WebSphere Business Integration

IBM

Using Multi-Language Message Service

Version 1.0

SC34-6363-00

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices," on page 177.

First edition (May 2004)

This edition applies to IBM Multi-Language Message Service, Version 1.0 and to any subsequent releases and modifications until otherwise indicated in new editions.

Contents

Figures

Tables

About this book

This book is about IBM® Multi-Language Message Service (XMS), Version 1.0. The book has the following parts:

- Part 1, "Getting started with XMS," on page 1, which describes what XMS is, and how to install and use XMS
- Part 2, "Programming with XMS," on page 19, which describes how to write XMS applications
- Part 3, "XMS API reference," on page 39, which documents the XMS classes and their methods, and the properties of XMS objects

For the latest information about XMS, see the product readme.txt file, which is in the zipped file supplied with XMS.

Who this book is for

This book is primarily for application programmers who write XMS applications. Some of the information might also be useful to system administrators who manage systems on which XMS applications run, or who manage WebSphere® Business Integration Event Broker or WebSphere Business Integration Message Broker brokers to which XMS applications connect.

What you need to know to understand this book

To understand this book, you need the following skills, knowledge, and experience:

- C or C++ application programming skills. If you are not a C++ programmer, you need some knowledge of object oriented concepts and terminology.
- A working knowledge of the operating system that you are using.
- Experience in using TCP/IP as a communications protocol.
- Some knowledge of the concepts and terminology associated with WebSphere Business Integration Event Broker or WebSphere Business Integration Message Broker.
- Some knowledge of the *Java™ Message Service Specification, Version 1.1* and the WebSphere MQ implementation of JMS, WebSphere MQ classes for Java Message Service, might be beneficial, but is not absolutely necessary. You do not need to be a Java or JMS application programmer.

Terms used in this book

The term *XMS* is used as an abbreviation for Multi-Language Message Service.

The term *JMS* means Java Message Service.

The term *WebSphere MQ JMS* means WebSphere MQ classes for Java Message Service.

The term *Linux* means SUSE LINUX Enterprise Server.

The term *Windows®* means Windows XP.

How to use this book

Certain sections in this book refer you to *WebSphere MQ Using Java* for more information. You can download the latest edition of *WebSphere MQ Using Java* from http://www.ibm.com/software/integration/mqfamily/library/manualsa/

Part 1. Getting started with XMS

Chapter 1. Introducing XMS

This chapter introduces Multi-Language Message Service. The chapter contains the following sections:

- "What is Multi-Language Message Service?"
- "Styles of messaging"
- "Messaging transports" on page 4
- "The XMS object model" on page 5
- "The first release of XMS" on page 7

What is Multi-Language Message Service?

Multi-Language Message Service (XMS) is an application programming interface (API) that is based on the Java Message Service (JMS) API. With the first release of XMS, you can write XMS applications that use the publish/subscribe style of messaging. An XMS application acts as a client application to a broker of WebSphere Business Integration Event Broker or WebSphere Business Integration Message Broker, and uses WebSphere MQ Real-Time Transport or WebSphere MQ Multicast Transport to communicate with the broker. You can write XMS applications in either the C or C++ programming language.

Styles of messaging

Point-to-point and *publish/subscribe* are two styles of messaging. Styles of messaging are also called *messaging domains*.

Point-to-point messaging

A common form of point-to-point messaging uses queuing. In the simplest case, an application sends a message to another application by identifying, implicitly or explicitly, a destination queue. The underlying messaging and queuing system receives the message from the sending application and routes the message to its destination queue. The receiving application can then retrieve the message from the queue.

If the underlying messaging and queuing system contains a message broker, the broker might replicate a message and route copies of the message to different queues so that more than one application can receive the message. The broker might also transform a message and add data to it.

A key characteristic of point-to-point messaging is that an application identifies a destination queue when it sends a message. The configuration of the underlying messaging and queuing system then determines precisely which queue the message is put on so that it can be retrieved by the receiving application.

Publish/subscribe messaging

In publish/subscribe messaging, there are two types of application: publisher and subscriber.

A *publisher* supplies information in the form of messages. When a publisher publishes a message, it specifies a topic, which identifies the subject of the information inside the message.

A *subscriber* is a consumer of the information that is published. A subscriber specifies the topics it is interested in by sending subscription requests to a publish/subscribe broker in the form of messages. The broker receives published messages from publishers and subscription requests from subscribers, and routes published messages to subscribers. A subscriber receives messages on all topics, and only those topics, to which it has subscribed.

A key characteristic of publish/subscribe messaging is that a publisher identifies a topic when it publishes a message, and a subscriber receives the message only if has subscribed to the topic. If a message is published on a topic for which there are no subscribers, no application receives the message.

An application can be both a publisher and a subscriber.

Note: The first release of XMS supports only the publish/subscribe messaging domain.

Messaging transports

A *messaging transport* is a way in which an application can exchange messages with a broker.

XMS supports two brokers for the publish/subscribe domain:

- WebSphere Business Integration Event Broker
- WebSphere Business Integration Message Broker

Each broker provides the following three transports:

WebSphere MQ Enterprise Transport
All communication between a publisher and a broker, or between a subscriber and a broker, uses WebSphere MQ.

If a publisher uses this transport, the publisher publishes messages by putting them on a queue that is monitored by the broker.

If a subscriber uses this transport, the subscriber sends subscription requests to the broker by putting messages on a queue that is monitored by the broker. In turn, the broker puts messages, published on topics to which the subscriber has subscribed, on a queue that is monitored by the subscriber.

WebSphere MQ Real-Time Transport
All communication between a publisher and a broker, or between a subscriber and a broker, uses a TCP connection.

If a publisher uses this transport, the publisher publishes messages by sending them directly to the broker over a TCP connection.

If a subscriber uses this transport, the subscriber sends subscription requests directly to the broker over a TCP connection. In turn, the broker sends messages, published on topics to which the subscriber has subscribed, directly to the subscriber over a TCP connection.

WebSphere MQ Multicast Transport
A subscriber can use this transport to receive published messages from a broker. The transport cannot be used for any other purpose.

The transport works by associating a multicast IP address with a topic. When a message is published, the broker transmits one copy of the message to the multicast IP address associated with the topic. IP then routes the message to all subscribers that have subscribed to the topic.

WebSphere MQ Multicast Transport is a high performance transport. Using this transport, a broker transmits only one copy of a published message over the network. Using WebSphere MQ Real-Time Transport, by comparison, a broker must transmit a copy of a published message to each subscriber.

Note: The first release of XMS supports only WebSphere MQ Real-Time Transport and WebSphere MQ Multicast Transport.

The XMS object model

The XMS API is an object oriented interface. The XMS object model is based on the JMS 1.1 object model.

The following list summarizes the main XMS classes, or types of object:

ConnectionFactory
A ConnectionFactory object encapsulates a set of configuration parameters for a connection. An application uses a connection factory to create a connection.

Connection
A Connection object encapsulates an application's active connection to a broker. An application uses a connection to create sessions.

Destination
A destination is where an application sends messages, or it is a source from which an application receives messages, or both. In the publish/subscribe domain, a Destination object encapsulates a topic.

Session
A session is a single threaded context for sending and receiving messages. An application uses a session to create messages, message producers, and message consumers.

Message
A Message object encapsulates a message that an application sends or receives.

MessageProducer
An application uses a message producer to send messages to a destination.

MessageConsumer
An application uses a message consumer to receive messages sent to a destination.

Figure 1 on page 6 shows these objects and their relationships.

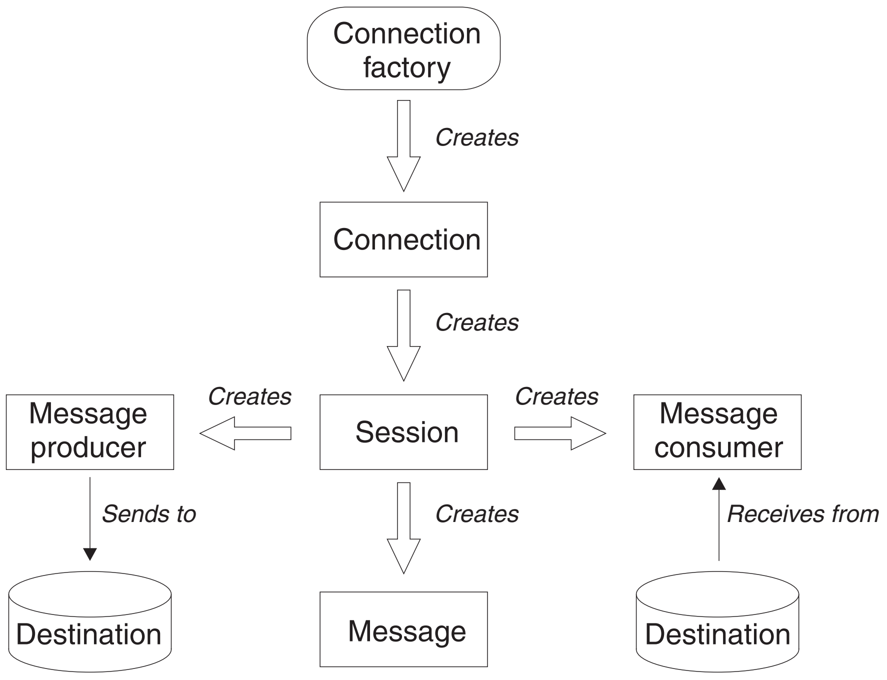

Figure 1. XMS objects and their relationships

XMS applications written in C++ use these classes and their methods. XMS applications written in C use the same object model even though C is not an object oriented language. When a C application creates an object, XMS stores the object internally and returns a handle for the object to the application. The application can then use the handle subsequently to reference the object. For example, if a C application creates a connection factory, XMS returns a handle for the connection factory to the application. In general, for each C++ method in the C++ interface, there is an equivalent C function in the C interface.

The XMS object model is based upon the domain independent interfaces that are described in the *Java Message Service Specification, Version 1.1*. Domain specific classes, such as Topic, TopicPublisher, and TopicSubscriber, are not provided.

Attributes and properties of objects

An XMS object can have attributes and properties, which are characteristics of the object. Attributes and properties, however, are implemented in different ways.

Attributes
An attribute of an object is always present and occupies storage, even if the attribute does not have a value. In this respect, an attribute is similar in concept to a field in a fixed length data structure. Another distinguishing feature is that each attribute has its own methods for setting and getting its value.

Properties
A property of an object is present and occupies storage only after its value is set. However, a property cannot be deleted, and the storage recovered, after its value has been set, although you can change its value. A property does not have its own methods for setting and getting its value. Instead, XMS provides a set of generic methods for setting and getting the values of properties.

The first release of XMS

This section specifies the supported operating environments for the first release of XMS. It also summarizes the function supported in the first release, and the function that is not supported. If you need an explanation of any of the function mentioned in this section, see the following sources of information:

- The WebSphere Business Integration Event Broker or WebSphere Business Integration Message Broker Information Center.
- *WebSphere MQ Using Java*
- *Java Message Service Specification, Version 1.1*

Supported operating environments

An XMS client is supplied for each of the operating systems listed in Table 1. The table also lists the supported C and C++ compiler for each client platform.

Table 1. XMS client platforms and compilers

Supported operating system	Supported C and C++ compiler
Microsoft® Windows XP Professional with Service Pack 1	Microsoft Visual C++, Version 6.0 with Service Pack 5
SUSE LINUX Enterprise Server 8 (Intel™ only)	gcc 3.2 (supplied with the operating system)

Function supported

The following function is supported in the first release of XMS:

- An XMS publisher can publish messages using WebSphere MQ Real-Time Transport. To receive the messages and forward them to subscribers, the broker must be configured with a message flow that contains a Real-timeOptimizedFlow message processing node.
- An XMS subscriber can send subscription requests using WebSphere MQ Real-Time Transport. The subscriber can then receive messages, published on the topics to which it has subscribed, using WebSphere MQ Real-Time Transport or WebSphere MQ Multicast Transport.
- The XMS message model is the same as the WebSphere MQ JMS message model. In particular, XMS implements the same message header fields and message properties that WebSphere MQ JMS implements:
  - JMS header fields. These are fields whose names commence with the prefix JMS.
  - JMS defined properties. These are properties whose names commence with the prefix JMSX.
  - IBM defined properties. These are the properties whose names commence with the prefix JMS_IBM_.

  As a result, XMS subscribers can receive messages published by WebSphere MQ JMS publishers, and WebSphere MQ JMS subscribers can receive messages published by XMS publishers. For each message that is published, some of the header fields and properties are set by the publisher and others are set by the XMS or WebSphere MQ JMS client when the message is sent. Where appropriate, these header fields and properties are propagated with a message through the broker and are made available to each subscriber that receives the message. This level of interoperability is also available if a WebSphere MQ JMS publisher or subscriber uses WebSphere MQ Enterprise Transport.

Function not supported

The following function is not supported in the first release of XMS:

- An XMS application cannot connect to a WebSphere MQ queue manager and perform messaging and queuing operations.
- Administered objects are not supported. ConnectionFactory and Destination objects are administered objects in JMS but, in XMS, only applications can create these objects and set their attributes and properties.
- Object messages, stream messages, and text messages are not supported. Only bytes messages, map messages, and messages without bodies are supported.
- Durable topic subscribers are not supported. Only nondurable message consumers are supported.
- Persistent messages are not supported. Only nonpersistent messages are supported.
- Transacted sessions are not supported.
- When an application connects to a broker, the application can supply a user identifier and a password, which the broker can use to authenticate the application. In the WebSphere Business Integration Event Broker and WebSphere Business Integration Message Broker Information Centers, this is referred to as "simple telnet-like password authentication". This is the only form of authentication supported by XMS, and it means that you cannot use the message protection facilities provided by the broker. Authentication using Secure Sockets Layer (SSL) is not supported.
- A TopicRequestor class is not provided, which means that the request/reply style of messaging is not directly supported.

Chapter 2. Installing XMS

This chapter describes how to install the Multi-Language Message Service (XMS) client on Windows. For instructions on how to install the XMS client on Linux, and for the latest information about installing the product, see the product readme.txt file, which is in the zipped file supplied with XMS.

On all platforms, XMS is installed using an InstallShield MultiPlatform 5 installer. The procedures in this chapter describe how to use the installer in the form of a Wizard with a graphical user interface. However, if you invoke the installer from a command prompt using the runtime command line option **-silent**, you can perform an unattended, or silent, installation, which requires no interaction with the Wizard. Other runtime command line options allow you to have more control over the installation. For general information about MultiPlatform 5, see the InstallShield Web site at http://www.installshield.com/. For more specific information about the runtime command line options, see the *InstallShield MultiPlatform 5 User's Guide*, which you can download from the same Web site.

This chapter contains the following section:
- "Installing on Windows" on page 10

Installing on Windows

To install the XMS client on Windows, follow this procedure. You must be logged on to Windows as an administrator. The installed code requires 40 MB of disk space.

1. Create a temporary directory and extract the contents of zipped file supplied with XMS into the directory.

   A subdirectory of the temporary directory is created. The subdirectory is called gxixms_install and contains the files needed for the installation.
2. Run the file setup.exe that is in the gxixms_install directory.

   A command prompt window opens to run setup.exe. The messages in the window inform you that setup.exe is searching for, and preparing, a Java Virtual Machine (JVM). If you do not have a JVM on your system, setup.exe uses its own JVM.

   Eventually, the Installer window opens and displays the following message:

   ```
   Welcome to the InstallShield Wizard for IBM Multi-Language Message Service
   ```
3. Click **Next**.

   The Installer windows asks you where you want to install XMS. If you do not want to install XMS in the directory suggested, you can choose another directory.
4. Click **Next**.

   The Installer window asks you which features you want to install. Ensure that **IBM Multi-Language Message Service Toolkit Feature** is selected.
5. Click **Next**.

   The Installer window displays details of what is about to be installed.
6. Click **Next** to start the installation.

   The Installer windows displays a bar showing the progress of the installation. Wait for the progress bar to complete. When the installation completes successfully, the window displays the following message:

   ```
   The InstallShield Wizard has successfully installed IBM Multi-Language Message
   Service. Choose Finish to exit the wizard.
   ```
7. Click **Finish** to close the Installer window.

You have now successfully installed the XMS client, which is ready to use.

What is installed

XMS is installed in the C:\Program Files\IBM\gxixms directory unless you choose to install it in a different directory. Table 2 lists the installation directories and their contents.

