow all of this value to be used. See Socket Buffer Sizes for platform-dependent information. A value of 0 instructs UM to use the OS default.

Scope:	context
Type:	**lbm_ulong_t**
Units:	bytes
Default value:	1048576 (1MB)
When to Set:	Can only be set during object initialization.

17.4.9 transport_lbtrm_transmission_window_limit (source)

Caps the total amount of memory that a transmission window uses, which includes data and overhead.

For example, if the transport_lbtrm_transmission_window_size (source) is 24 MB (default) and the source sends (with flush flag set) 1.2 million messages with a 20-byte payload and 230-byte header, the actual amount of memory used can approximate 300 MB. The default value of 0 (zero) disables the transmission window size limit.

Scope:	source
Type:	**size_t**
Units:	bytes
Default value:	0 (disables limit)
When to Set:	Can only be set during object initialization.

17.4.10 transport_lbtrm_transmission_window_size (source)

The maximum amount of buffered payload data, excluding UM headers, that the LBT-RM source is allowed to retain for retransmissions.

The minimum valid value is 65,536 bytes. This option affects the transport session underlying the source rather than the source itself. The transport session uses the value from the first source created on the session and ignores subsequent sources' configuration.

Scope:	source
Type:	**size_t**
Units:	bytes
Default value:	25165824 (24 MB)
When to Set:	Can only be set during object initialization.

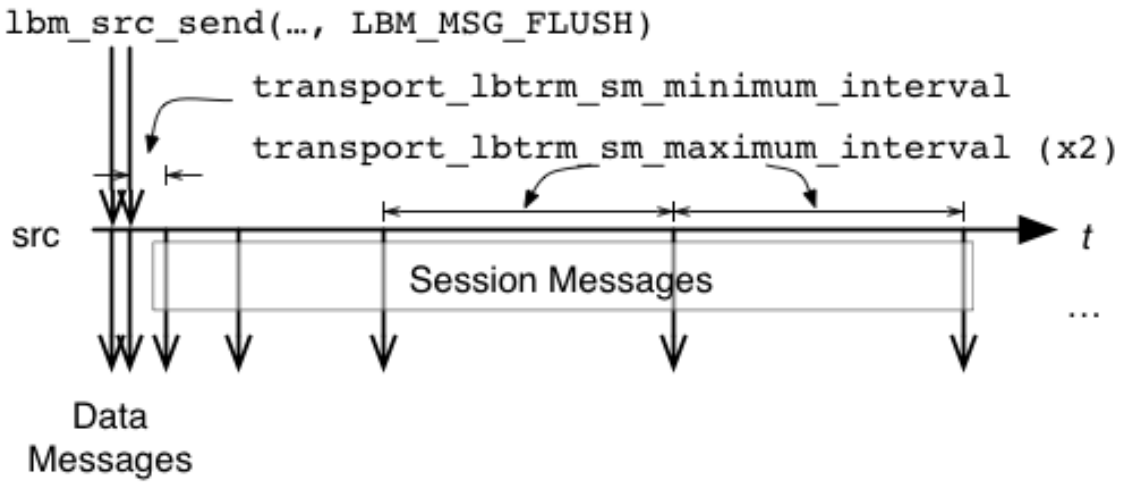
Chapter 18

Transport LBT-RM Operation Options

Reliable multicast protocols like LBT-RM rely on sequence numbers and the arrival of data after a loss as evidence that the loss happened. What would happen if the last packet sent by a source was lost? How would receivers learn of the loss if no further messages were sent?

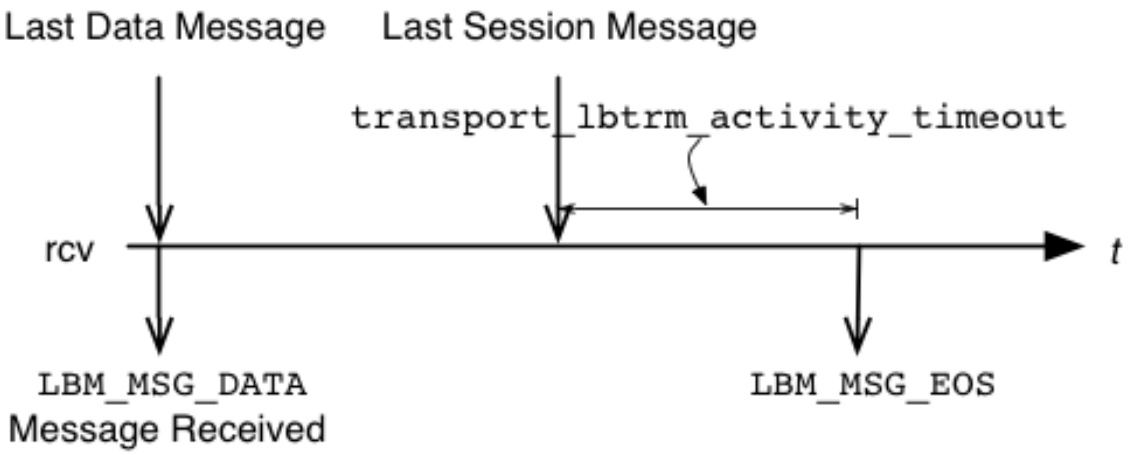
LBT-RM generates session messages when the sources on a transport session stop sending. These messages contain the expected last sequence number for the session so that receivers can detect loss even when sources aren't sending. Session messages also help to maintain state in multicast routers and switches that require regular traffic to prevent the reclamation of unused forwarding entries.

The image below illustrates the sending of session messages.



No session messages are generated as long as the interval between **lbm_src_send()** calls that generate writes to LBT-RM is less than transport_lbtrm_sm_minimum_interval (source) option. The interval between session messages starts at transport_lbtrm_sm_minimum_interval (source) and doubles till it reaches transport_lbtrm_sm_↩ maximum_interval (source) at which point the interval continues at that level.

The absence of activity on a transport session is the only indication receivers get that a source is gone or no longer available through any network path. LBT-RM receivers reset a session activity timer for each data message or session message that arrives. If the activity timer ever expires, all receivers on the transport session receive an LBM_MSG_EOS event. This is illustrated in the following timeline:



The activity timer is controlled with the transport_lbtrm_activity_timeout (receiver) option.

18.1 Reference

18.1.1 transport_lbtrm_activity_timeout (receiver)

The maximum time that an LBT-RM session may be quiescent before it is deleted and an EOS event is delivered for all topics using this transport session.

For LBT-RM transport sessions only. This option affects the transport session underlying the receiver rather than the receiver itself. The transport session uses the value from the first receiver created on the session and ignores subsequent receivers' configuration.

Refer to **Receiver Object** for additional information.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	60000 (60 seconds)
When to Set:	Can only be set during object initialization.

18.1.2 transport_lbtrm_coalesce_threshold (source)

UM passes implicitly batched messages to the Operating System sendmsg() as a set unless the size of the set exceeds the coalescing threshold at which point the messages are coalesced and passed to the O/S as one copy.

This option accommodates the different number of iovecs supported by different O/Ss. Tuning this option balances the efficiency of less iovecs handled by the OS vs. the expense of an additional copy operation of the messages before sending. The default value is also the maximum allowable value for Solaris, AIX and HPUX. For Linux and Microsoft Windows and Darwin, the maximum allowable value is 1023. These maximum allowable values are one less than what the O/S provides. This option affects the transport session underlying the source rather than the source itself. The transport session uses the value from the first source created on the session and ignores subsequent sources' configuration.

Refer to **Source Object** for additional information.

Scope:	source
Type:	**int**
Units:	number of individual messages
Default value:	15
When to Set:	Can only be set during object initialization.

18.1.3 transport_lbtrm_data_rate_limit (context)

Maximum aggregate transmission rate of all LBT-RM sessions' original data plus retransmissions for this particular context.

Refer to **Rate Controls** for additional information about the UM rate limiting algorithm.

Note: For backwards compatibility with earlier versions, the **lbm_context_attr_setopt()** function will accept both 32 and 64 bit values for this option. Note however that a 32-bit value can only specify a rate limit a little larger than 4 Gbps.

Scope:	context
Type:	**lbm_uint64_t**
Units:	bits per second
Default value:	10000000 (10 Mbps)
When to Set:	Can only be set during object initialization.

18.1.4 transport_lbtrm_datagram_max_size (context)

The maximum UDP datagram payload size that can be generated for a LBT-RM transport session. Note that this does not include UDP, IP, or packet overhead added by the network stack. The default value is 8192, the minimum is 500 bytes, and the maximum is 65535.

See **Message Fragmentation and Reassembly** for more information.

Informatica does not recommend setting datagram max size options to the network MTU. See **Datagram Max Size and Network MTU**.

Warning

When the DRO is in use, it is recommended that all UM applications and components (including the DRO and Persistent Store) share the same maximum datagram size setting. See **Protocol Conversion**.

Users of **kernel-bypass drivers** should also see **Dynamic Fragmentation Reduction**.

Scope:	context
Type:	**lbm_uint_t**
Units:	bytes
Default value:	8192
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1

18.1.5 transport_lbtrm_preactivity_timeout (receiver)

The time that a newly-joined LBT-RM transport session can have no activity before the receiver decides the transport session is dead.

This option typically does not need to be set for deployments using UM version 3.3 and beyond. If this option is

set to 0 (the default), then the activity timeout for a newly-joined transport session is the same as transport_lbtrm_activity_timeout (receiver).

The purpose of this option is for a receiver to allow an extended timeout for a newly-created source transport session to have no activity prior to the first application message (or TSNI) being sent.

This option is most useful when sending applications use UM versions prior to 3.3, which did not use Topic Sequence Number Information messages (TSNIs; see transport_topic_sequence_number_info_interval (source)). In these cases, the source does not start the transport session until the first application message is sent. If the sending application might delay sending its first message for more than transport_lbtrm_activity_timeout (receiver) (60 seconds by default), the receiver will decide that the transport session is dead and will disconnect. Assuming that the source is still actually alive, the receiver will subsequently re-join the session, which can lead to "flapping".

This flapping can be prevented by setting transport_lbtrm_preactivity_timeout to a value greater than the worst-case delay before the sending application sends its first message.

In UM version 3.3 and beyond, LBT-RM sources enable TSNIs by default, which ensures that some transport session activity will happen within 5 seconds, by default. Thus, there is no longer any need to set a different timeout for a newly-joined transport session. But note that it also extends the time required for a receiver to detect that a newly-joined source transport session is actually dead.

This option may still have some utility in UM version 3.3 and beyond if TSNIs need to be disabled.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	0 (zero)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.4.1/UME 2.1.1.

18.1.6 transport_lbtrm_rate_interval (context)

Period that LBT-RM rate limiter runs.

Reducing period reduces burst intensity, but also increases CPU load. Refer to **Rate Controls** for additional information about the UM rate limiting algorithm.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	10
When to Set:	Can only be set during object initialization.

String value	Integer value	Description
"10"	10	LBT-RM rate limiter runs every 10 milliseconds. **Default for all.**
"20"	20	LBT-RM rate limiter runs every 20 milliseconds.
"50"	50	LBT-RM rate limiter runs every 50 milliseconds.
"100"	100	LBT-RM rate limiter runs every 100 milliseconds.

18.1.7 transport_lbtrm_receiver_timestamp (context)

Controls whether high-resolution timestamps for received packets are delivered to the receiver callback.

For LBT-RM transport sessions only.

Refer to **High-resolution Timestamps** for additional information.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.9

Value	Description
1	Receive timestamps delivered to receive callback.
0	Receive timestamps not delivered. **Default for all.**

18.1.8 transport_lbtrm_recycle_receive_buffers (context)

Enables the use of buffer recycling for socket operations.

See **Receive Buffer Recycling** for more information, including restrictions on the use of this feature.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.12

Value	Description
1	Use buffer recycling.
0	Buffer recycling is not used. **Default for all.**

18.1.9 transport_lbtrm_retransmit_rate_limit (context)

Maximum aggregate transmission rate of all LBT-RM sessions' retransmissions for this particular context.

This should always be less than the value used for original data. Refer to **Rate Controls** for additional information about the UM rate limiting algorithm.

Note: For backwards compatibility with earlier versions, the **lbm_context_attr_setopt()** function will accept both 32 and 64 bit values for this option. Note however that a 32-bit value can only specify a rate limit a little larger than 4 Gbps.

Scope:	context
Type:	**lbm_uint64_t**
Units:	bits per second

Default value:	5000000 (5 Mbps)
When to Set:	Can only be set during object initialization.

18.1.10 transport_lbtrm_sm_maximum_interval (source)

The maximum interval between LBT-RM session messages.

In lieu of data being sent, LBT-RM sends session messages to inform receivers of sequence numbers and to let receivers know that the sender is still transmitting. This option affects the transport session underlying the source rather than the source itself. The transport session uses the value from the first source created on the session and ignores subsequent sources' configuration.

Refer to **Source Object** for additional information.

Scope:	source
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	10000 (10 seconds)
When to Set:	Can only be set during object initialization.

18.1.11 transport_lbtrm_sm_minimum_interval (source)

The minimum interval between LBT-RM session messages.

In lieu of data being sent, LBT-RM sends session messages to inform receivers of sequence numbers and to let receivers know that the sender is still transmitting. This option affects the transport session underlying the source rather than the source itself. The transport session uses the value from the first source created on the session and ignores subsequent sources' configuration.

Refer to **Source Object** for additional information.

Scope:	source
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	200 (0.2 seconds)
When to Set:	Can only be set during object initialization.

18.1.12 transport_lbtrm_source_timestamp (context)

Controls whether high-resolution timestamps for transmitted packets are delivered to the source event callback.

For LBT-RM transport sessions only. Refer to **High-resolution Timestamps** for additional information.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.9

Value	Description
1	Transmit timestamps delivered to receive callback.
0	Transmit timestamps not delivered. **Default for all.**

18.1.13 transport_lbtrm_tgsz (source)

The transmission group size used for this Topic when LBT-RM is used.

This value must be greater than 0 and must be a power of 2 no greater than 32K. This option affects the transport session underlying the source rather than the source itself. The transport session uses the value from the first source created on the session and ignores subsequent sources' configuration.

Refer to **Source Object** for additional information.

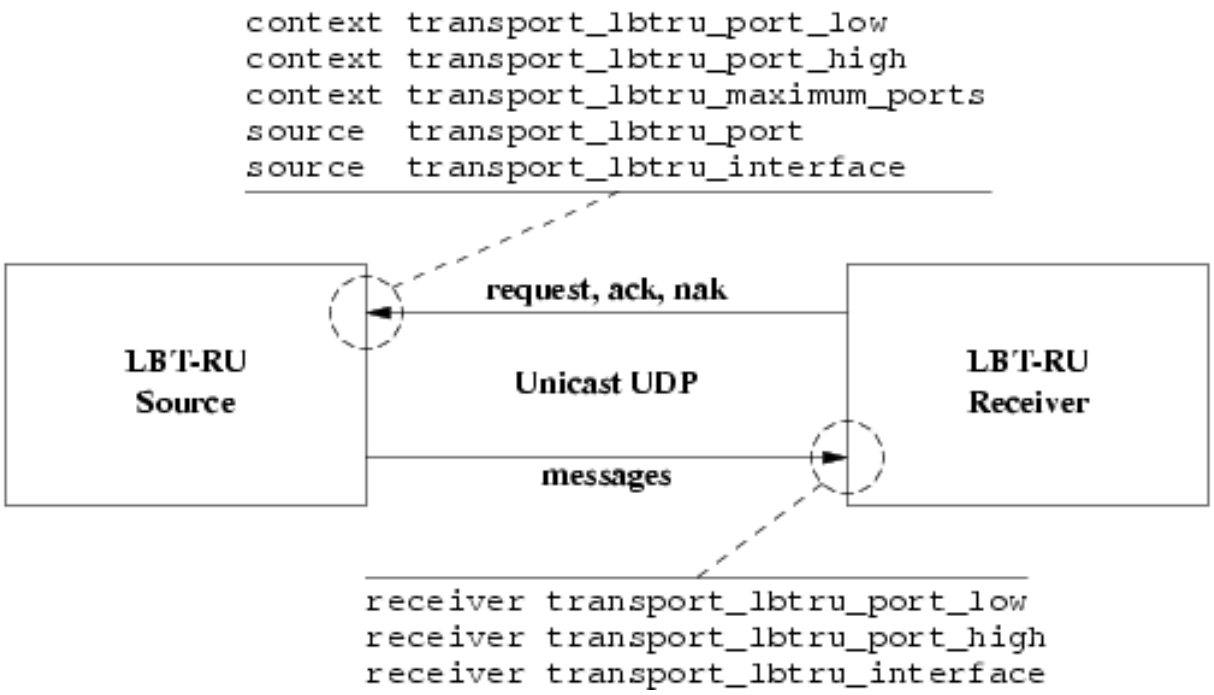
Scope:	source
Type:	**lbm_uint16_t**
Units:	packets
Default value:	8
When to Set:	Can only be set during object initialization.

Chapter 19

Transport LBT-RU Network Options

19.1 LBT-RU Transport Session Management

The image below illustrates the relationship between the primary LBT-RU network options.



When a source is created, the application can explicitly map it to a transport session by setting the transport_lbtru↩_port (source) option. If a previous source was created on the same context with the same port, this new source will be mapped to the same transport session. However, two different contexts on the same host may not share the same port number. If a source is created with a port number that is already in use, UM will return an error.

Alternatively, if the application does not explicitly specify a source port, UM will implicitly assign the new source to a pool of transport sessions defined when the context was created. The pool is defined as a range of port numbers specified by the options transport_lbtru_port_low (context) and transport_lbtru_port_high (context). In addition, the option transport_lbtru_maximum_ports (context) defines the number of transport sessions in the pool.

When a new source is created and the source port is not explicitly defined, UM will check to see how many transport

sessions are currently active from the pool within the context. If it is less than the number in the pool, then UM will activate the next transport session in the range. However, if the context already has activated all transport sessions in the pool, then the new topic is mapped to one of the existing transport sessions, in round-robin fashion.

Notice that the default range of ports, 14380 to 14389, is 10 ports. But the default number of transport sessions in the pool is 5. This allows two contexts to be created on the same host and use the same configuration. If more than 2 contexts are intended to co-exist on the same host, the port range and number of transport session per context must be managed to give a unique port number to every transport session.

The option transport_lbtru_interface (source) may be used on LBT-RU sources to choose particular interface, overriding the default INADDR_ANY which accepts connections on all interfaces. Similarly, transport_lbtru_interface (receiver) may be used on receivers to choose a particular interface for outgoing connections.

19.2 Reference

19.2.1 transport_lbtru_interface (receiver)

Specifies the network interface over which UM LBT-RU receivers read application data messages.

Can specify full IP address of interface, or just network part (see Specifying Interfaces for details).

Default is set to default_interface (context), if specified. Otherwise, it is set to INADDR_ANY, meaning that it will accept incoming connection requests from any interface.

Note: if specifying an interface name in an XML-format file, see Interface Device Names and XML.

Scope:	receiver
Type:	**lbm_ipv4_address_mask_t**
Default value:	0.0.0.0 (INADDR_ANY)
When to Set:	Can only be set during object initialization.

19.2.2 transport_lbtru_interface (source)

Specifies the network interface over which UM LBT-RU sources receive connection requests from topic receivers.

Can specify full IP address of interface, or just network part (see Specifying Interfaces for details).

Be aware that this option is applied to the transport session when the first topic is created on that session. Thus, setting a different interface for a subsequent topic that maps onto the same transport session will have no effect.

Default is set to default_interface (context), if specified. Otherwise, it is set to INADDR_ANY, meaning that it will accept incoming connection requests from any interface.

Note: if specifying an interface name in an XML-format file, see Interface Device Names and XML.

Scope:	source
Type:	**lbm_ipv4_address_mask_t**
Default value:	0.0.0.0 (INADDR_ANY)
When to Set:	Can only be set during object initialization.

19.2.3 transport_lbtru_maximum_ports (context)

Maximum size of LBT-RU source transport session pool.

See LBT-RU Transport Session Management for how LBT-RU source transport sessions are managed.

Scope:	context
Type:	**lbm_uint16_t**
Units:	number of ports
Default value:	5
When to Set:	Can only be set during object initialization.

19.2.4 transport_lbtru_port (source)

The UDP port to be used for the source transport session.

This is the source-side option. For receive-side ports, see transport_lbtru_port_low (receiver).

Setting this option to non-zero overrides the use of the pool of LBT-RU source transport sessions.

See LBT-RU Transport Session Management for how LBT-RU source transport sessions are managed.

See Port Assignments for more information about configuring ports.

Scope:	source
Type:	**lbm_uint16_t**
Default value:	0 (use open port)
Byte order:	Network
When to Set:	Can only be set during object initialization.

19.2.5 transport_lbtru_port_high (context)

High UDP port number of range for pool of LBT-RU source transport sessions.

When transport_lbtru_port (source) is not specified, a newly-created transport session will use an unused port from this range. Receivers that join the source's transport session send connection requests, acknowledgements, and NAKs to the source port.

See also transport_lbtru_port_high (context).

This is the source-side option. For the corresponding receiver option, see transport_lbtru_port_high (receiver).

See LBT-RU Transport Session Management for how LBT-RU source transport sessions are managed.

See Port Assignments for more information about configuring ports.

Scope:	context
Type:	**lbm_uint16_t**
Default value:	14389
Byte order:	Host
When to Set:	Can only be set during object initialization.

19.2.6 transport_lbtru_port_high (receiver)

High port number to use for receiving LBT-RU data.

This is the receive-side option. For the corresponding source option, see transport_lbtru_port_high (context).

When a newly-created receiver joins a source's transport session, it finds a free port from this range, binds to it, and informs the source of the receiver's IP and port. The UM source will send message data to that IP and port.

Unlike most UM port ranges, if the library is not able to find an unused port in this range, it will log a warning (Core-5688-3300), but instead of failing the receiver creation, it will allocate a port from the host's ephemeral pool and operate normally. Thus, it is possible for a receiver to get messages on a port outside of the configured range.

See Port Assignments for more information about configuring ports.

Scope:	receiver
Type:	**lbm_uint16_t**
Default value:	14379
Byte order:	Host
When to Set:	Can only be set during object initialization.

19.2.7 transport_lbtru_port_low (context)

Low UDP port number of range for pool of LBT-RU source transport sessions.

When transport_lbtru_port (source) is not specified, a newly-created transport session will use an unused port from this range. Receivers that join the source's transport session send connection requests, acknowledgements, and NAKs to the source port.

See also transport_lbtru_port_high (context).

This is the source-side option. For the corresponding receiver option, see transport_lbtru_port_low (receiver).

See LBT-RU Transport Session Management for how LBT-RU source transport sessions are managed.

See Port Assignments for more information about configuring ports.

Scope:	context
Type:	**lbm_uint16_t**
Default value:	14380
Byte order:	Host
When to Set:	Can only be set during object initialization.

19.2.8 transport_lbtru_port_low (receiver)

Low port number to use for receiving LBT-RU data.

This is the receive-side option. For the corresponding source option, see transport_lbtru_port_low (context).

When a newly-created receiver joins a source's transport session, it finds a free port from this range, binds to it, and informs the source of the receiver's IP and port. The UM source will send message data to that IP and port.

Unlike most UM port ranges, if the library is not able to find an unused port in this range, it will log a warning (Core-5688-3300), but instead of failing the receiver creation, it will allocate a port from the host's ephemeral pool and operate normally. Thus, it is possible for a receiver to get messages on a port outside of the configured range.

See Port Assignments for more information about configuring ports.

Scope:	receiver
Type:	**lbm_uint16_t**
Default value:	14360
Byte order:	Host
When to Set:	Can only be set during object initialization.

Chapter 20

Transport LBT-RU Reliability Options

LBT-RU's reliability options closely model LBT-RM's. The descriptions and illustrations in Transport LBT-RM Reliability Options generally apply to LBT-RU, with appropriate option name changes.

20.1 Reference

20.1.1 transport_lbtru_ignore_interval (source)

The interval to ignore NAKs after a retransmission is sent.

This option affects the transport session underlying the source rather than the source itself. The transport session uses the value from the first source created on the session and ignores subsequent sources' configuration.

Refer to **Source Object** for additional information.

Scope:	source
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	500 (0.5 seconds)
When to Set:	Can only be set during object initialization.

20.1.2 transport_lbtru_nak_backoff_interval (receiver)

The maximum interval between transmissions of LBT-RU NAKs for a given sequence number, after the first NAK.

When an LBT-RU receiver detects a sequence number gap, it delays an initial amount before sending its first NAK (controlled by transport_lbtru_nak_initial_backoff_interval (receiver)), and then delays an a separately configurable time between sending subsequent NAKs for the same sequence number. This configuration option controls those subsequent delays.

The actual time the receiver will wait to NAK again is random. The algorithm used to determine the time range is (1/2 ∗ backoff_interval - 3/2 ∗ backoff_interval). This will result in a delay longer or shorter than the specified value.

This option affects the transport session underlying the receiver rather than the receiver itself. The transport session uses the value from the first receiver created on the session and ignores subsequent receivers' configuration.

Refer to **Receiver Object** and Interrelated Configuration Options for additional information.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	200 (0.2 seconds)
When to Set:	Can only be set during object initialization.

20.1.3 transport_lbtru_nak_generation_interval (receiver)

The maximum time that a piece of data may be outstanding before the data is unrecoverably lost.

For LBT-RU transport sessions only. Although the minimum valid value is 5 milliseconds, larger values are advisable. This option affects the transport session underlying the receiver rather than the receiver itself. The transport session uses the value from the first receiver created on the session and ignores subsequent receivers' configuration.

Refer to **Receiver Object** and Interrelated Configuration Options for additional information.

Scope:	receiver
Type:	**lbm_ulong_t**

Units:	milliseconds
Default value:	10000 (10 seconds)
When to Set:	Can only be set during object initialization.

20.1.4 transport_lbtru_nak_initial_backoff_interval (receiver)

The interval between loss detection and transmission of the first LBT-RU NAK.

When an LBT-RU receiver detects a sequence number gap, it delays an initial amount before sending its first NAK controlled by this option, and then delays an a separately configurable time between sending subsequent NAKs for the same sequence number, controlled by transport_lbtru_nak_backoff_interval (receiver).

The actual time the receiver will wait to NAK is random. The algorithm used to determine the time range is (1/2 $*$ initial_backoff_interval - 3/2 $*$ initial_backoff_interval). This can result in a wait interval longer than the specified value. A value of 0 indicates that the receiver should immediately send a NAK.

This option affects the transport session underlying the receiver rather than the receiver itself. The transport session uses the value from the first receiver created on the session and ignores subsequent receivers' configuration.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	0 (disabled)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.10

20.1.5 transport_lbtru_nak_suppress_interval (receiver)

The time that an LBT-RU receiver will suppress sending a NAK for a missing datagram after an NCF is received from the source.

The source sends an NCF in response to a NAK which the source temporarily cannot retransmit. For example,

if the source gets a NAK for a sequence number for which it has recently sent a retransmission, it will send an NCF with reason code "ignored". The receiver responds by suppressing NAKs for that sequence number for the interval configured by this option. See **NAK Suppression** for more information about NCFs. For LBT-RU transport sessions only.

This option affects the transport session underlying the receiver rather than the receiver itself. The transport session uses the value from the first receiver created on the session and ignores subsequent receivers' configuration.

Refer to **Receiver Object** for additional information.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	1000 (1 second)
When to Set:	Can only be set during object initialization.

20.1.6 transport_lbtru_receiver_socket_buffer (context)

Value used to set SO_RCVBUF value of the LBT-RU receiver unicast socket (both sender and receiver sides).

In some cases the OS will not allow all of this value to be used. See Socket Buffer Sizes for platform-dependent information.

See also our white paper Topics in High Performance Messaging for background and guidelines on UDP buffer sizing.

Scope:	context
Type:	**lbm_ulong_t**
Units:	bytes
Default value:	8388608 (8MB)
When to Set:	Can only be set during object initialization.

20.1.7 transport_lbtru_source_socket_buffer (context)

Value used to set SO_SNDBUF value of the LBT-RU send multicast socket.

In some cases the OS will not allow all of this value to be used. See Socket Buffer Sizes for platform-dependent information. A value of 0 instructs UM to use the OS default.

Scope:	context
Type:	**lbm_ulong_t**
Units:	bytes
Default value:	1048576 (1MB)
When to Set:	Can only be set during object initialization.

20.1.8 transport_lbtru_transmission_window_limit (source)

Caps the total amount of memory that a transmission window uses, which includes data and overhead.

For example, if the transport_lbtru_transmission_window_size (source) is 24 MB (default) and the source sends 20 byte messages with the "flush" flag, the actual amount of memory used can approximate 300 MB. The default value of this option does not limit the transmission window.

Scope:	source
Type:	**size_t**
Units:	bytes
Default value:	0 (no limit)
When to Set:	Can only be set during object initialization.

20.1.9 transport_lbtru_transmission_window_size (source)

The maximum amount of buffered data that the LBT-RU source is allowed to retain for retransmissions.

The minimum valid value is 65536 bytes. This option affects the transport session underlying the source rather

than the source itself. The transport session uses the value from the first source created on the session and ignores subsequent sources' configuration.

Refer to **Source Object** for additional information.

Scope:	source
Type:	**size_t**
Units:	bytes
Default value:	25165824 (24 MB)
When to Set:	Can only be set during object initialization.

Chapter 21

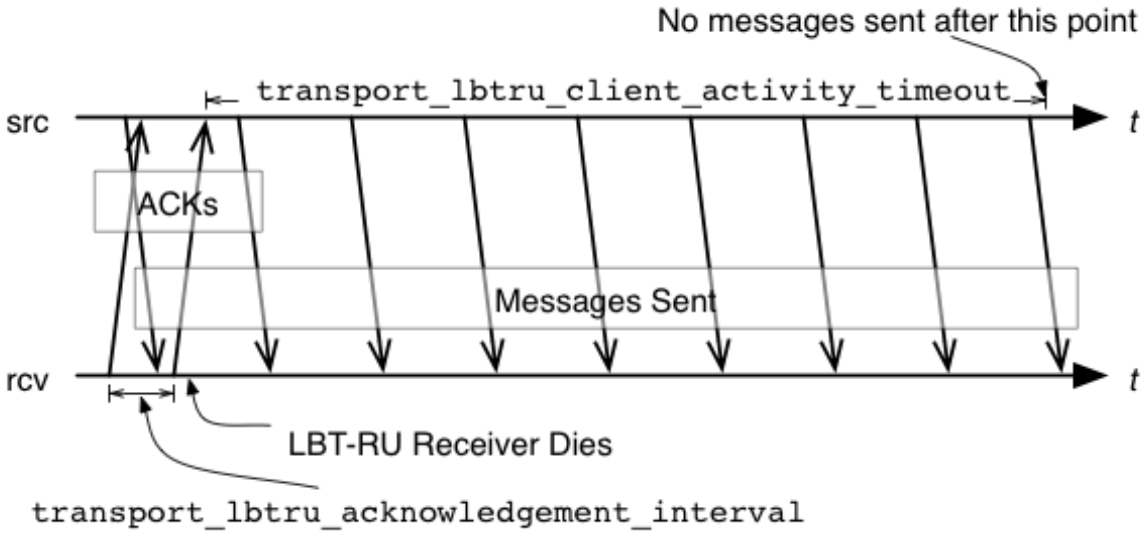
Transport LBT-RU Operation Options

LBT-RU's operational options closely model LBT-RM's. The descriptions and illustrations in Transport LBT-RM Operation Options generally apply to LBT-RU, with appropriate option name changes.

The following options are present for LBT-RU but not LBT-RM:

- transport_lbtru_client_map_size (source)
- transport_lbtru_connect_interval (receiver)
- transport_lbtru_acknowledgement_interval (receiver)
- transport_lbtru_client_activity_timeout (source)

The image below illustrates the timing of the latter two LBT-RU unique options:



21.1 Reference

21.1.1 transport_lbtru_acknowledgement_interval (receiver)

The interval between sending acknowledgements.

For LBT-RU transport session only. Each client continually sends acknowledgements to let the source know that the client is still alive. This option affects the transport session underlying the receiver rather than the receiver itself. The transport session uses the value from the first receiver created on the session and ignores subsequent receivers' configuration.

Refer to **Receiver Object** for additional information.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	500 (0.5 seconds)
When to Set:	Can only be set during object initialization.

21.1.2 transport_lbtru_activity_timeout (receiver)

The maximum time that an LBT-RU session may be quiescent before it is deleted and an EOS event is delivered for all topics using this transport session.

For LBT-RU transport sessions only. This option affects the transport session underlying the receiver rather than the receiver itself. The transport session uses the value from the first receiver created on the session and ignores subsequent receivers' configuration.

Refer to **Receiver Object** for additional information.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	60000 (60 seconds)
When to Set:	Can only be set during object initialization.

21.1.3 transport_lbtru_client_activity_timeout (source)

The maximum time that an LBT-RU client may be quiescent, i.e. not sending ACKs, before the sender assumes that it is dead and stops sending to it.

This option affects the transport session underlying the source rather than the source itself. The transport session uses the value from the first source created on the session and ignores subsequent sources' configuration.

Refer to **Source Object** for additional information.

Scope:	source
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	10000 (10 seconds)
When to Set:	Can only be set during object initialization.

21.1.4 transport_lbtru_client_map_size (source)

The size of the hash table used to store client information and state.

This option affects the transport session underlying the source rather than the source itself. The transport session uses the value from the first source created on the session and ignores subsequent sources' configuration.

Refer to **Source Object** for additional information.

Scope:	source
Type:	**size_t**
Units:	table entries
Default value:	7
When to Set:	Can only be set during object initialization.

21.1.5 transport_lbtru_coalesce_threshold (source)

UM passes implicitly batched messages to the Operating System sendmsg() as a set unless the size of the set exceeds the coalescing threshold at which point the messages are coalesced and passed to the O/S as one copy.

This option accommodates the different number of iovecs supported by different O/Ss. Tuning this option balances the efficiency of less iovecs handled by the OS vs. the expense of an additional copy operation of the messages before sending. The default value is also the maximum allowable value for Solaris, AIX and HPUX. For Linux and Microsoft Windows and Darwin, the maximum allowable value is 1023. These maximum allowable values are one less than what the O/S provides. This option affects the transport session underlying the source rather than the source itself. The transport session uses the value from the first source created on the session and ignores subsequent sources' configuration.

Refer to **Source Object** for additional information.

Scope:	source
Type:	**int**
Units:	number of messages
Default value:	15
When to Set:	Can only be set during object initialization.

21.1.6 transport_lbtru_connect_interval (receiver)

The interval between sending connection requests.

For LBT-RU transport session only. This option affects the transport session underlying the receiver rather than the receiver itself. The transport session uses the value from the first receiver created on the session and ignores subsequent receivers' configuration.

Refer to **Receiver Object** for additional information.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	100 (0.1 seconds)
When to Set:	Can only be set during object initialization.

21.1.7 transport_lbtru_data_rate_limit (context)

Maximum aggregate transmission rate of all LBT-RU sessions original data for this particular context.

Refer to **Rate Controls** for additional information about the UM rate limiting algorithm.

Note: For backwards compatibility with earlier versions, the **lbm_context_attr_setopt()** function will accept both 32 and 64 bit values for this option. Note however that a 32-bit value can only specify a rate limit a little larger than 4 Gbps.

Scope:	context
Type:	**lbm_uint64_t**
Units:	bits per second
Default value:	10000000 (10 Mbps)
When to Set:	Can only be set during object initialization.

21.1.8 transport_lbtru_datagram_max_size (context)

The maximum UDP datagram payload size that can be generated for a LBT-RU transport session. Note that this does not include UDP, IP, or packet overhead added by the network stack. The default value is 8192, the minimum is 500 bytes, and the maximum is 65535.

See **Message Fragmentation and Reassembly** for more information.

Informatica does not recommend setting datagram max size options to the network MTU. See **Datagram Max Size and Network MTU**.

Warning

When the DRO is in use, it is recommended that all UM applications and components (including the DRO and Persistent Store) share the same maximum datagram size setting. See **Protocol Conversion**.

Users of **kernel-bypass drivers** should also see **Dynamic Fragmentation Reduction**.

Scope:	context
Type:	**lbm_uint_t**
Units:	bytes
Default value:	8192
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1

21.1.9 transport_lbtru_maximum_connect_attempts (receiver)

The maximum number of connect attempts to make before this transport session is deleted and an EOS event is delivered for all topics using this transport session.

This option affects the transport session underlying the receiver rather than the receiver itself. The transport session uses the value from the first receiver created on the session and ignores subsequent receivers' configuration.

Refer to **Receiver Object** for additional information.

Scope:	receiver
Type:	**lbm_ulong_t**
Default value:	600
When to Set:	Can only be set during object initialization.

21.1.10 transport_lbtru_rate_interval (context)

Period that LBT-RU rate limiter runs.

Reducing period reduces burst intensity, but also increases CPU load. Refer to **Rate Controls** for additional information about the UM rate limiting algorithm.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	100
When to Set:	Can only be set during object initialization.

String value	Integer value	Description
"10"	10	LBT-RU rate limiter runs every 10 milliseconds.
"20"	20	LBT-RU rate limiter runs every 20 milliseconds.
"50"	50	LBT-RU rate limiter runs every 50 milliseconds.
"100"	100	LBT-RU rate limiter runs every 100 milliseconds. **Default for all.**

21.1.11 transport_lbtru_recycle_receive_buffers (context)

Enables the use of buffer recycling for socket operations.

See **Receive Buffer Recycling** for more information, including restrictions on the use of this feature.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.12

Value	Description
1	Use buffer recycling.
0	Buffer recycling is not used. **Default for all.**

21.1.12 transport_lbtru_retransmit_rate_limit (context)

Maximum aggregate transmission rate of all LBT-RU sessions retransmissions for this particular context.

This should always be less than the value used for original data. Refer to **Rate Controls** for additional information about the UM rate limiting algorithm.

Note: For backwards compatibility with earlier versions, the **lbm_context_attr_setopt()** function will accept both 32 and 64 bit values for this option. Note however that a 32-bit value can only specify a rate limit a little larger than 4 Gbps.

Scope:	context
Type:	**lbm_uint64_t**
Units:	bits per second
Default value:	5000000 (5 Mbps)
When to Set:	Can only be set during object initialization.

21.1.13 transport_lbtru_sm_maximum_interval (source)

The maximum interval between LBT-RU session messages.

In lieu of data being sent, LBT-RU sends session messages to each client to inform them of sequence numbers and to let receivers know that the sender is still transmitting. This option affects the transport session underlying the source rather than the source itself. The transport session uses the value from the first source created on the session and ignores subsequent sources' configuration.

Refer to **Source Object** for additional information.

Scope:	source
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	10000 (10 seconds)
When to Set:	Can only be set during object initialization.

21.1.14 transport_lbtru_sm_minimum_interval (source)

The minimum interval between LBT-RU session messages.

In lieu of data being sent, LBT-RU sends session messages to each client to inform them of sequence numbers and to let receivers know that the sender is still transmitting. This option affects the transport session underlying the source rather than the source itself. The transport session uses the value from the first source created on the session and ignores subsequent sources' configuration.

Refer to **Source Object** for additional information.

Scope:	source
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	200 (0.2 seconds)
When to Set:	Can only be set during object initialization.

21.1.15 transport_lbtru_use_session_id (source)

Control whether a session ID is used for LBT-RU Transport sessions.

This option should be set to 0 if a version 3.3 (and beyond) LBT-RU source must interoperate with a version pre-3.3 receiver.

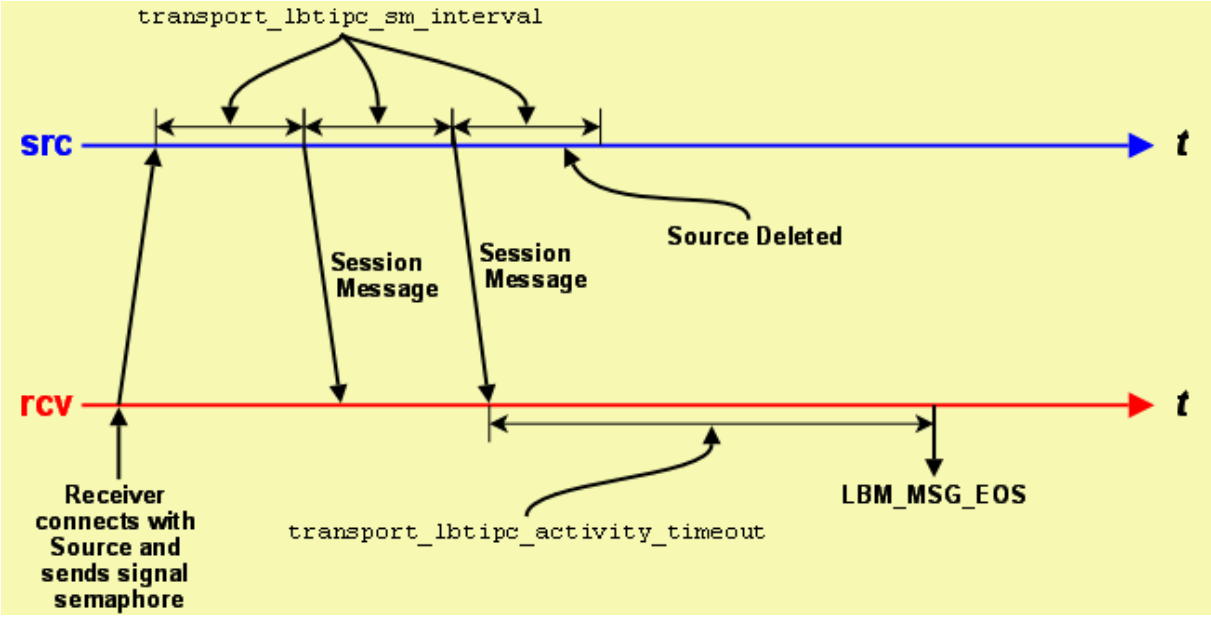
Scope:	source
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 3.3

Value	Description
1	Indicates the application desires LBT-RU to use a session ID. **Default for all.**
0	Indicates the application does not desire LBT-RU to use a session ID. For use when version pre-3.3 receivers must be used with TCP sources that are version 3.3 and beyond.

Chapter 22

Transport LBT-IPC Operation Options

The image below illustrates the timing of an LBT-IPC transport session.



The Source Session Message mechanism enables the receiver to detect when a source goes away and works similarly to LBT-RU. It operates independently of message writes/reads in the Shared Memory Area.

22.1 LBT-IPC Transport Session Management

When a source is created, the application can explicitly map it to a transport session by setting the transport_lbtipc_id (source) option. If a previous source was created on the same context with the same ID number, this new source will be mapped to the same transport session. Note that ID numbers can be re-used by different contexts on the same host. The resulting transport sessions will be separate, independent, and non-interfering.

Alternatively, if the application does not explicitly specify a source ID, UM will implicitly assign the new source to a pool of transport sessions defined when the context was created. The pool is defined as a range of ID numbers specified by the options transport_lbtipc_id_low (context) and transport_lbtipc_id_high (context). The numeric range defines the number of transport sessions in the pool.

When a new source is created and the source port is not explicitly defined, UM will check to see how many transport sessions are currently active from the pool within the context. If it is less than the configured range of IDs then UM

will use the next ID in the range transport_lbtipc_id_low (context) to transport_lbtipc_id_high (context). However, if the context already has activated all transport sessions in the pool, then the new topic is mapped to one of the existing transport sessions, in round-robin fashion.

22.2 Reference

22.2.1 transport_lbtipc_activity_timeout (receiver)

The maximum period of inactivity (lack of session messages) from an IPC source before the UM delivers an EOS event for all topics using the transport session.

Refer to **Receiver Object** and Interrelated Configuration Options for additional information.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	60,000 (60 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.5ea2/UME 2.2ea1

22.2.2 transport_lbtipc_behavior (source)

Desired flow control behavior when multiple receivers have joined the same LBT-IPC transport session.

See also **Transport LBT-IPC**. This option affects the transport session underlying the source rather than the source itself. The transport session uses the value from the first source created on the session and ignores subsequent sources' configuration.

Refer to **Source Object** for additional information.

Scope:	source
Type:	**lbm_ushort_t**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0

String value	Integer value	Description
"source_paced"	LBM_SRC_TOPIC_ATTR_LBTIPC_BE↩ HAVIOR_SOURCE_PACED	Your application writes as fast as it can to the LBT-IPC shared memory area. Slower receivers can experience loss. A source does not consider if any receivers have successfully read a message before it reclaims it. **Default for all.**
"receiver_paced"	LBM_SRC_TOPIC_ATTR_LBTIPC_BE↩ HAVIOR_RECEIVER_PACED	Your application writes to the LBT-IPC shared memory area only as fast as the slowest receiver consumes data. A source will not reclaim a message until all receivers have successfully read the message. This slows down all receiver on the LBT-IPC transport session.

22.2.3 transport_lbtipc_datagram_max_size (context)

The maximum datagram size that can be generated for a LBT-IPC transport session. While IPC does not use UDP datagrams, this option limits the size of the UM message which is given to the underlying transport type, including all UM headers and overhead. The default value is 65535, the minimum is 500 bytes, and the maximum is 65535.

See **Message Fragmentation and Reassembly** for more information.

Informatica does not recommend setting datagram max size options to the network MTU. See **Datagram Max Size and Network MTU**.

Warning

When the DRO is in use, it is recommended that all UM applications and components (including the DRO and Persistent Store) share the same maximum datagram size setting. See **Protocol Conversion**.

Users of **kernel-bypass drivers** should also see **Dynamic Fragmentation Reduction**.

Scope:	context
Type:	**lbm_uint_t**
Units:	bytes

Default value:	65535
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1

22.2.4 transport_lbtipc_dro_loss_recovery_timeout (receiver)

For IPC transport sessions originating from a DRO endpoint portal, delay declaring as unrecoverable a lost message.

Message streams traversing a DRO can have the message order changed. If the DRO's outgoing transport session uses the IPC protocol, these out-of-order messages will normally trigger immediate unrecoverable loss. This timeout allows an opportunity for the messages to be re-ordered properly.

The value 0 disables this delay (i.e. receivers immediately declare unrecoverable loss).

See **DRO Reliable Loss** for more information.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	0
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 6.12

22.2.5 transport_lbtipc_id (source)

The preferred Transport ID for a specific source's LBT-IPC session.

If 0, the UM context attempts to find one in the given Transport ID range of transport_lbtipc_id_low (context) and transport_lbtipc_id_high (context).

See LBT-RU Transport Session Management and **Sources and LBT-IPC** for more information.

Scope:	source
Type:	**lbm_uint16_t**
Default value:	0 (use open ID)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.5ea2/UME 2.2ea1

22.2.6 transport_lbtipc_id_high (context)

Highest transport ID of the range of available LBT-IPC Transport IDs.

See LBT-RU Transport Session Management and **Sources and LBT-IPC** for more information.

Scope:	context
Type:	**lbm_uint16_t**
Default value:	20,005
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.5ea2/UME 2.2ea1

22.2.7 transport_lbtipc_id_low (context)

Lowest transport ID of the range of available LBT-IPC Transport IDs.

See LBT-RU Transport Session Management and **Sources and LBT-IPC** for more information.

Scope:	context
Type:	**lbm_uint16_t**
Default value:	20,001
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.5ea2/UME 2.2ea1

22.2.8 transport_lbtipc_maximum_receivers_per_transport (source)

The maximum number of receiving contexts that can join an IPC transport session.

Once a receiving context joins an IPC transport session, it can receive messages on multiple topics. Increasing this value increases the amount of shared memory allocated per transport session by a negligible amount.

Scope:	source
Type:	**lbm_ushort_t**
Default value:	20
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0

22.2.9 transport_lbtipc_pend_behavior_linger_loop_count (context)

When using pend as the LBTIPC receiver thread behavior, the receiver loop can linger in a temporary busy wait behavior before pending again.

At high sustained rates or during short bursts of data, this can result in a significant reduction in the number of kernel calls if more data arrives relatively quickly. Once the burst subsides, the CPU utilization drops again since the receiver would be pending. The default value of 1 results in legacy pend behavior. If the value is set large, significant CPU will be consumed.

Scope:	context
Type:	**lbm_ulong_t**
Default value:	1
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.10

22.2.10 transport_lbtipc_receiver_operational_mode (context)

The mode in which UM operates to process LBT-IPC messages.

See **Embedded Mode** for additional information.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.

String value	Integer value	Description
"embedded"	LBM_CTX_ATTR_OP_EMBEDDED	UM spawns a thread to process received LBT-IPC messages. **Default for all.**
"sequential"	LBM_CTX_ATTR_OP_SEQUENTIAL	Your application must call **lbm_context_process_lbtipc_messages()** to process received LBT-IPC messages. If you also set the context's operational_mode option to sequential, your application must donate an additional thread to service the **lbm_context_process_events()** calls. Note: You can use sequential mode with the C API, but not with the Java API or .NET API. The Java and .NET APIs do not provide an equivalent **lbm_context_process_lbtipc_messages()** API for LBT- IPC.

22.2.11 transport_lbtipc_receiver_thread_behavior (context)

Receiver behavior for monitoring the signaling semaphore set by the IPC source when it writes new data to the shared memory area.

Note that the IPC thread is not the same as the Context thread.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.5ea2/UME 2.2ea1

String value	Integer value	Description
"pend"	LBM_CTX_ATTR_IPC_RCV_THREAD_↩ PEND	Receiver waits (sleep) for notification from OS that IPC source has updated the signaling semaphore. This option is best when the IPC source frequently writes new data to the shared area. **Default for all.**
"busy_wait"	LBM_CTX_ATTR_IPC_RCV_THREAD_↩ BUSY_WAIT	Provides the lowest latency as the receiver monopolizes the CPU core looking for an incremented semaphore. This option works best for infrequent or sporadic message delivery from the IPC source, but involves a CPU cost.

22.2.12 transport_lbtipc_recycle_receive_buffers (context)

Enables the use of buffer recycling for IPC operations.

See **Receive Buffer Recycling** for more information, including restrictions on the use of this feature.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.12

Value	Description
1	Use buffer recycling.
0	Buffer recycling is not used. **Default for all.**

22.2.13 transport_lbtipc_sm_interval (source)

Time period between sessions message sent from source to receivers.

Refer to **Source Object** and Interrelated Configuration Options for additional information.

Scope:	source
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	10,000 (10 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.5ea2/UME 2.2ea1

22.2.14 transport_lbtipc_transmission_window_size (source)

Size of an LBT-IPC transport's shared memory area.

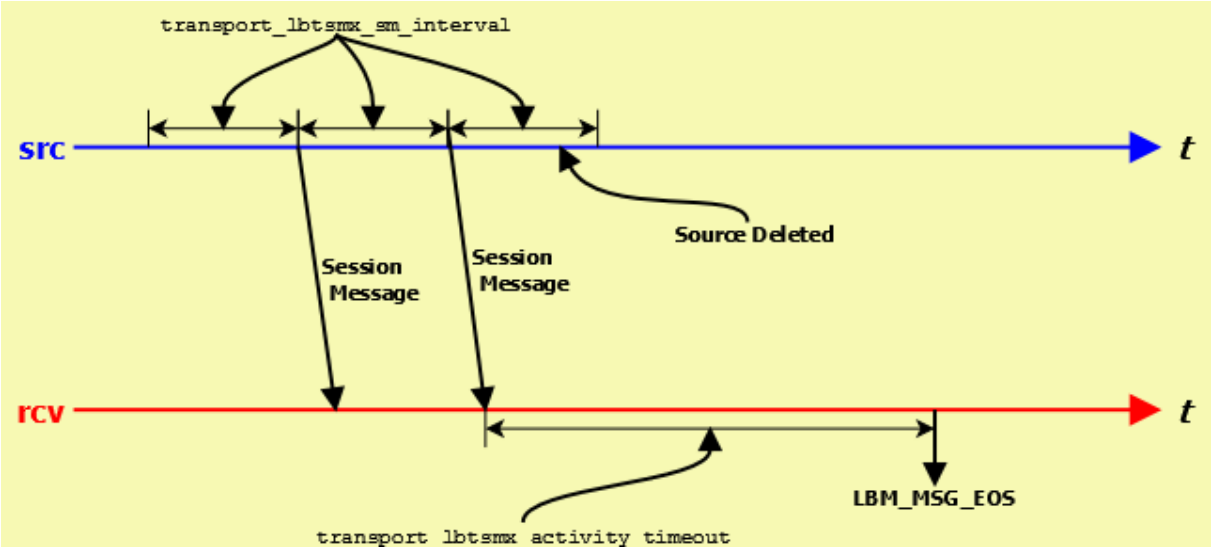
This value may vary across platforms. The actual size of the shared memory area equals the value you specify for this option plus about 64 KB for header information. The minimum value for this option is 65,536. Refer to **Source Object** for additional information.

Scope:	source
Type:	**size_t**
Units:	bytes
Default value:	25165824 (24 MB)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.5ea2/UME 2.2ea1

Chapter 23

Transport LBT-SMX Operation Options

The image below illustrates the timing of an LBT-SMX transport session.



The Source Session Message mechanism enables the receiver to detect when a source goes away and works similarly to LBT-RU. It operates independently of message writes/reads in the Shared Memory Area.

23.1 LBT-SMX Transport Session Management

When a source is created, the application can explicitly map it to a transport session by setting the transport_↩ lbtsmx_id (source) option. If a previous source was created on the same context with the same ID number, this new source will be mapped to the same transport session. Note that ID numbers can be re-used by different contexts on the same host. The resulting transport sessions will be separate, independent, and non-interfering.

Alternatively, if the application does not explicitly specify a source ID, UM will implicitly assign the new source to a pool of transport sessions defined when the context was created. The pool is defined as a range of ID numbers specified by the options transport_lbtsmx_id_low (context) and transport_lbtsmx_id_high (context). The numeric range defines the number of transport sessions in the pool.

When a new source is created and the source port is not explicitly defined, UM will check to see how many transport sessions are currently active from the pool within the context. If it is less than the configured range of IDs then UM will use the next ID in the range transport_lbtsmx_id_low (context) to transport_lbtsmx_id_high (context). However,

if the context already has activated all transport sessions in the pool, then the new topic is mapped to one of the existing transport sessions, in round-robin fashion.

23.2 Reference

23.2.1 transport_lbtsmx_activity_timeout (receiver)

The maximum period of inactivity (lack of updates to the source's shared activity counter) from an SMX source before UM delivers an EOS event for all topics using the transport session.

You should configure this option to a value greater than the source's transport_lbtsmx_sm_interval so receivers do not erroneously report a source as inactive.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	60,000 (60 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.1

23.2.2 transport_lbtsmx_datagram_max_size (source)

The maximum datagram size that can be sent for an LBT-SMX transport session.

While SMX does not use UDP datagrams, this option limits the size of the UM message which is given to the underlying transport type, including all UM headers and overhead. This value includes 16 bytes of header information per message, plus an additional 24 bytes of reserved space for compatibility with other egress transports when re-sending SMX messages through a UM Dynamic Router. Therefore, the largest usable message size for the default setting of 8192 bytes would be 8176 bytes (8192 - 16 - 24). The minimum is 32 bytes. The maximum size is limited by available memory.

This option imposes a hard limit on message size because the LBT-SMX transport does not support datagram fragmentation or reassembly. Unlike other transports that do support fragmentation, attempts to send messages larger than the datagram size configured by this option fail.

The minimum value for this option is 32 bytes. Unlike other transports, there is no hard-coded maximum value; the maximum is limited only by the amount of memory available.

Note: The source's configured transport_lbtsmx_transmission_window_size (source) must be at least twice as large as the source's configured transport_lbtsmx_datagram_max_size. If the transmission window has not been configured to be large enough to hold at least two maximum-sized SMX datagrams, then a warning will be issued and the source's transport_lbtsmx_transmission_window_size option will be automatically adjusted upwards to the nearest power-of-2 size in bytes that can fit at least two maximum-sized datagrams.

See **Message Fragmentation and Reassembly** for more information.

Informatica does not recommend setting datagram max size options to the network MTU. See **Datagram Max Size and Network MTU**.

Warning

When the DRO is in use, it is recommended that all UM applications and components (including the DRO and Persistent Store) share the same maximum datagram size setting. See **Protocol Conversion**.

Users of **kernel-bypass drivers** should see **Dynamic Fragmentation Reduction**.

Scope:	source
Type:	**lbm_uint_t**
Units:	bytes
Default value:	8192
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.1

23.2.3 transport_lbtsmx_id (source)

The preferred Transport ID for a specific source's LBT-SMX session.

To use this option, configure a non-zero value. For the default value of 0 (zero), the UM context selects the next available Transport ID in the Transport ID range of transport_lbtsmx_id_low (context) and transport_lbtsmx_↩ id_high (context).

See LBT-RU Transport Session Management and **Sources and LBT-SMX** for more information.

Scope:	source
Type:	**lbm_uint16_t**
Default value:	0 (select next ID in range)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.1

23.2.4 transport_lbtsmx_id_high (context)

Highest transport ID in the range of available LBT-SMX Transport IDs.

See LBT-RU Transport Session Management and **Sources and LBT-SMX** for more information.

Scope:	context
Type:	**lbm_uint16_t**
Default value:	30,005
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.1

23.2.5 transport_lbtsmx_id_low (context)

Lowest transport ID in the range of available LBT-SMX Transport IDs.

See LBT-RU Transport Session Management and **Sources and LBT-SMX** for more information.

Scope:	context
Type:	**lbm_uint16_t**
Default value:	30,001
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.1

23.2.6 transport_lbtsmx_maximum_receivers_per_transport (source)

The maximum number of receiving contexts that can join an SMX transport session.

Once a receiving context joins an SMX transport session, it can receive messages on multiple topics. Increasing this value increases the amount of shared memory allocated per transport session by a negligible amount.

Scope:	source
Type:	**lbm_ushort_t**
Default value:	64
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.1

23.2.7 transport_lbtsmx_message_statistics_enabled (context)

Controls whether or not UM records LBT-SMX transport statistics

Enabling statistics gives better visibility of application behavior, at the expense of a small but measurable amount of latency.

Scope:	context
Type:	**int**
Default value:	0
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.1

Value	Description
1	UM records source and receiver LBT-SMX transport statistics.

Value	Description
0	UM does not record source and receiver LBT-SMX transport statistics. **Default for all.**

23.2.8 transport_lbtsmx_sm_interval (source)

Time period between updates to an LBT-SMX source's shared activity counter, which enables connected receivers to determine the source's liveness.

You should configure this option to a value less than the receivers' corresponding transport_lbtsmx_activity_timeout (receiver) setting so receivers do not time out sources too early.

Refer to **Source Object** for additional information.

Scope:	source
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	10,000 (10 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.1

23.2.9 transport_lbtsmx_transmission_window_size (source)

Size of an LBT-SMX transport's shared memory area.

Must be a power of two and be twice a large as the source's transport_lbtsmx_datagram_max_size (source). If you configure a value that is not a power of 2 or is less than twice the size of the maximum datagram size, UM issues a warning log message and automatically rounds up the value of this option to the next power of 2 window size that can fit at least two maximum-sized datagrams. The minimum value for this option is 64 bytes.

Refer to **Source Object** for additional information.

Scope:	source
Type:	**size_t**

Units:	bytes
Default value:	131072 (128 KB)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.1

Chapter 24

Transport Acceleration Options

Transport acceleration options enable kernel-bypass acceleration in conjunction with the following vendor solutions:

- Myricom Datagram Bypass Layer (DBL)
- Solarflare Onload
- Mellanox 10-Gigabit Ethernet or InfiniBand hardware

24.1 Myricom Datagram Bypass Layer (DBL)

DBL is a kernel-bypass technology that accelerates sending and receiving UDP traffic and operates with DBL-enabled Myricom 10-Gigabit Ethernet adapter cards for Linux and Microsoft Windows.

DBL does not support fragmentation and reassembly, so do not send messages larger than the MTU size configured on the DBL interface.

DBL acceleration is compatible with the following Ultra Messaging transport types:

- LBT-RM (UDP-based reliable multicast)
- LBT-RU (UDP-based reliable unicast)
- Multicast Immediate Messaging
- Multicast Topic Resolution

To use DBL Transport Acceleration, perform the following steps:

1. Install the Myricom 10-Gigabit Ethernet NIC.
2. Install the DBL shared library.
3. Update your search path to include the location of the DBL shared library.
4. Set option transport_∗_datagram_max_size and option resolver_datagram_max_size (context) to a value of no more than 28 bytes smaller than the Myricom interface's configured MTU size.

Users of DBL are advised to make use of **Dynamic Fragmentation Reduction**.

24.2 Reference

24.2.1 dbl_lbtrm_acceleration (context)

Flag indicating if DBL acceleration is enabled for LBT-RM transports.

See Myricom Datagram Bypass Layer (DBL).

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0.

Value	Description
1	DBL acceleration is enabled for LBT-RM.
0	DBL acceleration is not enabled for LBT-RM. **Default for all.**

24.2.2 dbl_lbtru_acceleration (context)

Flag indicating if DBL acceleration is enabled for LBT-RU transports.

See Myricom Datagram Bypass Layer (DBL).

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0.