Table 2. The XMS installation directories and their contents

Installation directory	**Content**
\gxixms	The readme.txt file
\gxixms\bin	The *.dll and *.pdb files required to run XMS applications
\gxixms\doc	This book as a PDF file
\gxixms\tools\c\include	The XMS header files for C
\gxixms\tools\cpp\include	The XMS header files for C++
\gxixms\tools\lib	The XMS link libraries for C and C++

Table 2. The XMS installation directories and their contents (continued)

Installation directory	Content
\gxixms\tools\samples	The readme.txt file for the samples
\gxixms\tools\samples\bin	The compiled sample applications and the command files to run them
\gxixms\tools\samples\c\RTTconsumer	The source and makefile for the C message consumer sample application that uses WebSphere MQ Real-Time Transport or WebSphere MQ Multicast Transport
\gxixms\tools\samples\c\RTTconsumersync	The source and makefile for the C message consumer application that uses synchronous message delivery and WebSphere MQ Real-Time Transport
\gxixms\tools\samples\c\RTTproducer	The source and makefile for the C message producer sample application that uses WebSphere MQ Real-Time Transport
\gxixms\tools\samples\cpp\RTTcons	The source and makefile for the C++ message consumer sample application that uses WebSphere MQ Real-Time Transport
\gxixms\tools\samples\cpp\RTTprod	The source and makefile for the C++ message producer sample application that uses WebSphere MQ Real-Time Transport
\gxixms\tools\samples\java\RTTpublisher	The source for the WebSphere MQ JMS message producer sample application that uses WebSphere MQ Real-Time Transport
\gxixms\tools\samples\java\RTTsubscriber	The source for the WebSphere MQ JMS message consumer sample application that uses WebSphere MQ Real-Time Transport
\gxixms\tools\samples\java\RTTsubscribersync	The source for the WebSphere MQ JMS message consumer application that uses synchronous message delivery and WebSphere MQ Real-Time Transport

Uninstalling XMS

To remove the XMS client from your system, follow this procedure. You must be logged on to Windows as an administrator.

1. From the Windows task bar, click **Start —> Settings —> Control Panel**.

   The Control Panel window opens.
2. Double-click **Add/Remove Programs**.

   The Add/Remove Programs window opens.
3. Click **IBM Multi-Language Message Service** to select it.
4. Click **Change/Remove**.

   The Uninstaller window opens and displays the following message:
   ```
   Welcome to the InstallShield Wizard for IBM Multi-Language Message Service
   ```
5. Click **Next**.

The Uninstaller window provides information about what is about to be uninstalled.

6. Click **Next** to start the removal of the XMS.

   The Uninstaller window confirms that XMS is being uninstalled. When XMS has been removed successfully, the window displays the following message:

   ```
   The InstallShield Wizard has successfully uninstalled IBM Multi-Language Message
   Service. Choose Finish to exit the wizard.
   ```
7. Click **Finish** to close the Uninstaller window.

You have now successfully removed the XMS client from your system.

Chapter 3. Using XMS

This chapter provides information about how to use XMS after you have installed it. It describes the sample applications provided with XMS and how to use them to verify your installation. It explains how to build the sample applications and your own applications. The chapter ends by describing how to produce a trace to help diagnose a problem.

The information in this chapter applies only to Windows. For the equivalent information about how to use XMS on Linux, see the samples readme.txt file in the samples installation directory and the product readme.txt file in the gxixms installation directory.

The chapter contains the following sections:
- "The sample applications"
- "Running the C sample applications" on page 15
- "Building the sample applications" on page 16
- "Building your own applications" on page 17
- "Problem determination" on page 17

The sample applications

This section describes the sample applications supplied with XMS. Both the source and an executable version are provided for each application. To find out where the applications are installed, see Table 2 on page 10.

Three sets of sample applications are supplied with XMS:
- "C sample applications"
- "C++ sample applications" on page 14
- "WebSphere MQ JMS sample applications" on page 14

C sample applications

The following C sample applications are supplied with XMS:

RTTproducer
: This application publishes a bytes message every 2 seconds. Each bytes message has a string property and a body that contains a string encoded in UTF-8 format.

RTTconsumer
: This application receives bytes messages asynchronously using WebSphere MQ Real-Time Transport. For each message received, the application reads a string, encoded in UTF-8 format, from the body of the message and gets the value of a string property of the message. The application then displays the two strings on the screen.

  If you start the application with one of the following optional arguments, the application uses WebSphere MQ Multicast Transport instead:

```
multicast:XMSC_MULTICAST_ENABLED
multicast:XMSC_MULTICAST_NOT_RELIABLE
multicast:XMSC_MULTICAST_RELIABLE
```

RTTconsumersync
This application receives bytes messages synchronously using WebSphere MQ Real-Time Transport. The application calls the Receive method with a specified wait interval. For each message received, the application reads a string, encoded in UTF-8 format, from the body of the message and gets the value of a string property of the message. The application then displays the two strings on the screen.

C++ sample applications

The following C++ sample applications are supplied with XMS:

RTTpub
This application publishes a bytes message every 2 seconds. Each bytes message has a string property and a body that contains a string encoded in UTF-8 format.

The application has two classes:

RTTpub
This class contains the main method.

SampleExpListener
This is an exception listener class defined by the application and contains the onException method.

RTTcons
This application receives bytes messages asynchronously using WebSphere MQ Real-Time Transport. For each message received, the application reads a string, encoded in UTF-8 format, from the body of the message and gets the value of a string property of the message. The application then displays the two strings on the screen.

If you start the application with one of the following optional arguments, the application uses WebSphere MQ Multicast Transport instead:

```
multicast:XMSC_MULTICAST_ENABLED
multicast:XMSC_MULTICAST_NOT_RELIABLE
multicast:XMSC_MULTICAST_RELIABLE
```

The application has three classes:

RTTcons
This class contains the main method.

SampleExpListener
This is an exception listener class defined by the application and contains the onException method.

SampleMsgListener
This is a message listener class defined by the application and contains the onMessage method.

WebSphere MQ JMS sample applications

The following WebSphere MQ JMS sample applications are supplied with XMS. Using these applications, you can demonstrate XMS applications exchanging messages with WebSphere MQ JMS applications.

RTTpublisher
This application publishes a bytes message every 2 seconds. Each bytes message has a string property and a body that contains a string encoded in UTF-8 format.

RTTsubscriber
This application receives bytes messages asynchronously using WebSphere MQ Real-Time Transport. For each message received, the application reads a string, encoded in UTF-8 format, from the body of the message and gets the value of a string property of the message. The application then displays the two strings on the screen.

RTTsubscribersync
This application receives bytes messages synchronously using WebSphere MQ Real-Time Transport. The application calls the receive() method with a specified timeout interval. For each message received, the application reads a string, encoded in UTF-8 format, from the body of the message and gets the value of a string property of the message. The application then displays the two strings on the screen.

Running the C sample applications

This section describes how to run the executable versions of the RTTproducer and RTTconsumer applications on Windows. You can use these applications to verify that you have installed XMS correctly.

The RTTproducer and RTTconsumer applications connect to a WebSphere Business Integration Event Broker or WebSphere Business Integration Message Broker broker using WebSphere MQ Real-Time Transport. Before you can run the applications, you must create a message flow that contains a Real-timeOptimizedFlow message processing node and deploy the message flow to a broker that is running on your system. If you need more information about how to do this, see the WebSphere Business Integration Event Broker or WebSphere Business Integration Message Broker Information Center.

After you have prepared the broker, follow this procedure to run the applications. The **cd** (change directory) command in the procedure assumes that you have installed XMS in the C:\Program Files\IBM\gxixms directory. If you have installed XMS in a different directory, you must make the appropriate modification to the command.

1. Open a command prompt window.
2. At the command prompt, enter the following command:
   ```
   cd \Program Files\IBM\gxixms\tools\samples\bin
   ```
3. At the command prompt, enter the following command:
   ```
   xmsdemo port_number
   ```
   where *port_number* is the port number on which the Real-timeOptimizedFlow message processing node listens for publish and subscribe requests.

The command file, xmsdemo.cmd, starts the RTTconsumer application and then the RTTproducer application, passing the following arguments to each application:

```
host:localhost port:port_number topic:xms/test
```

RTTconsumer subscribes to the topic xms/test, and then receives each message published on that topic. RTTproducer publishes a message on the topic xms/test every 2 seconds.

Building the sample applications

This section describes how to build the sample applications on Windows. The **cd** (change directory) command in each procedure in this section assumes that you have installed XMS in the C:\Program Files\IBM\gxixms directory. If you have installed XMS in a different directory, you must make the appropriate modification to the command.

Building the C sample applications

Before you can build the C sample applications, you must ensure that you have set up the Microsoft Visual C++ build environment. You can do this by running vcvars32.

To build a C sample application, follow this procedure:

1. Open a command prompt window.
2. At the command prompt, enter the following command:
   ```
   cd \Program Files\IBM\gxixms\tools\samples\c\application_name
   ```
   where *application_name* is one of the following names:
   - RTTproducer
   - RTTconsumer
   - RTTconsumersync
3. At the command prompt, enter the following command:
   ```
   nmake
   ```
   The command builds an executable version of the application, *application_name*.exe, in the current directory.

Building the C++ sample applications

Before you can build the C++ sample applications, you must ensure that you have configured the Microsoft Visual C++ build environment. You can do this by running vcvars32.

To build a C++ sample application, follow this procedure:

1. Open a command prompt window.
2. At the command prompt, enter the following command:
   ```
   cd \Program Files\IBM\gxixms\tools\samples\cpp\application_name
   ```
   where *application_name* is one of the following names:
   - RTTprod
   - RTTcons
3. At the command prompt, enter the following command:
   ```
   nmake
   ```
   The command builds an executable version of the application, *application_name*.exe, in the current directory.

Building your own applications

This section provides the information you need to build your own XMS applications on Windows. To find out where the files and libraries mentioned in this section are installed, see Table 2 on page 10.

Building your own C applications

Your C applications must include the file xms.h, which defines the function prototypes for the XMS methods. The file also includes the file xmsc.h, which defines the data types, enumerated data types, and constants used by the XMS API.

The makefile called Makefile, which is provided for each of the C sample applications, shows you how to build your applications. Note, in particular, that you must link your applications using the library gxi.lib.

Before running your applications, make sure that the gxixms\bin directory is in the path specified by the PATH environment variable.

Building your own C++ applications

Your C++ applications must include the file xms.hpp, which defines the XMS classes and their methods. The file also includes the file xmsc.h, which defines the data types, enumerated data types, and constants used by the XMS API.

The makefile called Makefile, which is provided for each of the C++ sample applications, shows you how to build your applications. Note, in particular, that you must link your applications using the library gxi01vn.lib.

Before running your applications, make sure that the gxixms\bin directory is in the path specified by the PATH environment variable.

Problem determination

If you experience a problem with XMS, your IBM Support Center might ask you to produce a trace to help diagnose the problem.

To enable tracing for an application, set the environment variable XMS_TRACE_ON to 1 and then start the application.

To disable tracing for an application, clear the environment variable XMS_TRACE_ON. Tracing ends only after the application ends.

XMS creates a trace file in the current working directory unless you specify an alternative location by setting the environment variable XMS_TRACE_FILE_PATH to the fully qualified path name of the directory where you want XMS to create the trace file. You must set the environment variable before you start the application that you want to trace, and you must make sure that the user identifier under which the application runs has the authority to write to the directory where XMS creates the trace file. The trace file has the extension .trc.

Part 2. Programming with XMS

Chapter 4. Writing XMS applications

This chapter provides information that you might find useful when writing Multi-Language Message Service (XMS) applications. If you are writing applications in C, see also Chapter 5, “Writing XMS applications in C,” on page 27.

The chapter contains the following sections:
- “The threading model”
- “Connections”
- “Sessions” on page 22
- “Uniform resource identifiers (URIs)” on page 23
- “Deleting objects” on page 24
- “Iterators” on page 24

The threading model

The following general rules govern how a multithreaded application can use XMS objects:
- Only objects of the following types can be used concurrently on different threads:
  - ConnectionFactory
  - Connection
  - ConnectionMetaData
  - Destination
- A Session object can be used only on the thread on which it is created.
- All other objects can be used only on the same thread as the session in which they are created.

Exceptions to these rules are indicated by entries labelled “Thread context” in the interface definitions of the methods in Chapter 7, “XMS classes,” on page 43.

Connections

An application uses a ConnectionFactory object to create a Connection object, and uses a Connection object to create a Session object. Creating a connection is relatively expensive in terms of system resources because it involves establishing a communications connection and authenticating the application.

An instance of an XMS client can support multiple connections. A multithreaded application can use a single Connection object concurrently on multiple threads.

A connection serves several purposes:
- A connection encapsulates a communications connection. If the connection uses WebSphere MQ Real-Time Transport, the connection encapsulates a TCP connection between the application and WebSphere Business Integration Event Broker or WebSphere Business Integration Message Broker broker.
- When an application creates a connection, the application can be authenticated.
- A connection can specify a unique client identifier.

- An application can register an ExceptionListener object with a connection.

An XMS application typically creates a connection, one or more sessions, and a number of message producers and consumers.

Starting and stopping a connection

A connection can operate in either a started or a stopped mode. When an application creates a connection, the connection is in stopped mode. In this mode, the application can receive no messages, either synchronously or asynchronously, until the application starts the connection using the Start Connection method. The application can use the time while the connection is stopped to initialize sessions. After a connection has started, an application can stop and restart the connection using the Stop Connection and Start Connection methods.

Note that an application can still send messages when a connection is stopped.

Closing a connection

Closing a connection has the following impacts:

- XMS closes all the sessions associated with the connection, and deletes certain objects associated those sessions. For more information, see “Deleting objects” on page 24.
- XMS ends the communications connection with the broker.
- XMS releases the memory and other internal resources used by the connection.

Handling exceptions

If an application registers an exception listener with a connection, XMS notifies the application asynchronously whenever a serious problem occurs with the connection. The ExceptionListener object contains a pointer to an onException() method, which XMS calls if a failure occurs. If an application uses a connection only to receive messages asynchronously, then the only way the application can learn about a problem with the connection is by using an exception listener.

Sessions

A session is a single threaded context for sending and receiving messages. Using a Session object, an application can create MessageProducer, MessageConsumer, and TemporaryTopic objects, and dynamically create Destination objects that represent topics.

An application can create multiple sessions that produce and consume messages independently. If two sessions subscribe to the same topic, they each receive a copy of any message published on the topic.

Unlike a Connection object, a Session object cannot be shared across threads. Only the Close Session method of a Session object can be called from a thread other than the one on which the Session object was created. The Close Session method ends a session and releases any system resources allocated to the session.

Asynchronous message delivery

To receive messages asynchronously, an application must register a message listener with one or more message consumers. In the C API, a message listener is a callback function and a pointer to application defined context data. In the C++

API, a message listener is an object with an onMessage() method. When messages arrive for a message consumer, XMS calls the message listener function or onMessage() method to deliver them.

A session uses its thread to handle all asynchronous message delivery. This means that only one message listener can run at a time. If more than one message consumer in a session has a registered message listener, and one message listener is currently delivering a message, other messages waiting to be delivered to the session must wait.

If an application needs concurrent delivery of messages, it must create more than one session, where each session runs in its own thread. In this way, a message listener can run concurrently in each session.

Synchronous message delivery

Messages can be delivered synchronously to an application if the application uses the Receive methods of a MessageConsumer object. Using the Receive methods, an application can wait a specified period of time for a message, or it can wait indefinitely.

Uniform resource identifiers (URIs)

A *uniform resource identifier (URI)* is a string that identifies a destination and, optionally, specifies one or more properties of the destination.

In its simplest form, a URI for a topic has the following format:

```
topic://topic_name
```

where *topic_name* is the name of the topic. If you want to specify one or more properties of a topic, a URI has the following extended format:

```
topic://topic_name?prop_name1=prop_value1&prop_name2=prop_value2& ...
```

where *prop_name* is the name of a property and *prop_value* is the value of a property.

In a URI, you cannot use named constants for the names and values of properties. Table 3 shows, for each property of a topic, the name and valid values that you can use in a URI. For more information about the properties of a topic, see “Properties of Destination” on page 173.

Table 3. The names and valid values of properties that you can use in a topic URI

Name of property	The name that you can use in a URI	The valid values that you can use in a URI (default value in bold)
XMSC_MULTICAST	multicast	**-1 (= XMSC_MULTICAST_ASCF)** 0 (= XMSC_MULTICAST_DISABLED) 3 (= XMSC_MULTICAST_NOT_RELIABLE) 5 (= XMSC_MULTICAST_RELIABLE) 7 (= XMSC_MULTICAST_ENABLED)
XMSC_PRIORITY	priority	**-2 (= XMSC_PROPERTY_AS_APP)** An integer in the range 0 to 9

An application can use a topic URI as a parameter when it calls the Create Destination method of the Destination class. Here is an example in a fragment of C code:

```
rc = xmsDestCreate("topic://Sport/Football/Results?multicast=0&priority=9");
```

A C++ application can also use a topic URI as a parameter when it calls the Create Topic method of the Session class. Here is an example in a fragment of C++ code:

```
topic = session.createTopic("topic://Sport/Football/Results?multicast=7");
```

Deleting objects

When an application creates an XMS object, XMS allocates memory and other internal resources to the object. XMS retains these internal resources until the application explicitly deletes the object by calling the object's close or delete method, at which point XMS releases the internal resources. In a C++ application, an object is also deleted when it goes out of scope. If an application tries to delete an object that is already deleted, the call is ignored.

When an application deletes a Connection or Session object, XMS deletes certain associated objects automatically and releases their internal resources. These are objects that were created by the Connection or Session object and depend for their existence upon the connection or session. These objects are shown in Table 4. Note that, if an application closes a connection with dependent sessions, all objects dependent on those sessions are also deleted. Only a Connection or Session object can have dependent objects.

Table 4. Objects that are deleted automatically

Deleted object	Method	Dependent objects that are deleted automatically
Connection	Close Connection	ConnectionMetaData and Session objects
Session	Close Session	MessageConsumer and MessageProducer objects

Iterators

Using an iterator, an application can retrieve the elements in a list. The iterator encapsulates a list and a cursor to the list. When an iterator is created, the position of the cursor is before the first element in the list.