Value	Description
1	DBL acceleration is enabled for LBT-RU.
0	DBL acceleration is not enabled for LBT-RU. **Default for all.**

24.2.3 dbl_mim_acceleration (context)

Flag indicating if DBL acceleration is enabled for MIM.

See Myricom Datagram Bypass Layer (DBL).

See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0.

Value	Description
1	DBL acceleration is enabled for MIM.
0	DBL acceleration is not enabled for MIM. **Default for all.**

24.2.4 dbl_resolver_acceleration (context)

Flag indicating if DBL acceleration is enabled for topic resolution.

See Myricom Datagram Bypass Layer (DBL).

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0.

Value	Description
1	DBL acceleration is enabled for topic resolution.
0	DBL acceleration is not enabled for topic resolution. **Default for all.**

24.3 Solarflare Onload

Solarflare Onload is a kernel-bypass technology that accelerates message traffic and operates with Solarflare 10↩ GbE Ethernet NICs.

Note

Onload does not support fragmentation and reassembly, so do not send messages larger than the MTU size configured on the Solarflare interface.

Warning

Onload does not support both accelerated and non-accelerated processes subscribing to the same multicast group on the same host. An attempt to do so will result in the non-accelerated process becoming "deaf" to the shared multicast group. See "Onload User Guide", chapter 9.11 *"Multicast Operation and Stack Sharing"*, sub-section *"Multicast Receive - Onload Stack and Kernel Stack"*.

Ultra Messaging loads the Onload library dynamically during Ultra Messaging initialization on Linux-based (X86) platforms.

Onload default behavior accelerates all sockets. You can access the Onload onload_set_stackname API extension to select the sockets you want to accelerate by using UM configuration options. Selecting sockets with a stackname lets you accelerate data transmission sockets and not sockets for control messages, topic resolution, or responses.

You can select a stackname with the configuration options onload_acceleration_stack_name (receiver) and onload↩ _acceleration_stack_name (source) for the following Ultra Messaging transport types:

- LBT-RM (UDP-based reliable multicast)
- LBT-RU (UDP-based reliable unicast)
- TCP

Note

If you set the LBM_SUPPRESS_ONLOAD environment variable to any value, Ultra Messaging does not dynamically load the Onload library at runtime. In this case, you cannot use the onload_acceleration_stack_↩ name options.

If you use the onload_set_stackname API directly for any other accelerated sockets, note that after Ultra Messaging accelerates a transport socket, Ultra Messaging resets the stackname to the default for all threads by calling:

onload_set_stackname(ONLOAD_ALL_THREADS, ONLOAD_SCOPE_NOCHANGE, "");

Ultra Messaging resets the stackname during source creation and when a receiver matched topic opens a transport session.

To enable Onload socket acceleration for selected transports, perform the following steps:

1. Install Onload.
2. Set the Onload environment variable EF_DONT_ACCELERATE = 1 to disable Onload default behavior.
3. To enable acceleration for all applications in an environment, export the following environment variable:
   export LD_PRELOAD=libonload.so
4. To enable acceleration on a per-application basis, start the application as in the following example:
   onload <app_name> [app_options]
5. Set UM configuration option onload_acceleration_stack_name (source) according to the thread the source uses.
   Note: Disable batching to ensure that it is the application thread that sends the data out.
6. Set UM configuration option onload_acceleration_stack_name (receiver) according to the thread the receiver uses.
   Note: Receiver transports might not share the same thread if MTT is enabled.
7. Set option transport_∗_datagram_max_size and option resolver_datagram_max_size (context) to a value of no more than 28 bytes smaller than the Solarflare interface's configured MTU size.

Users of Onload are advised to make use of **Dynamic Fragmentation Reduction**.

For detailed information about onload stack names, refer to the Solarflare Onload User Guide.

24.4 Reference

24.4.1 onload_acceleration_stack_name (receiver)

The stackname to use when creating an OpenOnload transport data socket.

The stackname must be eight characters or less. Because this is a transport setting, the first receiver applies its configuration option setting, and other receivers that join the transport inherit the setting of the first source. To disable the stackname, set this option to NULL, which must be all uppercase.

Note: Use of this option requires Solarflare OpenOnload and applies to LBT-RM, LBT-RU, and TCP transports.

Scope:	receiver
Type:	**string**
Default value:	NULL
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.5.

24.4.2 onload_acceleration_stack_name (source)

The stackname to use when creating an OpenOnload transport data socket.

The stackname must be eight characters or less. Because this is a transport setting, the first source applies its configuration option setting, and other sources that join the transport inherit the setting of the first source. To disable the stackname, set this option to NULL, which must be all uppercase.

Note: Use of this option requires Solarflare OpenOnload and applies to LBT-RM, LBT-RU, and TCP transports.

Scope:	source
Type:	**string**
Default value:	NULL
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.5.

24.5 UD Acceleration for Mellanox Hardware Interfaces

UD (Unreliable Datagram) acceleration is a kernel-bypass technology that accelerates sending and receiving UDP traffic and operates with Mellanox 10-Gigabit Ethernet or InfiniBand adapter cards for 64-bit Linux on X86 platforms.

UD acceleration does not support fragmentation and reassembly, so do not send messages larger than the MTU size configured on the Mellanox interface.

UD acceleration is available for the following Ultra Messaging transport types:

- LBT-RM (UDP-based reliable multicast)
- LBT-RU (UDP-based reliable unicast)

- Multicast Immediate Messaging
- Multicast Topic Resolution

To use UD acceleration, perform the following steps:

1. Install the Mellanox NIC.
2. Install the VMA package, which is part of the UD acceleration option .
3. Include the appropriate transport acceleration options in your Ultra Messaging Configuration File.
4. Set option transport_∗_datagram_max_size and option resolver_datagram_max_size (context) to a value of no more than 28 bytes smaller than the Mellanox interface's configured MTU size.

Users of UD acceleration are advised to make use of **Dynamic Fragmentation Reduction**.

24.6 Reference

24.6.1 resolver_ud_acceleration (context)

Flag indicating if Accelerated Multicast is enabled for Topic Resolution. Accelerated Multicast requires Mellanox InfiniBand or 10 Gigabit Ethernet hardware.

UD Acceleration of topic resolution relies on hardware-supported loopback, which InfiniBand provides, but which the 10 Gigabit Ethernet ConnectX hardware does not.

Note: If 10 Gigabit Ethernet ConnectX hardware is used and multiple UM contexts are desired on the host, this option must be disabled.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 5.2.

Value	Description
1	Accelerated Topic Resolution is enabled.
0	Accelerated Topic Resolution is not enabled. **Default for all.**

24.6.2 ud_acceleration (context)

Flag indicating if Accelerated Multicast is enabled for LBT-RM.

Accelerated Multicast requires InfiniBand or 10 Gigabit Ethernet hardware and the purchase and installation of the Ultra Messaging Accelerated Multicast Module. See your Ultra Messaging representative for licensing specifics.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1.

Value	Description
1	Accelerated Multicast is enabled.
0	Accelerated Multicast is not enabled. **Default for all.**

Chapter 25

Smart Source Options

See **Smart Sources** for introductory information on Smart Sources.

25.1 Reference

25.1.1 mem_mgt_callbacks (source)

Callback functions (and optional associated client data pointer) that are called when a Smart Source allocates, reallocates, and deallocates memory.

The callbacks are called by the user thread that invokes **lbm_ssrc_create()** for create, and by **lbm_ssrc_delete()** for delete. See **lbm_mem_mgt_malloc_cb_func**, **lbm_mem_mgt_realloc_cb_func**, **lbm_mem_mgt_free_cb_func**.

See **Smart Sources and Memory Management** for restrictions.

See **Smart Sources** for more information about Smart Sources.

Scope:	source
Type:	**lbm_mem_mgt_callbacks_t**
Default value:	NULL
When to Set:	Can only be set during object initialization.
Config File:	Cannot be set from an UM configuration file.
Version:	This option was implemented in UM 6.11

25.1.2 smart_src_enable_spectrum_channel (source)

This option enables spectrum channel use with Smart Sources.

See **Smart Sources and Spectrum** for restrictions.

See **Smart Sources** for more information about Smart Sources.

Scope:	source
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.11

Value	Description
1	The source will allocate spectrum channel resources.
0	The source will not allocate spectrum channel resources. **Default for all.**

25.1.3 smart_src_max_message_length (source)

The number of bytes allocated for application messages to each Smart Source buffer.

Smart Source buffers are pre-allocated when the source is created. The final allocation size is the value specified for this option, plus the sizes required for internal headers, plus a possible padding value intended to ensure that the final internal buffer allocation is a power of 2. Because of these additions, the actual amount of memory allocated can be over twice as much as requested.

There are three types of buffers sized by smart_src_max_message_length: user buffers, retention buffers (for late join), and transmission window buffers (for transport retransmissions). User buffers and retention buffers are created by **lbm_ssrc_create()**, and are deleted by **lbm_ssrc_delete()**. Transmission window buffers are created only when the first Smart Source on a transport session is created, and are deleted when the last Smart Source on a transport session is deleted.

Different numbers of buffers can be allocated for each buffer type. See smart_src_user_buffer_count (source) for user buffers, transport_lbtrm_smart_src_transmission_window_buffer_count (source) and transport_lbtru_smart_src_transmission_window_buffer_count (source) for transmission window buffers, and smart_src_retention_buffer_count (source) for retention buffers.

The smart_src_max_message_length option affects both the transport session underlying the source and also the source itself. The transport session uses the value from the first source created on the session when it allocates the transmission window; subsequent sources created on the same session do not affect the transmission window. However, the sizes of the user buffers and retention buffers are specific to each Smart Source on a session.

The default value was specifically chosen so that for a Smart Source with no optional headers (no message properties, no spectrum channel, etc.), the total memory consumed per buffer, including internal headers, is 512 bytes.

Note that unlike most UM configuration options, the default value for smart_src_max_message_length is likely to change with new versions of UM. This is because the addition of new capabilities to the Smart Sources feature often requires the addition of internal headers to the message buffer, thus reducing the available user space while staying within the 512-byte total buffer size default target. To assist application designers who want to use the default, the constant **SSRC_DEFAULT_MAX_MSG_LEN** is defined in **lbm.h**.

Also note that the application designer can avoid that uncertainty by simply defining smart_src_max_message_length to be the maximum size of his messages, and allowing the final allocation size of the message buffer to vary by UM version. This is the recommended approach.

See **Smart Sources** for more information about Smart Sources.

Scope:	source
Type:	**int**
Units:	bytes
Default value:	SSRC_DEFAULT_MAX_MSG_LEN (368)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.10

25.1.4 smart_src_message_property_int_count (source)

The maximum number of 32-bit integer message properties that can be set on messages for a particular Smart Source.

See **Smart Sources and Message Properties** for restrictions.

See **Smart Sources** for more information about Smart Sources.

Scope:	source
Type:	**int**
Units:	32-bit integer properties
Default value:	0
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.11

25.1.5 smart_src_retention_buffer_count (source)

The number of Smart Source buffers that are allocated for Late Join and other topic level retransmission features such as Off Transport Recovery.

Once created, the application cannot change the number of buffers. Also, the number of buffers should be a power of 2. If a value is supplied that is not a power of 2, the value is increased to the next larger power of two and a warning message is logged.

The buffer size is determined by smart_src_max_message_length (source), see that option description for more details.

The normal Late Join options "retransmit_retention_∗" do not apply to Smart Sources.

See **Smart Sources** for more information about Smart Sources.

Scope:	source
Type:	**int**
Units:	buffers
Default value:	1024
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.10

25.1.6 smart_src_user_buffer_count (source)

The number of Smart Source buffers that are allocated when the source is created.

Once created, the application cannot change the number of buffers. Also, the number of buffers should be a power of 2. If a value is supplied that is not a power of 2, the value is increased to the next larger power of two and a warning message is logged.

The buffer is sized by the smart_src_max_message_length (source) option.

See **Smart Sources** for more information about Smart Sources.

Scope:	source
Type:	**int**
Units:	buffers
Default value:	32
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.10

25.1.7 transport_lbtrm_smart_src_transmission_window_buffer_count (source)

The number of Smart Source buffers allocated for transport-level retransmissions.

Once created, the application cannot change the number of buffers. Also, the number of buffers should be a power of 2. If a value is supplied that is not a power of 2, the value is increased to the next larger power of two and a warning message is logged.

This option affects the transport session underlying the source rather than the source itself. The transport session uses the value from the first source created on the session and ignores subsequent sources' configuration.

The option smart_src_max_message_length (source) is used to size the buffers (see that option description for more details). This means that the first Smart Source created on the session defines the maximum possible size of user messages for all Smart Sources on the transport session. It is not legal to create a subsequent Smart Source on the same transport session with a larger max message size, although smaller values are permissible.

The normal LBT-RM transmission window options "transport_lbtrm_transmission_window_$*$" do not apply to Smart Sources.

See **Smart Sources** for more information about Smart Sources.

Scope:	source
Type:	**int**
Units:	buffers
Default value:	16384 (16K)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.10

25.1.8 transport_lbtru_smart_src_transmission_window_buffer_count (source)

The number of Smart Source buffers allocated for transport-level retransmissions.

Once created, the application cannot change the number of buffers. Also, the number of buffers should be a power of 2. If a value is supplied that is not a power of 2, the value is increased to the next larger power of two and a warning message is logged.

This option affects the transport session underlying the source rather than the source itself. The transport session uses the value from the first source created on the session and ignores subsequent sources' configuration.

The option smart_src_max_message_length (source) is used to size the buffers (see that option description for more details). This means that the first Smart Source created on the session defines the maximum possible size of user messages for all Smart Sources on the transport session. It is not legal to create a subsequent Smart Source on the same transport session with a larger max message size, although smaller values are permissible.

The normal LBT-RU transmission window options "transport_lbtru_transmission_window_$*$" do not apply to Smart Sources.

Note

If transport_source_side_filtering_behavior (source) is enabled, each connecting receiver will be assigned its own transmission window buffer. As the number of connecting receivers increases, the total memory consumption of the source can become very large.

See **Smart Sources** for more information about Smart Sources.

Scope:	source
Type:	**int**
Units:	buffers
Default value:	16384 (16K)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.11

Chapter 26

Encrypted TCP Options

26.1 Reference

26.1.1 tls_certificate (context)

When TLS is enabled, this option specifies the path to a file containing an OpenSSL-compatible PEM-formatted certificate that will be presented as the TLS server certificate when a TLS connection is established by a client.

For more information, see **Encrypted TCP**.

Scope:	context
Type:	**string**
Default value:	NULL
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.9

26.1.2 tls_certificate_key (context)

When TLS is enabled, this option specifies the path to a file containing the private key associated with the "server" certificate.

The server certificate is specified by the tls_certificate (context) option. Note that this private key must be protected from intruders. For that reason, when the certificate and private key files are generated, the private key file is typically encrypted with a passphrase. The passphrase is supplied using the tls_certificate_key_password (context) option.

For more information, see **Encrypted TCP**.

Scope:	context
Type:	**string**
Default value:	NULL
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.9

26.1.3 tls_certificate_key_password (context)

When TLS is enabled, this option specifies the passphrase needed to decrypt the server private key file.

The private key file is specified by the tls_certificate_key (context) option.

For more information, see **Encrypted TCP**.

Scope:	context
Type:	**string**
Default value:	NULL
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.9

26.1.4 tls_cipher_suites (context)

When TLS is enabled, this option defines the list of one or more (comma separated) names of cipher suites that will be accepted by this context.

See OpenSSL's Cipher Suite Names for the full list of suite names. When configuring UM, use the OpenSSL names (with dashes), not∗ the IANA names (with underscores).

If more than one name is supplied, they should be in descending order of preference. When a remote context negotiates encrypted TCP, the two sides must find a cipher suite in common, otherwise the connection will be canceled.

The default cipher suite is highly secure and is recommended.

For more information, see **Encrypted TCP**.

Scope:	context
Type:	**string**
Default value:	DHE-RSA-AES256-GCM-SHA384
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.9

26.1.5 tls_compression_negotiation_timeout (context)

The number of milliseconds allowed for TLS and/or compression handshake and negotiation.

This negotiation happens when the TCP connection is initiated. If the negotiation does not complete within this amount of time, the connection is canceled. Note that in many cases, this will result in a retry a short time later. If the timeout is caused by mismatched endpoints, it can result in unbounded flapping of the connection.

For more information, see **Encrypted TCP** and/or **Compressed TCP**.

Scope:	context
Type:	**int**
Units:	milliseconds
Default value:	5000
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.9

26.1.6 tls_trusted_certificates (context)

When TLS is enabled, this option specifies the path to a file containing one or more OpenSSL-compatible PEM-formatted TLS client certificates and certificate authorities.

If this option is not supplied, the default behavior is to use the system-level trusted certificates and certificate authorities (operating-system dependent). The TLS server uses these trusted certificates to verify the identity of connecting clients. If a client connects and presents a certificate which is not in the server's trusted certificates file, the connection will be canceled. Note that in many cases, this will result in a retry a short time later, which can lead to unbounded flapping of the connection.

For more information, see **Encrypted TCP**.

Scope:	context
Type:	**string**
Default value:	NULL
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.9

26.1.7 use_tls (context)

This option enables data encryption on all TCP links established within the context.

This includes but may not be limited to TCP transports, Late Join, and Request/Response.

For more information, see **Encrypted TCP**.

Scope:	context
Type:	**int**
Default value:	0
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.9

String value	Integer value	Description
"1"	1	All TCP data will be encrypted.
"0"	0	No encryption will be implemented. **Default for all.**

Chapter 27

Compressed TCP Options

27.1 Reference

27.1.1 compression (context)

This option enables compression and sets the desired data compression algorithm on all TCP links established within the context.

This includes but may not be limited to TCP transports, Late Join, and Request/Response. Currently, only LZ4 lossless data compression is supported.

For more information, see **Compressed TCP**.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.9.

String value	Integer value	Description
"none"	LBM_CTX_ATTR_COMPRESSION_NONE	No compression will be implemented. **Default for all.**
"lz4"	LBM_CTX_ATTR_COMPRESSION_LZ4	All TCP data will be compressed using LZ4.

Chapter 28

Multicast Immediate Messaging Network Options

The multicast address and port used for incoming and outgoing multicast immediate messages can be set with mim_address (context) and mim_destination_port (context) options.

A context may use different multicast addresses and/or ports for incoming and outgoing messages by setting one or more of:

- mim_incoming_address (context)
- mim_outgoing_address (context)
- mim_incoming_destination_port (context)
- mim_outgoing_destination_port (context)

In case of conflict, the most recently set option wins.

As with LBT-RM on multi-homed hosts, the interface UM uses for MIM follows the interface used with multicast topic resolution. See resolver_multicast_interface (context).

Warning

> The addresses and ports you configure for MIM traffic should not overlap with any addresses or ports - or address and port ranges - configured for LBT-RM transports or Topic Resolution traffic. For example, do not use the same multicast address for both Topic Resolution (resolver_multicast_address (context)) and MIM (mim_address (context)). Use different addresses and ports for all multicast address options and port options.

See also **Multicast Immediate Messaging** for general information on MIM.

28.1 Reference

28.1.1 mim_address (context)

> Convenience option to set both the incoming and outgoing multicast addresses for multicast immediate messages.

See mim_outgoing_address (context) and mim_incoming_address (context) for their respective default values. See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**struct in_addr**
Default value:	n.a.
When to Set:	Can only be set during object initialization.

28.1.2 mim_destination_port (context)

The UDP destination port that multicast immediate messages are sent to and received from.

See Port Assignments for more information about configuring ports. See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**lbm_uint16_t**
Default value:	14401
Byte order:	Network
When to Set:	Can only be set during object initialization.

28.1.3 mim_incoming_address (context)

The IP multicast address (or domain name of the multicast address) that multicast immediate messages are received from.

The value 0.0.0.0 disables reception of multicast immediate messages. See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**struct in_addr**

Default value:	0.0.0.0
When to Set:	Can only be set during object initialization.

28.1.4 mim_incoming_destination_port (context)

The UDP destination port that multicast immediate messages are received from.

See Port Assignments for more information about configuring ports. See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**lbm_uint16_t**
Default value:	14401
Byte order:	Network
When to Set:	Can only be set during object initialization.

28.1.5 mim_outgoing_address (context)

The IP multicast address (or domain name of the multicast address) that multicast immediate messages are sent to.

See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**struct in_addr**
Default value:	224.10.10.21
When to Set:	Can only be set during object initialization.

28.1.6 mim_outgoing_destination_port (context)

The UDP destination port that multicast immediate messages are sent to.

See Port Assignments for more information about configuring ports. See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**lbm_uint16_t**
Default value:	14401
Byte order:	Network
When to Set:	Can only be set during object initialization.

Chapter 29

Multicast Immediate Messaging Reliability Options

For every MIM reliability option, there is a corresponding LBT-RM reliability option. For more information on how MIM reliability options interact and for illustrations, see Transport LBT-RM Reliability Options.

See also **Multicast Immediate Messaging** for general information on MIM.

29.1 Reference

29.1.1 mim_ignore_interval (context)

The interval to ignore NAKs after a retransmission is sent.

For multicast immediate message senders only. Similar to transport_lbtrm_ignore_interval (source).

See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	500 (0.5 seconds)
When to Set:	Can only be set during object initialization.

29.1.2 mim_nak_backoff_interval (context)

The maximum interval between transmissions of MIM NAKs for a given sequence number, after the first NAK.

Similar to transport_lbtrm_nak_backoff_interval (receiver).

See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	200 (0.2 seconds)
When to Set:	Can only be set during object initialization.

29.1.3 mim_nak_generation_interval (context)

The maximum time that a piece of data may be outstanding before the data is unrecoverably lost.

For multicast immediate message receivers only. Similar to transport_lbtrm_nak_generation_interval (receiver).

See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	10000 (10 seconds)
When to Set:	Can only be set during object initialization.

29.1.4 mim_nak_initial_backoff_interval (context)

The interval between loss detection and transmission of the first MIM NAK.

For multicast immediate message receivers only. Similar to transport_lbtrm_nak_initial_backoff_interval (receiver).

See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	50 (0.05 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.4/UME 2.1.

29.1.5 mim_nak_suppress_interval (context)

The time that an LBT-RM receiver will suppress sending a NAK for a missing datagram after an NCF is received from the source.

The source sends an NCF in response to a NAK which the source temporarily cannot retransmit. For example, if the source gets a NAK for a sequence number for which it has recently sent a retransmission, it will send an NCF with reason code "ignored". The receiver responds by suppressing NAKs for that sequence number for the interval configured by this option. See **NAK Suppression** for more information about NCFs.

For multicast immediate message receivers only.

See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	1000 (1 second)
When to Set:	Can only be set during object initialization.

29.1.6 mim_send_naks (context)

This flag indicates whether LBT-RM should send negative acknowledgements (NAKs) for missing packets or not.

For multicast immediate message receivers only. Similar to transport_lbtrm_send_naks (receiver).

See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.

Value	Description
1	NAKs are sent for missing packets to request retransmission. **Default for all.**
0	Do not send NAKs for missing packets.

29.1.7 mim_transmission_window_limit (context)

Caps the total amount of memory that a transmission window uses, which includes data and overhead.

For multicast immediate message senders only. Similar to transport_lbtrm_transmission_window_limit (source).

See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**size_t**
Units:	bytes
Default value:	0 (zero)
When to Set:	Can only be set during object initialization.

29.1.8 mim_transmission_window_size (context)

The maximum amount of buffered payload data, excluding UM headers, that the LBT-RM source is allowed to retain for retransmissions.

For multicast immediate message senders only. Similar to transport_lbtrm_transmission_window_size (source).

See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**size_t**
Units:	bytes
Default value:	25165824 (24 MB)
When to Set:	Can only be set during object initialization.

Chapter 30

Multicast Immediate Messaging Operation Options

For many MIM operation options, there is a corresponding LBT-RM operation option. For more information on how MIM operation options interact and for illustrations, see Transport LBT-RM Operation Options.

Note that the LBT-RM rate controller also governs MIM transmission rates. Hence there is no separate option for setting MIM transmission rate.

See also **Multicast Immediate Messaging** for general information on MIM.

30.1 Reference

30.1.1 immediate_message_receiver_function (context)

Callback function (and associated event queue and client data pointer) called when a topicless immediate message is received.

A value of NULL (the default) disables this feature.

Alternatively, the API **lbm_context_rcv_immediate_msgs()** can be used.

See **Immediate Messaging** for general information on immediate messages.

Scope:	context
Type:	**lbm_context_rcv_immediate_msgs_func_t**
Default value:	NULL
When to Set:	Can only be set during object initialization.
Config File:	Cannot be set from an UM configuration file.

30.1.2 immediate_message_topic_receiver_function (context)

Callback function (and associated event queue and client data pointer) that is called when an immediate message is received for a topic for which there is no receiver.

A value of NULL (the default) disables this feature.

Alternatively, the API **lbm_context_rcv_immediate_topic_msgs()** can be used.

See **Immediate Messaging** for general information on immediate messages.

Scope:	context
Type:	**lbm_context_rcv_immediate_msgs_func_t**
Default value:	NULL
When to Set:	Can only be set during object initialization.
Config File:	Cannot be set from an UM configuration file.

30.1.3 mim_activity_timeout (context)

The maximum time that an LBT-RM session may be quiescent before it is deleted and an EOS event is delivered for all topics using this transport session.

For multicast immediate message receivers only. Similar to transport_lbtrm_activity_timeout (receiver). However, multicast immediate message channels do not deliver an EOS indication.

See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	60000 (60 seconds)

When to Set:	Can only be set during object initialization.

30.1.4 mim_delivery_control_activity_check_interval (context)

The interval between activity checks of a Multicast Immediate Messaging delivery controller.

Multiple MIM delivery controllers may exist to accommodate multiple messages from a single MIM sender received across more than one DRO. These multiple delivery controllers allow for duplicate message detection.

See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	5000 (5 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0.

30.1.5 mim_delivery_control_activity_timeout (context)

The maximum time that a Multicast Immediate Messaging delivery controller may be quiescent before it is deleted.

MIM delivery controllers may be created to accommodate multiple messages from a single MIM sender received across more than one DRO. These multiple delivery controllers allow for duplicate message detection.

See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds

Default value:	60000 (60 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0.

30.1.6 mim_delivery_control_order_tablesz (context)

For multicast immediate messages with ordered delivery, this controls the size of the hash table used to hold data.

See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**size_t**
Units:	table entries
Default value:	1031
When to Set:	Can only be set during object initialization.

30.1.7 mim_implicit_batching_interval (context)

The maximum timeout between when the first message of an implicitly batched immediate message is queued until the batch is sent. A message will not stay in the queue longer than this value before being sent in the worst case.

See **Implicit Batching** for details. See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	200 (0.2 seconds)
When to Set:	Can only be set during object initialization.

30.1.8 mim_implicit_batching_minimum_length (context)

The minimum length of an implicitly batched multicast immediate message. When the total length of the implicitly batched messages reaches or exceeds this value, the batch is sent.

See **Implicit Batching** for details. See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**size_t**
Units:	bytes
Default value:	2048 (8192 for Microsoft Windows)
When to Set:	Can only be set during object initialization.

30.1.9 mim_ordered_delivery (context)

For multicast immediate messages only. Indicates whether or not the MIM source should have its data delivered in order.

The default value also guarantees fragmentation and reassembly of large messages. Changing this option from the default value results in large messages being delivered as individual fragments of less than 8K each, requiring the application to reassemble them. See also **Ordered Delivery** for more information about large message fragmentation and reassembly.

See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.

Value	Description
1	Indicates the source should have its data delivered in order. **Default for all.**
0	The source should have its data delivered as soon as possible and may come in out of order.

30.1.10 mim_sm_maximum_interval (context)

The maximum interval between LBT-RM session messages.

For multicast immediate message senders only. Similar to transport_lbtrm_sm_maximum_interval (source).

See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	10000 (10 seconds)
When to Set:	Can only be set during object initialization.

30.1.11 mim_sm_minimum_interval (context)

The minimum interval between LBT-RM session messages.

For multicast immediate message senders only. Similar to transport_lbtrm_sm_minimum_interval (source).

See **Unicast Immediate Messaging** for more information.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	200 (0.2 seconds)
When to Set:	Can only be set during object initialization.

30.1.12 mim_sqn_window_increment (context)

Determines the increment by which the sequence number window is moved when detecting the receipt of duplicate multicast immediate messages.

For multicast immediate message receivers only.

Must be a multiple of 8 and an even divisor of mim_sqn_window_size (context).

See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**lbm_ulong_t**
Units:	messages
Default value:	8192
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.2.8/UME 3.2.8/UMQ 2.1.8

30.1.13 mim_sqn_window_size (context)

For multicast immediate message receivers only. Determines the window size used to detect the receipt of duplicate multicast immediate messages. Must be a multiple of 8.

See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**lbm_ulong_t**
Units:	messages
Default value:	16384
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.2.8/UME 3.2.8/UMQ 2.1.8

30.1.14 mim_src_deletion_timeout (context)

The timeout after a multicast immediate message is sent before the internal source is deleted and cleaned up.

See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	30000 (30 seconds)
When to Set:	Can only be set during object initialization.

30.1.15 mim_tgsz (context)

The transmission group size used for this Topic when LBT-RM is used.

For multicast immediate message senders only. Similar to transport_lbtrm_tgsz (source).

See **Unicast Immediate Messaging** for more information.

Scope:	context
Type:	**lbm_uint16_t**
Units:	packets
Default value:	8
When to Set:	Can only be set during object initialization.

30.1.16 mim_unrecoverable_loss_function (context)

Callback function (and associated client data pointer) that is called when a MIM receiver has unrecoverable loss.

This callback is called by the context thread and can not use an event queue. Therefore the callback function used should not block or it will delay reception of latency-sensitive messages.

See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**lbm_mim_unrecloss_func_t**
Default value:	NULL
When to Set:	Can only be set during object initialization.
Config File:	Cannot be set from an UM configuration file.

Chapter 31

Late Join Options

Late Join allows sources to save a predefined amount of their messaging traffic for late-joining receivers. Sources set the configuration options that determine whether they use Late Join or not, and receivers set options that determine whether they will participate in Late Join recovery if sources use Late Join.

UMP's persistent store is built on Late Join technology. In the Estimating Recovery Time discussion below, the terms Late Join buffers and UMP store are roughly equivalent.

For more, review *Late Join* in the *Ultra Messaging Concepts Guide*, especially *Configuring Late Join for Large Numbers of Messages*.

31.1 Estimating Recovery Time

Late Join message recovery time is a function of how much data must be recovered and how fast messages are retransmitted. To estimate Late Join recovery time **R** in minutes, use the formula:

```
R = D / ( 1 - ( txrate / rxrate ) )
```

where:

D is the downtime (in minutes) across all receivers
txrate is the average source transmission rate of normal (live stream) messages during recovery (in kmsgs/sec).
rxrate is the average source retransmission rate from source-side Late Join buffers during recovery (in kmsgs/sec). This rate needs to be greater than **txrate**.