The Iterator class provides the methods for using an iterator and is equivalent to the Enumerator class in Java. The Iterator class provides three methods that an application can use to retrieve the elements in a list sequentially:

- Check for More Properties
- Get Next Property
- Reset Iterator

An application can use an iterator to retrieve the properties of a message, and to retrieve the name-value pairs in the body of a map message. The following code fragment shows how a C application can use an iterator to print out all properties of a message:

```
/********************************************************/
/* XMS Sample using an iterator to browse properties    */
/********************************************************/
rc = xmsMsgGetProperties(hMsg, &it, xmsError);
if (rc == XMS_OK)
{
  rc = xmsIteratorHasNext(it, &more, xmsError);
  while (more)
  {
    rc = xmsIteratorGetNextProperty(it, &p, xmsError);
    if (rc == XMS_OK)
    {
      xmsPropertyGetName(p, name, 100, &len, xmsError);
      printf("Property name=\"%s\"\n", name);
      xmsPropertyGetType(p, &type, xmsError);
      switch (type)
      {
        case XMS_PROPERTY_TYPE_INT:
        {
          xmsINT value=0;
          xmsPropertyGetInt(p, &value, xmsError);
          printf("Property value=%d\n", value);
          break;
        }
        case XMS_PROPERTY_TYPE_STRING:
        {
          xmsSIZE len=0;
          char value[100];
          xmsPropertyGetString(p, value, 100, &len, xmsError);
          printf("Property value=\"%s\"\n", value);
          break;
        }
        default:
        {
          printf("Unhandled property type (%d)\n", (int)type);
        }
      }
      xmsPropertyDispose(&p, xmsError);
    }
    rc = xmsIteratorHasNext(it, &more, xmsError);
  }
  printf("Finished iterator....\n");
  xmsIteratorDispose(&it, xmsError);
}
/*****************************************************/
```

Chapter 5. Writing XMS applications in C

This chapter provides information that you might find useful when writing Multi-Language Message Service (XMS) applications in C. The chapter contains the following sections:

- "Object handles in C"
- "Listeners and callbacks"
- "C functions that return a string or byte array by value" on page 28
- "C functions that return a string or byte array by reference" on page 29
- "Handling errors in C" on page 30

Object handles in C

When writing an XMS application in C, every object handle has a data type, where the data type of the handle relates specifically to the type of object. Table 5 shows the handle data type for each type of object.

Note, however, that BytesMessage, MapMessage, and Message objects all have handles with data type xmsHMsg. For more information about how to use handles for messages, see "The body of an XMS message" on page 34.

Table 5. Object handle data types

Type of object	Object handle data type
BytesMessage	xmsHMsg
Connection	xmsHConn
ConnectionFactory	xmsHConnFact
ConnectionMetaData	xmsHConnMetaData
Destination	xmsHDest
ErrorBlock	xmsHErrorBlock
Iterator	xmsHIterator
MapMessage	xmsHMsg
Message	xmsHMsg
MessageConsumer	xmsHMsgConsumer
MessageProducer	xmsHMsgProducer
Property	xmsHProperty

Listeners and callbacks

The API supports two types of asynchronous callback: message callbacks and exception callbacks.

A C application calls the Set Exception Listener or Set Message Listener method to pass a pointer to a callback function to XMS. On the same call, the application passes a pointer to context data, which XMS subsequently passes to the callback function when XMS calls the function.

The context data is in an area of memory defined by the application. For example, it might be a structure allocated on the heap. The context data contains all the information that the application needs to refer to when XMS calls the onMessage() or onException() function.

XMS does not take a copy of the context data and considers the area of memory occupied by the context data to be owned by the application. Your application must ensure that the context data is still available and in scope when XMS calls the callback function.

The signature of the C message callback function is:

```
xmsVOID onMessage(xmsCONTEXT context, xmsHMsg message);
```

The application must free the resources used by the message when the onMessage() function returns. XMS does not free these resources.

The signature of the C exception callback function is:

```
xmsVOID onException(xmsCONTEXT context, xmsHErrorBlock errorBlock);
```

The application must free the resources used by the error block when the onException() function returns. XMS does not free these resources.

An application can stop the asynchronous delivery of messages or exceptions by passing a null function pointer to the Set Message Listener method or Set Exception Listener method respectively.

C functions that return a string or byte array by value

In the C API, certain functions return a string or byte array as a parameter. Each of these functions provides essentially the same interface for the purpose of retrieving a string or byte array.

Here is an example of one of these functions. The function implements the Get String Property method in the Message class and returns a string.

```
xmsRC xmsMsgGetStringProperty(xmsHMsg message,
                              xmsCHAR *propertyName,
                              xmsCHAR *propertyValue,
                              xmsSIZE length,
                              xmsSIZE *actualLength,
                              xmsHErrorBlock errorBlock);
```

Three parameters control the retrieval of the string:

propertyValue
: This parameter is a pointer to an buffer provided by the application into which XMS copies the characters in the string. If data conversion is required, XMS converts the characters into the code page of the application before copying them into the buffer.

length This parameter is the length of the buffer in bytes. This is an input parameter that must be set by the application before the call.

actualLength
: This output parameter is the length of the string that XMS stores in the buffer. The length is measured in bytes. If data conversion is required, this is the length after conversion.

If the buffer is not large enough to contain the whole string, XMS returns the string truncated to the length of the buffer and sets error code XMS_E_DATA_TRUNCATED in the error block.

If the length parameter is zero, XMS returns the length of the string in the actualLength parameter but does not copy the string into the buffer. XMS sets error code XMS_E_DATA_TRUNCATED in this case as well.

If the length of the buffer is larger than the length of the string that XMS copies into the buffer, XMS appends a null character to the end of the string. If the length of the buffer is less than or equal to the length of the string, XMS does not append a null character. The length of the string in the buffer, as reported by XMS in the actualLength parameter, does not include this null character. If a function returns a byte array instead of a string, XMS still appends a null character to the end of the byte array, if there is room in the buffer.

Note that, if an XMS application receives a message sent by a WebSphere MQ JMS application, the values of header fields, properties, and application data in the message that are strings might contain embedded null characters.

C functions that return a string or byte array by reference

When a C application calls one of the functions discussed in “C functions that return a string or byte array by value” on page 28, XMS must copy the string or byte array into the buffer provided by the application. If the string or byte array is very large, the time taken to copy it might have a significant impact on performance.

To deliver better performance in this situation, the C API provides another set of functions. When an application calls one of these functions, one parameter returns a pointer to a string or byte array that is stored in memory owned by XMS, and another parameter returns the length of the string or byte array.

Here are some examples of these functions:
- xmsMsgGetStringPropertyByRef(), which implements the Get String Property by Reference method in the Message class
- xmsBytesMsgReadBytesByRef(), which implements the Read Bytes by Reference method in the BytesMessage class

If data conversion is required for a string, XMS converts the characters into the code page of the application and returns a pointer to the converted string. The length returned to the application is the length of the converted string.

If data conversion is required, the first time an application retrieves a string by reference might take as long as retrieving the string by value. However, XMS caches the converted string and so subsequent calls to retrieve the same string do not take as long.

Because these functions return a pointer to memory owned by XMS, the application must not attempt to free or modify the contents of this memory. Attempting to do so might cause unpredictable results.

The pointer returned to the application remains valid until the XMS object, with which string or byte array is associated, is deleted. The application must copy the string or byte array if it needs the data after the object is deleted.

Handling errors in C

Most functions in the C API return a value that is a return code, and have an optional input parameter that is a handle for an error block. This section describes the respective roles of the return code and the error block.

Return codes

The return code from a C function call indicates whether the call was successful. The return code has data type xmsRC. Table 6 shows the possible return codes and their meaning.

Table 6. Return codes from C function calls

Return code	Meaning
XMS_OK	The call completed successfully.
Any other value	The call failed. The error block contains more details about why the call failed.

The error block

When an application calls a C function, the application can include a handle for an error block as an input parameter on the call. If the call fails, XMS stores information in the error block about why the call failed. The application can then retrieve this information from the error block.

An error block contains the following information:

Exception code
An integer representing the exception. The header file xmsc.h defines a named constant for each exception code.

Error code
An integer representing the error. The header file xmsc.h defines a named constant for each error code.

Error string
A null terminated string of characters that describes the error. The characters in the string are the same as those in the named constant that represents the error code.

Error data
A null terminated string of characters that provides additional information about the error. The information is free format.

Linked error
The handle for an linked error block. If XMS needs to report more information about an error, XMS can create one or more additional error blocks and chain them from the error block provided by the application.

XMS provides a set of helper functions to create an error block and extract information from it. An application must use a helper function to create an error block and obtain a handle for it before calling the first function that can accept the handle as an input parameter. If the function call fails, the application can then use other helper functions to extract information about the error that XMS has stored in the error block. For details of these helper functions, see "ErrorBlock" on page 80.

Chapter 6. XMS messages

This chapter describes the structure and content of Multi-Language Message Service (XMS) messages and how applications process XMS messages.

An XMS message has the following parts:

A header
: The header of a message contains fields, and all messages contain the same set of header fields. XMS and applications use the values of the header fields to identify and route messages. For more information about header fields, see "Header fields in an XMS message."

A set of properties
: The properties of a message specify additional information about the message. Although all messages have the same set of header fields, every message can have a different set of properties. For more information about the properties of a message, see "Properties of an XMS message" on page 32.

A body
: The body of a message contains the application data. For more information about the body of a message, see "The body of an XMS message" on page 34.

An application can select which messages it wants to receive. It does this by using message selectors, which specify the selection criteria. The criteria can be based on the values of certain header fields and the values of any of the properties of a message. For more information about message selectors, see "Message selectors" on page 37.

Header fields in an XMS message

To allow an XMS application to exchange messages with a WebSphere MQ JMS application, the header of an XMS message contains the JMS message header fields. The names of these header fields commence with the prefix JMS. For a description of the JMS message header fields, see the *Java Message Service Specification, Version 1.1*.

XMS implements the JMS message header fields as attributes of a Message object. Each header field has its own methods for setting and getting its value. For a description of these methods, see "Message" on page 106. A header field is always readable and writable.

Table 7 lists the JMS message header fields and indicates how the value of each field is set for a transmitted message. Note that some of the fields are set automatically by XMS when an application sends a message or, in the case of JMSRedelivered, when an application receives a message.

Table 7. JMS message header fields

Name of the JMS message header field	**How the value is set for a transmitted message (in the format** *method* [*class*]**)**
JMSCorrelationID	Set JMSCorrelationID [Message]

Table 7. JMS message header fields (continued)

Name of the JMS message header field	How the value is set for a transmitted message (in the format *method* [*class*])
JMSDeliveryMode	Send [MessageProducer]
JMSDestination	Send [MessageProducer]
JMSExpiration	Send [MessageProducer]
JMSMessageID	Send [MessageProducer]
JMSPriority	Send [MessageProducer]
JMSRedelivered	Receive [MessageConsumer]
JMSReplyTo	Set JMSReplyTo [Message]
JMSTimestamp	Send [MessageProducer]
JMSType	Set JMSType [Message]

Properties of an XMS message

To allow an XMS application to exchange messages with a WebSphere MQ JMS application, XMS supports the following predefined properties of a Message object:

- The same JMS defined properties that WebSphere MQ JMS supports. The names of these properties commence with the prefix JMSX.
- The same IBM defined properties that WebSphere MQ JMS supports. The names of these properties commence with the prefix JMS_IBM_.

Each predefined property has two names:

- An XMS name. Use this name in an XMS application to identify the property, except in a message selector expression.
- A JMS name. This is the name by which the property is known in JMS, and is also the name that is transmitted with a message that has this property. Use this name in an XMS application to identify the property in a message selector expression.

In addition to the predefined properties, an XMS application can create and use its own set of message properties. These properties are called *application defined properties*.

A message property does not have its own methods for setting and getting its value. Instead, the Message class provides the following generic methods:

- A set and a get method for each of the following data types: xmsBOOL, xmsBYTE, xmsSHORT, xmsINT, xmsLONG, xmsFLOAT, xmsDOUBLE, and String (or character array, if you are using the C interface)
- A Set Property method and a Get Property method

For more information about these methods, see "Message" on page 106.

After an application creates a message, the properties of the message are readable and writable. The properties remain readable and writable after the application sends the message. When an application receives a message, the properties of the message are read-only. If an application calls the Clear Properties method of the Message class when the properties of a message are read-only, the properties become readable and writable. The method also clears the properties.

To determine the values of all the properties of a message, an application can call the Get Properties method of the Message class. The method creates an iterator that encapsulates a list of Property objects, where each Property object represents a property of the message. The application can then use the methods of the Iterator class to retrieve each Property object in turn, and use the methods of the Property class to retrieve the name, data type, and value of each property. For a sample fragment of C code that performs a similar function, see "Iterators" on page 24.

JMS defined properties of a message

Table 8 lists the JMS defined properties of a message that are supported by both XMS and WebSphere MQ JMS. For a description of the JMS defined properties, see the *Java Message Service Specification, Version 1.1*.

The table specifies the data type of each property and indicates how the value of the property is set for a transmitted message. Note that some of the properties are set automatically by XMS when an application sends a message or, in the case of JMSXDeliveryCount, when an application receives a message.

Table 8. JMS defined properties of a message

XMS name of the JMS defined property	**JMS name**	**Data type**	**How the value is set for a transmitted message (in the format** *method* [*class*]**)**
JMSX_APPID	JMSXAppID	String[1]	Send [MessageProducer]
JMSX_DELIVERY_COUNT	JMSXDeliveryCount	xmsINT	Receive [MessageConsumer]
JMSX_GROUPID	JMSXGroupID	String[1]	Set String Property [Message]
JMSX_GROUPSEQ	JMSXGroupSeq	xmsINT	Set Integer Property [Message]
JMSX_USERID	JMSXUserID	String[1]	Send [MessageProducer]

Notes:

1. This is the data type if you are using C++. If you are programming in C, it is a character array.

IBM defined properties of a message

Table 9 lists the IBM defined properties of a message that are supported by both XMS and WebSphere MQ JMS. For more information about the IBM defined properties, see *WebSphere MQ Using Java*.

The table specifies the data type of each property and indicates how the value of the property is set for a transmitted message. Note that some of the properties are set automatically by XMS when an application sends a message.

Table 9. IBM defined properties of a message

XMS name of the IBM defined property	**WebSphere MQ JMS name**	**Data type**	**How the value is set for a transmitted message** *method* [*class*]**)**
JMS_IBM_CHARACTER_SET	JMS_IBM_Character_Set	xmsINT	Set Integer Property [Message]
JMS_IBM_ENCODING	JMS_IBM_Encoding	xmsINT	Set Integer Property [Message]
JMS_IBM_FEEDBACK	JMS_IBM_Feedback	xmsINT	Set Integer Property [Message]
JMS_IBM_FORMAT	JMS_IBM_Format	String[1]	Set String Property [Message]
JMS_IBM_LAST_MSG_IN_GROUP	JMS_IBM_Last_Msg_In_Group	xmsINT	Set Integer Property [Message]
JMS_IBM_MSGTYPE	JMS_IBM_MsgType	xmsINT	Set Integer Property [Message]

Table 9. IBM defined properties of a message (continued)

XMS name of the IBM defined property	WebSphere MQ JMS name	Data type	How the value is set for a transmitted message *method* [*class*])
JMS_IBM_PUTAPPLTYPE	JMS_IBM_PutApplType	xmsINT	Send [MessageProducer]
JMS_IBM_PUTDATE	JMS_IBM_PutDate	String[1]	Send [MessageProducer]
JMS_IBM_PUTTIME	JMS_IBM_PutTime	String[1]	Send [MessageProducer]
JMS_IBM_REPORT_COA	JMS_IBM_Report_COA	xmsINT	Set Integer Property [Message]
JMS_IBM_REPORT_COD	JMS_IBM_Report_COD	xmsINT	Set Integer Property [Message]
JMS_IBM_REPORT_DISCARD_MSG	JMS_IBM_Report_Discard_Msg	xmsINT	Set Integer Property [Message]
JMS_IBM_REPORT_EXCEPTION	JMS_IBM_Report_Exception	xmsINT	Set Integer Property [Message]
JMS_IBM_REPORT_EXPIRATION	JMS_IBM_Report_Expiration	xmsINT	Set Integer Property [Message]
JMS_IBM_REPORT_NAN	JMS_IBM_Report_NAN	xmsINT	Set Integer Property [Message]
JMS_IBM_REPORT_PAN	JMS_IBM_Report_PAN	xmsINT	Set Integer Property [Message]
JMS_IBM_REPORT_PASS_CORREL_ID	JMS_IBM_Report_Pass_Correl_ID	xmsINT	Set Integer Property [Message]
JMS_IBM_REPORT_PASS_MSG_ID	JMS_IBM_Report_Pass_Msg_ID	xmsINT	Set Integer Property [Message]

Notes:

1. This is the data type if you are using C++. If you are programming in C, it is a character array.

Application defined properties of a message

An XMS application can create and use its own set of message properties. When an application sends a message, these properties are also transmitted with the message. A receiving application, using message selectors, can then select which messages it wants to receive based upon the values of these properties.

To allow a JMS application to select and process messages sent by an XMS application, the name of an application defined property must conform to the rules documented in *WebSphere MQ Using Java*. The value of an application defined property must have one of the following data types: xmsBOOL, xmsBYTE, xmsSHORT, xmsINT, xmsLONG, xmsFLOAT, xmsDOUBLE, or String (or character array, if you are using the C interface).

The body of an XMS message

The body of a message contains the application data. However, a message can contain no application data and therefore have no body. In this case, the message comprises only the header fields and properties.

The first release of XMS supports two types of message body:

Bytes The body contains a stream of bytes. A message with this type of body is called a *bytes message*. The BytesMessage class contains the methods to process the body of a bytes message. For more information about bytes messages, see "Bytes messages" on page 36.

Map The body contains a set of name-value pairs. A message with this type of body is called a *map message*. The MapMessage class contains the methods to process the body of a map message. For more information about map messages, see "Map messages" on page 36.

In the C interface, XMS returns a message handle to an application when the application creates a bytes message or map message. The application can use this handle to call any of the methods of the Message class, and any of the methods of the BytesMessage or MapMessage class, whichever is appropriate for the type of message body. However, if an application tries to call a method that is inappropriate for the type of message body, the call fails and XMS returns error code XMS_E_BAD_PARAMETER.

A C application can use the Get Type method of the Message class to determine the body type of a message. The Get Type method returns one of the following values:

XMSC_T_BYTES_MSG
: If the message is a bytes message

XMSC_T_MAP_MSG
: If the message is a map message

XMSC_T_MSG
: If the message contains no application data and therefore has no body.

See the following fragment of C code, for example:

```
xmsMESSAGE_TYPE msgtype;
xmsMsgConsumerReceive(messageConsumer, &msg, errorBlock);
xmsMsgGetTypeId(msg, &msgtype, errorBlock);
if (msgtype == XMSC_T_BYTES_MSG)
{
  xmsBytesMsgGetBodyLength(msg, &length, errorBlock);
}
```

In the C++ interface, BytesMessage and MapMessage are subclasses of the Message class.

To ensure that XMS applications can exchange messages with WebSphere MQ JMS applications, an XMS application and a WebSphere MQ JMS application must be able to interpret the application data in the body of a message in the same way. For this reason, each element of application data written in the body of a message by an XMS application must have one of the data types listed in Table 10. For each XMS data type, the table shows the compatible Java data type. XMS provides the methods to write elements of application data with these data types, and only these data types.

Table 10. XMS data types that are compatible with Java data types

XMS data type	Represents	Size	Compatible Java data type
xmsBOOL	The boolean value `xmsTRUE` or `xmsFALSE`	32 bits	boolean
xmsCHAR16	Double byte character	16 bits	char
xmsBYTE	Signed 8-bit integer	8 bits	byte
xmsSHORT	Signed 16-bit integer	16 bits	short
xmsINT	Signed 32-bit integer	32 bits	int
xmsLONG	Signed 64-bit integer	64 bits	long
xmsFLOAT	Signed floating point number	32 bits	float
xmsDOUBLE	Signed double precision floating point number	64 bits	double
String[1]	String of characters	-	String

Notes:

1. This is the data type if you are using C++. If you are programming in C, it is a character array.

Bytes messages

The body of a bytes message contains a stream of bytes. After an application creates a bytes message, the body of the message is write-only. The application assembles the application data into the body by calling the appropriate write methods of the BytesMessage class. Each time the application writes a value to the bytes message stream, the value is assembled immediately after the previous value written by the application. XMS maintains an internal cursor to remember the position of the last byte that was assembled.

When the application sends the message, the body of the message becomes read-only. In this mode, the application can send the message multiple times.

When an application receives a bytes message, the body of the message is read-only. The application can use the appropriate read methods of the BytesMessage class to read the contents of the bytes message stream. The application reads the bytes in sequence, and XMS maintains an internal cursor to remember the position of the last byte that was read. The application can skip over bytes without reading them by calling a read method with a null pointer for the `value` parameter (or for the `buffer` parameter, if the application calls Read Bytes).

If an application calls the Reset method of the BytesMessage class when the body of a bytes message is write-only, the body becomes read-only. The method also repositions the cursor at the beginning of the bytes message stream.

If an application calls the Clear Body method of the Message class when the body of a bytes message is read-only, the body becomes write-only. The method also clears the body.

When a bytes message is transported over WebSphere MQ Real-Time Transport or WebSphere MQ Multicast Transport, no data conversion is performed on the body of the message, except for character data encoded in UTF-8 format, which is converted.

Map messages

The body of a map message contains a set of name-value pairs. In each name-value pair, the name is a string that identifies the value, and the value is an element of application data that has one of the XMS data types listed in Table 10 on page 35. The order of the name-value pairs is not defined. The MapMessage class contains the methods to set and get name-value pairs.

An application can access a name-value pair randomly by specifying its name. Alternatively, the application can access the name-value pairs sequentially using an iterator. The application can call the Get Name-Value Pairs method of the MapMessage class to create an iterator that encapsulates a list of Property objects, where each Property object encapsulates a name-value pair. The application can then use the methods of the Iterator class to retrieve each Property object in turn, and use the methods of the Property class to retrieve the name, data type, and value of each name-value pair. Although a name-value pair is not a property, the methods of the Property class treat a name-value pair like a property.

After an application creates a map message, the body of the message is readable and writable. The body remains readable and writable after the application sends the message. When an application receives a map message, the body of the message is read-only. If an application calls the Clear Body method of the Message class when the body of a map message is read-only, the body becomes readable and writable. The method also clears the body.

If a connection uses WebSphere MQ Real-Time Transport, XMS converts the character data in the bodies of outgoing map messages into Unicode format so that the messages can be received and processed by WebSphere MQ JMS applications.

Message selectors

An XMS application uses messages selectors to select which messages it wants to receive.

When an application creates a message consumer, it can associate a message selector expression with the message consumer. The message selector expression specifies the selection criteria. XMS determines whether each incoming message satisfies the selection criteria. If a message satisfies the selection criteria, XMS delivers the message to the message consumer. If a message does not satisfy the selection criteria, XMS does not deliver the message.

An application can create more than one message consumer, each with its own message selector expression. If an incoming message satisfies the selection criteria of more than one message consumer, XMS delivers the message to each of these consumers.

A message selector expression can reference the following properties of a message:
- JMS defined properties
- IBM defined properties
- Application defined properties

It can also reference the following message header fields:
- JMSCorrelationID
- JMSDeliveryMode
- JMSMessageID
- JMSPriority
- JMSTimestamp
- JMSType

A message selector expression, however, cannot reference data in the body of a message.

Here is an example of a message selector expression:

```
JMSPriority > 3 AND manufacturer = 'Jaguar' AND model in ('xj6','xj12')
```

XMS delivers a message to a message consumer with this message selector expression only if the message has a priority greater than `3`, an application defined property, manufacturer, with a value of `Jaguar`, and another application defined property, model, with a value of `xj6` or `xj12`.

The syntax rules for forming a message selector expression in XMS are the same as those in WebSphere MQ JMS. For information about how to construct a message selector expression, therefore, see *WebSphere MQ Using Java*. Note, in particular, that, in a message selector expression, the names of JMS defined properties and IBM defined properties must be the JMS names, not the XMS names.

Part 3. XMS API reference

Chapter 7. XMS classes

This chapter documents the XMS classes and their methods. Table 11 summarizes all the classes.

Table 11. A summary of the XMS classes

Class	Description	Page
BytesMessage	A bytes message is a message whose body comprises a stream of bytes.	45
Connection	A Connection object represents an application's active connection to a broker.	58
ConnectionFactory	An application uses a connection factory to create a connection.	63
ConnectionMetaData	A ConnectionMetaData object provides information about a connection.	70
Destination	A destination is where an application sends messages, or it is a source from which an application receives messages, or both.	73
ErrorBlock	If a C function call fails, XMS can store information in an error block about why the call failed.	80
Exception	If a call of a C++ method fails, XMS creates an Exception object, which encapsulates information about why the call failed.	84
ExceptionListener	An application uses an exception listener to be notified asynchronously of a problem with a connection.	86
Iterator	An iterator encapsulates a list of Property objects. An application can use an iterator to retrieve each Property object in turn.	87
MapMessage	A map message is a message whose body comprises a set of name-value pairs.	90
Message	A Message object represents a message that an application sends or receives.	106
MessageConsumer	An application uses a message consumer to receive messages sent to a destination.	131
MessageListener	An application uses a message listener to receive messages asynchronously.	136
MessageProducer	An application uses a message producer to send messages to a destination.	137
Property	A Property object represents a property of an object.	147
Session	A session is a single threaded context for sending and receiving messages.	163

Exceptions

Table 12 on page 44 lists all the exception codes that XMS can return to a C application. For each exception code, the table shows the corresponding C++ exception class.

Table 12. C exception codes and C++ exception classes

C exception code	C++ exception class	Explanation
XMS_X_GENERAL_EXCEPTION	Exception	
XMS_X_ILLEGAL_STATE_EXCEPTION	IllegalStateException	
XMS_X_INVALID_CLIENTID_EXCEPTION	InvalidClientIDException	
XMS_X_INVALID_DESTINATION_EXCEPTION	InvalidDestinationException	
XMS_X_INVALID_SELECTOR_EXCEPTION	InvalidSelectorException	The application attempted to use a message selector expression that is not valid.
XMS_X_MESSAGE_EOF_EXCEPTION	MessageEOFException	
XMS_X_MESSAGE_FORMAT_EXCEPTION	MessageFormatException	
XMS_X_MESSAGE_NOT_READABLE_EXCEPTION	MessageNotReadableException	
XMS_X_MESSAGE_NOT_WRITABLE_EXCEPTION	MessageNotWritableException	
XMS_X_RESOURCE_ALLOCATION_EXCEPTION	ResourceAllocationException	
XMS_X_SECURITY_EXCEPTION	SecurityException	

BytesMessage

A bytes message is a message whose body comprises a stream of bytes.

Methods

Get Body Length

C interface:

```
xmsRC xmsBytesMsgGetBodyLength(xmsHMsg message,
                               xmsLONG *bodyLength,
                               xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsLONG getBodyLength() const;
```

Get the length of the body of the message when the message is in read-only mode.

Parameters:

message (input)
The handle for the message.

bodyLength (output)
The length of the body of the message in bytes. The call returns the length of the whole body regardless of where the cursor for reading the message is currently positioned.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_READABLE_EXCEPTION

Read Boolean Value

C interface:

```
xmsRC xmsBytesMsgReadBoolean(xmsHMsg message,
                             xmsBOOL *value,
                             xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsBOOL readBoolean() const;
```

Read the next byte from the bytes message stream as a boolean value .

Parameters:

message (input)
The handle for the message.

value (output)
The boolean value that is read.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_READABLE_EXCEPTION

- XMS_X_MESSAGE_EOF_EXCEPTION

Read Byte

C interface:
```
xmsRC xmsBytesMsgReadByte(xmsHMsg message,
                          xmsBYTE *value,
                          xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsBYTE readByte() const;
```

Read the next byte from the bytes message stream as a signed 8-bit integer.

Parameters:

message (input)
The handle for the message.

value (output)
The byte that is read.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_READABLE_EXCEPTION
- XMS_X_MESSAGE_EOF_EXCEPTION

Read Bytes

C interface:
```
xmsRC xmsBytesMsgReadBytes(xmsHMsg message,
                           xmsBYTE *buffer,
                           xmsSIZE bufferLength,
                           xmsSIZE *returnedLength,
                           xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsINT readBytes(xmsBYTE *buffer,
                 const xmsSIZE bufferLength,
                 xmsSIZE *returnedLength) const;
```

Read an array of bytes from the bytes message stream starting from the current position of the cursor.

Parameters:

message (input)
The handle for the message.

buffer (output)
The buffer to contain the array of bytes that is read. If the number of bytes remaining to be read from the stream before the call is greater than or equal to the length of the buffer, the buffer is filled. Otherwise, the buffer is partially filled with all the remaining bytes.

bufferLength (input)
The length of the buffer in bytes.

returnedLength (output)
The number of bytes that are read into the buffer. If the buffer is partially filled, the value is less than the length of the buffer, indicating that there are no more bytes remaining to be read. If there are no bytes remaining to be read from the stream before the call, the value is XMSC_END_OF_STREAM.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_READABLE_EXCEPTION
- XMS_X_MESSAGE_EOF_EXCEPTION

Read Bytes by Reference

C interface:

```
xmsRC xmsBytesMsgReadBytesByRef(xmsHMsg message,
                                xmsBYTE **stream,
                                xmsSIZE *length,
                                xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Get a pointer to the start of the bytes message stream and get the length of the stream.

For more information about how to use this function, see “C functions that return a string or byte array by reference” on page 29.

Parameters:

message (input)
The handle for the message.

stream (output)
A pointer to the start of the bytes message stream.

length (output)
The number of bytes in the bytes message stream.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_READABLE_EXCEPTION
- XMS_X_MESSAGE_EOF_EXCEPTION

Read Character

C interface:
```
xmsRC xmsBytesMsgReadChar(xmsHMsg message,
                          xmsCHAR16 *value,
                          xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsCHAR16 readChar() const;
```

Read the next 2 bytes from the bytes message stream as a character.

Parameters:

message (input)
The handle for the message.

value (output)
The character that is read.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_READABLE_EXCEPTION
- XMS_X_MESSAGE_EOF_EXCEPTION

Read Double Precision Floating Point Number

C interface:
```
xmsRC xmsBytesMsgReadDouble(xmsHMsg message,
                            xmsDOUBLE *value,
                            xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsDOUBLE readDouble() const;
```

Read the next eight bytes from the bytes message stream as a double precision floating point number.

Parameters:

message (input)
The handle for the message.

value (output)
The double precision floating point number that is read.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_READABLE_EXCEPTION
- XMS_X_MESSAGE_EOF_EXCEPTION

Read Floating Point Number

C interface:

```
xmsRC xmsBytesMsgReadFloat(xmsHMsg message,
                           xmsFLOAT *value,
                           xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsFLOAT readFloat() const;
```

Read the next four bytes from the bytes message stream as a floating point number.

Parameters:

message (input)
The handle for the message.

value (output)
The floating point number that is read.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_READABLE_EXCEPTION
- XMS_X_MESSAGE_EOF_EXCEPTION

Read Integer

C interface:

```
xmsRC xmsBytesMsgReadInt(xmsHMsg message,
                         xmsINT *value,
                         xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsINT readInt() const;
```

Read the next four bytes from the bytes message stream as a signed 32-bit integer.

Parameters:

message (input)
The handle for the message.

value (output)
The integer that is read.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_READABLE_EXCEPTION
- XMS_X_MESSAGE_EOF_EXCEPTION

Read Long Integer

C interface:
```
xmsRC xmsBytesMsgReadLong(xmsHMsg message,
                          xmsLONG *value,
                          xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsLONG readLong() const;
```

Read the next eight bytes from the bytes message stream as a signed 64-bit integer.

Parameters:

message (input)
The handle for the message.

value (output)
The long integer that is read.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_READABLE_EXCEPTION
- XMS_X_MESSAGE_EOF_EXCEPTION

Read Short Integer

C interface:
```
xmsRC xmsBytesMsgReadShort(xmsHMsg message,
                           xmsSHORT *value,
                           xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsSHORT readShort() const;
```

Read the next two bytes from the bytes message stream as a signed 16-bit integer.

Parameters:

message (input)
The handle for the message.

value (output)
The short integer that is read.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_READABLE_EXCEPTION
- XMS_X_MESSAGE_EOF_EXCEPTION

Read Unsigned Byte

C interface:

```
xmsRC xmsBytesMsgReadUnsignedByte(xmsHMsg message,
                                  xmsUINT8 *value,
                                  xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsUINT8 readUnsignedByte() const;
```

Read the next byte from the bytes message stream as an unsigned 8-bit integer.

Parameters:

message (input)
The handle for the message.

value (output)
The byte that is read.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_READABLE_EXCEPTION
- XMS_X_MESSAGE_EOF_EXCEPTION

Read Unsigned Short Integer

C interface:

```
xmsRC xmsBytesMsgReadUnsignedShort(xmsHMsg message,
                                   xmsUINT16 *value,
                                   xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsUINT16 readUnsignedShort() const;
```

Read the next two bytes from the bytes message stream as an unsigned 16-bit integer.

Parameters:

message (input)
The handle for the message.

value (output)
The unsigned short integer that is read.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_READABLE_EXCEPTION
- XMS_X_MESSAGE_EOF_EXCEPTION

Read UTF String

C interface:
```
xmsRC xmsBytesMsgReadUTF(xmsHMsg message,
                         xmsCHAR *buffer,
                         xmsSIZE bufferLength,
                         xmsSIZE *actualLength,
                         xmsHErrorBlock errorBlock);
```

C++ interface:
```
String readUTF() const;
```

Read a string, encoded in UTF-8 format, from the bytes message stream.

Parameters:

message (input)
The handle for the message.

buffer (output)
The buffer to contain the string that is read.

bufferLength (input)
The length of the buffer in bytes. If you specify a length of zero, the string is not returned, but its length is returned in the actualLength parameter.

actualLength (output)
The length of the string in bytes.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_READABLE_EXCEPTION
- XMS_X_MESSAGE_EOF_EXCEPTION

Reset

C interface:
```
xmsRC xmsBytesMsgReset(xmsHMsg message,
                       xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID reset() const;
```

Put the body of the message into read-only mode and reposition the cursor at the beginning of the bytes message stream.

Parameters:

message (input)
The handle for the message.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_READABLE_EXCEPTION

- XMS_X_MESSAGE_EOF_EXCEPTION

Write Boolean Value

C interface:
```
xmsRC xmsBytesMsgWriteBoolean(xmsHMsg message,
                              xmsBOOL value,
                              xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID writeBoolean(const xmsBOOL value);
```

Write a boolean value to the bytes message stream as 1 byte.

Parameters:

message (input)
The handle for the message.

value (input)
The boolean value to be written. The value `xmsTRUE` is written as `X'01'`, and the value `xmsFALSE` is written as `X'00'`.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_WRITABLE_EXCEPTION

Write Byte

C interface:
```
xmsRC xmsBytesMsgWriteByte(xmsHMsg message,
                           xmsBYTE value,
                           xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID writeByte(const xmsBYTE value);
```

Write a byte to the bytes message stream.

Parameters:

message (input)
The handle for the message.

value (input)
The byte to be written.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_WRITABLE_EXCEPTION

Write Bytes

C interface:

```
xmsRC xmsBytesMsgWriteBytes(xmsHMsg message,
                            xmsBYTE *value,
                            xmsSIZE length,
                            xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsVOID writeBytes(const xmsBYTE *value,
                   const xmsSIZE length);
```

Write an array of bytes to the bytes message stream.

Parameters:

message (input)
The handle for the message.

value (input)
The array of bytes to be written.

length (input)
The number of bytes in the array.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_WRITABLE_EXCEPTION

Write Character

C interface:

```
xmsRC xmsBytesMsgWriteChar(xmsHMsg message,
                           xmsCHAR16 value,
                           xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsVOID writeChar(const xmsCHAR16 value);
```

Write a character to the bytes message stream as 2 bytes, high order byte first.

Parameters:

message (input)
The handle for the message.

value (input)
The character to be written.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_WRITABLE_EXCEPTION

Write Double Precision Floating Point Number

C interface:

```
xmsRC xmsBytesMsgWriteDouble(xmsHMsg message,
                             xmsDOUBLE value,
                             xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsVOID writeDouble(const xmsDOUBLE value);
```

Convert a double precision floating point number to a long integer and write the long integer to the bytes message stream as 8 bytes, high order byte first.

Parameters:

message (input)
The handle for the message.

value (input)
The double precision floating point number to be written.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_WRITABLE_EXCEPTION

Write Floating Point Number

C interface:

```
xmsRC xmsBytesMsgWriteFloat(xmsHMsg message,
                            xmsFLOAT value,
                            xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsVOID writeFloat(const xmsFLOAT value);
```

Convert a floating point number to an integer and write the integer to the bytes message stream as 4 bytes, high order byte first.

Parameters:

message (input)
The handle for the message.

value (input)
The floating point number to be written.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_WRITABLE_EXCEPTION

Write Integer

C interface:
```
xmsRC xmsBytesMsgWriteInt(xmsHMsg message,
                          xmsINT value,
                          xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID writeInt(const xmsINT value);
```

Write an integer to the bytes message stream as 4 bytes, high order byte first.

Parameters:

message (input)
The handle for the message.

value (input)
The integer to be written.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_WRITABLE_EXCEPTION

Write Long Integer

C interface:
```
xmsRC xmsBytesMsgWriteLong(xmsHMsg message,
                           xmsLONG value,
                           xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID writeLong(const xmsLONG value);
```

Write a long integer to the bytes message stream as 8 bytes, high order byte first.

Parameters:

message (input)
The handle for the message.

value (input)
The long integer to be written.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_WRITABLE_EXCEPTION

Write Short Integer

C interface:
```
xmsRC xmsBytesMsgWriteShort(xmsHMsg message,
                            xmsSHORT value,
                            xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsVOID writeShort(const xmsSHORT value);
```

Write a short integer to the bytes message stream as 2 bytes, high order byte first.

Parameters:

message (input)
: The handle for the message.

value (input)
: The short integer to be written.

errorBlock (input)
: The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_WRITABLE_EXCEPTION

Write UTF String

C interface:

```
xmsRC xmsBytesMsgWriteUTF(xmsHMsg message,
                          xmsCHAR *value,
                          xmsSIZE length,
                          xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsVOID writeUTF(const String & value);
```

Write a string, encoded in UTF-8 format, to the bytes message stream.

Parameters:

message (input)
: The handle for the message.

value (input)
: The string, which must be encoded in UTF-8 format.

length (input)
: The length of the string in bytes.

errorBlock (input)
: The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_WRITABLE_EXCEPTION

Connection

A Connection object represents an application's active connection to a broker.

Methods

Close Connection

C interface:
```
xmsRC xmsConnClose(xmsHConn *connection,
                   xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID close();
```

Close the connection.

If an application tries to close a connection that is already closed, the call is ignored.

Parameters:

connection (input/output)
: On input, the handle for the connection. On output, the call returns a null handle.

errorBlock (input)
: The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Create Session

C interface:
```
xmsRC xmsConnCreateSession(xmsHConn connection,
                           xmsBOOL transacted,
                           xmsACKNOWLEDGE_MODE acknowledgeMode,
                           xmsHSess *session,
                           xmsHErrorBlock errorBlock);
```

C++ interface:
```
Session createSession(const xmsBOOL transacted = xmsFALSE,
                      const xmsACKNOWLEDGE_MODE
                         acknowledgeMode = XMSC_AUTO_ACKNOWLEDGE);
```

Create a session.

Parameters:

connection (input)
: The handle for the connection.

transacted (input)
: The value must be `xmsFALSE`.

acknowledgeMode (input)
: Indicates how messages received by an application are acknowledged. The value must be `XMSC_AUTO_ACKNOWLEDGE`.

session (output)
: The handle for the session.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Exception Listener

C interface:
```
xmsRC xmsConnGetExceptionListener(xmsHConn connection,
                                  fpXMS_EXCEPTION_CALLBACK *lsr,
                                  xmsCONTEXT *context,
                                  xmsHErrorBlock errorBlock);
```

C++ interface:
```
ExceptionListener * getExceptionListener() const;
```

Get pointers to the exception listener function and context data that are registered with the connection.

For more information about using exception listeners and context data, see "Listeners and callbacks" on page 27.

Parameters:

connection (input)
The handle for the connection.

lsr (output)
A pointer to the exception listener function. If no exception listener function is registered with the connection, the call returns a null pointer.

context (output)
A pointer to the context data. If no exception listener function is registered with the connection, the call returns a null pointer.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Metadata

C interface:
```
xmsRC xmsConnGetMetaData(xmsHConn connection,
                         xmsHConnMetaData *connectionMetaData,
                         xmsHErrorBlock errorBlock);
```

C++ interface:
```
ConnectionMetaData getMetaData() const;
```

Get the metadata for the connection.

Parameters:

connection (input)
The handle for the connection.

connectionMetaData (output)
: The handle for the connection metadata.

errorBlock (input)
: The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Property

C interface:
```
xmsRC xmsConnGetProperty(xmsHConn connection,
                         xmsCHAR *propertyName,
                         xmsHProperty *property,
                         xmsHErrorBlock errorBlock);
```

C++ interface:
```
virtual Property getProperty(const String & propertyName) const;
```

Get a Property object for the property identified by name.

Parameters:

connection (input)
: The handle for the connection.

propertyName (input)
: The name of the property in the format of a null terminated string.

property (output)
: The handle for the Property object.

errorBlock (input)
: The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Exception Listener

C interface:
```
xmsRC xmsConnSetExceptionListener(xmsHConn connection,
                                  fpXMS_EXCEPTION_CALLBACK lsr,
                                  xmsCONTEXT context,
                                  xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID setExceptionListener(const ExceptionListener * lsr);
```

Register an exception listener function and context data with the connection.

For more information about using exception listeners and context data, see "Listeners and callbacks" on page 27.

Parameters:

connection (input)
: The handle for the connection.

lsr (input)
A pointer to the exception listener function. If an exception listener function is already registered with the connection, you can cancel the registration by specifying a null pointer instead.

context (input)
A pointer to the context data.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Property

C interface:

```
xmsRC xmsConnSetProperty(xmsHConn connection,
                         xmsHProperty property,
                         xmsHErrorBlock errorBlock);
```

C++ interface:

```
virtual xmsVOID setProperty(const Property & property);
```

Set the value of a property using a Property object.

Parameters:

connection (input)
The handle for the connection.

property (input)
The handle for the Property object.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Start Connection

C interface:

```
xmsRC xmsConnStart(xmsHConn connection,
                   xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsVOID start() const;
```

Start, or restart, the delivery of incoming messages for the connection. The call is ignored if the connection is already started.

Parameters:

connection (input)
The handle for the connection.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Stop Connection

C interface:
```
xmsRC xmsConnStop(xmsHConn connection,
                  xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID stop() const;
```

Stop temporarily the delivery of incoming messages for the connection. The call is ignored if the connection is already stopped.

Parameters:

connection (input)
The handle for the connection.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

ConnectionFactory

An application uses a connection factory to create a connection.

For a list of the XMS defined properties of a ConnectionFactory object, see "Properties of ConnectionFactory" on page 171.

Constructor

Create Connection Factory

C interface:
```
xmsRC xmsConnFactCreate(xmsHConnFact *factory,
                        xmsHErrorBlock errorBlock);
```

C++ interface:
```
ConnectionFactory();
```

Create a connection factory with the default properties.

Parameters:

factory (output)
The handle for the connection factory.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Methods

Create Connection (using the default user identity)

C interface:
```
xmsRC xmsConnFactCreateConnection(xmsHConnFact factory,
                                  xmsHConn *connection,
                                  xmsHErrorBlock errorBlock);
```

C++ interface:
```
Connection createConnection();
```

Create a connection using the default user identity. The connection factory properties XMSC_USER and XMSC_PASSWORD, if they are set, are used to authenticate the application. If these properties are not set, the connection is created without authenticating the client, provided the broker permits a connection without authentication.

The connection is created in stopped mode. No messages are delivered until Start Connection is called explicitly.

Parameters:

factory (input)
The handle for the connection factory.

connection (output)
The handle for the connection.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_SECURITY_EXCEPTION

Create Connection (using a specified user identity)

C interface:

```
xmsRC xmsConnFactCreateConnectionForUser(xmsHConnFact factory,
                                         xmsCHAR *userID,
                                         xmsCHAR *password,
                                         xmsHConn *connection,
                                         xmsHErrorBock errorBlock);
```

C++ interface:

```
Connection createConnection(const String & userID,
                            const String & password);
```

Create a connection using a specified user identity. The specified user identifier and password are used to authenticate the application. The connection factory properties XMSC_USER and XMSC_PASSWORD, if they are set, are ignored.

The connection is created in stopped mode. No messages are delivered until Start Connection is called explicitly.

Parameters:

factory (input)
The handle for the connection factory.

userID (input)
The user identifier to be used to authenticate the application. The user identifier is in the format of a null terminated string.

password (input)
The password to be used to authenticate the application. The password is in the format of a null terminated string.

connection (output)
The handle for the connection.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_SECURITY_EXCEPTION

Delete Connection Factory

C interface:

```
xmsRC xmsConnFactDispose(xmsHConnFact *factory,
                         xmsHErrorBlock errorBlock);
```

C++ interface:

```
virtual ~ConnectionFactory();
```

Delete the connection factory.

If an application tries to delete a connection factory that is already deleted, the call is ignored.

Parameters:

factory (input/output)
On input, the handle for the connection factory. On output, the call returns a null handle.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Boolean Property

C interface:

```
xmsRC xmsConnFactGetBooleanProperty(xmsHConnFact factory,
                                    xmsCHAR *propertyName,
                                    xmsBOOL *propertyValue,
                                    xmsHErrorBlock errorBlock);
```

C++ interface:
Inherited from the PropertyContext class

Get the value of the boolean property identified by name.

Parameters:

factory (input)
The handle for the connection factory.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue (output)
The value of the property.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Integer Property

C interface:

```
xmsRC xmsConnFactGetIntProperty(xmsHConnFact factory,
                                xmsCHAR *propertyName,
                                xmsINT *propertyValue,
                                xmsHErrorBlock errorBlock);
```

C++ interface:
Inherited from the PropertyContext class

Get the value of the integer property identified by name.

Parameters:

factory (input)
The handle for the connection factory.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue (output)
The value of the property.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Property

C interface:

```
xmsRC xmsConnFactGetProperty(xmsHConnFact factory,
                             xmsCHAR *propertyName,
                             xmsHProperty *property,
                             xmsHErrorBlock errorBlock);
```

C++ interface:

```
Property getProperty(const String & propertyName) const;
```

Get a Property object for the property identified by name.

Parameters:

factory (input)
The handle for the connection factory.

propertyName (input)
The name of the property in the format of a null terminated string.

property (output)
The handle for the Property object.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get String Property

C interface:

```
xmsRC xmsConnFactGetStringProperty(xmsHConnFact factory,
                                   xmsCHAR *propertyName,
                                   xmsCHAR *propertyValue,
                                   xmsSIZE length,
                                   xmsSIZE *actualLength,
                                   xmsHErrorBlock errorBlock);
```

C++ interface:
Inherited from the PropertyContext class

Get the value of the string property identified by name.

For more information about how to use this function in a C application, see “C functions that return a string or byte array by value” on page 28.

Parameters:

factory (input)
The handle for the connection factory.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue(output)
The buffer to contain the value of the property.

length (input)
The length of the buffer in bytes. If you specify a length of zero, the value of the property is not returned, but its length is returned in the actualLength parameter.

actualLength (output)
The length of the value of the property in bytes.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get String Property by Reference

C interface:
```
xmsRC xmsConnFactGetStringPropertyByRef(xmsHConnFact factory,
                                        xmsCHAR *propertyName,
                                        xmsCHAR **propertyValue,
                                        xmsSIZE *length,
                                        xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Get a pointer to the value of the string property identified by name.

For more information about how to use this function, see “C functions that return a string or byte array by reference” on page 29.

Parameters:

factory (input)
The handle for the connection factory.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue(output)
A pointer to the value of the property.

length (output)
The length of the value of the property in bytes.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Boolean Property

C interface:

```
xmsRC xmsConnFactSetBooleanProperty(xmsHConnFact factory,
                                    xmsCHAR *propertyName,
                                    xmsBOOL propertyValue,
                                    xmsHErrorBlock errorBlock);
```

C++ interface:
Inherited from the PropertyContext class

Set the value of the boolean property identified by name.

Parameters:

factory (input)
The handle for the connection factory.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue (output)
The value of the property.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Integer Property

C interface:

```
xmsRC xmsConnFactSetIntProperty(xmsHConnFact factory,
                                xmsCHAR *propertyName,
                                xmsINT propertyValue,
                                xmsHErrorBlock errorBlock);
```

C++ interface:
Inherited from the PropertyContext class

Set the value of the integer property identified by name.

Parameters:

factory (input)
The handle for the connection factory.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue (output)
The value of the property.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Property

C interface:
```
xmsRC xmsConnFactSetProperty(xmsHConnFact factory,
                             xmsHProperty property,
                             xmsHErrorBlock errorBlock);
```

C++ interface:
```
virtual xmsVOID setProperty(const Property & property);
```

Set the value of a property using a Property object.

Parameters:

factory (input)
The handle for the connection factory.

property (input)
The handle for the Property object.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set String Property

C interface:
```
xmsRC xmsConnFactSetStringProperty(xmsHConnFact factory,
                                   xmsCHAR *propertyName,
                                   xmsCHAR *propertyValue,
                                   xmsSIZE length,
                                   xmsHErrorBlock errorBlock);
```

C++ interface:
Inherited from the PropertyContext class

Set the value of the string property identified by name.

Parameters:

factory (input)
The handle for the connection factory.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue(input)
The value of the property as a character array.

length (input)
The length of the value of the property in bytes.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

ConnectionMetaData

A ConnectionMetaData object provides information about a connection.

For a list of the XMS defined properties of a ConnectionMetaData object, see "Properties of ConnectionMetaData" on page 173.

Methods

Delete Connection Metadata

C interface:

```
xmsRC xmsConnMetaDataDispose(xmsHConnMetaData *connectionMetaData,
                             xmsHErrorBlock errorBlock);
```

C++ interface:

```
virtual ~ConnectionMetaData();
```

Delete the metadata for the connection.

If an application tries to delete connection metadata that is already deleted, the call is ignored.

Parameters:

connectionMetaData (input/output)
On input, the handle for the connection metadata. On output, the call returns a null handle.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Integer Property

C interface:

```
xmsRC xmsConnMetaDataGetIntProperty(xmsHConnMetaData connectionMetaData,
                                    xmsCHAR *propertyName,
                                    xmsINT *propertyValue,
                                    xmsHErrorBlock errorBlock);
```

C++ interface:
Inherited from the PropertyContext class

Get the value of the integer property identified by name.

Parameters:

connectionMetaData (input)
The handle for the connection metadata.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue (output)
The value of the property.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Property

C interface:
```
xmsRC xmsConnMetaDataGetProperty(xmsHConnMetaData connectionMetaData,
                                 xmsCHAR *propertyName,
                                 xmsHProperty *property,
                                 xmsHErrorBlock errorBlock);
```

C++ interface:
```
virtual Property getProperty(const String & propertyName) const;
```

Get a Property object for the property identified by name.