For example, consider the following scenario:

D = 10 minutes
txrate = 10k messages / second
rxrate = 25k messages / second

Plugging these values into the formula gives an estimated recovery time in minutes:

```
R = 10 / ( 1 - ( 10 / 25 ) )
```

or 16.67 minutes. Note that this formula assumes the following:

- Retransmit rate(rxrate) is as linear as possible with use of option response_tcp_nodelay (context) to 1.
- Transmit rate (txrate) from *all* relevant sources is fairly constant and equal

- Retransmit rate (rxrate) from Late Join buffers is fairly constant and equal, and should be measured in a live test, if possible. You can adjust the recovery rate with two Late Join configuration options: retransmit_request_outstanding_maximum (receiver) and retransmit_request_interval (receiver).

31.2 Reference

31.2.1 late_join (source)

Configure the source to enable both Late Join and Off-Transport Recovery operation for receivers.

See **Using Late Join** and **Off-Transport Recovery (OTR)**.

Scope:	source
Type:	**int**
When to Set:	Can only be set during object initialization.

Value	Description
1	Enable source for Late Join and OTR. (Forced on for Persistence.)
0	Disable source for Late Join and OTR. **Default for all.**

31.2.2 late_join_info_request_interval (receiver)

The interval at which the receiver requests a Late Join Information Record (LJI) from the source.

Controlling these requests helps reduce receiver start-up traffic on your network.

See **Late Join**.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	1000 (1 second)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMP 6.0

31.2.3 late_join_info_request_maximum (receiver)

The maximum number of requests the receiver issues for a Late Join Information Record (LJI) from the source.

If the receiver has not received an LJI after this number of requests, it stops requesting.

Scope:	receiver
Type:	**lbm_ulong_t**
Default value:	60
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMP 6.0

31.2.4 retransmit_initial_sequence_number_request (receiver)

When a late-joining receiver detects (from the topic advertisement) that a source is enabled for Late Join but has sent no messages, this flag option lets the receiver request an initial sequence number from a source.

Sources respond with a TSNI.

Scope:	receiver
Type:	**int**
Default value:	1
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.2.

Value	Description
1	The receiver requests an initial sequence number from Late Join enabled sources that have not sent any messages. **Default for all.**
0	The receiver does not request an initial sequence number.

31.2.5 retransmit_message_caching_proximity (receiver)

This option determines how a receiver handles new messages that are being published while the receiver is in the process of recovering older messages through the retransmit request mechanism.

A receiver has the ability to cache new messages during the recovery process in order to facilitate a smooth transition from recovery to live stream. This option value determines how close (proximate) a newly received message sequence number must be to the latest retransmitted sequence number for the receiver to cache it. New messages that arrive while the receiver is not within proximity will be discarded, and the receiver will attempt to recover those messages later via OTR.

An option value between 0 and 0x7FFFFFFE (2,147,483,646) enables proximity caching, with larger values allowing caching to begin earlier during recovery. Values 0x7FFFFFFF and above disable proximity caching. This value has meaning for only receivers using ordered delivery of data.

See **Configuring Late Join for Large Numbers of Messages** for additional information.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	messages
Default value:	5000 (was 0xFFFFFFFF = 4,294,967,295 in versions prior to 6.8)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.3.2/UME 2.0.

31.2.6 retransmit_request_interval (receiver)

The interval between retransmission request messages to the source.

See **Configuring Late Join for Large Numbers of Messages** for additional information.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	500 (0.5 seconds)
When to Set:	Can only be set during object initialization.

31.2.7 retransmit_request_maximum (receiver)

The maximum number of messages to request, counting backward from the current latest message, when late-joining a topic.

Due to network timing factors, UM may transmit an additional message. For example, a value of 5 sends 5 or possibly 6 retransmit messages to the new receiver. (Hence, you cannot request and be guaranteed to receive only 1 last message–you may get 2.) A value of 0 indicates no maximum.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	messages
Default value:	0
When to Set:	Can only be set during object initialization.

31.2.8 retransmit_request_message_timeout (receiver)

The maximum time from when a receiver first sends a retransmission request to when the receiver gives up on receiving the retransmitted message and reports loss.

See **Configuring Late Join for Large Numbers of Messages** for additional information.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	10000 (10 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.0

31.2.9 retransmit_request_outstanding_maximum (receiver)

The maximum number of messages to request and to remain active (pending) at a single time.

Value must be greater than zero.

If this option is increased significantly, retransmit_request_interval (receiver) should also be increased.

See **Configuring Late Join for Large Numbers of Messages** for additional information.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	messages
Default value:	10
When to Set:	Can only be set during object initialization.

31.2.10 retransmit_retention_size_limit (source)

Sets a maximum limit on the size of the source's retransmit retention buffer when using a persistent Store.

With UME, stability and delivery confirmation events can delay the deletion of retained messages, which can increase the size of the buffer above the retransmit_retention_size_threshold (source). Hence, this option

provides a hard size limit. UM sets a minimum value for this option of 8K for UDP and 64K for TCP, and issues a log warning if you set a value less than the minimum.

With **Smart Sources**, this option is ignored. Retention buffers are preallocated.

Scope:	source
Type:	**size_t**
Units:	bytes
Default value:	25165824 (24 MB)
When to Set:	Can only be set during object initialization.

31.2.11 retransmit_retention_size_threshold (source)

Specifies the minimum size of the retained message buffer before UM can delete messages.

The buffer must reach this size before UM can delete any messages older than retransmit_retention_age_↩ threshold (source).

For persistence, these options combined with retransmit_retention_size_limit (source) affect the retention buffer size. A value of 0 sets the size threshold to be always triggered, in which case deletion is determined by other threshold criteria.

With **Smart Sources**, this option is ignored. Retention buffers are preallocated and are never deleted.

Scope:	source
Type:	**size_t**
Units:	bytes
Default value:	0 (threshold always triggered)
When to Set:	Can only be set during object initialization.

31.2.12 use_late_join (receiver)

Flag indicating if the receiver should participate in a late join operation or not.

See **Late Join** for more information.

Scope:	receiver
Type:	**int**
When to Set:	Can only be set during object initialization.

Value	Description
1	The receiver will participate in using late join if requested to by the source. **Default for all.**
0	The receiver will not participate in using late join even if requested to by the source.

Chapter 32

Off-Transport Recovery Options

See also **Off-Transport Recovery (OTR)** for general information on OTR.

32.1 Reference

32.1.1 otr_message_caching_threshold (receiver)

Number of messages in the Delivery Controller's Order Map above which UM will trigger OTR to try to recover the messages.

This option only applies for receivers that are enabled for **Off-Transport Recovery (OTR)**. See **Delivery Controller** for a description of the Order Map.

The purpose for this option is to speed up recovery in the presence of loss. The delivery controller normally delays for otr_request_initial_delay (receiver) before initiating OTR. This is to give the transport layer time to recover the lost datagram through its more efficient protocol. However, if the number of datagrams waiting for recovery grows too large, it might indicate that the transport layer is unable to recover the datagrams. In this case, it can be helpful to bypass the normal OTR initial delay and immediately initiate OTR.

For environments that are subject to severe loss events, and has expanded the source's transport transmission windows to accommodate, this option should typically be increased above its default to prevent premature OTR.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	messages
Default value:	10000
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.0

32.1.2 otr_request_initial_delay (receiver)

The length of time a receiver waits before initiating OTR to recover lost datagrams.

Note that unlike transport-level NAKing, this setting is not specific to each lost datagram. Rather the Delivery Controller is either "in" OTR mode or it is not. This delay time controls the entry into OTR mode. Once that happens, the OTR feature will request individual datagrams according to its internal algorithms.

See **Off-Transport Recovery (OTR)**.

There are other conditions that can initiate OTR, like the Delivery Controller's Order Map growing too large. In these cases, OTR can begin prior to the configured initial delay time. See otr_message_caching_threshold (receiver).

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	2000 (2 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 5.3

32.1.3 otr_request_log_alert_cooldown (receiver)

Each OTR request generates a log message. The first request's log message is a WARNING-level log message, and subsequent requests that quickly follow generate INFO-level log messages. After a time interval defined by this option, the next request leading a new burst of requests again generates a WARNING-level log message.

See **Off-Transport Recovery (OTR)**.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	seconds

Default value:	300 (5 minutes)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 5.3

32.1.4 otr_request_maximum_interval (receiver)

The maximum time interval between a receiver's OTR lost-message requests.

After the receiver initiates OTR and is waiting to receive the retransmission, the initial interval (set by otr_request_minimum_interval (receiver)) doubles in length for each request until it reaches this option's value, then continues at this interval (until timeout or UM recovers messages).

Note

When using TCP Request/Response, this value must be shorter than response_tcp_deletion_timeout (context).

See **Off-Transport Recovery (OTR)**.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	10000 (10 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 5.3

32.1.5 otr_request_message_timeout (receiver)

The maximum time from when a receiver first sends an OTR lost-message request to when the receiver gives up on receiving the retransmitted message and reports loss.

See **Off-Transport Recovery (OTR)**.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	60000 (60 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.0

32.1.6 otr_request_minimum_interval (receiver)

The initial time interval between a receiver's OTR lost-message requests.

While the receiver is waiting to receive the retransmission, the interval doubles in length for each request until it reaches the maximum interval set by otr_request_maximum_interval (receiver).

See **Off-Transport Recovery (OTR)**.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	1000 (1 second)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 5.2

32.1.7 otr_request_outstanding_maximum (receiver)

The maximum number of OTR lost-message requests outstanding at any given time. Each message specifies an individual lost message. A value of 0 indicates no maximum.

See **Off-Transport Recovery (OTR)**.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	messages
Default value:	200
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 5.2

32.1.8 use_otr (receiver)

Flag indicating if the receiver can use OTR or not.

See **Off-Transport Recovery (OTR)**.

Scope:	receiver
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 5.2

String value	Integer value	Description
"0"	0	The receiver is not enabled to use OTR to recover lost messages.
"1"	1	The receiver is enabled to use OTR to recover lost messages.
"2"	2	If the receiver is a persistent receiver, the receiver is enabled to use OTR to recover lost messages. **Default for all.**

Chapter 33

Unicast Immediate Messaging Network Options

In early versions of UM, the Unicast Immediate Messaging (UIM) feature was primarily used to support the **Request**/**Response** feature. Therefore, the configuration options related to UIMs have names that start with "request" and "response". However, as UM has evolved, the UIM feature has come to be used by a great many UM features, such as Late Join, Persistence, and Queuing.

To maintain backwards compatibility, the old names of the configuration options have been retained. The reader must simply be aware that the "request_..." and "response_..." options affect more than just the request/response feature.

See **Unicast Immediate Messaging** for general information on UIM. See also Unicast Immediate Messaging Operation Options for operationally-oriented options.

33.1 Reference

33.1.1 request_tcp_bind_request_port (context)

Allows you to turn off UIM port binding (also known as "request port binding").

Setting this option to 0 prevents sockets from being bound to the UIM port. Turning off UIM port binding also turns off several UM features such as: **Request**/**Response Model**, **Using Late Join**, **Off-Transport Recovery (OTR)**, the reception of Unicast Immediate Messages, persistence, brokered queuing, and ULB.

See **Unicast Immediate Messaging** for general information on UIM.

Scope:	context
Type:	**int**
Default value:	1

When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.3.7/UME 2.0.5.

Value	Description
1	Set UIM port binding. **Default for all.**
0	Turn off UIM port binding.

33.1.2 request_tcp_interface (context)

Specifies the network interface over which UM accepts TCP connections for reception of UIM messages.

You can specify a full IP address of interface, or just the network part (see Specifying Interfaces for details).

Default is set to default_interface (context), if specified. Otherwise, it is set to INADDR_ANY, meaning that it will not bind to a specific interface. You can also set this option to 0.0.0.0/0 which produces the same result.

See **Unicast Immediate Messaging** for general information on UIM.

Note: if specifying an interface name in an XML-format file, see Interface Device Names and XML.

Scope:	context
Type:	**lbm_ipv4_address_mask_t**
Default value:	0.0.0.0 (INADDR_ANY)
When to Set:	Can only be set during object initialization.

33.1.3 request_tcp_port (context)

Port number used for UIM port (also known as "request port").

A context binds to and listens on the UIM port to be able to accept TCP connections for reception of Unicast Immediate Messages (UIMs). The port is either explicitly specified by request_tcp_port (context), or is selected from the range: [request_tcp_port_low (context), request_tcp_port_high (context)].

If request_tcp_port (context) is 0, the context binds to the first open port within the range of [request_tcp_port_low (context), request_tcp_port_high (context)]. If nonzero, the specific port number is used instead.

See **Unicast Immediate Messaging** for general information on UIM. See Port Assignments for more information about configuring ports.

Scope:	context
Type:	**lbm_uint16_t**
Default value:	0 (use open port)
Byte order:	Network
When to Set:	Can only be set during object initialization.

33.1.4 request_tcp_port_high (context)

High port number to use for UIM port (also known as "request port").

A context binds to and listens on the UIM port to be able to accept TCP connections for reception of Unicast Immediate Messages (UIMs). The port is either explicitly specified by request_tcp_port (context), or is selected from the range: [request_tcp_port_low (context), request_tcp_port_high (context)].

See **Unicast Immediate Messaging** for more information about UIM. See Port Assignments for more information about configuring ports.

Scope:	context
Type:	**lbm_uint16_t**
Default value:	14395
Byte order:	Host
When to Set:	Can only be set during object initialization.

33.1.5 request_tcp_port_low (context)

Low port number to use for UIM port (also known as "request port").

A context binds to and listens on the UIM port to be able to accept TCP connections for reception of Unicast Immediate Messages (UIMs). The port is either explicitly specified by request_tcp_port (context), or is selected from the range: [request_tcp_port_low (context), request_tcp_port_high (context)].

See **Unicast Immediate Messaging** for general information on UIM. See Port Assignments for more information about configuring ports.

Scope:	context
Type:	**lbm_uint16_t**
Default value:	14391
Byte order:	Host
When to Set:	Can only be set during object initialization.

Chapter 34

Unicast Immediate Messaging Operation Options

In early versions of UM, the Unicast Immediate Messaging (UIM) feature was primarily used to support the **Request/Response** feature. Therefore, the configuration options related to UIMs have names that start with "request" and "response". However, as UM has evolved, the UIM feature has come to be used by a great many UM features, such as Late Join, Persistence, and Queuing.

To maintain backwards compatibility, the old names of the configuration options have been retained. The reader must simply be aware that the "request_..." and "response_..." options affect more than just the request/response feature.

See **Unicast Immediate Messaging** for general information on UIM. See also Unicast Immediate Messaging Network Options for network-oriented options.

34.1 Reference

34.1.1 request_tcp_exclusiveaddr (context)

Controls whether the context sets SO_EXCLUSIVEADDRUSE before it binds to the UIM port (also known as the "Request Port").

Applicable only to Windows.

The default setting in Windows allows multiple binds to the same port. By default, UM will set SO_EXCLUSIVEADDRUSE to minimize port sharing. Refer to Microsoft's web site for more information on SO_EXCLUSIVEADDRUSE.

See **Unicast Immediate Messaging** for general information on UIM.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.

Value	Description
1	Set SO_EXCLUSIVEADDRUSE. **Default for Windows.**
0	Do not set SO_EXCLUSIVEADDRUSE.

34.1.2 request_tcp_listen_backlog (context)

The backlog used in the TCP listen() call to set the queue length for incoming UIM connections (also known as "request connections" or "response connections").

See **Unicast Immediate Messaging** for general information on UIM.

Scope:	context
Type:	**int**
Default value:	5
When to Set:	Can only be set during object initialization.

34.1.3 request_tcp_reuseaddr (context)

Controls whether the context sets SO_REUSEADDR before it binds to the UIM port (also known as the "↩ Request Port").

See **Unicast Immediate Messaging** for general information on UIM.

Warning

This option is not recommended for Microsoft Windows users because the SO_REUSEADDR socket option in Windows allows a socket to forcibly bind to a port in use by another socket. Multiple sockets using the same port results in indeterminate behavior.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.

Value	Description
1	Set SO_REUSEADDR.
0	Do not set SO_REUSEADDR. **Default for all.**

34.1.4 response_session_maximum_buffer (context)

Maximum number of bytes of application data which can be queued for a UIM connection.

When the application sends a UIM message via a UIM API function, UM may not be able to immediately send the message. For example, if many messages are bring sent but the receiver is slow, TCP flow control may prevent messages from being sent. UM will queue outgoing UIM messages that cannot be sent immediately. If that queue fills, then the UIM send API will either block, or will return -1 with the error code LBM_EWOULD_↩ BLOCK.

This queue is shared across all API methods of sending UIMs, including **lbm_unicast_immediate_message()**, **lbm_send_response()**, etc.

See **Unicast Immediate Messaging** for general information on UIM.

Scope:	context
Type:	**lbm_ulong_t**
Units:	bytes
Default value:	65536
When to Set:	Can only be set during object initialization.

34.1.5 response_session_sender_socket_buffer (context)

Value used to set the SO_SNDBUF value of the UIM connection.

In some cases the OS will not allow all of this value to be used. A value of 0 instructs UM to use the OS defaults. See Socket Buffer Sizes for platform-dependent information.

See **Unicast Immediate Messaging** for general information on UIM.

Scope:	context
Type:	**lbm_ulong_t**
Units:	bytes
Default value:	0 (use TCP autotuning)
When to Set:	Can only be set during object initialization.

34.1.6 response_tcp_deletion_timeout (context)

Time period that the context waits before deleting a UIM connection.

UIM connections are dynamic, being created when needed and deleted when no longer needed. The purpose of this timer is to keep the TCP connection up for a time after it is no longer needed, just in case it becomes needed again. The exact semantics of this timer are described in **Unicast Immediate Messaging**.

NOTE: When using Off-Transport Recovery (OTR), this value must be longer than otr_request_maximum_↩ interval (receiver).

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	20,000 (20 seconds)
When to Set:	Can only be set during object initialization.

34.1.7 response_tcp_interface (context)

Specifies the network interface over which UM initiates outgoing TCP connections for UIMs.

You can specify the full IP address of interface, or just the network part (see Specifying Interfaces for details).

Default is set to default_interface (context), if specified. Otherwise, it is set to INADDR_ANY, meaning that it will not bind to a specific interface. You can also set this option to 0.0.0.0/0 which produces the same result.

See **Unicast Immediate Messaging** for general information on UIM.

Note: if specifying an interface name in an XML-format file, see Interface Device Names and XML.

Scope:	context
Type:	**lbm_ipv4_address_mask_t**
Default value:	0.0.0.0 (INADDR_ANY)
When to Set:	Can only be set during object initialization.

34.1.8 response_tcp_nodelay (context)

Controls whether the context sets TCP_NODELAY before it binds to the UIM port (also known as the "Request Port").

Setting TCP_NODELAY disables Nagle's algorithm, which somewhat decreases the efficiency and throughput of TCP, but decreases the latency of individual messages.

See **Unicast Immediate Messaging** for general information on UIM.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.

Value	Description
1	TCP response sockets should set TCP_NODELAY (disable Nagle).
0	TCP response sockets should not set TCP_NODELAY (leave Nagle enabled). **Default for all.**

Chapter 35

Implicit Batching Options

35.1 Reference

35.1.1 implicit_batching_interval (source)

The maximum timeout between when the first message of an implicit batch is queued until the batch is sent. A message will not stay in the queue longer than this value before being sent in the worst case.

See **Implicit Batching** for details.

Scope:	source
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	200 (0.2 seconds)
When to Set:	May be set during operation.

35.1.2 implicit_batching_minimum_length (source)

The minimum length of an implicitly batched message. When the total length of the implicitly batched messages reaches or exceeds this value, the batch is sent.

See **Implicit Batching** for details.

Scope:	source
Type:	**size_t**
Units:	bytes
Default value:	2048 (8192 for Microsoft Windows)
When to Set:	May be set during operation.

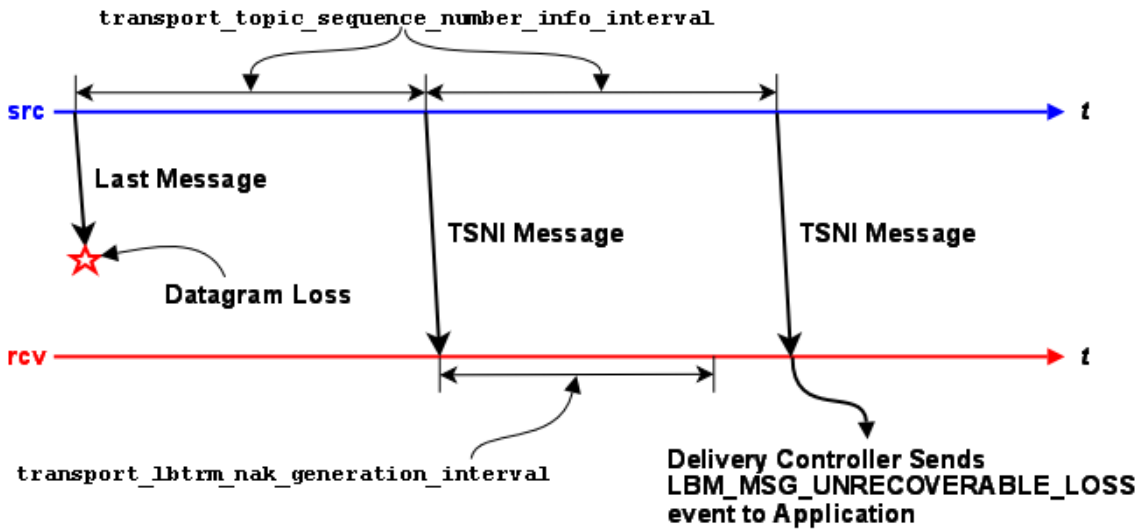
Chapter 36

Delivery Control Options

A Delivery Controller is a receiver-side object created for each source identified by the receiver through topic resolution. A delivery controller performs the following.

- Delivers messages to multiple receivers subscribed to the same topic.
- Orders received topic messages if ordered_delivery (receiver) is set to 1 (default). This option applies to LBT-RU and LBT-RM transports.
- Determines unrecoverable loss and burst loss events for the receiver's topic over LBT-RU and LBT-RM transports.

Unlike the loss depicted in LBT-RM, the image below illustrates how a receiver's Delivery Controller detects unrecoverable tail loss on a topic.



In a non-tail-loss case, the TSNI messages shown above can also be application messages. The point being that the delivery controller does not send NAKs, and instead waits for a transport_lbtrm_nak_generation_interval (receiver) period after the point where the gap is detected (either by an application message or by a TSNI). During that wait interval, the transport may deliver retransmitted message. If not, it is the reception of *another* message or TSNI after the NAK generation interval expires which triggers delivery of the unrecoverable loss event.

Note

if the source disables TSNIs, tail loss can go undetected unless and until another application is sent on that topic.

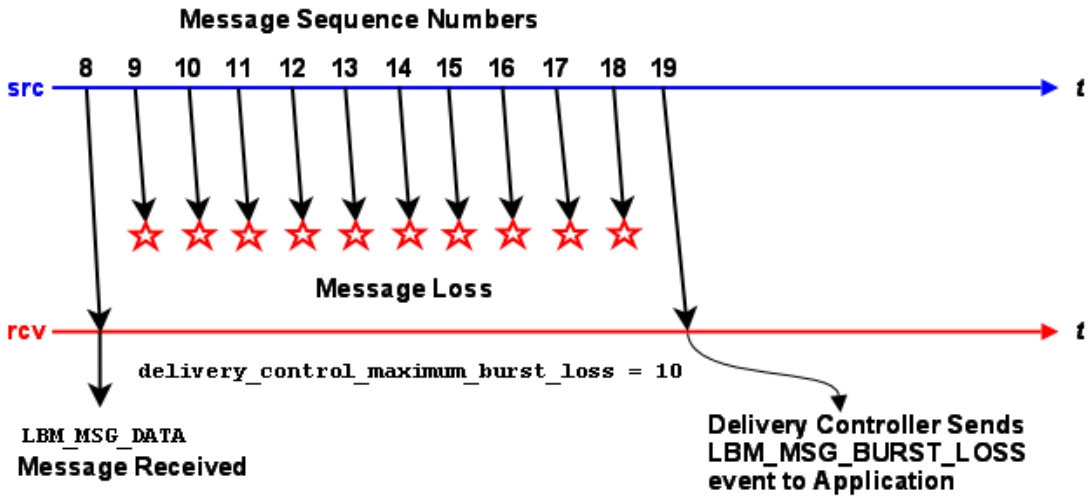
36.1 Burst Loss

Normally, when the delivery controller detects a gap in topic sequence numbers of received message fragments, it waits for a NAK generation interval (defaults to 10 seconds) before declaring the missing message fragments unrecoverably lost. This wait time allows the underlying transport layer to attempt to retrieve the missing message fragments.

The configuration options delivery_control_maximum_burst_loss (receiver) and delivery_control_maximum_burst_loss (hfx) specify a size for a contiguous gap in topic sequence numbers beyond which the gap is defined to be a "burst loss". When this happens, the delivery controller *immediately* declares the entire gap to be unrecoverably lost and resets its loss-handling structures. Thus, even if the underlying transport layer is subsequently able to retrieve some or all of the missing message fragments, the delivery controller will discard them (since they are already declared unrecoverably lost).

The purpose of this is to prevent long delays for large loss events for which the chances of successful recover are very small.

The image below illustrates this.



For burst loss, a single **LBM_MSG_UNRECOVERABLE_LOSS_BURST** event is delivered for the entire sequence number gap. (Contrast this with simple (not burst) loss events, where a separate **LBM_MSG_UNRECOVERABLE_LOSS** event will be delivered to the receiver for each lost sequence number.)

Note

The burst loss control takes priority over all recovery methods. For example, if the receiver is reading a persistent stream and OTR is enabled, a gap longer than delivery_control_maximum_burst_loss will immediately declare the gap as unrecoverable without even trying to use OTR to recover. If gapless message delivery is a high priority, delivery_control_maximum_burst_loss should be set to a very large value.

There is a possibility of successfully-received messages being discarded when a burst loss is detected. Let's say a minor loss event is followed by several successful message fragments. The delivery of those successfully-received message fragments will be delayed in hopes that the underlying transport layer can retrieve the missing data. However, if a burst loss is detected while the delivery controller is still waiting for recovery, the pending messages will be deleted as the loss-handling structures are cleaned up.

36.2 Reference

36.2.1 channel_map_tablesz (receiver)

The size of the hash table that the receiver uses to store channel subscriptions.

A larger table means more channels can be stored more efficiently, but takes up more memory. A smaller table uses less memory, but costs more CPU time for large numbers of channel subscriptions.

See **Spectrum** for more information.

Scope:	receiver
Type:	**size_t**
Default value:	10273
When to Set:	Can only be set during object initialization.

36.2.2 delivery_control_loss_check_interval (receiver)

This controls the interval between mandatory topic loss checks for a receiver. See Preventing Undetected Unrecoverable Loss.

A value of 0 turns this loss check off.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds

Default value:	0 (disabled)
When to Set:	Can only be set during object initialization.

36.2.3 delivery_control_maximum_burst_loss (receiver)

This controls the size of a topic sequence number gap past which the gap is declared a "burst loss".

See Burst Loss for a detailed explanation of burst loss and its semantics.

Note

the burst loss control takes priority over all recovery methods. For example, if the receiver is reading a persistent stream and OTR is enabled, a gap longer than delivery_control_maximum_burst_loss will immediately declare the gap as unrecoverable without even trying to use OTR to recover. If message integrity is a high priority, delivery_control_maximum_burst_loss should be set to a very large value.

Scope:	receiver
Type:	**lbm_uint_t**
Units:	number of messages (fragments)
Default value:	1024
When to Set:	Can only be set during object initialization.

36.2.4 delivery_control_maximum_total_map_entries (context)

The maximum number of messages that can be buffered in the Delivery Controller's Order Map.

When the number of messages stored in a Delivery Controller's Order Map is exceeded, unrecoverable loss is signaled for the oldest gaps and older data is delivered until the Order Map size is below delivery_control_↩ maximum_total_map_entries.

A value of 0 implies no maximum value setting and allows unbounded growth of the Delivery Controller's Order Map.

See **Delivery Controller** for a description of the Order Map. Also see otr_message_caching_threshold (receiver).

For a persistent receiver that has OTR enabled, this option is typically set to 0 (no limit). This is because the option retransmit_message_caching_proximity (receiver) prevents unbounded growth of the Order Map.

Note

Although this option is context scoped, understand that there is a separate Order Map for each Delivery Controller. Those Order Maps are sized independently.

Scope:	context
Type:	**size_t**
Units:	map entries
Default value:	200000
When to Set:	Can only be set during object initialization.

36.2.5 delivery_control_message_batching (context)

Controls whether or not to use receive-side batching, which can improve receiver throughput when using event queues, but might add latency in other cases.

If you enable this option, and you use an event queue that is in polling mode, using lbm_event_dispatch(evq, LBM_EVENT_QUEUE_POLL), then rather than dispatching exactly one event per call to lbm_event_dispatch, you may get multiple events dispatched with a single call.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.

Value	Description
1	Receive-side batching is enabled.

Value	Description
0	Receive-side batching is disabled. **Default for all.**

36.2.6 mim_delivery_control_loss_check_interval (context)

This controls the interval between mandatory loss checks for MIM.

A value of 0 turns this loss check off.

See **Multicast Immediate Messaging** for general information about MIM.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	0 (disabled)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.2

36.2.7 null_channel_behavior (receiver)

Behavior desired when a message without channel information (i.e. a standard UM message) is received by UM.

See **Spectrum** for more information.

Scope:	receiver
Type:	**int**
When to Set:	Can only be set during object initialization.

String value	Integer value	Description
"deliver"	LBM_RCV_TOPIC_ATTR_CHANNEL_B↩ EHAVIOR_DELIVER_MSGS	Messages sent without channel information will be delivered to the callback specified upon receiver creation. **Default for all.**
"discard"	LBM_RCV_TOPIC_ATTR_CHANNEL_B↩ EHAVIOR_DISCARD_MSGS	Messages sent without channel information will be discarded.

36.2.8 source_notification_function (receiver)

Callback functions (and associated client data pointer) that are called when a UM receiver creates or deletes a delivery controller associated with a source.

A receiver can have zero or more **Delivery Controllers**. Each Delivery Controller maintains internal state for a specific source that the receiver has joined. The application callbacks associated with this configuration option allow the application to track the receiver joining and exiting of the individual source.

For the creation function, the application has the ability to set the source client data pointer to be used in each message received from the source.