Parameters:

connectionMetaData (input)
The handle for the connection metadata.

propertyName (input)
The name of the property in the format of a null terminated string.

property (output)
The handle for the Property object.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get String Property

C interface:
```
xmsRC xmsConnMetaDataGetStringProperty(xmsHConnMetaData connectionMetaData,
                                       xmsCHAR *propertyName,
                                       xmsCHAR *propertyValue,
                                       xmsSIZE length,
                                       xmsSIZE *actualLength,
                                       xmsHErrorBlock errorBlock);
```

C++ interface:
Inherited from the PropertyContext class

Get the value of the string property identified by name.

For more information about how to use this function in a C application, see “C functions that return a string or byte array by value” on page 28.

Parameters:

connectionMetaData (input)
The handle for the connection metadata.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue(output)
The buffer to contain the value of the property.

length (input)
The length of the buffer in bytes. If you specify a length of zero, the value of the property is not returned, but its length is returned in the actualLength parameter.

actualLength (output)
The length of the value of the property in bytes.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get String Property by Reference

C interface:

```
xmsRC
xmsConnMetaDataGetStringPropertyByRef(xmsHConnMetaData connectionMetaData,
                                      xmsCHAR *propertyName,
                                      xmsCHAR **propertyValue,
                                      xmsSIZE *length,
                                      xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Get a pointer to the value of the string property identified by name.

For more information about how to use this function, see "C functions that return a string or byte array by reference" on page 29.

Parameters:

connectionMetaData (input)
The handle for the connection metadata.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue(output)
A pointer to the value of the property.

length (output)
The length of the value of the property in bytes.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Destination

A destination is where an application sends messages, or it is a source from which an application receives messages, or both.

For a list of the XMS defined properties of a Destination object, see “Properties of Destination” on page 173.

Constructor

Create Destination (using a URI)

C interface:

```
xmsRC xmsDestCreate(xmsCHAR *URI,
                    xmsHDest *destination,
                    xmsHErrorBlock errorBlock);
```

C++ interface:

```
Destination(const String & URI);
```

Create a destination using the specified uniform resource identifier (URI). Properties of the destination that are not specified by the uniform resource identifier take the default values.

Parameters:

URI (input)
A uniform resource identifier in the format of a null terminated string.

destination (output)
The handle for the destination.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Create Destination (specifying a type and name)

C interface:

```
xmsRC xmsDestCreateByType(xmsDESTINATION_TYPE destinationType,
                          xmsCHAR *destinationName,
                          xmsHDest *destination,
                          xmsHErrorBlock errorBlock);
```

C++ interface:

```
Destination(const xmsDESTINATION_TYPE destinationType,
            const String & destinationName);
```

Create a destination using the specified destination type and name.

Parameters:

destinationType (input)
The type of the destination. The value must be `XMSC_TOPIC`.

destinationName (input)
The name of the destination in the format of a null terminated string.

destination (output)
The handle for the destination.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Methods

Delete Destination

C interface:
```
xmsRC xmsDestDispose(xmsHDest *destination,
                     xmsHErrorBlock errorBlock);
```

C++ interface:
```
virtual ~Destination();
```

Delete the destination.

If an application tries to delete a destination that is already deleted, the call is ignored.

Parameters:

destination (input/output)
On input, the handle for the destination. On output, the call returns a null handle.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Destination Name

C interface:
```
xmsRC xmsDestGetName(xmsHDest destination,
                     xmsCHAR *destinationName,
                     xmsSIZE length,
                     xmsSIZE *actualLength,
                     xmsHErrorBlock errorBlock);
```

C++ interface:
```
String getDestinationName() const;
```

Get the name of the destination.

For more information about how to use this function in a C application, see “C functions that return a string or byte array by value” on page 28.

Parameters:

destination (input)
The handle for the destination.

destinationName (output)
The buffer to contain the name of the destination.

length (input)
: The length of the buffer in bytes. If you specify a length of zero, the name of the destination is not returned, but its length is returned in the actualLength parameter.

actualLength (output)
: The length of the name of the destination in bytes.

errorBlock (input)
: The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Destination Name as URI

C interface:
```
xmsRC xmsDestToString(xmsHDest destination,
                      xmsCHAR *destinationName,
                      xmsSIZE length,
                      xmsSIZE *actualLength,
                      xmsHErrorBlock errorBlock);
```

C++ interface:
```
String toString() const;
```

Get the name of the destination in the format of a uniform resource identifier (URI).

For more information about how to use this function in a C application, see "C functions that return a string or byte array by value" on page 28.

Parameters:

destination (input)
: The handle for the destination.

destinationName (output)
: The buffer to contain the uniform resource identifier.

length (input)
: The length of the buffer in bytes. If you specify a length of zero, the URI is not returned, but its length is returned in the actualLength parameter.

actualLength (output)
: The length of the URI in bytes.

errorBlock (input)
: The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Destination Type

C interface:
```
xmsRC xmsDestGetTypeId(xmsHDest destination,
                       xmsDESTINATION_TYPE *destinationType,
                       xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsDESTINATION_TYPE GetType();
```

Get the type of the destination.

Parameters:

destination (input)
The handle for the destination.

destinationType (output)
The type of the destination. The value is always `XMSC_TOPIC`.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Integer Property

C interface:

```
xmsRC xmsDestGetIntProperty(xmsHDest destination,
                            xmsCHAR *propertyName,
                            xmsINT *propertyValue,
                            xmsHErrorBlock errorBlock);
```

C++ interface:
Inherited from the PropertyContext class

Get the value of the integer property identified by name.

Parameters:

destination (input)
The handle for the destination.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue (output)
The value of the property.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Property

C interface:

```
xmsRC xmsDestGetProperty(xmsHDest destination,
                         xmsCHAR *propertyName,
                         xmsHProperty *property,
                         xmsHErrorBlock errorBlock);
```

C++ interface:

```
virtual Property getProperty(const String & propertyName) const;
```

Get a Property object for the property identified by name.

Parameters:

destination (input)
The handle for the destination.

propertyName (input)
The name of the property in the format of a null terminated string.

property (output)
The handle for the Property object.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get String Property

C interface:

```
xmsRC xmsDestGetStringProperty(xmsHDest destination,
                               xmsCHAR *propertyName,
                               xmsCHAR *propertyValue,
                               xmsSIZE length,
                               xmsSIZE *actualLength,
                               xmsHErrorBlock errorBlock);
```

C++ interface:
Inherited from the PropertyContext class

Get the value of the string property identified by name.

For more information about how to use this function in a C application, see "C functions that return a string or byte array by value" on page 28.

Parameters:

destination (input)
The handle for the destination.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue(output)
The buffer to contain the value of the property.

length (input)
The length of the buffer in bytes. If you specify a length of zero, the value of the property is not returned, but its length is returned in the actualLength parameter.

actualLength (output)
The length of the value of the property in bytes.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get String Property by Reference

C interface:

```
xmsRC xmsDestGetStringPropertyByRef(xmsHDest destination,
                                    xmsCHAR *propertyName,
                                    xmsCHAR **propertyValue,
                                    xmsSIZE *length,
                                    xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Get a pointer to the value of the string property identified by name.

For more information about how to use this function, see "C functions that return a string or byte array by reference" on page 29.

Parameters:

destination (input)
The handle for the destination.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue(output)
A pointer to the value of the property.

length (output)
The length of the value of the property in bytes.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Integer Property

C interface:

```
xmsRC xmsDestSetIntProperty(xmsHDest destination,
                            xmsCHAR *propertyName,
                            xmsINT propertyValue,
                            xmsHErrorBlock errorBlock);
```

C++ interface:
Inherited from the PropertyContext class

Set the value of the integer property identified by name.

Parameters:

destination (input)
The handle for the destination.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue (output)
The value of the property.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Property

C interface:
```
xmsRC xmsDestSetProperty(xmsHDest destination,
                         xmsHProperty property,
                         xmsHErrorBlock errorBlock);
```

C++ interface:
```
virtual xmsVOID setProperty(const Property & property);
```

Set the value of a property using a Property object.

Parameters:

destination (input)
The handle for the destination.

property (input)
The handle for the Property object.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set String Property

C interface:
```
xmsRC xmsDestSetStringProperty(xmsHDest destination,
                               xmsCHAR *propertyName,
                               xmsCHAR *propertyValue,
                               xmsSIZE length,
                               xmsHErrorBlock errorBlock);
```

C++ interface:
Inherited from the PropertyContext class

Set the value of the string property identified by name.

Parameters:

destination (input)
The handle for the destination.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue (input)
The value of the property as a character array.

length (input)
The length of the value of the property in bytes.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

ErrorBlock

If a C function call fails, XMS can store information in an error block about why the call failed. For more information about the error block and its contents, see "The error block" on page 30.

Only the C interface uses this class.

Methods

Clear Error Block

C interface:
```
xmsRC xmsErrorClear(xmsHErrorBlock errorBlock);
```

Clear the contents of the error block.

XMS automatically clears the contents of an error block before filling the block with information about why a C function call has failed.

Parameters:

errorBlock (input)
The handle for the error block.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Create Error Block

C interface:
```
xmsRC xmsErrorCreate(xmsHErrorBlock *errorBlock);
```

Create an error block.

Parameters:

errorBlock (output)
The handle for the error block.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Delete Error Block

C interface:
```
xmsRC xmsErrorDispose(xmsHErrorBlock *errorBlock);
```

Delete an error block and any linked error blocks.

If an application tries to delete an error block that is already deleted, the call is ignored.

Parameters:

errorBlock (input/output)
On input, the handle for the error block. On output the call returns a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Error Code

C interface:
```
xmsRC xmsErrorGetErrorCode(xmsHErrorBlock errorBlock,
                           xmsINT *errorCode);
```

Get the error code.

Parameters:

errorBlock (input)
The handle for the error block.

errorCode (output)
The error code.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Error Data

C interface:
```
xmsRC xmsErrorGetErrorData(xmsHErrorBlock errorBlock,
                           xmsCHAR *buffer,
                           xmsSIZE bufferLength,
                           xmsSIZE *actualLength);
```

Get the error data.

Parameters:

errorBlock (input)
The handle for the error block.

buffer (output)
The buffer to contain the error data.

bufferLength (input)
The length of the buffer in bytes. If you specify a length of zero, the error data is not returned, but its length is returned in the actualLength parameter.

actualLength (output)
The length of the error data in bytes.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Error Module

C interface:
```
xmsRC xmsErrorGetModule(xmsHErrorBlock errorBlock,
                        xmsMODULE_TYPE *errorModule);
```

Get the identifier of the XMS module where the error originated. This information might be useful to your IBM Support Center if an unexpected error occurs.

Parameters:

errorBlock (input)
The handle for the error block.

errorModule (output)
The identifier of the XMS module.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Error String

C interface:
```
xmsRC xmsErrorGetErrorString(xmsHErrorBlock errorBlock,
                             xmsCHAR *buffer,
                             xmsSIZE bufferLength,
                             xmsSIZE *actualLength);
```

Get the error string.

Parameters:

errorBlock (input)
The handle for the error block.

buffer (output)
The buffer to contain the error string.

bufferLength (input)
The length of the buffer in bytes. If you specify a length of zero, the error string is not returned, but its length is returned in the actualLength parameter.

actualLength (output)
The length of the error string in bytes.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Exception Code

C interface:

```
xmsRC xmsErrorGetJMSException(xmsHErrorBlock errorBlock,
                              xmsJMSEXP_TYPE *exceptionCode);
```

Get the exception code.

Parameters:

errorBlock (input)
The handle for the error block.

exceptionCode (output)
The exception code.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Linked Error

C interface:

```
xmsRC xmsErrorGetLinkedError(xmsHErrorBlock errorBlock,
                             xmsHErrorBlock *linkedError);
```

Get the handle for the next error block in the chain of error blocks.

Parameters:

errorBlock (input)
The handle for the error block.

linkedError (output)
The handle for the next error block in the chain. The call returns a null handle if there are no more error blocks in the chain.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Exception

If a call of a C++ method fails, XMS creates an Exception object, which encapsulates information about why the call failed.

Only the C++ interface uses this class.

Methods

Delete Exception

C++ interface:
```
virtual ~Exception();
```

Delete the exception and any linked exceptions.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Error Code

C++ interface:
```
xmsINT getErrorCode() const;
```

Get the error code.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Error Data

C++ interface:
```
String getErrorData() const;
```

Get the free format data that provides additional information about the error.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Error String

C++ interface:
```
String getErrorString() const;
```

Get the string of characters that describes the error. The characters in the string are the same as those in the named constant that represents the error code.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Exception Code

C++ interface:

```
xmsJMSEXP_TYPE getJMSException() const;
```

Get the exception code.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Linked Exception

C++ interface:

```
Exception getLinkedException() const;
```

Get the next exception in the chain of exceptions. The call returns a null Exception object if there are no more exceptions in the chain.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

ExceptionListener

An application uses an exception listener to be notified asynchronously of a problem with a connection.

If an application uses a connection only to consume messages asynchronously, and for no other purpose, then the only way the application can learn about a problem with the connection is by using an exception listener.

Methods

On Exception

C interface:

```
xmsVOID onException(xmsCONTEXT context,
                    xmsHErrorBlock errorBlock);
```

C++ interface:

```
virtual xmsVOID onException(const Exception * exception);
```

Notify the application of a problem with a connection.

For a C application, onException() is the exception listener function that is registered with the connection. For a C++ application, onException() is a method of the exception listener that is registered with the connection.

For more information about using exception listeners, see “Listeners and callbacks” on page 27.

Parameters:

context (input)
A pointer to the context data that is registered with the connection.

errorBlock (input)
The handle for an error block created by the connection.

Iterator

An iterator encapsulates a list of Property objects and a cursor that maintains the current position in the list. When an iterator is created, the cursor is positioned before the first Property object.

An application can use an iterator to retrieve each Property object in turn.

Methods

Check for More Properties

C interface:

```
xmsRC xmsIteratorHasNext(xmsHIterator iterator,
                         xmsBOOL *moreProperties,
                         xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsBOOL xmsIterator::hasNext();
```

Check whether there are any more Property objects beyond the current position of the cursor. The call does not move the cursor.

Parameters:

iterator (input)
The handle for the iterator.

moreProperties (output)
If the value is `xmsTRUE`, more Property objects are available for retrieval beyond the current position of the cursor. If the value is `xmsFALSE`, no more Property objects are available for retrieval beyond the current position of the cursor.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Session

Exceptions:
XMS_X_GENERAL_EXCEPTION

Delete Iterator

C interface:

```
xmsRC xmsIteratorDispose(xmsHIterator *iterator,
                         xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsVOID xmsIterator::~xmsIterator();
```

Delete the iterator.

If an application tries to delete an iterator that is already deleted, the call is ignored.

Parameters:

iterator (input/output)
On input, the handle for the iterator. On output, the call returns a null handle.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Session

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Next Property

C interface:

```
xmsRC xmsIteratorGetNextProperty(xmsHIterator iterator,
                                 xmsHProperty *property,
                                 xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsProperty xmsIterator::getNextProperty();
```

Move the cursor to the next Property object and get the Property object at the new position of the cursor.

Parameters:

iterator (input)
The handle for the iterator.

property (output)
The handle for the Property object.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Session

Exceptions:
XMS_X_GENERAL_EXCEPTION

Reset Iterator

C interface:

```
xmsRC xmsIteratorReset(xmsHIterator iterator,
                       xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsVOID xmsIterator::reset();
```

Move the cursor back to a position before the first Property object.

Parameters:

iterator (input)
The handle for the iterator.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Session

Exceptions:
XMS_X_GENERAL_EXCEPTION

MapMessage

A map message is a message whose body comprises a set of name-value pairs.

Methods

Check Name-Value Pair Exists

C interface:
```
xmsRC xmsMapMsgItemExists(xmsHMsg message,
                          xmsCHAR *name,
                          xmsBOOL *pairExists,
                          xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsBOOL itemExists(const String & name) const;
```

Check whether the body of the map message contains a name-value pair with the specified name.

Parameters:

message (input)
The handle for the message.

name (input)
The name of the name-value pair in the format of a null terminated string.

pairExists (output)
If the value is `xmsTRUE`, the body of the map message contains a name-value pair with the specified name. If the value is `xmsFALSE`, the body of the map message does not contain a name-value pair with the specified name.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Boolean Value

C interface:
```
xmsRC xmsMapMsgGetBoolean(xmsHMsg message,
                          xmsCHAR *name,
                          xmsBOOL *value,
                          xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsBOOL getBoolean(const String & name) const;
```

Get the boolean value identified by name from the body of the map message.

Parameters:

message (input)
The handle for the message.

name (input)
The name that identifies the boolean value. The name is in the format of a null terminated string.

value (output)
: The boolean value that is returned.

errorBlock (input)
: The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Byte

C interface:
```
xmsRC xmsMapMsgGetByte(xmsHMsg message,
                       xmsCHAR *name,
                       xmsBYTE *value,
                       xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsBYTE getByte(const String & name) const;
```

Get the byte identified by name from the body of the map message.

Parameters:

message (input)
: The handle for the message.

name (input)
: The name that identifies the byte. The name is in the format of a null terminated string.

value (output)
: The byte that is returned. No data conversion is performed on the byte.

errorBlock (input)
: The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Bytes

C interface:
```
xmsRC xmsMapMsgGetBytes(xmsHMsg message,
                        xmsCHAR *name,
                        xmsBYTE *buffer,
                        xmsSIZE bufferLength,
                        xmsSIZE *actualLength,
                        xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsINT getBytes(const String & name,
                xmsBYTE *buffer,
                const xmsSIZE bufferLength,
                xmsSIZE *actualLength) const;
```

Get the array of bytes identified by name from the body of the map message.

For more information about how to use this function in a C application, see "C functions that return a string or byte array by value" on page 28.

Parameters:

message (input)
The handle for the message.

name (input)
The name that identifies the array of bytes. The name is in the format of a null terminated string.

buffer (output)
The buffer to contain the array of bytes. No data conversion is performed on the bytes that are returned.

bufferLength (input)
The length of the buffer in bytes. If you specify a length of zero, the array of bytes is not returned, but its length is returned in the actualLength parameter.

actualLength (output)
The number of bytes in the array.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Bytes by Reference

C interface:

```
xmsRC xmsMapMsgGetBytesByRef(xmsHMsg message,
                             xmsCHAR *name,
                             xmsBYTE **array,
                             xmsSIZE *length,
                             xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Get a pointer to an array of bytes in the body of the map message and get the length of the array. The array of bytes is identified by name.

For more information about how to use this function, see “C functions that return a string or byte array by reference” on page 29.

Parameters:

message (input)
The handle for the message.

name (input)
The name that identifies the array of bytes. The name is in the format of a null terminated string.

array (output)
A pointer to the array of bytes.

length (output)
The number of bytes in the array.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Character

C interface:
```
xmsRC xmsMapMsgGetChar(xmsHMsg message,
                       xmsCHAR *name,
                       xmsCHAR16 *value,
                       xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsCHAR16 getChar(const String & name) const;
```

Get the character identified by name from the body of the map message.

Parameters:

message (input)
The handle for the message.

name (input)
The name that identifies the character. The name is in the format of a null terminated string.

value (output)
The character that is returned.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Double Precision Floating Point Number

C interface:
```
xmsRC xmsMapMsgGetDouble(xmsHMsg message,
                         xmsCHAR *name,
                         xmsDOUBLE *value,
                         xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsDOUBLE getDouble(const String & name) const;
```

Get the double precision floating point number identified by name from the body of the map message.

Parameters:

message (input)
The handle for the message.

name (input)
The name that identifies the double precision floating point number. The name is in the format of a null terminated string.

value (output)
The double precision floating point number that is returned.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Floating Point Number

C interface:
```
xmsRC xmsMapMsgGetFloat(xmsHMsg message,
                        xmsCHAR *name,
                        xmsFLOAT *value,
                        xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsFLOAT getFloat(const String & name) const;
```

Get the floating point number identified by name from the body of the map message.

Parameters:

message (input)
The handle for the message.

name (input)
The name that identifies the floating point number. The name is in the format of a null terminated string.

value (output)
The floating point number that is returned.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Integer

C interface:
```
xmsRC xmsMapMsgGetInt(xmsHMsg message,
                      xmsCHAR *name,
                      xmsINT *value,
                      xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsINT getInt(const String & name) const;
```

Get the integer identified by name from the body of the map message.

Parameters:

message (input)
The handle for the message.

name (input)
The name that identifies the integer. The name is in the format of a null terminated string.

value (output)
: The integer that is returned.

errorBlock (input)
: The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Long Integer

C interface:

```
xmsRC xmsMapMsgGetLong(xmsHMsg message,
                       xmsCHAR *name,
                       xmsLONG *value,
                       xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsLONG getLong(const String & name) const;
```

Get the long integer identified by name from the body of the map message.

Parameters:

message (input)
: The handle for the message.

name (input)
: The name that identifies the long integer. The name is in the format of a null terminated string.

value (output)
: The long integer that is returned.

errorBlock (input)
: The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Name-Value Pairs

C interface:

```
xmsRC xmsMapMsgGetMapNames(xmsHMsg message,
                           xmsHIterator *iterator,
                           xmsHErrorBlock errorBlock);
```

C++ interface:

```
Iterator getMapNames() const;
```

Get a list of the name-value pairs in the body of the map message.

The call returns a handle for an iterator, which the application can then use to access each of the name-value pairs in turn. The iterator encapsulates a list of Property objects, where each Property object encapsulates a name-value pair.

Note: The equivalent JMS method performs a slightly different function. The JMS method returns an enumeration of only the names, not the values, in the body of the map message.

Parameters:

message (input)
The handle for the message.

iterator (output)
The handle for the iterator.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Object

C interface:
```
xmsRC xmsMapMsgGetObject(xmsHMsg message,
                         xmsCHAR *name,
                         xmsBYTE *buffer,
                         xmsSIZE bufferLength,
                         xmsSIZE *actualLength,
                         xmsOBJECT_TYPE *objectType,
                         xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsOBJECT_TYPE getObject(const String & name,
                         xmsBYTE *buffer,
                         const xmsSIZE bufferLength,
                         xmsSIZE *actualLength,
                         xmsOBJECT_TYPE *objectType) const;
```

Get the value of a name-value pair, and its type, from the body of the map message. The name-value pair is identified by name.

Parameters:

message (input)
The handle for the message.

name (input)
The name of the name-value pair in the format of a null terminated string.

buffer (output)
The buffer to contain the value, which is returned as an array of bytes. If the type of the value is XMS_OBJECT_TYPE_STRING and data conversion is required, this is the value after conversion. No data conversion is performed on a value of any other type.

bufferLength (input)
The length of the buffer in bytes. If you specify a length of zero, the object is not returned, but its length is returned in the actualLength parameter.

actualLength (output)
The length of the value in bytes. If data conversion is required, this is the length after conversion.

objectType (output)
The type of the value, which is one of the following types:

XMS_OBJECT_TYPE_BOOL

XMS_OBJECT_TYPE_BYTE
XMS_OBJECT_TYPE_CHAR
XMS_OBJECT_TYPE_DOUBLE
XMS_OBJECT_TYPE_FLOAT
XMS_OBJECT_TYPE_INT
XMS_OBJECT_TYPE_LONG
XMS_OBJECT_TYPE_SHORT
XMS_OBJECT_TYPE_STRING

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Short Integer

C interface:
```
xmsRC xmsMapMsgGetShort(xmsHMsg message,
                        xmsCHAR *name,
                        xmsSHORT *value,
                        xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsSHORT getShort(const String & name) const;
```

Get the short integer identified by name from the body of the map message.

Parameters:

message (input)
The handle for the message.

name (input)
The name that identifies the short integer. The name is in the format of a null terminated string.

value (output)
The short integer that is returned.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get String

C interface:
```
xmsRC xmsMapMsgGetString(xmsHMsg message,
                         xmsCHAR *name,
                         xmsCHAR *buffer,
                         xmsSIZE bufferLength,
                         xmsSIZE *actualLength,
                         xmsHErrorBlock errorBlock);
```

C++ interface:
```
String getString(const String & name) const;
```

Get the string identified by name from the body of the map message.

Parameters:

message (input)
The handle for the message.

name (input)
The name that identifies the string. The name is in the format of a null terminated string.

buffer (output)
The buffer to contain the string. If data conversion is required, this is the string after conversion.

bufferLength (input)
The length of the buffer in bytes. If you specify a length of zero, the string is not returned, but its length is returned in the actualLength parameter.

actualLength (output)
The length of the string in bytes. If data conversion is required, this is the length of the string after conversion.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get String by Reference

C interface:

```
xmsRC xmsMapMsgGetStringByRef(xmsHMsg message,
                              xmsCHAR *name,
                              xmsCHAR **string,
                              xmsSIZE *length,
                              xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Get a pointer to the string identified by name and get the length of the string.

For more information about how to use this function, see “C functions that return a string or byte array by reference” on page 29.

Parameters:

message (input)
The handle for the message.

name (input)
The name that identifies the string. The name is in the format of a null terminated string.

string (output)
A pointer to the string. If data conversion is required, this is the string after conversion.

length (output)
: The length of the string in bytes. If data conversion is required, this is the length after conversion.

errorBlock (input)
: The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Boolean Value

C interface:

```
xmsRC xmsMapMsgSetBoolean(xmsHMsg message,
                          xmsCHAR *name,
                          xmsBOOL value,
                          xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsVOID setBoolean(const String & name,
                   const xmsBOOL value);
```

Set a boolean value in the body of the map message.

Parameters:

message (input)
: The handle for the message.

name (input)
: The name to identify the boolean value in the body of the map message. The name is in the format of a null terminated string.

value (input)
: The boolean value to be set.

errorBlock (input)
: The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Byte

C interface:

```
xmsRC xmsMapMsgSetByte(xmsHMsg message,
                       xmsCHAR *name,
                       xmsBYTE value,
                       xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsVOID setByte(const String & name,
                const xmsBYTE value);
```

Set a byte in the body of the map message.

Parameters:

message (input)
: The handle for the message.

name (input)
The name to identify the byte in the body of the map message. The name is in the format of a null terminated string.

value (input)
The byte to be set.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Bytes

C interface:
```
xmsRC xmsMapMsgSetBytes(xmsHMsg message,
                        xmsCHAR *name,
                        xmsBYTE *value,
                        xmsSIZE length,
                        xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID setBytes(const String & name,
                 const xmsBYTE *value,
                 const xmsSIZE length);
```

Set an array of bytes in the body of the map message.

Parameters:

message (input)
The handle for the message.

name (input)
The name to identify the array of bytes in the body of the map message. The name is in the format of a null terminated string.

value (input)
The array of bytes to be set.

length (input)
The number of bytes in the array.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Character

C interface:
```
xmsRC xmsMapMsgSetChar(xmsHMsg message,
                       xmsCHAR *name,
                       xmsCHAR16 value,
                       xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID setChar(const String & name,
                const xmsCHAR16 value);
```

Set a 2-byte character in the body of the map message.

Parameters:

message (input)
The handle for the message.

name (input)
The name to identify the character in the body of the map message. The name is in the format of a null terminated string.

value (input)
The character to be set.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Double Precision Floating Point Number

C interface:

```
xmsRC xmsMapMsgSetDouble(xmsHMsg message,
                         xmsCHAR *name,
                         xmsDOUBLE value,
                         xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsVOID setDouble(const String & name,
                  const xmsDOUBLE value);
```

Set a double precision floating point number in the body of the map message.

Parameters:

message (input)
The handle for the message.

name (input)
The name to identify the double precision floating point number in the body of the map message. The name is in the format of a null terminated string.

value (input)
The double precision floating point number to be set.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Floating Point Number

C interface:

```
xmsRC xmsMapMsgSetFloat (xmsHMsg message,
                         xmsCHAR *name,
                         xmsFLOAT value,
                         xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID setFloat(const String & name,
                 const xmsFLOAT value);
```

Set a floating point number in the body of the map message.

Parameters:

message (input)
The handle for the message.

name (input)
The name to identify the floating point number in the body of the map message. The name is in the format of a null terminated string.

value (input)
The floating point number to be set.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Integer

C interface:
```
xmsRC xmsMapMsgSetInt(xmsHMsg message,
                      xmsCHAR *name,
                      xmsINT value,
                      xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID setInt(const String & name,
               const xmsINT value);
```

Set an integer in the body of the map message.

Parameters:

message (input)
The handle for the message.

name (input)
The name to identify the integer in the body of the map message. The name is in the format of a null terminated string.

value (input)
The integer to be set.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Long Integer

C interface:
```
xmsRC xmsMapMsgSetLong (xmsHMsg message,
                        xmsCHAR *name,
                        xmsLONG value,
                        xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID setLong(const String & name,
                const xmsLONG value);
```

Set a long integer in the body of the map message.

Parameters:

message (input)
The handle for the message.

name (input)
The name to identify the long integer in the body of the map message. The name is in the format of a null terminated string.

value (input)
The long integer to be set.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Object

C interface:
```
xmsRC xmsMapMsgSetObject(xmsHMsg message,
                         xmsCHAR *name,
                         xmsBYTE *value,
                         xmsSIZE length,
                         xmsOBJECT_TYPE objectType
                         xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID setObject(const String & name,
                  const xmsBYTE *value,
                  const xmsSIZE length
                  const xmsOBJECT_TYPE objectType);
```

Set a value, with a specified type, in the body of the map message.

Parameters:

message (input)
The handle for the message.

name (input)
The name to identify the value in the body of the map message. The name is in the format of a null terminated string.

value (input)
The array of bytes representing the value to be set.

length (input)
The number of bytes in the array.

objectType (input)
The type of the value, which must be one of the following types:

XMS_OBJECT_TYPE_BOOL
XMS_OBJECT_TYPE_BYTE
XMS_OBJECT_TYPE_CHAR
XMS_OBJECT_TYPE_DOUBLE
XMS_OBJECT_TYPE_FLOAT
XMS_OBJECT_TYPE_INT
XMS_OBJECT_TYPE_LONG
XMS_OBJECT_TYPE_SHORT
XMS_OBJECT_TYPE_STRING

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Short Integer

C interface:
```
xmsRC xmsMapMsgSetShort(xmsHMsg message,
                        xmsCHAR *name,
                        xmsSHORT value,
                        xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID setShort(const String & name,
                 const xmsSHORT value);
```

Set a short integer in the body of the map message.

Parameters:

message (input)
The handle for the message.

name (input)
The name to identify the short integer in the body of the map message. The name is in the format of a null terminated string.

value (input)
The short integer to be set.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set String

C interface:
```
xmsRC xmsMapMsgSetString(xmsHMsg message,
                         xmsCHAR *name,
                         xmsCHAR *value,
                         xmsSIZE length,
                         xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID setString(const String & name,
                  const String value);
```

Set a string in the body of the map message.

Parameters:

message (input)
The handle for the message.

name (input)
The name to identify the string in the body of the map message. The name is in the format of a null terminated string.

value (input)
The character array representing the string to be set.

length (input)
The length of the string in bytes.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Message

A Message object represents a message that an application sends or receives.

For a list of the JMS message header fields in a Message object, see “Header fields in an XMS message” on page 31. For a list of the JMS defined properties of a Message object, see “JMS defined properties of a message” on page 33. For a list of the IBM defined properties of a Message object, see “IBM defined properties of a message” on page 33.

Methods

Check Property Exists

C interface:

```
xmsRC xmsMsgPropertyExists(xmsHMsg message,
                           xmsCHAR *propertyName,
                           xmsBOOL *valueExists,
                           xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsBOOL propertyExists(const String & propertyName) const;
```

Check whether the property identified by name has a value.

Parameters:

message (input)
The handle for the message.

propertyName (input)
The name of the property in the format of a null terminated string.

valueExists (output)
If the value is `xmsTRUE`, the property has a value. If the value is `xmsFALSE`, the property does not have a value.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Clear Body

C interface:

```
xmsRC xmsMsgClearBody(xmsHMsg message,
                      xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsVOID clearBody();
```

Clear the body of the message. The header fields and message properties are not cleared.

If you clear a message body that is read-only, the body becomes writable.

Parameters:

message (input)
The handle for the message.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Clear Properties

C interface:
```
xmsRC xmsMsgClearProperties(xmsHMsg message,
                            xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID clearProperties();
```

Clear the properties of the message. The header fields and the message body are not cleared.

If you clear message properties that are read-only, the properties become writable.

Parameters:

message (input)
The handle for the message.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Delete Message

C interface:
```
xmsRC xmsMsgDispose(xmsHMsg *message,
                    xmsHErrorBlock errorBlock);
```

C++ interface:
```
virtual ~Message();
```

Delete the message.

If an application tries to delete a message that is already deleted, the call is ignored.

Parameters:

message (input)
On input, the handle for the message. On output, the call returns a null handle.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Boolean Property

C interface:

```
xmsRC xmsMsgGetBooleanProperty(xmsHMsg message,
                               xmsCHAR *propertyName,
                               xmsBOOL *propertyValue,
                               xmsHErrorBlock errorBlock);
```

C++ interface:
Inherited from the PropertyContext class

Get the value of the boolean property identified by name.

Parameters:

message (input)
The handle for the message.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue (output)
The value of the property.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Byte Property

C interface:

```
xmsRC xmsMsgGetByteProperty(xmsHMsg message,
                            xmsCHAR *propertyName,
                            xmsBYTE *propertyValue,
                            xmsHErrorBlock errorBlock);
```

C++ interface:
Inherited from the PropertyContext class

Get the value of the byte property identified by name.

Parameters:

message (input)
The handle for the message.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue (output)
The value of the property.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Double Precision Floating Point Property

C interface:

```
xmsRC xmsMsgGetDoubleProperty(xmsHMsg message,
                              xmsCHAR *propertyName,
                              xmsDOUBLE *propertyValue,
                              xmsHErrorBlock errorBlock);
```

C++ interface:
Inherited from the PropertyContext class

Get the value of the double precision floating point property identified by name.

Parameters:

message (input)
The handle for the message.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue (output)
The value of the property.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Floating Point Property

C interface:

```
xmsRC xmsMsgGetFloatProperty(xmsHMsg message,
                             xmsCHAR *propertyName,
                             xmsFLOAT *propertyValue,
                             xmsHErrorBlock errorBlock);
```

C++ interface:
Inherited from the PropertyContext class

Get the value of the floating point property identified by name.