Contrast this with resolver_source_notification_function (context).

This callback is called by the context thread and can not use an event queue. Therefore the callback function used should not block or it will delay reception of latency-sensitive messages.

Scope:	receiver
Type:	**lbm_rcv_src_notification_func_t**
Default value:	NULL
When to Set:	Can only be set during object initialization.
Config File:	Cannot be set from an UM configuration file.

36.2.9 unrecognized_channel_behavior (receiver)

Behavior desired when a message with channel information for a channel not in the receiver's set of subscribed channels is received by UM.

See **Spectrum** for more information.

Scope:	receiver
Type:	**int**
When to Set:	Can only be set during object initialization.

String value	Integer value	Description
"deliver"	LBM_RCV_TOPIC_ATTR_CHANNEL_B↩ EHAVIOR_DELIVER_MSGS	Messages sent with channel information for a channel not in the receiver's set of subscribed channels will be delivered to the callback specified upon receiver creation. **Default for all.**
"discard"	LBM_RCV_TOPIC_ATTR_CHANNEL_B↩ EHAVIOR_DISCARD_MSGS	Messages sent with channel information for a channel not in the receiver's set of subscribed channels will be discarded.

Chapter 37

Wildcard Receiver Options

37.1 Reference

37.1.1 pattern_type (wildcard_receiver)

The type of expression UM uses to compare wildcard receiver patterns to new topics seen in topic advertisements or responses to wildcard receiver queries.

As of UM Version 6.1, wildcard receivers must use PCRE expressions.

Scope:	wildcard_receiver
Type:	**int**
When to Set:	Can only be set during object initialization.

String value	Integer value	Description
"pcre"	LBM_WILDCARD_RCV_PATT↩ ERN_TYPE_PCRE	The pattern is a regular expression usable by PCRE (Perl Compatible Regular Expressions) library. **Default for all.**
"regex" Deprecated in UM Version 6.1.	LBM_WILDCARD_RCV_PATT↩ ERN_TYPE_REGEX	The pattern is a regular expression usable by POSIX Extended Regular Expressions.

String value	Integer value	Description
"appcb" Deprecated in UM Version 6.1.	LBM_WILDCARD_RCV_PATTERN_TYPE_APP_CB	The wildcard receiver ignores the pattern and calls an application callback set by the pattern_callback (wildcard_receiver) option.

37.1.2 receiver_create_callback (wildcard_receiver)

Callback function (and associated client data pointer) that is called when a receiver is about to be created for a topic which matched a wildcard receiver pattern.

This callback is called by the context thread and can not use an event queue. Therefore the callback function used should not block or it will delay reception of latency-sensitive messages.

The callback function should always return 0.

Scope:	wildcard_receiver
Type:	**lbm_wildcard_rcv_create_func_t**
Default value:	NULL
When to Set:	Can only be set during object initialization.
Config File:	Cannot be set from an UM configuration file.
Version:	This option was implemented in LBM 3.4/UME 2.1.

37.1.3 receiver_delete_callback (wildcard_receiver)

Callback function (and associated client data pointer) that is called when a receiver is about to be deleted.

This callback is called by the context thread and can not use an event queue. Therefore the callback function used should not block or it will delay reception of latency-sensitive messages.

The callback function should always return 0.

Scope:	wildcard_receiver
Type:	**lbm_wildcard_rcv_delete_func_t**
Default value:	NULL
When to Set:	Can only be set during object initialization.
Config File:	Cannot be set from an UM configuration file.
Version:	This option was implemented in LBM 3.4/UME 2.1.

37.1.4 resolver_no_source_linger_timeout (wildcard_receiver)

This sets the linger timeout value before a topic with no sources is removed and cleaned up.

Since wildcard receivers set the resolution_no_source_notification_threshold (receiver) to 10, the linger timer starts after the wildcard receiver sends 10 queries and subsequently receives a no-source notification.

Scope:	wildcard_receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	1000 (1 second)
When to Set:	Can only be set during object initialization.

37.1.5 resolver_query_maximum_interval (wildcard_receiver)

The longest - and last - interval in wildcard receiver topic querying.

A value of 0 disables wildcard receiver topic querying.

See also Disabling Aspects of Topic Resolution.

Scope:	wildcard_receiver
Type:	**lbm_ulong_t**
Units:	milliseconds

Default value:	1000 (1 second)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0

37.1.6 resolver_query_minimum_duration (wildcard_receiver)

The duration of wildcard queries in wildcard receiver topic querying.

Only PCRE and regex pattern types can use wildcard queries. A value of 0 guarantees that wildcard receiver topic querying never completes.

Scope:	wildcard_receiver
Type:	**lbm_ulong_t**
Units:	seconds
Default value:	60 (1 minute)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0

37.1.7 resolver_query_minimum_interval (wildcard_receiver)

Interval between the first topic query sent upon creation of the wildcard receiver and the second query sent by the receiver.

A value of 0 disables wildcard receiver topic querying. This option has an effective minimum of 30 ms. See UDP-Based Resolver Operation Options.

See also Disabling Aspects of Topic Resolution.

Scope:	wildcard_receiver
Type:	**lbm_ulong_t**
Units:	milliseconds

Default value:	50 (0.05 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0

37.1.8 resolver_wildcard_queries_per_second (context)

Maximum number of queries sent within a one second period during wildcard receiver topic querying.

A value of 0 means that queries for the wildcard topic are not limited to a maximum number of queries per second.

Note that the topic's queries are still subject to being rate limited by resolver_wildcard_query_bps (context).

Refer to **Rate Controls** for additional information.

Scope:	context
Type:	**lbm_ulong_t**
Units:	advertisements
Default value:	0
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0

37.1.9 resolver_wildcard_query_bps (context)

Maximum query rate during wildcard receiver topic querying.

A value of 0 means that queries for the wildcard topic are not limited to a maximum number of bits per second. Note that the topic's queries are still subject to being rate limited by resolver_wildcard_queries_per_second (context).

Refer to **Rate Controls** for additional information.

Scope:	context
Type:	**lbm_uint64_t**
Units:	bits per second
Default value:	1000000
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.0

37.1.10 resolver_wildcard_receiver_map_tablesz (context)

The size of the hash table used for storing wildcard receiver patterns.

A value of 0 disables caching wildcard receiver patterns. This value should be a prime number.

Scope:	context
Type:	**size_t**
Units:	map entries
Default value:	10273
When to Set:	Can only be set during object initialization.

Chapter 38

Event Queue Options

38.1 Reference

38.1.1 event_queue_name (event_queue)

The name of an event queue, limited to 128 alphanumeric characters, hyphens or underscores.

This is only used for XML Configuration Files.

Scope:	event_queue
Type:	**string**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.3/UME 3.3/UMQ 2.3.

38.1.2 queue_age_enabled (event_queue)

Controls whether the length of time each event spends on the event queue is measured.

Useful only if you are monitoring event queue statistics.

See **Automatic Monitoring**.

Scope:	event_queue
Type:	**int**
Default value:	0
When to Set:	May be set during operation.

Value	Description
1	Enables measuring of event queue entry ages.
0	Disables measuring of event queue entry ages. **Default for all.**

38.1.3 queue_cancellation_callbacks_enabled (event_queue)

Flag indicating whether the event queue is to do appropriate locking to provide cancellation callback support for cancel/delete functions.

This must be enabled if you want to use the extended form of object deletion with a callback that indicates completion of the deletion.

For example, see **lbm_src_delete_ex()**.

Scope:	event_queue
Type:	**int**
When to Set:	Can only be set during object initialization.

Value	Description
1	Provide support for cancellation callbacks.
0	Do not provide cancellation callback support. **Default for all.**

38.1.4 queue_count_enabled (event_queue)

Controls whether the numbers of each type of queue entry are counted.

Useful only if you are monitoring event queue statistics.

Scope:	event_queue
Type:	**int**
Default value:	0
When to Set:	May be set during operation.

Value	Description
1	Enables counting event queue entries.
0	Disables counting of event queue entries. **Default for all.**

38.1.5 queue_delay_warning (event_queue)

The event queue delay threshold (in microseconds) at which the monitor function for the event queue is called.

This delay is the time that an event has been queued before being dispatched. A value of 0 indicates the event queue delay is not to be monitored and checked.

Scope:	event_queue
Type:	**lbm_ulong_t**
Units:	microseconds
Default value:	0 (not monitored)
When to Set:	May be set during operation.

38.1.6 queue_enqueue_notification (event_queue)

Flag indicating whether to call the monitor function when an event is enqueued into the given event queue.

The thread enqueuing the event is the one that calls this function. So, when this is called, the monitoring function in use should only assume this is only notification of enqueuing. The monitor function should not dispatch events directly.

Scope:	event_queue
Type:	**int**
When to Set:	May be set during operation.

Value	Description
1	Enable notification.
0	Disable notification. **Default for all.**

38.1.7 queue_objects_purged_on_close (event_queue)

Flag indicating whether the event queue should be immediately purged of any pending events associated with a recently closed object (e.g. source, receiver) during the close operation, or be left on the queue to be discarded as the event queue drains normally.

In either case, UM does not deliver the defunct events to the application. The `Immediate purge` setting reclaims memory immediately, while the `Delay purge` setting spreads the reclamation work over time, reducing the CPU impact of closing objects associated with the queue.

Scope:	event_queue
Type:	**int**
When to Set:	Can only be set during object initialization.

Value	Description
1	Immediate purge. **Default for all.**
0	Delay purge.

38.1.8 queue_service_time_enabled (event_queue)

Controls whether the amount of time required to service each event on the event queue is measured.

Useful only if you are monitoring event queue statistics.

See **Automatic Monitoring**.

Scope:	event_queue
Type:	**int**
Default value:	0
When to Set:	May be set during operation.

Value	Description
1	Enables measuring of event queue service times.
0	Disables measuring of event queue service times. **Default for all.**

38.1.9 queue_size_warning (event_queue)

The event queue size threshold (in number of events) at which the monitor function for the event queue is called.

A value of 0 indicates the event queue size is not to be monitored and checked.

Scope:	event_queue
Type:	**lbm_ulong_t**
Units:	number of events
Default value:	0 (not monitored)
When to Set:	May be set during operation.

Chapter 39

Ultra Messaging Persistence Options

The options described in this section are for persistence, and are invalid for users of the UMS (streaming-only) product.

See the *Guide for Persistence* for more information.

39.1 Reference

39.1.1 ume_ack_batching_interval (context)

The interval between checks by a persistent receiver of consumed, unacknowledged messages.

This option is used in conjunction with ume_use_ack_batching (receiver).

See **Persistence Message Consumption** for a full explanation of consumption acknowledgements.

For RPP, see **RPP Configuration Specifics** for interactions between this and other configuration options. See **RPP: Receiver-Paced Persistence** for general information on RPP.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	100 (0.1 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMS 5.0, UME 5.0, UMQ 5.0.

39.1.2 ume_activity_timeout (receiver)

Establishes the period of time from a receiver's last activity to the release of the receiver's Reg ID. Stores return an error to any new request for the receiver's Reg ID during this period.

Overrides the **receiver-activity-timeout** setting configured for the receiver's topic on the Store. The default value of 0 (zero) disables this option.

See also **Persistence Proxy Sources**.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	0 (disables timeout)
When to Set:	Can only be set during object initialization.

39.1.3 ume_activity_timeout (source)

Establishes the period of time from a source's last activity to the release of the source's Reg ID. Stores return an error to any new source requesting the source's Reg ID during this period.

If proxy sources are enabled (ume_proxy_source (source)), the Store does not release the source's Reg ID and UM elects a proxy source. Overrides the **source-activity-timeout** setting configured for the source's topic on the Store. The default value of 0 (zero) disables this option.

If neither proxy sources nor ume_state_lifetime (source) are configured, the Store also deletes the source's state and cache.

See also **Persistence Proxy Sources**.

Scope:	source
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	0 (disables timeout)
When to Set:	Can only be set during object initialization.

39.1.4 ume_allow_confirmed_delivery (receiver)

Specifies whether a persistent receiver sends confirmed delivery notifications back to the source.

See also ume_confirmed_delivery_notification (source).

For more information, see **Delivery Confirmation Concept**.

Scope:	receiver
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 5.0.

Value	Description
1	Indicates that receivers can send confirmed delivery notifications. **Default for all.**
0	Indicates that receivers can not send confirmed delivery notifications.

39.1.5 ume_application_outstanding_maximum (receiver)

This persistent receiver option enables the persistence Throttled Delivery feature and sets an upper threshold on the number of message fragments from a single source that are delivered or in an event queue, but not yet consumed.

When the number of message fragments exceeds this threshold, the receiver stops buffering all incoming message fragments. Thus, messages from the source transport stream might be dropped and recovered via OTR or persistence late-join mechanisms.

This feature effectively limits the recovery rate and live stream rate to the receiver message consumption rate. If OTR is disabled for the receiver, this threshold applies only during initial Late Join recovery. Setting this option to 0 (zero) disables the persistence persistence Throttled Delivery feature.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	message fragments
Default value:	0 (disabled)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMP 6.7

39.1.6 ume_confirmed_delivery_notification (source)

Flag indicating the application is interested in receiving notifications of consumption of messages by receivers (confirmed delivery) via the source event mechanism.

Generates the source events **LBM_SRC_EVENT_UME_DELIVERY_CONFIRMATION** and/or **LBM_SRC_E↩ VENT_UME_DELIVERY_CONFIRMATION_EX**. When turned off, receivers do not send delivery confirmation notifications to the source unless the release policy dictates the need for them. For more information, see **Delivery Confirmation Concept**.

Note

Smart Sources do not support delivery confirmation.

Scope:	source
Type:	**int**
When to Set:	Can only be set during object initialization.

String value	Integer value	Description
"0"	LBM_SRC_TOPIC_ATTR_UME_CDELV↩ _EVENT_NONE	The source does not wish to receive delivery confirmation notifications. **Default for all.**
"1"	LBM_SRC_TOPIC_ATTR_UME_CDELV↩ _EVENT_PER_FRAGMENT	The source wishes to receive delivery confirmation notifications for all messages and message fragments.
"2"	LBM_SRC_TOPIC_ATTR_UME_CDELV↩ _EVENT_PER_MESSAGE	The source wishes to receive only one delivery confirmation for a message regardless of how many fragments it comprised.

String value	Integer value	Description
"3"	LBM_SRC_TOPIC_ATTR_UME_CDELV↩ _EVENT_FRAG_AND_MSG	The source wishes to receive delivery confirmation notifications for all messages and message fragments. In addition, the notification contains a WHOLE_MESSAGE_C↩ ONFIRMED flag when the last fragment of a message has been delivered.

39.1.7 ume_consensus_sequence_number_behavior (receiver)

The behavior that the receiver will follow when determining the consensus sequence number used as the sequence number to begin reception at upon re-registration after a failure or suspension.

This setting is only used when quorum-consensus is also used on the source.

Scope:	receiver
Type:	**int**
When to Set:	Can only be set during object initialization.

String value	Integer value	Description
"lowest"	LBM_RCV_TOPIC_ATTR_UME_QC_SQ↩ N_BEHAVIOR_LOWEST	Consensus is determined as the lowest of the latest sequence numbers seen from any Store.
"majority"	LBM_RCV_TOPIC_ATTR_UME_QC_SQ↩ N_BEHAVIOR_MAJORITY	Consensus is determined as the latest sequence number agreed upon by the majority of Stores within a group. Between groups, the latest of all majority decisions is used. **Default for all.**
"highest"	LBM_RCV_TOPIC_ATTR_UME_QC_SQ↩ N_BEHAVIOR_HIGHEST	Consensus is determined as the highest of the latest sequence numbers seen from any Store.

39.1.8 ume_consensus_sequence_number_behavior (source)

The behavior that the source follows when determining the consensus sequence number used as the first message of a source upon re-registration after a failure or suspension.

This setting is only used when quorum-consensus is also used.

Scope:	source
Type:	**int**
When to Set:	Can only be set during object initialization.

String value	Integer value	Description
"lowest"	LBM_SRC_TOPIC_ATTR_UME_QC_SQ↩ N_BEHAVIOR_LOWEST	Consensus is determined as the lowest of the latest sequence numbers seen from any Store.
"majority"	LBM_SRC_TOPIC_ATTR_UME_QC_SQ↩ N_BEHAVIOR_MAJORITY	Consensus is determined as the latest sequence number agreed upon by the majority of Stores within a group. Between groups, the latest of all majority decisions is used.
"highest"	LBM_SRC_TOPIC_ATTR_UME_QC_SQ↩ N_BEHAVIOR_HIGHEST	Consensus is determined as the highest of the latest sequence numbers seen from any Store. **Default for all.**

39.1.9 ume_explicit_ack_only (receiver)

Transfers responsibility for sending consumption ACKs to the application.

Normally, the client UM library performs implicit batching of consumption acknowledgements to the Store. This option tells UM that the application will call **lbm_msg_ume_send_explicit_ack()** to explicitly trigger consumption acknowledgements to the Store. This has historically been done by the application to implement ACK batching.

However, as of UM 5.0 with the introduction of ume_use_ack_batching (receiver), UM will now perform ACK batching by default.

Warning

If explicit ACKs are used, the application must ensure that messages are ACKed in the order received. See **ACK Ordering**.

See **Persistence Message Consumption** for a full explanation of consumption acknowledgements.

Note

This option is incompatible with ume_use_ack_batching (receiver). If ACK batching is turned on, this option is silently turned off. If both are turned on, the last one configured turns off the other.

Scope:	receiver
Type:	**int**
When to Set:	Can only be set during object initialization.

Value	Description
1	The receiving application will generate acknowledgements explicitly and the persistent receiver should not automatically generate them.
0	The persistent receiver will automatically generate and send acknowledgements based on message consumption. **Default for all.**

39.1.10 ume_flight_size (source)

Specifies the number of persisted message fragments allowed to be in flight (not stabilized at a Store and without delivery confirmation).

If an application attempts to exceed flight size, the message send function will either block or trigger a **notification** source event, depending on ume_flight_size_behavior (source).

Note that the flight size can also be limited by ume_flight_size_bytes (source), if supplied. Flight size would be exceeded if either one is exceeded.

For RPP, see **RPP Configuration Specifics** for interactions between this and other configuration options. See **RPP: Receiver-Paced Persistence** for general information on RPP.

Note: for very small flight sizes, it is recommended to configure the Store's UM config option response_tcp_↩ nodelay (context) to 1.

Scope:	source
Type:	**unsigned int**
Units:	messages
Default value:	1000
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1.1/UME 3.1.1

39.1.11 ume_flight_size_behavior (source)

The behavior that UM follows when a persistent message send exceeds the source's flight size.

See ume_flight_size (source).

Scope:	source
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1.1/UME 3.1.1

String value	Integer value	Description
"Block"	LBM_FLIGHT_SIZE_BEHAVIOR_BLOCK	The send call blocks when a source sends a message that exceeds its flight size. If the source uses a non-blocking send, the send returns an LBM_EWOULD_BLOCK. **Default for all.**
"Notify"	LBM_FLIGHT_SIZE_BEHAVIOR_NOTIFY	A message send that exceeds the configured flight size does not block but triggers a flight size notification (source event), indicating that the flight size has been surpassed. UM also sends a source event notification if the number of in-flight messages falls below the configured flight size.

39.1.12 ume_flight_size_bytes (source)

Specifies the number of bytes of persisted message payload allowed to be in flight (not stabilized at a Store and without delivery confirmation).

If an application attempts to exceed flight size, the message send function will either block or trigger a **notification** source event, depending on ume_flight_size_behavior (source).

If this option is not supplied (defaults to 0), a persisted source uses ume_flight_size (source) to determine if a send would exceed flight size.

If this option is supplied, a persisted source uses both ume_flight_size_bytes and ume_flight_size (source). Flight size would be exceeded if either one is exceeded.

If the source uses RPP, then this option must be supplied, and its value is sent to the Store during registration. The Store compares it to the configured value of **source-flight-size-bytes-maximum**. The source's registration will be rejected if ume_flight_size_bytes exceeds the Store's source-flight-size-bytes-maximum.

For RPP, see **RPP Configuration Specifics** for interactions between this and other configuration options. See **RPP: Receiver-Paced Persistence** for general information on RPP.

Note: for very small flight sizes, it is recommended to configure the Store's UM config option response_tcp_↩nodelay (context) to 1.

Scope:	source
Type:	**lbm_uint64_t**
Units:	bytes
Default value:	0 (disabled)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UME 5.3

39.1.13 ume_force_reclaim_function (source)

Callback function (and associated client data pointer) that is called when a source is forced to release a retained message due to size limitations specified.

This callback is called by the context thread and can not use an event queue. Therefore the callback function used should not block or it will delay reception of latency-sensitive messages.

Scope:	source
Type:	**lbm_ume_src_force_reclaim_func_t**
Default value:	NULL
When to Set:	Can only be set during object initialization.
Config File:	Cannot be set from an UM configuration file.

39.1.14 ume_late_join (source)

Flag indicating the source should allow late join operation for receivers and persistent Stores.

This option is retained for backwards compatibility. The late_join (source) setting should be used instead.

Scope:	source
Type:	**int**
When to Set:	Can only be set during object initialization.

Value	Description
1	The source allows late join receivers and persistent Stores.
0	The source does not allow late join receivers or persistent Stores. **Default for all.**

39.1.15 ume_message_stability_lifetime (source)

The total time in milliseconds from the initial send of a message before a persistent source gives up entirely on receiving a stability acknowledgement for the message.

The source then delivers a forced reclaim notice to the application. This option is part of the Proactive Retransmissions feature.

Scope:	source
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	1200000 (20 minutes)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UME 6.0

39.1.16 ume_message_stability_notification (source)

Flag indicating the source is interested in receiving notifications of message stability from persistent Stores via the source event mechanism.

Even when turned off, Stores continue to send message stability notifications to the source for retention purposes. However, no notification will be delivered to the application.

Note

Smart Sources only support "0" (none) or "2" (per-message).

Scope:	source
Type:	**int**
When to Set:	Can only be set during object initialization.

String value	Integer value	Description
"0"	LBM_SRC_TOPIC_ATTR_UME_STABL↩ E_EVENT_NONE	The source does not wish to receive message stability notifications from the Store.
"1"	LBM_SRC_TOPIC_ATTR_UME_STABL↩ E_EVENT_PER_FRAGMENT	The source wishes to receive all message and message fragment stability notifications from the Store. **Default for all.**
"2"	LBM_SRC_TOPIC_ATTR_UME_STABL↩ E_EVENT_PER_MESSAGE	The source wishes to receive only a single message stability notifications from the Store when the entire message has been stabilized. This notification contains the Sequence Number of the last fragment of the whole message but does NOT contain Store information.
"3"	LBM_SRC_TOPIC_ATTR_UME_STABL↩ E_EVENT_FRAG_AND_MSG	The source wishes to receive all message and message fragment stability notifications from the Store. In addition, the notification contains a WHOLE_MESSAGE_STA↩ BLE flag when the last fragment of a message has been stabilized.

39.1.17 ume_message_stability_timeout (source)

The time in milliseconds from initial send of a message until it is resent by the source because the source has not received a stability acknowledgement for the Store (or a quorum of Stores).

Setting this option to 0 (zero) disables the Proactive Retransmissions feature.

Scope:	source
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	5000 (5 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UME 6.0

39.1.18 ume_proactive_keepalive_interval (context)

Maximum period of inactivity after which a persistent receiver proactively sends an acknowledgement to the Store.

A persistent receiver sends consumption acknowledgements to the Store to update that receiver's state in the Store. In the absence of new consumption acknowledgments, a receiver will re-send the most-recent acknowledgement periodically to maintain that state. The ume_proactive_keepalive_interval option specifies the maximum interval between successive acknowledgements. This value should be set less than the ume_activity_timeout (receiver) and the state lifetime, ideally no more than 1/3 of the lesser of those two. Valid settings are greater than or equal to 1500 (1.5 seconds, the effective minimum), or zero to disable proactive keepalives and revert to pre-UM 6.9 keepalive behavior.

Note that disabling proactive keepalives is generally not recommended, and cannot be done for a persistent receiver which is assigned to a **Transport Services Provider (XSP)**.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	3000 (3 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UME 6.9.1

39.1.19 ume_proxy_source (source)

Controls whether any Stores with which the source registers should provide a proxy source in the event the actual source terminates.

Proxy source support is only available for quorum/consensus Store configurations. In addition, proxy source support requires that the source register with an actual registration ID, and not request that the Store assign it a registration ID.

Scope:	source
Type:	**int**
Default value:	0
When to Set:	Can only be set during object initialization.

Value	Description
1	Enables proxy source support.
0	Disables proxy source support. **Default for all.**

39.1.20 ume_receiver_liveness_interval (context)

The maximum interval between delivery confirmations or keepalive messages send to the source.

Expiration of this interval triggers another keepalive and an interval reset.

Scope:	context
Type:	**int**
Units:	milliseconds
Default value:	0 (disable; do not send keepalives)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UME 5.2.

39.1.21 ume_receiver_paced_persistence (receiver)

Enables Receiver-paced Persistence (RPP) for the receiver, and specifies the blocking behavior.

The source controls whether the Store's repository behavior is SPP or RPP (see ume_receiver_paced_persistence (source)). The receiver's configuration must match how the source set the repository. If the repository and receiver disagree, the receiver's registration is rejected.

See **RPP: Receiver-Paced Persistence** for general information on RPP.

Scope:	receiver
Type:	**lbm_uint8_t**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UME 5.3. Value "2" was added in UME 6.9

String value	Integer value	Description
"0"	0	Indicates that the receiver is not a RPP receiver. **Default for all.**
"1"	1	Indicates that the receiver is a blocking RPP receiver.
"2"	2	Indicates that the receiver is a non-blocking RPP receiver.

39.1.22 ume_receiver_paced_persistence (source)

Requests that the Store operate its repository with Receiver-paced Persistence (RPP).

During registration with the Store, this option controls whether the source requests RPP behavior. If the source requests RPP, and the Store's repository is configured with **repository-allow-receiver-paced-persistence** set to 1 (enabled), the registration is accepted and the Store operates with receiver pacing. If **repository-allow-receiver-paced-persistence** is 0 (disabled), the source's registration is rejected.

If this option is not set, the Store will default to Source-paced Persistence (SPP), independent of the setting of **repository-allow-receiver-paced-persistence**. The Store cannot be directly configured to enable RPP; the source must be configured to request it.

See **RPP: Receiver-Paced Persistence** for general information on RPP.

Scope:	source
Type:	**lbm_uint8_t**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UME 5.3

Value	Description
1	Source requests RPP operation.
0	Source does not request RPP operation. Store will operate with Source-paced Persistence (SPP). **Default for all.**

39.1.23 ume_recovery_sequence_number_info_function (receiver)

Callback function (and associated client data pointer) that is called when a receiver is about to complete registration from the Stores in use by the source and the low sequence number is to be determined.

The application has the ability to modify the sequence number to use if it desires.

This callback is called by the context thread and can not use an event queue. Therefore the callback function used should not block or it will delay reception of latency-sensitive messages.

Scope:	receiver
Type:	**lbm_ume_rcv_recovery_info_ex_func_t**
Default value:	NULL
When to Set:	Can only be set during object initialization.
Config File:	Cannot be set from an UM configuration file.

39.1.24 ume_registration_extended_function (receiver)

Callback function (and associated client data pointer) that is called when a receiver is about to attempt to register with a persistent Store.

The app must return the registration ID to request from the Store or 0 if it will allow the Store to allocate one. This function passes additional extended information, such as the Store being used and a source client data pointer, etc.

This callback is called by the context thread and can not use an event queue. Therefore the callback function used should not block or it will delay reception of latency-sensitive messages.

Scope:	receiver
Type:	**lbm_ume_rcv_regid_ex_func_t**
Default value:	NULL
When to Set:	Can only be set during object initialization.
Config File:	Cannot be set from an UM configuration file.

39.1.25 ume_registration_function (receiver)

Callback function (and associated client data pointer) that is called when a receiver is about to attempt to register with a persistent Store.

The app must return the registration ID to request from the Store or 0 if it will allow the Store to allocate one.

This callback is called by the context thread and can not use an event queue. Therefore the callback function used should not block or it will delay reception of latency-sensitive messages.

This option is retained for backwards compatibility. The ume_registration_extended_function (receiver) setting should be used instead.

Scope:	receiver
Type:	**lbm_ume_rcv_regid_func_t**
Default value:	NULL
When to Set:	Can only be set during object initialization.
Config File:	Cannot be set from an UM configuration file.

39.1.26 ume_registration_interval (receiver)

The interval between registration attempts by the receiver to a persistent Store in use by the source.

For networks with large numbers of receivers connecting to a Store, this value can be increased to reduce the registration load on the Store.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	3000 (3 seconds)
When to Set:	Can only be set during object initialization.

39.1.27 ume_registration_interval (source)

The interval between registration attempts by the source. Before declaring Registration Complete, sources wait at least one full interval, unless all Stores have registered.

When using the round-robin Store behavior, this is the value between registration attempts with the various Stores. In other words, attempt to register with primary, wait interval, attempt to register with secondary, wait interval, etc.

Scope:	source
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	3000 (3 seconds)
When to Set:	Can only be set during object initialization.

39.1.28 ume_repository_ack_on_reception (source)

For RPP, requests that the Store operate its repository with "ack on reception" behavior.

Normally, the Store sends stability ACKs to the source as messages are written to disk. With "ack on reception" behavior, the Store sends stability ACKs to the source as the messages are received and stored in its memory cache.

During registration with the Store, this option controls whether the source requests "ack on reception" behavior. If the source requests "ack on reception", and the Store's repository is configured with **repository-allow-ack-on-reception** set to 1 (enabled), the registration is accepted and the repository operates with "ack on reception" behavior. If **repository-allow-ack-on-reception** is 0 (disabled), the source's registration is rejected.

The Store cannot be directly configured to enable "ack on reception"; the source must be configured to request it.

This option is ignored for SPP repositories.

For RPP, see **RPP Configuration Specifics** for interactions between this and other configuration options. See **RPP: Receiver-Paced Persistence** for general information on RPP.

Scope:	source
Type:	**lbm_uint8_t**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UME 5.3

Value	Description
1	Source requests "ack on reception".
0	Source does not request "ack on reception". Store defaults to sending stability ACKs when messages are written to disk. **Default for all.**

39.1.29 ume_repository_disk_file_size_limit (source)

For Receiver-paced Persistence (RPP) sources, overrides the Store's configured repository disk file size limit.

The Store's repository disk file size limit is configured by the **repository-disk-file-size-limit** option. For RPP, this source option can be used to override the Store's setting.

See **repository-disk-file-size-limit** for more information on the option.

This option is ignored for SPP repositories.

When the source overrides the Store's configured value, it must not exceed that value. If the source's ume↩ _repository_disk_file_size_limit exceeds the Store's repository-disk-file-size-limit, the source's registration is rejected.

See **Implementing RPP** for more information on the coordination between RPP source and Store configuration options.

Scope:	source
Type:	**lbm_uint64_t**
Units:	bytes
Default value:	0 (disabled)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UME 5.3

39.1.30 ume_repository_size_limit (source)

For Receiver-paced Persistence (RPP) sources, overrides the Store's configured repository size limit.

The Store's repository size limit is configured by the **repository-size-limit** option. For RPP, this source option can be used to override the Store's setting.

See **repository-size-limit** for more information on the option.

This option is ignored for SPP repositories.

For RPP, see **RPP Configuration Specifics** for interactions between this and other configuration options. See **RPP: Receiver-Paced Persistence** for general information on RPP.

When the source overrides the Store's configured value, it must not exceed that value. If the source's ume_↩ repository_size_limit exceeds the Store's repository-size-limit, the source's registration is rejected.

Scope:	source
Type:	**size_t**
Units:	bytes
Default value:	0 (disabled)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UME 5.3

39.1.31 ume_repository_size_threshold (source)

For Receiver-paced Persistence (RPP) sources, overrides the Store's configured repository size threshold.

The Store's repository size threshold is configured by the Store option **repository-size-threshold** option. For RPP, this source option can be used to override the Store's setting.

See **repository-size-threshold** for more information on the option.

This option is ignored for SPP repositories.

For RPP, see **RPP Configuration Specifics** for interactions between this and other configuration options. See **RPP: Receiver-Paced Persistence** for general information on RPP.

When the source overrides the Store's configured value, it must not exceed that value. If the source's ume_↩ repository_size_threshold exceeds the Store's repository-size-threshold, the source's registration is rejected.

Scope:	source
Type:	**size_t**
Units:	bytes
Default value:	0 (disabled)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UME 5.3

39.1.32 ume_retention_intergroup_stability_behavior (source)

The behavior that the source will follow when determining, across Store groups, both message stability and registration completion.

A source cannot release a message until the message is stable. To be stable, a message must first be stable within the group and then stable between

Note

Multiple QC groups is deprecated and may be removed from a future version of UM. Users of multiple QC groups are encouraged to migrate their source configurations to use a single group of Stores for QC. groups.

Scope:	source
Type:	**int**
When to Set:	Can only be set during object initialization.

String value	Integer value	Description
"any", "any-group"	LBM_SRC_TOPIC_ATTR_UME_STABLE_BEHAVIOR_ANY	Registration is complete when it is complete in any group. Messages are stable when they are stable in any group. **Default for all.**
"all-active"	LBM_SRC_TOPIC_ATTR_UME_STABLE_BEHAVIOR_ALL_ACTIVE	A group is active if it has at least a quorum of registered Stores, or as determined by the ume_retention_intragroup_stability_behavior option. Registration is complete when it is complete in all active groups. At least one group must be active. Messages are stable when they are stable in all active groups.
"majority"	LBM_SRC_TOPIC_ATTR_UME_STABLE_BEHAVIOR_MAJORITY	Registration is complete when it is complete in a majority of groups. Messages are stable when they are stable in a majority of groups.
"all", "all-groups"	LBM_SRC_TOPIC_ATTR_UME_STABLE_BEHAVIOR_ALL	Registration is complete when it is complete in all groups. Messages are stable when they are stable in all groups.

39.1.33 ume_retention_intragroup_stability_behavior (source)

The behavior that the source will follow when determining, within a Store group, both message stability and group registration completion.

A source cannot release a message until the message is stable. To be stable, a message must first be stable within the group and then stable between groups.

Note

Multiple QC groups is deprecated and may be removed from a future version of UM. Users of multiple QC groups are encouraged to migrate their source configurations to use a single group of Stores for QC.

Scope:	source
Type:	**int**
When to Set:	Can only be set during object initialization.

String value	Integer value	Description
"quorum"	LBM_SRC_TOPIC_ATTR_UME_STAB↩ LE_BEHAVIOR_QUORUM	Registration is complete for the group when a majority of the Stores in the group are registered. A message is stable within the group when a majority of the Stores have acknowledged the message as stable. **Default for all.**
"all-active"	LBM_SRC_TOPIC_ATTR_UME_STAB↩ LE_BEHAVIOR_ALL_ACTIVE	Registration is complete for the group when a majority of the Stores in the group are registered. Stores registered with a source are active Stores. A message is stable within the group when each active Store in that group has acknowledged the message as stable.
"all", "all-stores"	LBM_SRC_TOPIC_ATTR_UME_STAB↩ LE_BEHAVIOR_ALL	Registration is complete for the group when all Stores in the group are registered. A message is stable within the group when all Stores in the group are registered and have acknowledged the message as stable.

39.1.34 ume_retention_size_limit (source)

The release policy regarding aggregate size limit before messages are forced to be released.

With **Smart Sources**, this option is ignored. Retention buffers are preallocated. This option is retained for backwards compatibility. The retransmit_retention_size_limit (source) setting should be used instead.

Scope:	source
Type:	**size_t**
Units:	bytes
Default value:	25165824 (24 MB)
When to Set:	Can only be set during object initialization.

39.1.35 ume_retention_size_threshold (source)

The release policy regarding aggregate size threshold before messages are released.

With **Smart Sources**, this option is ignored. Retention buffers are preallocated.

This option is retained for backwards compatibility. The retransmit_retention_size_threshold (source) setting should be used instead.

Scope:	source
Type:	**size_t**
Units:	bytes
Default value:	0 (no threshold)
When to Set:	Can only be set during object initialization.

39.1.36 ume_retention_unique_confirmations (source)

The release policy regarding the number of confirmations from different receivers required before the source can release a message.

This option enhances, but does not supersede, message stability notification from the Store(s). If the number of unique confirmations for a message is less than this amount, the message will not be released. If the number of unique confirmations for a message exceeds or equals this amount, then the message may be released if no other release policy setting overrides the decision. A value of 0 indicates there is no unique number of confirmations required for reclamation. For more information, see **Delivery Confirmation Concept**.

Note

Smart Sources do not support delivery confirmation.

Scope:	source
Type:	**size_t**
Units:	number of confirmations
Default value:	0 (none required)
When to Set:	Can only be set during object initialization.

39.1.37 ume_session_id (context)

Specifies the default Session ID to use for sources and receivers within a context. A value of 0 (zero) indicates no Session ID is to be set.

See also **Managing RegIDs with Session IDs**. Valid formats for session IDs are as follows: A hexadecimal string with a maximum value of FFFFFFFFFFFFFFFE, prefixed with '0x'. An octal string with a maximum value of 1777777777777777777776 prefixed with '0'. A decimal string with a maximum value of 18446744073709551614. Prior to LBM 5.2.2, all persistence session IDs were interpreted as hexadecimal, and did not accept the '0x' prefix. If upgrading from an earlier version to LBM 5.2.2 or later, prepend '0x' to the original setting to use the originally assigned session ID.

Scope:	context
Type:	**lbm_uint64_t**
Default value:	0 (no session ID)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.2/UME 3.2

39.1.38 ume_session_id (receiver)

Specifies the Session ID to use for a receiver. A value of 0 (zero) indicates the context ume_session_id will be used.

See also **Managing RegIDs with Session IDs**.

Valid formats for session IDs are as follows: A hexadecimal string with a maximum value of FFFFFFFFFFFF↩FFFE, prefixed with '0x'. An octal string with a maximum value of 1777777777777777777776 prefixed with '0'. A decimal string with a maximum value of 18446744073709551614. Prior to LBM 5.2.2, all persistence session IDs were interpreted as hexadecimal, and did not accept the '0x' prefix. If upgrading from an earlier version to LBM 5.2.2 or later, prepend '0x' to the original setting to use the originally assigned session ID.

Scope:	receiver
Type:	**lbm_uint64_t**
Default value:	0 (uses context session ID)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.2/UME 3.2

39.1.39 ume_session_id (source)

Specifies the Session ID to use for a source. A value of 0 (zero) indicates the context ume_session_id will be used.

See also **Managing RegIDs with Session IDs**.

Valid formats for session IDs are as follows: A hexadecimal string with a maximum value of FFFFFFFFFFFF↩ FFFE, prefixed with '0x'. An octal string with a maximum value of 1777777777777777777776 prefixed with '0'. A decimal string with a maximum value of 18446744073709551614. Prior to LBM 5.2.2, all persistence session IDs were interpreted as hexadecimal, and did not accept the '0x' prefix. If upgrading from an earlier version to LBM 5.2.2 or later, prepend '0x' to the original setting to use the originally assigned session ID.

Scope:	source
Type:	**lbm_uint64_t**
Default value:	0 (uses context session ID)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.2/UME 3.2

39.1.40 ume_source_liveness_timeout (context)

The expected maximum interval between keepalive or delivery confirmation messages from a receiver.

If neither are received within the interval, the source declares the receiver "dead".

Scope:	context
Type:	**int**
Units:	milliseconds
Default value:	0 (disable; do not track receivers)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UME 5.2.

39.1.41 ume_sri_flush_sri_request_response (source)

This option determines if a source flushes the Implicit Batching buffer after it sends a Source Registration Information (SRI) record in response to a SRI request from a receiver.

Flushing this buffer places the SRI record immediately on the transport, which speeds up the process of receivers registering, but also can impose a greater load on the overall network since it can reduce the amount of transport batching.

See ume_sri_immediate_sri_request_response (source) for more information on SRI messages.

Note

Smart Sources do not support batching, so this option is ignored by a Smart Source.

Scope:	source
Type:	**lbm_ulong_t**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMP 6.0

Value	Description
1	The source places a SRI record in the Implicit Batching buffer and then flushes the buffer.
0	The source places a SRI record in the Implicit Batching buffer and lets normal batch scheduling determine when to place the SRI on the transport. **Default for all.**

39.1.42 ume_sri_immediate_sri_request_response (source)

This option controls how quickly a source responds to a receiver's request for an SRI record.

A persistent source need to send information about its Stores so that the receivers can properly register with those Stores. The information messages sent by the sources, contained in a Source Registration Information

(SRI) record, is sent on the source's data transport session, and therefore have an effect on the transfer of application data messages. This configuration option is provided to assist you in managing the impact of SRI messages on the normal flow of data when a registering receiver requests the SRI record.

Note

Smart Sources do not support batching, so this option is ignored by a Smart Source.

Scope:	source
Type:	**lbm_ulong_t**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMP 6.0

Value	Description
1	Indicates that the source sends an SRI record and also flushes the implicit batching buffer to immediately put the SRI record on the transport. This maximizes the speed at which a receiver completes its registration, but also can impose a greater load on the overall network since it can reduce the amount of transport batching. **Default for all.**
0	Indicates that the source waits for the period of time defined by ume_sri_request_response_latency (source) before sending an SRI record. This reduces overall system load, especially if multiple receivers are registering, as it allows a single SRI record to satisfy the registration needs of multiple receivers.

39.1.43 ume_sri_inter_sri_interval (source)

This option controls how frequently a source sends SRI records in reaction to a change in the source's registration with its Stores.

Source Registration Information (SRI) records are sent by a source to its receivers for either of two reasons:

- a receiver has requested an SRI, usually because it is in the process of initializing and registering, or
- the source sees a change in its registration with its Stores. For example, if a Store becomes unresponsive and the source loses registration with it. Or if a previously failed Store returns to service, and the source successfully registers with it.

This configuration option is concerned with the latter case (change in a source's registration with its Stores): the source will send SRI records to receivers to inform them of the change. It sends multiple copies over time to maximize the chances of successful reception. It uses this configuration option to determine the interval between these SRI sends.

The default value results in the source sending 2 SRI packets every second. This value cannot be set to 0. See also ume_sri_max_number_of_sri_per_update (source).

Scope:	source
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	500 (0.5 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMP 6.0

39.1.44 ume_sri_max_number_of_sri_per_update (source)

The maximum number of SRI packets sent by a source after a change in the source's registration with its Stores.

For more information about these SRI messages, see ume_sri_inter_sri_interval (source).

Scope:	source
Type:	**lbm_uint16_t**
Default value:	20
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMP 6.0

39.1.45 ume_sri_request_interval (receiver)

The interval at which a registering receiver requests information about the persistent Store(s) from the source.

The receiver cannot complete registration with the Store(s) until the source supplies the information, in the form of a Store Information Record (SRI). If no SRI is received within this interval, the receiver will continue to send requests until either the information is received, or until the ume_sri_request_maximum (receiver) is reached. If that limit is reached without having received the SRI, the receiver registration fails.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	1000 (1 second)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMP 6.0

39.1.46 ume_sri_request_maximum (receiver)

The maximum number of requests the receiver issues for a Store Information Record (SRI) from the source.

If the receiver has not received an SRI after this number of requests, it stops requesting and fails its registration. See ume_sri_request_interval (receiver).

Scope:	receiver
Type:	**lbm_ulong_t**
Default value:	60
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMP 6.0

39.1.47 ume_sri_request_response_latency (source)

The interval a source waits before sending an SRI packet in response to a request from a receiver.

At the expiration of this interval, the SRI record may also be slightly delayed by normal batch scheduling unless ume_sri_flush_sri_request_response (source) is set to 1.

See ume_sri_immediate_sri_request_response (source) for more information about how and why to use this.

Note

Smart Sources do not support batching, so this option is ignored by a Smart Source.

Scope:	source
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	100 (0.1 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMP 6.0

39.1.48 ume_state_lifetime (receiver)

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 a **receiver-state-lifetime** for the receiver's topic on the Store. The Store uses whichever is shorter. The default value of 0 (zero) disables this option.

See also **Persistence Proxy Sources**.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	0 (disables lifetime)
When to Set:	Can only be set during object initialization.

39.1.49 ume_state_lifetime (source)

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 a **source-state-lifetime** for the source's topic on the Store. The Store uses whichever is shorter. The default value of 0 (zero) disables this option.

See also **Persistence Proxy Sources**.

Scope:	source
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	0 (disables lifetime)
When to Set:	Can only be set during object initialization.

39.1.50 ume_store (source)

Enable persistence for this source and add a Store specification to the current list of Stores specified for the source. Unlike most other UM settings, every time this setting is called, it adds another Store specification to the list and does NOT overwrite previous specifications.

Each entry contains the IP address, TCP port, registration ID, and group index for the Store. For the configuration file as well as API string setting functions, the string value for this option is formatted as "DomainID↩ :IP:port:RegID:GroupIDX" where DomainID is the Store's UM domain ID, IP is the Store's IP address, port is the TCP port for the Store, RegID is the registration ID that the source desires to use, and GroupIDX is the group index that the Store belongs to. The DomainID, RegID, and GroupIDX pieces may be left off the string if desired. If so, UM assumes the value of 0 for them.

Note

Multiple QC groups is deprecated and may be removed from a future version of UM. Users of multiple QC groups are encouraged to migrate their source configurations to use a single group of Stores for QC.

With most configuration options, a previously-specified value can be overridden by simply specifying the option again with a new value. However, because each occurrence of **`ume_store`** adds a new Store specification, use the IP address 0.0.0.0 and TCP port 0 to remove all previously specified Stores. This allows subsequent Store specifications to, in effect, override the earlier Stores.

One or more Stores means the source will use persistence. If no Stores are specified, then persistence will not be provided for the source.

When the binary form of option setting is used, UM does NOT expect an array of structures. Instead, only one Store specification can be supplied for each call to **lbm_src_topic_attr_setopt()**. However, when the binary

form of option retrieval **lbm_src_topic_attr_getopt()** is used, the list of Stores is returned as an array, and the **optlen** parameter should be set as:

```
optlen = (max_num_stores * sizeof(lbm_ume_store_entry_t));
```

Scope:	source
Type:	**lbm_ume_store_entry_t**
When to Set:	Can only be set during object initialization.

39.1.51 ume_store_activity_timeout (source)

The timeout value used to indicate when a Store is unresponsive.

The Store must not be active within this interval to be considered unresponsive. This value must be much larger than the check interval.

Scope:	source
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	10,000 (10 seconds)
When to Set:	Can only be set during object initialization.

39.1.52 ume_store_behavior (source)

The behavior that the source follows for handling Store failures.

Only quorum-consensus is allowed. The option is retained for backwards compatibility purposes.

Scope:	source
Type:	**int**
When to Set:	Can only be set during object initialization.

String value	Integer value	Description
"qc", "quorum-consensus"	LBM_SRC_TOPIC_ATTR_UME_S↩ TORE_BEHAVIOR_QC	The source uses multiple Stores at the same time based on Store and Store group configuration. **Default for all.**

39.1.53 ume_store_check_interval (source)

The interval between activity checks of the current Store.

This interval also governs how often a source checks outstanding unstabilized messages to see if they have reached the configured ume_message_stability_timeout (source) value yet.

Scope:	source
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	500 (0.5 seconds)
When to Set:	Can only be set during object initialization.

39.1.54 ume_store_group (source)

Add a Store group specification to the list of Store groups specified for the source.

Unlike other UM settings, every time this setting is called, it adds another Store group specification to the list and does NOT overwrite previous specifications. Each entry contains the group index and group size for the group. For the configuration file as well as string versions of setting this option, the string value is formatted as "GroupIDX:GroupSZ" where GroupIDX is the index of the group and GroupSZ is the size of the group. Because each entry adds a new Store specification and does not overwrite previous values, an entry or string with the group index of 0 and group size of 0 will cause all previous Store group specifications to be removed.

Note

Multiple QC groups is deprecated and may be removed from a future version of UM. Users of multiple QC groups are encouraged to migrate their source configurations to use a single group of Stores for QC.

Note: When setting this option multiple times, you must set this option in group-index order, from lowest to highest. In other words, do not set this option for a group index lower in value than any previously set group index value.

When the binary form of option setting is used, UM does NOT expect an array of structures. Instead, only one group specification can be supplied for each call to **lbm_src_topic_attr_setopt()**. However, when the binary form of option retrieval **lbm_src_topic_attr_getopt()** is used, the list of groups is returned as an array, and the **optlen** parameter should be set as:

```
optlen = (max_num_store_groups * sizeof(lbm_ume_store_group_entry_t));
```

Scope:	source
Type:	**lbm_ume_store_group_entry_t**
When to Set:	Can only be set during object initialization.

39.1.55 ume_store_name (source)

Add a named Store specification to the list of Stores specified for the source.

Unlike other UM settings, every time this setting is called, it adds another Store specification to the list and does NOT overwrite previous specifications. Each entry contains the Store context name, registration ID, and group index for the Store. For the configuration file as well as string versions of setting this option, the string value is formatted as **"name:RegID:GroupIDX"** where **"name"** is the context name of the Store, (configured with the Store option **context-name**), **"RegID"** is the registration ID that the source desires to use, and **"GroupIDX"** is the group index that the Store belongs to. The RegID and GroupIDX pieces may be left off the string if desired. If so, then the value of 0 is assumed for them. Store context names are restricted to 128 characters in length, and may contain only alphanumeric characters, hyphens, and underscores.

Note

Multiple QC groups is deprecated and may be removed from a future version of UM. Users of multiple QC groups are encouraged to migrate their source configurations to use a single group of Stores for QC.

When the binary form of option setting is used, UM does NOT expect an array of structures. Instead, only one named Store specification can be supplied for each call to **lbm_src_topic_attr_setopt()**. However, when the binary form of option retrieval **lbm_src_topic_attr_getopt()** is used, the list of named Stores is returned as an array, and the **optlen** parameter should be set as:

```
optlen = (max_num_stores * sizeof(lbm_ume_store_name_entry_t));
```

Scope:	source
Type:	**lbm_ume_store_name_entry_t**
When to Set:	Can only be set during object initialization.

39.1.56 ume_use_ack_batching (receiver)

Enables automatic implicit batching of consumption acknowledgements by a persistent receiver.

Automatic batching of consumption acknowledgements improves average latency and throughput of a persistent receiver. The batching timer is set with ume_ack_batching_interval (context).

Warning

If this option is disabled, the application must ensure that messages are ACKed in the order received. See **ACK Ordering**.

See **Persistence Message Consumption** for a full explanation of consumption acknowledgements.

Note

This option is incompatible with ume_explicit_ack_only (receiver). If explicit ACK is turned on, this option is silently turned off. If both are turned on, the last one configured turns off the other.

Scope:	receiver
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMS 5.0, UME 5.0, UMQ 5.0.

Value	Description
1	UM automatically sends acknowledges of batches of consumed messages to the persistent Store. **Default for all.**
0	The application directly controls the sending of acknowledgements to the persistent Store.

39.1.57 ume_use_late_join (receiver)

Flag indicating if the receiver should participate in late join operation or not.

This option is retained for backwards compatibility. The use_late_join (receiver) setting should be used instead.

Scope:	receiver
Type:	**int**
When to Set:	Can only be set during object initialization.

Value	Description
1	The receiver will participate in using late join if requested to by the source. **Default for all.**
0	The receiver will not participate in using late join even if requested to by the source.

39.1.58 ume_use_store (receiver)

Flag indicating if the receiver should participate in using a persistent Store or not.

If "0" is supplied, the receiver will join as a streaming receiver.

Scope:	receiver
Type:	**int**
When to Set:	Can only be set during object initialization.

Value	Description
1	The receiver will participate in using a persistent Store if requested to by the source. **Default for all.**
0	The receiver will not participate in using a persistent Store even if requested to by the source.

39.1.59 ume_user_receiver_registration_id (context)

32-bit value that is used as a user set identifier to be included as the receiver registration ID in acknowledgements send by any receivers in the context to sources as confirmed delivery notifications.

The value is not interpreted by UM in any way and has no relation to registration IDs used by the receiver. A value of 0 indicates no user set value is in use and should not be sent with acknowledgements

Scope:	context
Type:	**lbm_uint_t**
Units:	identifier
Default value:	0 (no user set value in use)
When to Set:	Can only be set during object initialization.

39.1.60 ume_write_delay (source)

For Receiver-paced Persistence (RPP) sources, overrides the Store's configured write delay setting.

The Store's write delay setting is configured by the **repository-disk-write-delay** option. For RPP, this source option can be used to override the Store's setting.

See **repository-disk-write-delay** for more information on the option.

This option is ignored for SPP repositories.

For RPP, see **RPP Configuration Specifics** for interactions between this and other configuration options. See **RPP: Receiver-Paced Persistence** for general information on RPP.

When the source overrides the Store's configured value, it must not exceed that value. If the source's ume_↩ write_delay exceeds the Store's repository-disk-write-delay, the source's registration is rejected.

Scope:	source
Type:	**lbm_uint32_t**
Units:	milliseconds
Default value:	0 (disabled)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UME 5.3

Chapter 40

Ultra Messaging Queuing Options

The options described in this section are for queuing, and are invalid for users of the UMS (streaming-only) and UMP (streaming and persistent) products.

See the *Guide for Queuing* for more information.

40.1 Reference

40.1.1 umq_command_interval (context)

The interval at which all currently outstanding UMQ commands (registrations, de-registrations, message list commands, indexed queueing commands, etc.) are re-sent if they have not yet been acknowledged by the queue.

Applies to brokered queue and ULB. For general information on queuing, see **Queuing**.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	500 (0.5 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.6/UME 3.0/UMQ 1.0.

40.1.2 umq_command_outstanding_maximum (context)

The maximum number of UMQ commands (registrations, de-registrations, message list commands, indexed queueing commands, etc.) that may be outstanding at one time for each configured queue.

This option value must be greater than 0. Reducing this value may help alleviate some load on the UMQ queue daemon, but may potentially cause registrations and other commands to take longer to complete.

Scope:	context
Type:	**lbm_uint32_t**
Units:	number of outstanding commands
Default value:	1000
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMQ 5.3.1.

40.1.3 umq_delayed_consumption_report_interval (receiver)

The maximum interval to delay sending consumption reports on the receiver.

Applies to ULB. For general information on queuing, see **Queuing**.

Delaying consumption reports allows them to be batched together for efficiency but at the expense of delaying the consumption reports themselves individually. The value of 0 indicates the consumption reports should not be delayed.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	0
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1.

40.1.4 umq_hold_interval (receiver)

The maximum interval to hold control and data information within the ULB delivery controller.

For ULB only. For general information on queuing, see **Queuing**.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	10000 (10 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.6/UME 3.0/UMQ 1.0.

40.1.5 umq_index_assignment_eligibility_default (receiver)

Controls whether new ULB receivers are immediately eligible for index assignment upon registration with a ULB source (the default), or whether they are ineligible upon registration and must be explicitly made eligible via a call to **lbm_rcv_umq_index_start_assignment()**.

For ULB only. For general information on queuing, see **Queuing**.

Scope:	receiver
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.2/UME 3.2/UMQ 1.2

String value	Integer value	Description
"Eligible"	LBM_RCV_TOPIC_ATTR_UMQ_INDEX↩ _ASSIGN_ELIGIBILITY_ELIGIBLE	The receiver may be assigned indices as soon as it registers with a queue. **Default for all.**
"Ineligible"	LBM_RCV_TOPIC_ATTR_UMQ_INDEX↩ _ASSIGN_ELIGIBILITY_INELIGIBLE	The receiver must first call **lbm_rcv_umq↩ _index_start_assignment()** before it can be assigned any indices.

40.1.6 umq_message_stability_notification (source)

Flag indicating the source is interested in receiving notifications of message stability from UMQ via the source event mechanism.

Even when turned off, UMQ continues to send message stability notifications to the source for retention purposes. However, UMQ delivers no notification to the application.

Scope:	source
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.6/UME 3.0/UMQ 1.0.

Value	Description
1	The source wishes to receive message stability notification. **Default for all.**
0	The source does not wish to receive message stability notifications.

40.1.7 umq_msg_total_lifetime (source)

Establishes the period of time from when a queue enqueues a message until the time the message cannot be assigned or reassigned to a receiver. The queue deletes the message upon expiration of the lifetime.

Applies to brokered queue and ULB. For general information on queuing, see **Queuing**.

The default value of 0 (zero) disables this option. See also **Message Lifetime**.

Scope:	source
Type:	**lbm_ulong_t**
Units:	milliseconds

Default value:	0 (disable lifetime)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.2 / UME 3.2 / UMQ 2.1

40.1.8 umq_queue_activity_timeout (context)

The timeout value used to indicate when a queue is marked inactive.

The queue must be active within this interval to be marked inactive. This value must be much larger than the check interval.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	3000 (3.0 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.6/UME 3.0/UMQ 1.0.

40.1.9 umq_queue_participation (receiver)

Flag indicating if the receiver desires to participate in Queuing operations or not.

For general information on queuing, see **Queuing**.

Scope:	receiver
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.6/UME 3.0/UMQ 1.0.

String value	Integer value	Description
"1"	1	The receiver desires to participate in Queuing operations. **Default for all.**
"0"	0	The receiver does not wish to participate in Queuing operations.

40.1.10 umq_queue_registration_id (context)

Add a broker/registration ID pair to the current list of broker/registration ID pairs.

Assigns a Registration ID when connected to the given broker name, using the format "BrokerName:RegID". If a broker is not named or a broker does not support names, the broker will be given the name "Default".

If a Registration ID is set for a given broker, that Registration ID is passed from the source through to the receiver. This information can be used to identify the source from which the data originated.

Each time you set this option, it adds another BrokerName:RegID pair to a list and does not overwrite previous specifications. If you supply an empty name, the list resets.

When the binary form of option setting is used, UM does NOT expect an array of structures. Instead, only one broker/registration ID pair specification can be supplied for each call to **lbm_context_attr_setopt()**. However, when the binary form of option retrieval **lbm_context_attr_getopt()** is used, the list of broker/registration ID pairs is returned as an array, and the **optlen** parameter should be set as:

```
optlen = (max_num_regid_broker_pairs * sizeof(lbm_umq_queue_entry_t));
```

Scope:	context
Type:	**lbm_umq_queue_entry_t**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.6/UME 3.0/UMQ 1.0.

40.1.11 umq_receiver_type_id (receiver)

32-bit value that is used as an identifier to instruct the ULB source as to the type of receiver the receiver should be.

Used by the ULB source to associate various settings with the connecting receiver.

For ULB receivers, see **Application Sets and Receiver Type IDs** for more information.

Scope:	receiver
Type:	**lbm_uint_t**
Units:	identifier
Default value:	0
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.6/UME 3.0/UMQ 1.0.

40.1.12 umq_retransmit_request_interval (receiver)

The interval between retransmission request messages to the ULB source.

Applies to ULB. For general information on queuing, see **Queuing**.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	500 (0.5 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.6/UME 3.0/UMQ 1.0.

40.1.13 umq_retransmit_request_outstanding_maximum (receiver)

The maximum number of messages to request at a single time from the ULB source.

Applies to ULB. For general information on queuing, see **Queuing**.

A value of 0 indicates no maximum.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	messages
Default value:	100
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.6/UME 3.0/UMQ 1.0.

40.1.14 umq_session_id (context)

Specifies the Session ID to use for managing ULB sources and receivers within a context.

For ULB only.

A value of 0 (zero) indicates no Session ID is to be set. Valid formats for session IDs are as follows: A hexadecimal string with a maximum value of FFFFFFFFFFFFFFFE, prefixed with '0x'. An octal string with a maximum value of 1777777777777777777776 prefixed with '0'. A decimal string with a maximum value of 18446744073709551614.

Scope:	context
Type:	**lbm_uint64_t**
Default value:	0 (no session ID)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMQ 5.3.

40.1.15 umq_ulb_application_set (source)

Defines the application sets for a ULB source. Format: "Index1:ID1,ID2,...;Index2:ID3,ID4,..."

"Index1" is the numeric index which defines an application set, and "ID1" is the numeric receiver type ID associated with one or more receivers (see umq_receiver_type_id (receiver)).

At least one application set must be specified for the source to use ULB.

The application set indices in the string can be specified in any order. However, they must be numbered contiguously starting with 0 when the topic is allocated.

When the binary form of option setting is used, UM expects an array of structures. See Setting an Option from Arrays of Binary Values.

For more information on application sets, see **Application Sets and Receiver Type IDs**.

Scope:	source
Type:	**lbm_umq_ulb_receiver_type_entry_t**
Default value:	empty (at least one is required)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1.

40.1.16 umq_ulb_application_set_assignment_function (source)

The assignment function for one or more application sets specified as a list of entries in the format, "Index1↩:value1;Index2:value2;..."

"Index1" is the numeric index which defines an application set, and "value1" is the desired assignment function associated that application set.

When the binary form of option setting is used, UM expects an array of structures. See Setting an Option from Arrays of Binary Values.

Scope:	source
Type:	**lbm_umq_ulb_application_set_attr_t**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1.

String value	Integer value	Description
"default"	LBM_SRC_TOPIC_ATTR_UMQ_ULB_A↩SSIGNMENT_DEFAULT	The default assignment function. **Default for all.**

String value	Integer value	Description
"random"	LBM_SRC_TOPIC_ATTR_UMQ_ULB_A↩ SSIGNMENT_RANDOM	Randomized assignment function.

40.1.17 umq_ulb_application_set_events (source)

The events mask of one or more application sets specified as a list of entries in the format, "Index1:value1;↩ Index2:value2;..."

"Index1" is the numeric index which defines an application set, and "value1" is the event mask to be set associated that application set.

The values may follow the same format as described in umq_ulb_events (source).

Application sets not listed default to a mask of 0.

When the binary form of option setting is used, UM expects an array of structures. See Setting an Option from Arrays of Binary Values.

Scope:	source
Type:	**lbm_umq_ulb_application_set_attr_t**
Default value:	empty (all application sets have a mask of 0)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1.

40.1.18 umq_ulb_application_set_load_factor_behavior (source)

The behavior for the load factor for one or more application sets specified as a list of entries in the format, "Index1:value1;Index2:value2;..."

"Index1" is the numeric index which defines an application set, and "value1" is the load factor behavior associated that application set.

When the binary form of option setting is used, UM expects an array of structures. See Setting an Option from Arrays of Binary Values.

Scope:	source
Type:	**lbm_umq_ulb_application_set_attr_t**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1.

String value	Integer value	Description
"ignored"	LBM_SRC_TOPIC_ATTR_UMQ_ULB_LF_BEHAVIOR_IGNORED	Load Factor information not sent and not processed or taken into assignment consideration. **Default for all.**
"provisioned"	LBM_SRC_TOPIC_ATTR_UMQ_ULB_LF_BEHAVIOR_PROVISIONED	Load Factor information on number of sources sent and processed as well as taken into consideration to reduce the active portion size for each receiver.
"dynamic"	LBM_SRC_TOPIC_ATTR_UMQ_ULB_LF_BEHAVIOR_DYNAMIC	Load Factor information sent and processed as well as taken into consideration during assignment to weight receiver choice.

40.1.19 umq_ulb_application_set_message_lifetime (source)

The message lifetime in milliseconds of one or more application sets specified as a list of entries in the format, "Index1:value1;Index2:value2;..."

"Index1" is the numeric index which defines an application set, and "value1" is the message lifetime to be set associated that application set. A message lifetime of 0 means UMQ never discards the message.

Application sets not listed default to a timeout of 0 [forever].

When the binary form of option setting is used, UM expects an array of structures. See Setting an Option from Arrays of Binary Values.

Scope:	source
Type:	**lbm_umq_ulb_application_set_attr_t**

Default value:	empty (all application sets have a timeout of 0 [forever])
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1.

40.1.20 umq_ulb_application_set_message_max_reassignments (source)

The maximum number of message reassignments before UMQ discards a message for one or more application sets specified as a list of entries in the format, "Index1:value1;Index2:value2;..."

"Index1" is the numeric index which defines an application set, and "value1" is the maximum number of reassignments associated that application set.

UMQ applies the initial assignment to this maximum. Setting this option to 1 means that the message will never be reassigned. The default value of 0 means UMQ never discards the message due to too many reassignments.

Application sets not listed default to a maximum of 0.

When the binary form of option setting is used, UM expects an array of structures. See Setting an Option from Arrays of Binary Values.

Scope:	source
Type:	**lbm_umq_ulb_application_set_attr_t**
Default value:	empty (all application sets have a maximum 0 [never discard due to reassignment])
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1.

40.1.21 umq_ulb_application_set_message_reassignment_timeout (source)

The message reassignment timeout (in milliseconds) of one or more application sets specified as a list of entries in the format, "Index1:value1;Index2:value2;..."

"Index1" is the numeric index which defines an application set, and "value1" is the message reassignment

timeout to be set associated that application set.

Application sets not listed default to a timeout of 10000 (10 seconds).

When the binary form of option setting is used, UM expects an array of structures. See Setting an Option from Arrays of Binary Values.

Scope:	source
Type:	**lbm_umq_ulb_application_set_attr_t**
Default value:	empty (all application sets have a timeout of 10000 [10 sec])
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1.

40.1.22 umq_ulb_application_set_receiver_activity_timeout (source)

The receiver activity timeout (in milliseconds) of one or more application sets specified as a list of entries in the format, "Index1:value1;Index2:value2;..."

"Index1" is the numeric index which defines an application set, and "value1" is the receiver activity timeout associated that application set.

Application sets not listed default to an activity timeout of 10000 (10 seconds).

When the binary form of option setting is used, UM expects an array of structures. See Setting an Option from Arrays of Binary Values.

Scope:	source
Type:	**lbm_umq_ulb_application_set_attr_t**
Default value:	empty (all application sets have a timeout of 10000 [10 sec])
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1.

40.1.23 umq_ulb_application_set_receiver_keepalive_interval (source)

The interval (in milliseconds) between keepalive messages to receivers for one or more application sets specified as a list of entries in the format, "Index1:value1;Index2:value2;..."

"Index1" is the numeric index which defines an application set, and "value1" is the receiver keepalive interval associated that application set.

Application sets not listed default to an activity timeout of 1000 (1 second).

When the binary form of option setting is used, UM expects an array of structures. See Setting an Option from Arrays of Binary Values.

Scope:	source
Type:	**lbm_umq_ulb_application_set_attr_t**
Default value:	empty (all application sets have a timeout of 1000 [1 sec])
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1.

40.1.24 umq_ulb_application_set_round_robin_bias (source)

The bias assignment towards unassigned receivers for one or more application sets specified as a list of entries in the format, "Index1:value1;Index2:value2;..."

"Index1" is the numeric index which defines an application set, and "value1" is the round robin bias associated that application set.

Large values increase the bias toward unassigned receivers. Zero (0) disables the bias.

When the binary form of option setting is used, UM expects an array of structures. See Setting an Option from Arrays of Binary Values.

Scope:	source
Type:	**lbm_umq_ulb_application_set_attr_t**
Default value:	1
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1.

40.1.25 umq_ulb_check_interval (source)

The interval upon which ULB sources check for message reassignment, message discards, and receiver liveness.

See **Ultra Load Balancing (ULB)**.

Scope:	source
Type:	**unsigned long int**
Units:	milliseconds
Default value:	1000 (1 second)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1.

40.1.26 umq_ulb_events (source)

A mask indicating what ULB events should be delivered to the source event callback. Applies to all application sets and receiver types for the source.

For the configuration file as well as string API method of setting this option, the string value may be formatted as hexadecimal value or a list of enumerated values separated by a '|' or ','.

Scope:	source
Type:	**lbm_ulong_t**
Units:	mask
Default value:	0
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1.

String value	Integer value	Description
"MSG_CONSUME", "Msg↩ Consume"	LBM_SRC_TOPIC_ATTR_UM↩ Q_ULB_EVENT_MSG_CONS↩ UME (0x1)	Deliver message consumption events.
"MSG_TIMEOUT", "MsgTimeout"	LBM_SRC_TOPIC_ATTR_UM↩ Q_ULB_EVENT_MSG_TIMEO↩ UT (0x2)	Deliver message timeout/discard events.
"MSG_ASSIGNMENT", "Msg↩ Assignment"	LBM_SRC_TOPIC_ATTR_UM↩ Q_ULB_EVENT_MSG_ASSIG↩ NMENT (0x4)	Deliver message assignment events.
"MSG_REASSIGNMENT", "↩ MsgReassignment"	LBM_SRC_TOPIC_ATTR_UM↩ Q_ULB_EVENT_MSG_REASS↩ IGNMENT (0x8)	Deliver message reassignment events.
"MSG_COMPLETE", "Msg↩ Complete"	LBM_SRC_TOPIC_ATTR_UM↩ Q_ULB_EVENT_MSG_COMP↩ LETE (0x10)	Deliver message completion events. Messages are complete once they are consumed or discarded from all application sets.
"RCV_TIMEOUT", "RcvTimeout"	LBM_SRC_TOPIC_ATTR_UM↩ Q_ULB_EVENT_RCV_TIMEO↩ UT (0x20)	Deliver receiver timeout events.
"RCV_REGISTRATION", "Rcv↩ Registration"	LBM_SRC_TOPIC_ATTR_UM↩ Q_ULB_EVENT_RCV_REGIS↩ TRATION (0x40)	Deliver receiver registration events.
"RCV_DEREGISTRATION", "↩ RcvDeregistration"	LBM_SRC_TOPIC_ATTR_UM↩ Q_ULB_EVENT_RCV_DEREG↩ ISTRATION (0x80)	Deliver receiver deregistration events.
"RCV_READY", "RcvReady"	LBM_SRC_TOPIC_ATTR_UM↩ Q_ULB_EVENT_RCV_READY (0x100)	Deliver receiver ready events.

40.1.27 umq_ulb_flight_size (source)

Specifies the number of messages allowed to be in flight (unconsumed) before a new message send either blocks or triggers a notification (source event).

See **Ultra Load Balancing (ULB)**.

Scope:	source
Type:	**unsigned int**
Units:	messages
Default value:	1000
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1.1/UME 3.1.1/UMQ 1.1.1

40.1.28 umq_ulb_flight_size_behavior (source)

The behavior that UMQ follows when a message send exceeds the source's flight size.

See umq_ulb_flight_size (source).

Scope:	source
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1.1/UME 3.1.1/UMQ 1.1.1

String value	Integer value	Description
"Block"	LBM_FLIGHT_SIZE_BEHAVIOR_BLOCK	The send call blocks when a message send exceeds the source's flight size. If the message send is a non-blocking send, the send returns an LBM_EWOULD_BLOCK. **Default for all.**
"Notify"	LBM_FLIGHT_SIZE_BEHAVIOR_NOTIFY	A message send that exceeds the configured flight size does not block but triggers a flight size notification (source event), indicating that the flight size has been surpassed. UMQ also sends a source event notification if the number of in-flight messages falls below the configured flight size.

40.1.29 umq_ulb_receiver_events (source)

Set the events mask of one or more receiver types specified as a list of entries in the format, "ID1:value1;ID2↩:value2;..."

"ID1" is the numeric receiver type ID associated with one or more receivers (see umq_receiver_type_id (receiver)), and "value1" is the evet mask to be associated with receivers of that type.

The values may follow the same format as described in umq_ulb_events (source).

Receivers with types not listed default to a mask of 0.

When the binary form of option setting is used, UM expects an array of structures. See Setting an Option from Arrays of Binary Values.

Scope:	source
Type:	**lbm_umq_ulb_receiver_type_attr_t**
Default value:	empty (all receiver types have a mask of 0)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1.

40.1.30 umq_ulb_receiver_portion (source)

The portion size of one or more receiver types specified as a list of entries in the format: "ID1:value1;ID2↩ :value2;..."

"ID1" is the numeric receiver type ID associated with one or more receivers (see umq_receiver_type_id (receiver)), and "value1" is the portion size to be associated with receivers of that type.

Receivers with types not listed default to a portion size of 1.

When the binary form of option setting is used, UM expects an array of structures. See Setting an Option from Arrays of Binary Values.

Scope:	source
Type:	**lbm_umq_ulb_receiver_type_attr_t**
Default value:	empty (all receivers have portion size of 1)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1.

40.1.31 umq_ulb_receiver_priority (source)

The priority of one or more receiver types specified as a list of entries in the format, "ID1:value1;ID2:value2;..."

"ID1" is the numeric receiver type ID associated with one or more receivers (see umq_receiver_type_id (receiver)), and "value1" is the priority to be associated with receivers of that type.

Receivers with types not listed default to a priority of 0.

When the binary form of option setting is used, UM expects an array of structures. See Setting an Option from Arrays of Binary Values.

Scope:	source
Type:	**lbm_umq_ulb_receiver_type_attr_t**
Default value:	empty (all receivers have priority of 0)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1.

40.1.32 umq_ulb_source_activity_timeout (receiver)

The timeout value used to indicate when a ULB source is unresponsive.

The ULB source must not be active within this interval to be considered unresponsive. This value must be much larger than the source check interval.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	10000 (10 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.2 / UME 3.2 / UMQ 2.1

40.1.33 umq_ulb_source_check_interval (receiver)

The interval between activity checks of a ULB source.

Allow a ULB receiver to proactively attempt re-registration with a ULB source if the receiver has not seen any activity (including keepalives) from that source in a specified amount of time, provided the source's transport session is still alive and valid.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	1000 (1 second)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.2 / UME 3.2 / UMQ 2.1

Chapter 41

Hot Failover Operation Options

Hot Failover (HF) allows your applications to build in sender redundancy. See *Hot Failover* in the *Ultra Messaging Concepts Guide* for a discussion of using Hot Failover within a single receiver context or across multiple receiver contexts.

41.1 Reference

41.1.1 delivery_control_loss_check_interval (hfx)

The interval between periodic forced loss checks.

This option defaults to 0, indicating that loss checks should only be made when a new message arrives.

Scope:	hfx
Type:	**lbm_ulong_t**
Units:	msec
Default value:	0 (no periodic loss checks)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.2

41.1.2 delivery_control_max_delay (hfx)

The minimum interval that must expire before the HFX Receiver declares a message unrecoverable and delivers an unrecoverable loss message the application.

By default, the HFX Receiver only checks loss when it receives new messages. To enable periodic loss checks, set the delivery_control_loss_check_interval (hfx) option.

Scope:	hfx
Type:	**lbm_ulong_t**
Units:	msec
Default value:	10000 (10 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.2

41.1.3 delivery_control_maximum_burst_loss (hfx)

This controls the size of a topic sequence number gap past which the gap is declared a "burst loss".

Note that the default value for HFX is different than for non-HFX receivers.

See Burst Loss for a detailed explanation of burst loss and its semantics.

Note

the burst loss control takes priority over all recovery methods. For example, if the receiver is reading a persistent stream and OTR is enabled, a gap longer than delivery_control_maximum_burst_loss will immediately declare the gap as unrecoverable without even trying to use OTR to recover. If message integrity is a high priority, delivery_control_maximum_burst_loss should be set to a very large value.

See **Hot Failover Across Multiple Contexts (HFX)**.

Scope:	hfx
Type:	**lbm_uint_t**
Units:	number of messages (fragments)
Default value:	512
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.2

41.1.4 delivery_control_maximum_total_map_entries (hfx)

The maximum number of map entries for the HFX order and loss maps.

This is a soft limit. When the sum of the number of loss records and the number of messages held for ordering (messages that will be delivered once all prior messages have been delivered) is greater than this value, the oldest consecutive sequence of loss records will be declared lost immediately to reduce the number of outstanding map entries. A value of 0 indicates that the map should be allowed to grow without bound.

Scope:	hfx
Type:	**size_t**
Units:	map entries
Default value:	200000
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.2

41.1.5 duplicate_delivery (hfx)

Flag indicating whether duplicate messages should be discarded or simply marked as duplicates.

Setting this to 1 overrides the hf_duplicate_delivery (receiver) setting on all underlying HFX Receivers.

Scope:	hfx
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.2

Value	Description
1	The HFX delivers duplicate messages.

Value	Description
0	The HFX does not deliver duplicate messages. **Default for all.**

41.1.6 hf_duplicate_delivery (receiver)

Flag indicating if the Hot Failover receiver delivers duplicate messages or not.

In normal operation, Hot Failover only delivers the first copy received of a message.

See **Hot Failover (HF)** for more information.

Scope:	receiver
Type:	**int**
When to Set:	Can only be set during object initialization.

Value	Description
1	The Hot Failover receiver delivers duplicate messages.
0	The Hot Failover receiver does not deliver duplicate messages. **Default for all.**

41.1.7 hf_optional_messages (receiver)

Indicates if a Hot Failover receiver can receive optional messages.

See also **Hot Failover (HF)**.

Scope:	receiver
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.2.5/UME 3.2.5/UMQ 2.1.5

Value	Description
1	Hot Failover receivers can receive optional messages. **Default for all.**
0	Hot Failover receivers do not receive optional messages.

41.1.8 hf_receiver (wildcard_receiver)

Specifies whether to create hot failover receivers for each topic that maps to the wildcard receiver pattern.

See **Hot Failover (HF)** for more information.

Scope:	wildcard_receiver
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMS 5.2.2

Value	Description
1	Create hot failover receivers for each matched topic.
0	Normal wildcard receiver operation. Hot failover sequence numbers are ignored. **Default for all.**

41.1.9 ordered_delivery (hfx)

Flag indicating if the HFX Receiver orders messages before delivery.

See **Hot Failover Across Multiple Contexts (HFX)**.

Scope:	hfx
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.2

String value	Integer value	Description
"1"	1	The HFX Receiver delivers messages in order. **Default for all.**
"-1"	-1	The HFX Receiver delivers messages as soon as they are received. In the case of fragmented messages, as soon as all fragments have been received and reassembled.

Chapter 42

Automatic Monitoring Options

The Monitoring Options below apply to a given UM context. You can override the default values of these options and apply monitoring option values to all UM contexts (transports and event queues) with the following environment variables.

- LBM_MONITOR_INTERVAL
- LBM_MONITOR_TRANSPORT
- LBM_MONITOR_TRANSPORT_OPTS
- LBM_MONITOR_APPID

These variables will not override any Monitoring Options you explicitly set. The environment variables only override Monitoring Options default values.

If you do not specify any monitoring options either in a UM configuration file or via **lbm_context_attr_setopt()** calls, no monitoring will occur. However, if you then set the LBM_MONITOR_INTERVAL environment variable to 5, you will turn on automatic monitoring for every UM context your application creates at 5 second intervals. If you then set monitor_interval to 10 for a particular context, all transport sessions in that context will be monitored every 10 seconds.

For XML configuration files, you can configure an automatic monitoring context by setting the <context> attribute **name=infa_statistics_context**.

See **Automatic Monitoring** for more information about this feature.

42.1 Reference

42.1.1 monitor_appid (context)

An application ID string used by automatic monitoring to identify the application generating the context and transport statistics.

See **Automatic Monitoring** for an overview.

Scope:	context
Type:	**string**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.4/UME 2.1.

42.1.2 monitor_appid (event_queue)

An application ID string used by automatic monitoring to identify the application generating the event queue statistics.

See **Automatic Monitoring** for an overview.

Scope:	event_queue
Type:	**string**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.4/UME 2.1.

42.1.3 monitor_format (context)

The LBMMON format module to be used for automatic monitoring.

See **Automatic Monitoring**.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.14.

String value	Integer value	Description
"csv"	LBM_CTX_ATTR_MON_FORMAT_CSV	Use the LBMMON comma-separated values format module. **Default for all.**
"pb"	LBM_CTX_ATTR_MON_FORMAT_PB	Use the LBMMON Protocol Buffers format module.

42.1.4 monitor_format (event_queue)

The LBMMON format module to be used for automatic monitoring.

See **Automatic Monitoring**.

Scope:	event_queue
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.14.

String value	Integer value	Description
"csv"	LBM_CTX_ATTR_MON_FORMAT_CSV	Use the LBMMON comma-separated values format module. **Default for all.**
"pb"	LBM_CTX_ATTR_MON_FORMAT_PB	Use the LBMMON Protocol Buffers format module.

42.1.5 monitor_format_opts (context)

An option string to be passed to the LBMMON format module for automatic monitoring of contexts and transports.

The option monitor_format (context) is used to select a formatting module for automatic monitoring. That module is passed the value string supplied for monitor_format_opts to configure the module. The format of the value string can vary, depending on the formatting module chosen. See **Monitoring Format Modules** for module-specific details.

For example, for the "pb" (protocol buffer) module, the following directs the formatting module to pass through and convert CSV data to protocol buffers:

```
context monitor_format_opts passthrough=convert
```

Scope:	context
Type:	**string**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.14

42.1.6 monitor_format_opts (event_queue)

An option string to be passed to the LBMMON format module for automatic monitoring of event queues.

The option monitor_format (context) is used to select a formatting module for automatic monitoring. That module is passed the value string supplied for monitor_format_opts to configure the module. The format of the value string can vary, depending on the formatting module chosen. See **Monitoring Format Modules** for module-specific details.

For example, for the "pb" (protocol buffer) module, the following directs the formatting module to pass through and convert CSV data to protocol buffers:

```
event_queue monitor_format_opts passthrough=convert
```

Scope:	event_queue
Type:	**string**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.14

42.1.7 monitor_interval (context)

Interval at which automatic monitoring retrieves the statistics for a context and all transport sessions on that context.

Setting this option to zero (the default) disables the automatic monitoring of a context's transport sessions.

See **Automatic Monitoring** for an overview.

Scope:	context
Type:	**lbm_ulong_t**
Units:	seconds
Default value:	0
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.4/UME 2.1.

42.1.8 monitor_interval (event_queue)

Interval at which automatic monitoring retrieves the statistics for an event queue.

Setting this option to zero (the default) disables the automatic monitoring of an event queue. When monitoring Event Queue statistics you must enable the Event Queue UM Configuration Options, queue_age_enabled (event_queue), queue_count_enabled (event_queue) and queue_service_time_enabled (event_queue). UM disables these options by default, which produces no event queue statistics.

See **Automatic Monitoring** for an overview.

Scope:	event_queue
Type:	**lbm_ulong_t**
Units:	seconds
Default value:	0
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.4/UME 2.1.

42.1.9 monitor_interval (receiver)

Interval at which automatic monitoring retrieves topic interest by the application.

Topic interest information contains source and topic information if the receiver has joined the source transport session. If the topic interest information is blank, the receiver has not joined a source transport session. UM System Monitoring uses this information to monitor the number of subscribed topics. Setting this option to zero (the default) disables the automatic monitoring of receiver interest.

Warning

Enabling this for applications with large numbers of receivers can produce very large amounts of monitoring data.

See **Automatic Monitoring** for an overview.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	seconds
Default value:	0
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.5.

42.1.10 monitor_interval (wildcard_receiver)

Interval at which automatic monitoring retrieves wildcard pattern interest by the application.

Topic interest information contains source and topic information if the receiver has joined the source transport session. If the topic interest information is blank, the receiver has not joined a source transport session. UM System Monitoring uses this information to monitor the number of subscribed topics. Setting this option to zero (the default) disables the automatic monitoring of a wildcard receiver interest.

Warning

Enabling this for applications with large numbers of receivers can produce very large amounts of monitoring data.

See **Automatic Monitoring** for an overview.

Scope:	wildcard_receiver
Type:	**lbm_ulong_t**
Units:	seconds
Default value:	0
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 6.5.

42.1.11 monitor_transport (context)

The LBMMON transport module to be used for automatic monitoring of a context and all transport sessions on that context.

The monitor_transport_opts (context) option passes configuration information to the selected transport module.

Note that the UDP monitoring transport cannot be selected for automatic monitoring.

See **Automatic Monitoring** for an overview.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.4/UME 2.1.

String value	Integer value	Description
"lbm"	LBM_CTX_ATTR_MON_TRANSPORT_LBM	Use the LBMMON lbm transport module. **Default for all.**
"lbmsnmp"	LBM_CTX_ATTR_MON_TRANSPORT_LBMSNMP	Use the LBMMON lbmsnmp transport module. This value is required if you use the UM SNMP Agent.

42.1.12 monitor_transport (event_queue)

The LBMMON transport module to be used for automatic monitoring of event queues.

The monitor_transport_opts (event_queue) option passes configuration information to the selected transport module.

Note that the UDP monitoring transport cannot be selected for automatic monitoring.

See **Automatic Monitoring** for an overview.

Scope:	event_queue
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.4/UME 2.1.

String value	Integer value	Description
"lbm"	LBM_CTX_ATTR_MON_TRANSPORT_LBM	Use the LBMMON lbm transport module. **Default for all.**
"lbmsnmp"	LBM_CTX_ATTR_MON_TRANSPORT_LBMSNMP	Use the LBMMON lbmsnmp transport module. This value is required if you use the UM SNMP Agent.

42.1.13 monitor_transport_opts (context)

An option string to be passed to the LBMMON transport module for automatic monitoring of contexts and transports.

The option monitor_transport (context) is used to select a transport module for automatic monitoring. That module is passed the value string supplied for monitor_transport_opts to configure the module. The format of the value string can vary, depending on the transport module chosen. See **Monitoring Transport Modules** for module-specific details.

For example, for the LBM and SNMP transport modules, the following directs the transport module to use a config file and overrides the resolver interface:

```
context monitor_transport_opts context|resolver_multicast_interface="en0";config=/um/
```

Note

Some UM options specify interfaces, which can be done by supplying the device name of the interface. Special care must be taken when including this option in XML configuration files. See Interface Device Names and XML for details.

See **Application Monitoring Context** for information about the context and limitations on how you can configure it.

Scope:	context
Type:	**string**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.4/UME 2.1.

42.1.14 monitor_transport_opts (event_queue)

An option string to be passed to the LBMMON transport module for automatic monitoring of event queues.

The option monitor_transport (event_queue) is used to select a transport module for automatic monitoring. That module is passed the value string supplied for monitor_transport_opts to configure the module. The format of the value string can vary, depending on the transport module chosen. See **Monitoring Transport Modules** for module-specific details.

For example, for the LBM and SNMP transport modules, the following directs the transport module to use a config file and overrides the resolver interface:

```
event_queue monitor_transport_opts context|resolver_multicast_interface="en0";config=
```

Note

Some UM options specify interfaces, which can be done by supplying the device name of the interface. Special care must be taken when including this option in XML configuration files. See Interface Device Names and XML for details.

See **Application Monitoring Context** for information about the context and limitations on how you can configure it.

Scope:	event_queue
Type:	**string**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.4/UME 2.1.

Chapter 43

Deprecated Options

43.1 Reference

43.1.1 datagram_acceleration_functions (context)

DEPRECATED, do not use. Specifies the callback functions that implement Datagram Acceleration.

Refer to the description of **lbm_datagram_acceleration_func_t** for the callback definitions.

Scope:	context
Type:	**lbm_datagram_acceleration_func_t**
Default value:	NULL
When to Set:	Can only be set during object initialization.
Config File:	Cannot be set from an UM configuration file.
Version:	This option was implemented in UM 6.10
Version:	This option was deprecated in UM 6.13.1

43.1.2 delivery_control_loss_tablesz (receiver)

DEPRECATED, this controls the size of the hash table index used for storing unrecoverable loss state on a per source per topic basis.

For LBT-RM and other datagram-based transport sessions only. Larger values mean larger hash tables and probably better CPU usage under loss scenarios at the cost of more memory per source per topic. Smaller values mean smaller hash tables and probably worse CPU usage under loss scenarios but with less memory usage. The value used should be a prime number for efficiency.

Scope:	receiver
Type:	**size_t**
Units:	table entries
Default value:	131
When to Set:	Can only be set during object initialization.
Version:	Deprecated

43.1.3 delivery_control_order_tablesz (receiver)

DEPRECATED, this controls the size of the hash table index used for storing buffered data on a per source per topic basis when ordered delivery is used.

For LBT-RM and other datagram-based transport sessions only. Larger values mean larger hash tables and probably better CPU usage under loss scenarios at the cost of more memory per source per topic. Smaller values mean smaller hash tables and probably worse CPU usage under loss scenarios but with less memory usage. The value used should be a prime number for efficiency.

Scope:	receiver
Type:	**size_t**
Units:	table entries
Default value:	131
When to Set:	Can only be set during object initialization.
Version:	Deprecated

43.1.4 implicit_batching_type (source)

DEPRECATED, determines the algorithm to use for implicit batching.

This option has been deprecated because the adaptive batching algorithm has the same worst case for latency as the default algorithm and is not better for throughput. This is because, even with adaptive batching, UM cannot predict when the application will stop sending, at which point (unless the application calls the flush API) the implicit batching interval must expire before the batch will be sent. To minimize latency while batching, it is most effective to call the flush API whenever the application will not immediately send another message.

Scope:	source
Type:	**int**
When to Set:	May be set during operation.
Version:	This option was deprecated in UM 6.9.

String value	Integer value	Description
"default"	LBM_SRC_TOPIC_ATTR_IMPLICIT_BATCH_TYPE_DEFAULT	Implicit batching is controlled entirely by the implicit_batching_minimum_length (source) and implicit_batching_interval (source) options. Refer to **Message Batching** for additional information. **Default for all.**
"adaptive"	LBM_SRC_TOPIC_ATTR_IMPLICIT_BATCH_TYPE_ADAPTIVE	Source-paced batching method that attempts to adjust the amount of messages sent in each batch automatically. The options, implicit_batching_minimum_length (source) and implicit_batching_interval (source), limit batch sizes and intervals but sizes and intervals will usually be much smaller. Setting this option may have a negative impact on maximum throughput.

43.1.5 network_compatibility_mode (context)

DEPRECATED, enable compatibility mode which allows UM versions LBM-4.2/UME-3.2/UMQ-2.1 through UM 5.∗ to interoperate with UM versions prior to LBM-4.2/UME-3.2/UMQ-2.1 by blocking the sending of some header option types.

This option has no effect on Ultra Messaging Versions 6.0 and later.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.

Version:	This option was implemented in LBM 4.2/UME 3.2/UMQ 2.1.
Version:	This option was deprecated in UM 6.0 (documentation was updated to reflect this deprecation in UM 6.9).

43.1.6 otr_request_duration (receiver)

DEPRECATED, the length of time a receiver continues to send OTR lost-message requests before giving up. This option is deprecated in favor of otr_request_message_timeout (receiver).

See **Off-Transport Recovery (OTR)**.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	20000 (20 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UM 5.2
Version:	This option was deprecated in UM 6.0

43.1.7 pattern_callback (wildcard_receiver)

DEPRECATED, callback function (and associated client data pointer) that is called when a pattern match is desired for a topic discovered for a wildcard receiver if the pattern type is set to "appcb".

This callback is called by the context thread and can not use an event queue. Therefore the callback function used should not block or it will delay reception of latency-sensitive messages.

A function return value of 0 indicates the given topic should be considered part of the wildcard. A value of 1 or more indicates the topic should NOT be considered matching the wildcard.

Scope:	wildcard_receiver
Type:	**lbm_wildcard_rcv_compare_func_t**

Default value:	NULL
When to Set:	Can only be set during object initialization.
Config File:	Cannot be set from an UM configuration file.

43.1.8 rcv_sync_cache (receiver)

DEPRECATED, UMCache only - a valid cache address (such as TCP:192.168.5.11:4567) in the standard form of `TCP:address:port` enables a UM receiver to use UMCache to receive a snapshot of larger, multiple-field messages stored by UMCache.

Receiving applications can then become synchronized with the live stream of messages sent on the receiver's topic. **`address`** is the IP address of the machine where the UMCache runs and **`port`** is the configured port where the cache request handler listens.

Scope:	receiver
Type:	**umcache_reqlib_request_info_t**
Default value:	NULL
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMS 5.0/UMP 5.0/UMQ 5.0
Version:	This option was deprecated in UM 6.9

43.1.9 rcv_sync_cache_timeout (receiver)

DEPRECATED, the maximum time period that a UM receiver waits for a snapshot message from the UMCache.

UMCache only.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	2000 (2 seconds)

When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMS 5.0/UMP 5.0/UMQ 5.0
Version:	This option was deprecated in UM 6.9

43.1.10 receive_thread_pool_size (context)

DEPRECATED, this option no longer functions.

It used to define the maximum number of threads available for transports (excluding the context thread). The MTT feature is replaced in 6.11 and beyond by **Transport Services Provider (XSP)**.

Scope:	context
Type:	**int**
Default value:	4
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1.
Version:	This option was deprecated in UM 6.9
Version:	This option was removed from UM 6.11

43.1.11 resolver_active_source_interval (context)

DEPRECATED, interval between sending Topic Resolution advertisements for active sources.

A value of 0 indicates that periodic advertisements should not be sent (sources will still respond to queries). When set to 0, the **`resolver_active_threshold`** should typically also be set to 0. See also Disabling Aspects of Topic Resolution.

Note: Although this option is eligible to be set during operation, two considerations exist. If this option is disabled at initialization (set to 0), you cannot re-set the option during operation.Disabling this option by setting it to 0 (zero) during operation prevents you from re-setting the option a second time during operation.

Scope:	context
Type:	**unsigned long int**
Units:	milliseconds
Default value:	1000 (1 second)
When to Set:	May be set during operation.
Version:	This option was deprecated in LBM 4.0

43.1.12 resolver_active_threshold (context)

DEPRECATED, number of seconds since the last application message was sent to a source that causes that source to be marked inactive.

Inactive sources are not advertised periodically (but will continue to respond to queries). A value of 0 indicates that sources will advertise periodically regardless of how often the application sends messages. Note that for publishers with large numbers of sources, this can increase the topic resolution traffic load.

However, also note that this option SHOULD be set to 0 if periodic advertisements are disabled. See Disabling Aspects of Topic Resolution and Interrelated Configuration Options.

Scope:	context
Type:	**unsigned long int**
Units:	seconds
Default value:	60
When to Set:	May be set during operation.
Version:	This option was deprecated in LBM 4.0

43.1.13 resolver_context_advertisement_interval (context)

DEPRECATED, interval between context advertisements.

Setting this option to 0 disables context advertisements, though DRO and other functionality depends upon context advertisements, so a value of 0 is not generally recommended.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	10000 (10 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was deprecated in UM 6.0

43.1.14 resolver_maximum_advertisements (context)

DEPRECATED, maximum number of topics that will be advertised per active source interval.

A value of 0 means to advertise all topics.

Scope:	context
Type:	**unsigned long int**
Units:	Number of topics
Default value:	0 (all topics)
When to Set:	May be set during operation.
Version:	This option was deprecated in LBM 4.0

43.1.15 resolver_maximum_queries (context)

DEPRECATED, maximum number of topics that will be queried for per query interval.

A value of 0 means to query for all topics that do not have at least one source.

Scope:	context
Type:	**unsigned long int**
Units:	Number of topics
Default value:	0 (all topics with no source)

When to Set:	May be set during operation.
Version:	This option was deprecated in LBM 4.0

43.1.16 resolver_query_interval (context)

DEPRECATED, interval between query transmissions for receivers attempting Topic Resolution.

A value of 0 indicates queries should not be sent. See also Disabling Aspects of Topic Resolution.

Note: Although this option is eligible to be set during operation, two considerations exist. If this option is disabled at initialization (set to 0), you cannot re-set the option during operation.Disabling this option by setting it to 0 (zero) during operation prevents you from re-setting the option a second time during operation.

Scope:	context
Type:	**unsigned long int**
Units:	milliseconds
Default value:	100 (0.1 seconds)
When to Set:	May be set during operation.
Version:	This option was deprecated in LBM 4.0

43.1.17 resolver_query_max_interval (wildcard_receiver)

DEPRECATED, this sets the maximum interval between wildcard queries in topic resolution (when used).

Only PCRE and regex pattern types can use wildcard queries. A value of 0 indicates wildcard queries should not be sent. UM currently queries a maximum of 250 unique wildcard patterns (receivers).

Note: Although this option is eligible to be set during operation, two considerations exist.

- If this option is disabled at initialization (set to 0), you cannot re-set the option during operation.
- Disabling this option by setting it to 0 (zero) during operation prevents you from re-setting the option a second time during operation.

Scope:	wildcard_receiver
Type:	**unsigned long int**
Units:	milliseconds
Default value:	0 (do not query)
When to Set:	Can only be set during object initialization.
Version:	This option was deprecated in LBM 4.0

43.1.18 resolver_unicast_address (context)

DEPRECATED, the IP address (or domain name of the IP address) to send unicast topic resolution messages to.

This option was deprecated in UMS 5.0. Use resolver_unicast_daemon (context) instead.

If set to 0.0.0.0 (INADDR_ANY), then topic resolution uses multicast (the default). If set to anything else, then topic resolution messages go to the IP address specified.

Scope:	context
Type:	**struct in_addr**
Default value:	0.0.0.0 (INADDR_ANY)
When to Set:	Can only be set during object initialization.
Version:	This option was deprecated in UMS 5.0.

43.1.19 resolver_unicast_destination_port (context)

DEPRECATED, the UDP port to send unicast topic resolution messages to. This is the UDP port used by the UM resolution daemon (lbmrd).

This option was deprecated in UMS 5.0. Use resolver_unicast_daemon (context) instead.

See Port Assignments for more information about configuring ports.

Scope:	context
Type:	**lbm_uint16_t**
Default value:	15380
Byte order:	Network
When to Set:	Can only be set during object initialization.
Version:	This option was deprecated in UMS 5.0.

43.1.20 resolver_unicast_port (context)

DEPRECATED, the local UDP port used for unicast topic resolution messages.

This option was deprecated in UMS 5.0. Use resolver_unicast_daemon (context) instead. The UM resolution daemon (lbmrd) will send unicast topic resolution messages to this UDP port. A value of 0 indicates that UM should pick an open port in the range (resolver_unicast_port_low (context), resolver_unicast_port_high (context)). See Port Assignments for more information about configuring ports.

Scope:	context
Type:	**lbm_uint16_t**
Default value:	0 (pick open port)
Byte order:	Network
When to Set:	Can only be set during object initialization.
Version:	This option was deprecated in UMS 5.0.

43.1.21 retransmit_message_map_tablesz (source)

DEPRECATED, the size of the hash table that the source uses to store messages for the retention policy in effect.

A larger table means more messages can be stored more efficiently, but takes up more memory. A smaller table uses less memory, but costs more CPU time as more messages are retained. See **Configuring Late Join for Large Numbers of Messages** for additional information.

Scope:	source
Type:	**size_t**
Default value:	131
When to Set:	Can only be set during object initialization.
Version:	This option has been deprecated.

43.1.22 retransmit_request_generation_interval (receiver)

DEPRECATED, the maximum interval between when a receiver first sends a retransmission request and when the receiver stops and reports loss on the remaining RXs not received.

See **Configuring Late Join for Large Numbers of Messages** for additional information.

This option is deprecated and has no effect. Use retransmit_request_message_timeout (receiver) instead.

Scope:	receiver
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	10000 (10 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was deprecated in UM 6.0

43.1.23 retransmit_retention_age_threshold (source)

DEPRECATED, specifies the minimum age of messages in the retained message buffer before UM can delete them. UM cannot delete any messages younger than this value.

For UMS Late Joins, this and retransmit_retention_size_threshold (source) are the only options that affect the retention buffer size. For UME, these two options combined with retransmit_retention_size_limit (source) affect the retention buffer size. UM deletes a message when it meets all configured threshold criteria, i.e., the message is older than this option (if set), and the size of the retention buffer exceeds the **`retransmit_retention_size_threshold`** (if set). A value of 0 sets the age threshold to be always triggered, in which case deletion is determined by other threshold criteria.

With **Smart Sources**, this option is ignored. Retention buffers are preallocated and are never deleted.

Scope:	source
Type:	**lbm_ulong_t**
Units:	seconds
Default value:	0 (threshold always triggered)
When to Set:	Can only be set during object initialization.
Version:	This option was deprecated in LBM 6.10

43.1.24 source_cost_evaluation_function (context)

DEPRECATED, callback function that you can use in the **lbm_src_cost_function_cb()** to evaluate or determine the cost of a message path.

The DRO evaluates the cost of any new topic it detects. The callback supplied with this option can affect the cost of topics to bias the DRO toward certain message paths.

Scope:	context
Type:	**lbm_src_cost_func_t**
Default value:	NULL
When to Set:	Can only be set during object initialization.
Config File:	Cannot be set from an UM configuration file.
Version:	This option was implemented in UMS 5.0/UMP 5.0/UMQ 5.0
Version:	This option was deprecated in UM 6.0

43.1.25 transport_datagram_max_size (context)

DEPRECATED, do not use. The maximum datagram size that can be generated by UM. The default value is 8192, the minimum is 400 bytes, and the maximum is 65535.

This configuration option is replaced by the following transport-specific options: transport_tcp_datagram_max_size (context), transport_lbtrm_datagram_max_size (context), transport_lbtru_datagram_max_size (context),

transport_lbtipc_datagram_max_size (context), transport_lbtsmx_datagram_max_size (source).

Scope:	context
Type:	**unsigned int**
Units:	bytes
Default value:	8192
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.3.5/UME 2.0.3.
Version:	This option was deprecated in LBM 4.1

43.1.26 transport_lbtipc_acknowledgement_interval (receiver)

DEPRECATED, period of time between acknowledgement (keepalive) messages sent from the receiver to the IPC source.

See also transport_lbtipc_client_activity_timeout (source).

Scope:	receiver
Type:	**unsigned long int**
Units:	milliseconds
Default value:	500 (0.5 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was deprecated in LBM 4.0

43.1.27 transport_lbtipc_client_activity_timeout (source)

DEPRECATED, the maximum period of inactivity (lack of acknowledgement keepalive messages) from a receiver before the source deletes the receiver from its active receiver table.

The IPC source signals all receivers in its active receiver's table when it writes new data to the shared memory area. See also transport_lbtipc_acknowledgement_interval (receiver).

Scope:	source
Type:	**unsigned long int**
Units:	milliseconds
Default value:	10,000 (10 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was deprecated in LBM 4.0

43.1.28 transport_lbtrdma_datagram_max_size (context)

DEPRECATED, the maximum datagram size that can be generated for a LBT-RDMA transport session. The default value is 4096, the minimum is 500 bytes, and the maximum is 4096.

See **Message Fragmentation and Reassembly** for more information.

Warning

When the DRO is in use, it is recommended that all UM applications and components (including the DRO and Persistent Store) share the same maximum datagram size setting. See **Protocol Conversion**.

Users of **kernel-bypass drivers** should also see **Dynamic Fragmentation Reduction**.

Scope:	context
Type:	**lbm_uint_t**
Units:	bytes
Default value:	4096
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1
Version:	This option was deprecated in UM 6.9

43.1.29 transport_lbtrdma_interface (source)

DEPRECATED, do not use. Specifies the network interface over which UM LBT-RDMA sources receive connection requests from topic receivers.

You can specify the full IP address of the interface, or just the network part (see Specifying Interfaces for details).

Default is set to INADDR_ANY, meaning that it accepts incoming connection requests from any interface.

Be aware that the first source joining a transport session sets the interface with this option. Thus, setting a different interface for a subsequent topic that maps onto the same transport session will have no effect.

Scope:	source
Type:	**lbm_ipv4_address_mask_t**
Default value:	0.0.0.0 (INADDR_ANY)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1
Version:	This option was deprecated in UM 6.9

43.1.30 transport_lbtrdma_maximum_ports (context)

DEPRECATED, maximum number of LBT-RDMA sessions to allocate.

See Port Assignments for more information about configuring ports.

Scope:	context
Type:	**lbm_uint16_t**
Units:	number of ports
Default value:	5
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1
Version:	This option was deprecated in UM 6.9

43.1.31 transport_lbtrdma_port (source)

DEPRECATED, port number for a specific source's LBT-RDMA session.

Must be outside the transport_lbtrdma_port_low (context) and transport_lbtrdma_port_high (context) range.

See Port Assignments for more information about configuring ports.

Scope:	source
Type:	**lbm_uint16_t**
Default value:	0 (zero)
Byte order:	Host
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1
Version:	This option was deprecated in UM 6.9

43.1.32 transport_lbtrdma_port_high (context)

DEPRECATED, highest port number that can be assigned to a LBT-RDMA session.

See Port Assignments for more information about configuring ports.

Scope:	context
Type:	**lbm_uint16_t**
Default value:	20,020
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1
Version:	This option was deprecated in UM 6.9

43.1.33 transport_lbtrdma_port_low (context)

DEPRECATED, lowest port number that can be assigned to a LBT-RDMA session.

See Port Assignments for more information about configuring ports.

Scope:	context
Type:	**lbm_uint16_t**
Default value:	20,001
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1
Version:	This option was deprecated in UM 6.9

43.1.34 transport_lbtrdma_receiver_thread_behavior (context)

DEPRECATED, receiver behavior for monitoring a LBT-RDMA source's shared memory area for new data.

LBT-RDMA is deprecated.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1
Version:	This option was deprecated in UM 6.9

String value	Integer value	Description
"pend"	LBM_CTX_ATTR_RDMA_RCV_THREAD_PEND	Receiver waits (sleep) for notification from RDMA that the source has updated the shared memory area with new data. Default. **Default for all.**
"busy_wait"	LBM_CTX_ATTR_RDMA_RCV_THREAD_BUSY_WAIT	UM polls the shared memory area for new data.

43.1.35 transport_lbtrdma_transmission_window_size (source)

DEPRECATED, size of an LBT-RDMA transport's shared memory area.

This value may vary across platforms. The actual size of the shared memory area equals the value you specify for this option plus about 64 KB for header information. The minimum value for this option is 65,536.

Refer to **Source Object** for additional information.

Scope:	source
Type:	**size_t**
Units:	bytes
Default value:	25165824 (24 MB)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1/UMQ 1.1
Version:	This option was deprecated in UM 6.9

43.1.36 ume_message_map_tablesz (source)

DEPRECATED, the size of the hash table that the source uses to store messages for the retention policy in effect.

A larger table means more messages can be stored more efficiently, but takes up more memory. A smaller table uses less memory, but costs more CPU time as more messages are retained. This setting no longer has any effect.

Scope:	source
Type:	**size_t**
Default value:	131
When to Set:	Can only be set during object initialization.
Version:	This option has been deprecated.

43.1.37 ume_primary_store_address (source)

DEPRECATED, IPv4 address of the persistent Store to be used as the primary Store.

A value of 0.0.0.0 (or INADDR_ANY) indicates no Store is set as the primary. In other words, persistence is not enabled for the source.

This setting is deprecated. Its use is not recommended except by legacy systems. Please use the ume_store (source) option instead.

Scope:	source
Type:	**struct in_addr**
Default value:	0.0.0.0 (INADDR_ANY)
When to Set:	Can only be set during object initialization.
Version:	This option was deprecated in UME 2.0

43.1.38 ume_primary_store_port (source)

DEPRECATED, TCP port of the primary persistent Store. This setting is deprecated. Its use is not recommended except by legacy systems. Please use the ume_store option instead.

See Port Assignments for more information about configuring ports.

Scope:	source
Type:	**lbm_uint16_t**
Default value:	14567
Byte order:	Network
When to Set:	Can only be set during object initialization.
Version:	This option was deprecated in UME 2.0

43.1.39 ume_registration_id (source)

DEPRECATED, 32-bit value that is used by a persistent Store to identify a source.

If a source desires to identify itself as a previously known source (after a crash or shutdown), it should set the ID to the value it was using before. A value of 0 indicates the source will allow the persistent Store to assign an ID. This setting is deprecated. Its use is not recommended except by legacy systems. Please use the ume_store option instead.

Scope:	source
Type:	**lbm_uint_t**
Units:	identifier
Default value:	0 (allow persistent Store to assign ID)
When to Set:	Can only be set during object initialization.
Version:	This option was deprecated in UME 2.0

43.1.40 ume_retransmit_request_generation_interval (receiver)

DEPRECATED, the maximum interval between when a retransmission request is first sent and when it is given up on and loss is reported.

This option is retained for backwards compatibility. The retransmit_request_generation_interval (receiver) setting should be used instead.

Scope:	receiver
Type:	**unsigned long int**
Units:	milliseconds
Default value:	10000 (10 seconds)
When to Set:	Can only be set during object initialization.

43.1.41 ume_retransmit_request_interval (receiver)

DEPRECATED, the interval between retransmission request messages to the persistent Store or to the source.

This option is retained for backwards compatibility. The retransmit_request_interval (receiver) setting should be used instead.

Scope:	receiver
Type:	**unsigned long int**
Units:	milliseconds
Default value:	500 (0.5 seconds)
When to Set:	Can only be set during object initialization.

43.1.42 ume_retransmit_request_maximum (receiver)

DEPRECATED, the maximum number of messages to request back from the current latest message when late joining a topic or when registering with a persistent Store.

A value of 0 indicates no maximum.

This option is retained for backwards compatibility. The retransmit_request_maximum (receiver) setting should be used instead.

Scope:	receiver
Type:	**unsigned long int**
Units:	messages
Default value:	0
When to Set:	Can only be set during object initialization.

43.1.43 ume_retransmit_request_outstanding_maximum (receiver)

DEPRECATED, the maximum number of messages to request at a single time from the Store or source.

A value of 0 indicates no maximum.

This option is retained for backwards compatibility. The retransmit_request_outstanding_maximum (receiver) setting should be used instead.

Scope:	receiver
Type:	**unsigned long int**
Units:	messages
Default value:	10
When to Set:	Can only be set during object initialization.

43.1.44 ume_secondary_store_address (source)

DEPRECATED, IPv4 address of the persistent Store to be used as the secondary Store.

A value of 0.0.0.0 (or INADDR_ANY) indicates no Store is set as the secondary. This setting is deprecated. Its use is not recommended except by legacy systems. Please use the ume_store (source) option instead.

Scope:	source
Type:	**struct in_addr**
Default value:	0.0.0.0 (INADDR_ANY)
When to Set:	Can only be set during object initialization.
Version:	This option was deprecated in UME 2.0

43.1.45 ume_secondary_store_port (source)

DEPRECATED, TCP port of the secondary persistent Store.

This setting is deprecated. Its use is not recommended except by legacy systems. Please use the ume_store (source) option instead.

See Port Assignments for more information about configuring ports.

Scope:	source
Type:	**lbm_uint16_t**
Default value:	14567
Byte order:	Network
When to Set:	Can only be set during object initialization.
Version:	This option was deprecated in UME 2.0

43.1.46 ume_tertiary_store_address (source)

DEPRECATED, IPv4 address of the persistent Store to be used as the tertiary Store.

A value of 0.0.0.0 (or INADDR_ANY) indicates no Store is set as the tertiary.

This setting is deprecated. Its use is not recommended except by legacy systems. Please use the ume_store (source) option instead.

Scope:	source
Type:	**struct in_addr**
Default value:	0.0.0.0 (INADDR_ANY)
When to Set:	Can only be set during object initialization.
Version:	This option was deprecated in UME 2.0

43.1.47 ume_tertiary_store_port (source)

DEPRECATED, TCP port of the tertiary persistent Store.

This setting is deprecated. Its use is not recommended except by legacy systems. Please use the ume_store (source) option instead.

See Port Assignments for more information about configuring ports.

Scope:	source
Type:	**lbm_uint16_t**
Default value:	14567
Byte order:	Network
When to Set:	Can only be set during object initialization.
Version:	This option was deprecated in UME 2.0

43.1.48 umq_flight_size (context)

DEPRECATED, specifies the number of Multicast Immediate Messages allowed to be in flight (unstabilized at a queue) before a new message send either blocks or triggers a notification (source event).

See **Ultra Load Balancing (ULB)**.

Scope:	context
Type:	**unsigned int**
Units:	messages
Default value:	1000
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1.1/UME 3.1.1/UMQ 1.1.1
Version:	This option was deprecated in UMQ 6.8

43.1.49 umq_flight_size (source)

DEPRECATED, specifies the number of messages allowed to be in flight (unstabilized at a queue) before a new message send either blocks or triggers a notification (source event).

See **Ultra Load Balancing (ULB)**.

Scope:	source
Type:	**unsigned int**

Units:	messages
Default value:	1000
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1.1/UME 3.1.1/UMQ 1.1.1
Version:	This option was deprecated in UM 6.8

43.1.50 umq_flight_size_behavior (context)

DEPRECATED, the behavior that UMQ follows when a MIM send exceeds the context's flight size.

See **Multicast Immediate Messaging** for general information about MIM.

See umq_flight_size (source).

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1.1/UME 3.1.1/UMQ 1.1.1
Version:	This option was deprecated in UMQ 6.8

String value	Integer value	Description
"Block"	LBM_FLIGHT_SIZE_BEHAVIOR_BLOCK	The send call blocks when a MIM send exceeds the context's flight size. If the MIM send is a non-blocking send, the send returns an LBM_EWOULD_BLOCK. **Default for all.**
"Notify"	LBM_FLIGHT_SIZE_BEHAVIOR_NOTIFY	A message send that exceeds the configured flight size does not block but triggers a flight size notification (context event), indicating that the flight size has been surpassed. UMQ also sends a context event notification if the number of in-flight messages falls below the configured flight size.

43.1.51 umq_flight_size_behavior (source)

DEPRECATED, the behavior that UMQ follows when a message send exceeds the source's flight size.

See umq_flight_size (source).

Scope:	source
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1.1/UME 3.1.1/UMQ 1.1.1
Version:	This option was deprecated in UM 6.8

String value	Integer value	Description
"Block"	LBM_FLIGHT_SIZE_BEHAVIOR_BLOCK	The send call blocks when a source sends a message that exceeds its flight size. If the source uses a non-blocking send, the send returns an LBM_EWOULD_BLOCK. **Default for all.**
"Notify"	LBM_FLIGHT_SIZE_BEHAVIOR_NOTIFY	A message send that exceeds the configured flight size does not block but triggers a flight size notification (source event), indicating that the flight size has been surpassed. UMQ also sends a source event notification if the number of in-flight messages falls below the configured flight size.

43.1.52 umq_message_retransmission_interval (context)

DEPRECATED, the interval between retransmissions of data messages when submitting to a Queue.

For general information on queuing, see **Queuing**.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	500 (0.5 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.6/UME 3.0/UMQ 1.0.
Version:	This option was deprecated in UMQ 6.8

43.1.53 umq_message_stability_notification (context)

DEPRECATED, flag indicating the context is interested in receiving notifications of message stability from Queues via the context event mechanism.

Even when turned off, Queues will continue to send message stability notifications to the context for retention purposes. However, no notification will be delivered to the application.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.6/UME 3.0/UMQ 1.0.
Version:	This option was deprecated in UMQ 6.8

Value	Description
1	The context wishes to receive message stability notification. **Default for all.**
0	The context does not wish to receive message stability notifications.

43.1.54 umq_msg_total_lifetime (context)

DEPRECATED, establishes the period of time from when a queue receives a message, or, for ULB, when a source sends a message, until the time the message cannot be assigned or reassigned to a receiver. The queue deletes the message upon expiration of the lifetime.

You can also set UMQ umestored option message-total-lifetime for the source's topic on the queue. However, the message-total-lifetime option is overridden by any value assigned to umq_msg_total_lifetime (source). The default value of 0 (zero) disables this option.

Note: This option is overridden by any message lifetime value set using send call, **lbm_src_send_ex()**.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	0 (zero)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.2 / UME 3.2 / UMQ 2.1
Version:	This option was deprecated in UMQ 6.8

43.1.55 umq_queue_check_interval (context)

DEPRECATED, the interval between activity checks of the individual UMQ queues.

For general information on queuing, see **Queuing**.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	500 (0.5 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.6/UME 3.0/UMQ 1.0.
Version:	This option was deprecated in UMQ 6.8

43.1.56 umq_queue_name (source)

DEPRECATED, the queue to submit messages to when sending.

For general information on queuing, see **Queuing**.

Scope:	source
Type:	**string**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.6/UME 3.0/UMQ 1.0.
Version:	This option was deprecated in UMQ 6.8

43.1.57 umq_queue_participants_only (source)

DEPRECATED, flag indicating the source only desires queue participants to listen to the topic.

For general information on queuing, see **Queuing**.

Scope:	source
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.6/UME 3.0/UMQ 1.0.
Version:	This option was deprecated in UMQ 6.8

Value	Description
1	The source desires that only queue participants listen to the topic.
0	The source desires anyone to listen to the topic without regard to queue participation. **Default for all.**

43.1.58 umq_queue_query_interval (context)

DEPRECATED, the interval between queries sent for resolving Queues.

This option is no longer functional.

Scope:	context
Type:	**lbm_ulong_t**
Units:	milliseconds
Default value:	200 (0.2 seconds)
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.6/UME 3.0/UMQ 1.0.
Version:	This option was deprecated in UMQ 6.8

43.1.59 umq_require_queue_authentication (context)

DEPRECATED, indicates if an application requires a queue to authenticate itself before accepting the queue's responses to Queue Browser commands.

For general information on queuing, see **Queuing**.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in UMQ 5.2.2.
Version:	This option was deprecated in UMQ 6.8

Value	Description
1	An application requires the queue to successfully authenticate before using browsing command responses from the queue. **Default for all.**

Value	Description
0	An application does not require queue authentication.

43.1.60 umq_retention_intergroup_stability_behavior (context)

DEPRECATED, the behavior that the context will follow when determining the stability of a message from an inter-group perspective.

This has a direct impact on the release policy for the context in that a message must be stable before it may be released. To be stable, a message must first be stable within the group and then stable between groups.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.6/UME 3.0/UMQ 1.0.
Version:	This option was deprecated in UMQ 6.8

String value	Integer value	Description
"any", "any-group"	LBM_SRC_TOPIC_ATTR_UMQ_STA↩ BLE_BEHAVIOR_ANY	Message is considered stable once it is stable in any group. **Default for all.**
"majority"	LBM_SRC_TOPIC_ATTR_UMQ_STA↩ BLE_BEHAVIOR_MAJORITY	Message is considered stable once it is stable in a majority of groups.
"all", "all-groups"	LBM_SRC_TOPIC_ATTR_UMQ_STA↩ BLE_BEHAVIOR_ALL	Message is considered stable once it is stable in all groups.
"all-active"	LBM_SRC_TOPIC_ATTR_UMQ_STA↩ BLE_BEHAVIOR_ALL_ACTIVE	Message is considered stable once it is stable in all active groups. A group is considered active if it has at least a quorum of active or registered queues. Inter-group stability requires at least one stable group.

43.1.61 umq_retention_intergroup_stability_behavior (source)

DEPRECATED, the behavior that the source will follow when determining the stability of a message from an inter-group perspective.

This has a direct impact on the release policy for the context in that a message must be stable before it may be released. To be stable, a message must first be stable within the group and then stable between groups.

Scope:	source
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.6/UME 3.0/UMQ 1.0.
Version:	This option was deprecated in UMQ 6.8

String value	Integer value	Description
"any", "any-group"	LBM_SRC_TOPIC_ATTR_UMQ_STABLE_BEHAVIOR_ANY	Message will be considered stable once any group has reached intra-group stability for the message. **Default for all.**
"majority"	LBM_SRC_TOPIC_ATTR_UMQ_STABLE_BEHAVIOR_MAJORITY	Message will be considered stable once a majority of groups have reached intra-group stability for the message.
"all", "all-groups"	LBM_SRC_TOPIC_ATTR_UMQ_STABLE_BEHAVIOR_ALL	Message will be considered stable once all groups have reached intra-group stability for the message.
"all-active"	LBM_SRC_TOPIC_ATTR_UMQ_STABLE_BEHAVIOR_ALL_ACTIVE	Message will be considered stable once all active groups have reached intra-group stability for the message.

43.1.62 umq_retention_intragroup_stability_behavior (context)

DEPRECATED, the behavior that the context will follow when determining the stability of a message from an intra-group perspective.

This has a direct impact on the release policy for the context in that a message must be stable before it may be released. To be stable, a message must first be stable within the group and then stable between groups.

Scope:	context
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.6/UME 3.0/UMQ 1.0.
Version:	This option was deprecated in UMQ 6.8

String value	Integer value	Description
"quorum"	LBM_SRC_TOPIC_ATTR_UMQ_STABLE_BEHAVIOR_QUORUM	Message is considered stable within the group once a quorum (or majority) of the queues have acknowledged the message as stable. **Default for all.**
"all", "all-stores"	LBM_SRC_TOPIC_ATTR_UMQ_STABLE_BEHAVIOR_ALL	Message is considered stable with the group once all queues have acknowledged the message as stable.
"all-active"	LBM_SRC_TOPIC_ATTR_UMQ_STABLE_BEHAVIOR_ALL_ACTIVE	Message is considered stable with the group once all active queues have acknowledged the message as stable.

43.1.63 umq_retention_intragroup_stability_behavior (source)

DEPRECATED, the behavior that the source will follow when determining the stability of a message from an intra-group perspective.

This has a direct impact on the release policy for the context in that a message must be stable before it may be released. To be stable, a message must first be stable within the group and then stable between groups.

Scope:	source
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 3.6/UME 3.0/UMQ 1.0.
Version:	This option was deprecated in UMQ 6.8

String value	Integer value	Description
"quorum"	LBM_SRC_TOPIC_ATTR_UMQ_STABLE_BEHAVIOR_QUORUM	Message will be considered stable within the group once a quorum (or majority) of the queues have acknowledged the message as stable. **Default for all.**
"all", "all-stores"	LBM_SRC_TOPIC_ATTR_UMQ_STABLE_BEHAVIOR_ALL	Message will be considered stable with the group once all queues have acknowledged the message as stable.
"all-active"	LBM_SRC_TOPIC_ATTR_UMQ_STABLE_BEHAVIOR_ALL_ACTIVE	Message will be considered stable with the group once all active queues have acknowledged the message as stable.

43.1.64 use_transport_thread (receiver)

DEPRECATED, this option no longer functions.

It used to determine whether UM uses a thread from the receiver thread pool to process message data or if it uses the context thread, which is the default. The MTT feature is replaced in 6.11 and beyond by **Transport Services Provider (XSP)**.

Scope:	receiver
Type:	**int**
When to Set:	Can only be set during object initialization.
Version:	This option was implemented in LBM 4.1/UME 3.1.
Version:	This option was deprecated in UM 6.9
Version:	This option was removed from UM in UM 6.11

String value	Integer value	Description
"1"	1	UM uses a thread from the receiver thread pool.
"0"	0	UM uses the context thread to process message data. **Default for all.**