Parameters:

message (input)
The handle for the message.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue (output)
The value of the property.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Integer Property

C interface:

```
xmsRC xmsMsgGetFloatProperty(xmsHMsg message,
                             xmsCHAR *propertyName,
                             xmsFLOAT *propertyValue,
                             xmsHErrorBlock errorBlock);
```

C++ interface:

Inherited from the PropertyContext class

Get the value of the integer property identified by name.

Parameters:

message (input)
The handle for the message.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue (output)
The value of the property.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

XMS_X_GENERAL_EXCEPTION

Get JMSCorrelationID

C interface:

```
xmsRC xmsMsgGetJMSCorrelationID(xmsHMsg message,
                                xmsCHAR *correlID,
                                xmsSIZE length,
                                xmsSIZE *actualLength,
                                xmsHErrorBlock errorBlock);
```

C++ interface:

```
String getJMSCorrelationID() const;
```

Get the correlation identifier of the message in the format of a string.

For more information about how to use this function in a C application, see "C functions that return a string or byte array by value" on page 28.

Parameters:

message (input)
The handle for the message.

correlID (output)
The buffer to contain the correlation identifier. If data conversion is required, this is the correlation identifier after conversion.

length (input)
The length of the buffer in bytes. If you specify a length of zero, the correlation identifier is not returned, but its length is returned in the actualLength parameter.

actualLength (output)
The length of the correlation identifier in bytes. If data conversion is required, this is the length after conversion.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get JMSDeliveryMode

C interface:
```
xmsRC xmsMsgGetJMSDeliveryMode(xmsHMsg message,
                               xmsDELIVERY_MODE *deliveryMode,
                               xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsDELIVERY_MODE getJMSDeliveryMode() const;
```

Get the delivery mode of the message.

Parameters:

message (input)
The handle for the message.

deliveryMode (output)
The delivery mode of the message, which is one of the following values:

`XMSC_NON_PERSISTENT`

`XMSC_PERSISTENT`

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get JMSDestination

C interface:
```
xmsRC xmsMsgGetJMSDestination(xmsHMsg message,
                              xmsHDest *destination,
                              xmsHErrorBlock errorBlock);
```

C++ interface:
```
Destination getJMSDestination() const;
```

Get the destination of the message. The destination is set by the Send call when the message is sent.

Parameters:

message (input)
The handle for the message.

destination (output)
The handle for the destination of the message.

For a newly created message that has not been sent, the call returns a null handle unless the sending application sets a destination by calling Set JMSDestination. For a message that has been received, the call returns a handle for the destination that was set by the Send call when the message was sent unless the receiving application changes the destination by calling Set JMSDestination.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get JMSExpiration

C interface:

```
xmsRC xmsMsgGetJMSExpiration(xmsHMsg message,
                             xmsLONG *expiration,
                             xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsLONG getJMSExpiration() const;
```

Get the expiration time of the message.

The expiration time is set by the Send call when the message is sent. Its value is calculated by adding the time to live, as specified by the sending application, to the time when the message is sent. The expiration time is expressed in milliseconds since 00:00:00 GMT on the 1 January 1970.

If the time to live is zero, the Send call sets the expiration time to zero to indicate that the message does not expire.

Parameters:

message (input)
The handle for the message.

expiration (output)
The expiration time of the message.

For a newly created message that has not been sent, the expiration time is undefined unless the sending application sets an expiration time by calling Set JMSExpiration. For a message that has been received, the call returns the expiration time that was set by the Send call when the message was sent unless the receiving application changes the expiration time by calling Set JMSExpiration.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get JMSMessageID

C interface:

```
xmsRC xmsMsgGetJMSMessageID(xmsHMsg message,
                            xmsCHAR *msgID,
                            xmsSIZE length,
                            xmsSIZE *actualLength,
                            xmsHErrorBlock errorBlock);
```

C++ interface:

```
String getJMSMessageID() const;
```

Get the message identifier of the message. The message identifier is set by the Send call when the message is sent.

For more information about how to use this function in a C application, see "C functions that return a string or byte array by value" on page 28.

Parameters:

message (input)
The handle for the message.

msgID (output)
The buffer to contain the message identifier. If data conversion is required, this is the message identifier after conversion.

For a newly created message that has not been sent, the call returns a null string unless the sending application sets a message identifier by calling Set JMSMessageID. For a message that has been received, the call returns the message identifier that was set by the Send call when the message was sent unless the receiving application changes the message identifier by calling Set JMSMessageID.

length (input)
The length of the buffer in bytes. If you specify a length of zero, the message identifier is not returned, but its length is returned in the actualLength parameter.

actualLength (output)
The length of the message identifier in bytes. If data conversion is required, this is the length after conversion.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get JMSPriority

C interface:

```
xmsRC xmsMsgGetJMSPriority(xmsHMsg message,
                           xmsINT *priority,
                           xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsINT getJMSPriority() const;
```

Get the priority of the message.

Parameters:

message (input)
The handle for the message.

priority (output)
The priority of the message. The value is an integer in the range `0`, the lowest priority, to `9`, the highest priority.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get JMSRedelivered

C interface:
```
xmsRC xmsMsgGetJMSRedelivered(xmsHMsg message,
                              xmsBOOL *redelivered,
                              xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsBOOL getJMSRedelivered() const;
```

Get an indication of whether the message is being re-delivered.

Parameters:

message (input)
The handle for the message.

redelivered (output)
If the value is `xmsTRUE`, it is likely, but not guaranteed, that the message was delivered earlier but its receipt was not acknowledged at that time. If the value is `xmsFALSE`, the message is not being re-delivered.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get JMSReplyTo

C interface:
```
xmsRC xmsMsgGetJMSReplyTo(xmsHMsg message,
                          xmsHDest *destination,
                          xmsHErrorBlock errorBlock);
```

C++ interface:
```
Destination getJMSReplyTo() const;
```

Get the destination where a reply to the message is to be sent.

Parameters:

message (input)
The handle for the message.

destination (output)
The handle for the destination where a reply to the message is to be sent.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get JMSTimestamp

C interface:

```
xmsRC xmsMsgGetJMSTimestamp(xmsHMsg message,
                            xmsLONG *timestamp,
                            xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsLONG getJMSTimestamp() const;
```

Get the time when the message was sent. The timestamp is set by the Send call when the message is sent and is expressed in milliseconds since 00:00:00 GMT on the 1 January 1970.

Parameters:

message (input)
The handle for the message.

timestamp (output)
The time when the message was sent.

For a newly created message that has not been sent, the timestamp is undefined unless the sending application sets a timestamp by calling Set JMSTimestamp. For a message that has been received, the call returns the timestamp that was set by the Send call when the message was sent unless the receiving application changes the timestamp by calling Set JMSTimestamp.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get JMSType

C interface:

```
xmsRC xmsMsgGetJMSType(xmsHMsg message,
                       xmsCHAR *type,
                       xmsSIZE length,
                       xmsSIZE *actualLength,
                       xmsHErrorBlock errorBlock);
```

C++ interface:

```
String getJMSType() const;
```

Get the type of the message.

For more information about how to use this function in a C application, see "C functions that return a string or byte array by value" on page 28.

Parameters:

message (input)
The handle for the message.

type (output)
The buffer to contain the type of the message. If data conversion is required, this is the type after conversion.

length (input)
The length of the buffer in bytes. If you specify a length of zero, the type of the message is not returned, but its length is returned in the actualLength parameter.

actualLength (output)
The length of the type of the message in bytes. If data conversion is required, this is the length after conversion.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Long Integer Property

C interface:

```
xmsRC xmsMsgGetLongProperty(xmsHMsg message,
                            xmsCHAR *propertyName,
                            xmsLONG *propertyValue,
                            xmsHErrorBlock errorBlock);
```

C++ interface:
Inherited from the PropertyContext class

Get the value of the long property identified by name.

Parameters:

message (input)
The handle for the message.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue (output)
The value of the property.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Properties

C interface:

```
xmsRC xmsMsgGetProperties(xmsHMsg message,
                          xmsHIterator *iterator,
                          xmsHErrorBlock errorBlock);
```

C++ interface:

```
Iterator getProperties() const;
```

Get a list of the properties of the message.

The call returns a handle for an iterator, which the application can then use to access each of the properties in turn.

Parameters:

message (input)
The handle for the message.

iterator (input)
The handle for the iterator.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Property

C interface:

```
xmsRC xmsMsgGetProperty(xmsHMsg message,
                        xmsCHAR *propertyName,
                        xmsHProperty *property,
                        xmsHErrorBlock errorBlock);
```

C++ interface:

```
virtual Property getProperty(String & propertyName) const;
```

Get a Property object for the property identified by name.

Parameters:

message (input)
The handle for the message.

propertyName (input)
The name of the property in the format of a null terminated string.

property (output)
The handle for the Property object.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Short Integer Property

C interface:

```
xmsRC xmsMsgGetShortProperty(xmsHMsg message,
                             xmsCHAR *propertyName,
                             xmsSHORT *propertyValue,
                             xmsHErrorBlock errorBlock);
```

C++ interface:
Inherited from the PropertyContext class

Get the value of the short property identified by name.

Parameters:

message (input)
The handle for the message.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue (output)
The value of the property.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get String Property

C interface:

```
xmsRC xmsMsgGetStringProperty(xmsHMsg message,
                              xmsCHAR *propertyName,
                              xmsCHAR *propertyValue,
                              xmsSIZE length,
                              xmsSIZE *actualLength,
                              xmsHErrorBlock errorBlock);
```

C++ interface:
Inherited from the PropertyContext class

Get the value of the string property identified by name.

For more information about how to use this function in a C application, see "C functions that return a string or byte array by value" on page 28.

Parameters:

message (input)
The handle for the message.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue (output)
The buffer to contain the value of the property. If data conversion is required, this is the value after conversion.

length (input)
: The length of the buffer in bytes. If you specify a length of zero, the value of the property is not returned, but its length is returned in the actualLength parameter.

actualLength (output)
: The length of the value of the property in bytes. If data conversion is required, this is the length after conversion.

errorBlock (input)
: The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get String Property by Reference

C interface:
```
xmsRC xmsMsgGetStringPropertyByRef(xmsHMsg message,
                                   xmsCHAR *propertyName,
                                   xmsCHAR **propertyValue,
                                   xmsSIZE *length,
                                   xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsCHAR * Message::getStringPropertyByRef(const xmsString & propertyName,
                                          xmsCHAR *propertyValue,
                                          xmsSIZE *length);
```

Get a pointer to the value of the string property identified by name.

For more information about how to use this function, see “C functions that return a string or byte array by reference” on page 29.

Parameters:

message (input)
: The handle for the message.

propertyName (input)
: The name of the property in the format of a null terminated string.

propertyValue (output)
: A pointer to the value of the property. If data conversion is required, this is the value after conversion.

length (output)
: The length of the value of the property in bytes. If data conversion is required, this is the length of the value after conversion.

errorBlock (input)
: The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Type

C interface:

```
xmsRC xmsMsgGetTypeId(xmsHMsg message,
                      xmsMESSAGE_TYPE *type,
                      xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Get the body type of the message.

For information about message body types, see "The body of an XMS message" on page 34.

Parameters:

message (input)
The handle for the message.

type (output)
The body type of the message, which is one of the following values:

```
XMSC_T_BYTES_MSG
XMSC_T_MAP_MSG
XMSC_T_MSG
```

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Boolean Property

C interface:

```
xmsRC xmsMsgSetBooleanProperty(xmsHMsg message,
                               xmsCHAR *propertyName,
                               xmsBOOL propertyValue,
                               xmsHErrorBlock errorBlock);
```

C++ interface:
Inherited from the PropertyContext class

Set the value of the boolean property identified by name.

Parameters:

message (input)
The handle for the message.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue (input)
The value of the property.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION

- XMS_X_MESSAGE_NOT_WRITABLE_EXCEPTION

Set Byte Property

C interface:

```
xmsRC xmsMsgSetByteProperty(xmsHMsg message,
                            xmsCHAR *propertyName,
                            xmsBYTE propertyValue,
                            xmsHErrorBlock errorBlock);
```

C++ interface:
Inherited from the PropertyContext class

Set the value of the byte property identified by name.

Parameters:

message (input)
The handle for the message.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue (input)
The value of the property.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_WRITABLE_EXCEPTION

Set Double Precision Floating Point Property

C interface:

```
xmsRC xmsMsgSetDoubleProperty(xmsHMsg message,
                              xmsCHAR *propertyName,
                              xmsDOUBLE propertyValue,
                              xmsHErrorBlock errorBlock);
```

C++ interface:
Inherited from the PropertyContext class

Set the value of the double precision floating point property identified by name.

Parameters:

message (input)
The handle for the message.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue (input)
The value of the property.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_WRITABLE_EXCEPTION

Set Floating Point Property

C interface:

```
xmsRC xmsMsgSetFloatProperty(xmsHMsg message,
                             xmsCHAR *propertyName,
                             xmsFLOAT propertyValue,
                             xmsHErrorBlock errorBlock);
```

C++ interface:
Inherited from the PropertyContext class

Set the value of the floating point property identified by name.

Parameters:

message (input)
The handle for the message.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue (input)
The value of the property.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_WRITABLE_EXCEPTION

Set Integer Property

C interface:

```
xmsRC xmsMsgSetIntProperty(xmsHMsg message,
                           xmsCHAR *propertyName,
                           xmsINT propertyValue,
                           xmsHErrorBlock errorBlock);
```

C++ interface:
Inherited from the PropertyContext class

Set the value of the integer property identified by name.

Parameters:

message (input)
The handle for the message.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue (input)
The value of the property.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_WRITABLE_EXCEPTION

Set JMSCorrelationID

C interface:

```
xmsRC xmsMsgSetJMSCorrelationID(xmsHMsg message,
                                xmsCHAR *correlID,
                                xmsSIZE length,
                                xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsVOID setJMSCorrelationID(const String correlID);
```

Set the correlation identifier of the message.

Parameters:

message (input)
The handle for the message.

correlID (input)
The correlation identifier as a character array.

length (input)
The length of the correlation identifier in bytes.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set JMSDeliveryMode

C interface:

```
xmsRC xmsMsgSetJMSDeliveryMode(xmsHMsg message,
                               xmsDELIVERY_MODE deliveryMode,
                               xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsVOID setJMSDeliveryMode(const xmsDELIVERY_MODE deliveryMode);
```

Set the delivery mode of the message.

A delivery mode set by this method before the message is sent is ignored and replaced by the Send call when the message is sent. However, you can use this method to change the delivery mode of a message that has been received.

Parameters:

message (input)
The handle for the message.

deliveryMode (input)
The delivery mode of the message. The value must be `XMSC_NON_PERSISTENT`.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set JMSDestination

C interface:
```
xmsRC xmsMsgSetJMSDestination(xmsHMsg message,
                              xmsHDest destination,
                              xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID setJMSDestination(const Destination & destination);
```

Set the destination of the message.

A destination set by this method before the message is sent is ignored and replaced by the Send call when the message is sent. However, you can use this method to change the destination of a message that has been received.

Parameters:

message (input)
The handle for the message.

destination (input)
The handle for the destination of the message.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set JMSExpiration

C interface:
```
xmsRC xmsMsgSetJMSExpiration(xmsHMsg message,
                             xmsLONG expiration,
                             xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID setJMSExpiration(const xmsLONG expiration);
```

Set the expiration time of the message.

An expiration time set by this method before the message is sent is ignored and replaced by the Send call when the message is sent. However, you can use this method to change the expiration time of a message that has been received.

Parameters:

message (input)
The handle for the message.

expiration (input)
The expiration time of the message expressed in milliseconds since 00:00:00 GMT on the 1 January 1970.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set JMSMessageID

C interface:
```
xmsRC xmsMsgSetJMSMessageID(xmsHMsg message,
                            xmsCHAR *msgID,
                            xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID setJMSMessageID(const String & msgID);
```

Set the message identifier of the message.

A message identifier set by this method before the message is sent is ignored and replaced by the Send call when the message is sent. However, you can use this method to change the message identifier of a message that has been received.

Parameters:

message (input)
The handle for the message.

msgID (input)
The message identifier in the format of a null terminated string.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set JMSPriority

C interface:
```
xmsRC xmsMsgSetJMSPriority(xmsHMsg message,
                           xmsINT priority,
                           xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID setJMSPriority(const xmsINT priority);
```

Set the priority of the message.

A priority set by this method before the message is sent is ignored and replaced by the Send call when the message is sent. However, you can use this method to change the priority of a message that has been received.

Parameters:

message (input)
The handle for the message.

priority (input)
The priority of the message. The value can be an integer in the range `0`, the lowest priority, to `9`, the highest priority.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set JMSRedelivered

C interface:
```
xmsRC xmsMsgSetJMSRedelivered(xmsHMsg message,
                              xmsBOOL redelivered,
                              xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID setJMSRedelivered(const xmsBOOL redelivered);
```

Indicate whether the message is being re-delivered.

An indication of re-delivery set by this method before the message is sent is ignored by the Send call when the message is sent, and is ignored and replaced by the Receive call when the message is received. However, you can use this method to change the indication for a message that has been received.

Parameters:

message (input)
The handle for the message.

redelivered (input)
The value `xmsTRUE` means that the message is being re-delivered. The value `xmsFALSE` means that the message is not being re-delivered.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set JMSReplyTo

C interface:
```
xmsRC xmsMsgSetJMSReplyTo(xmsHMsg message,
                          xmsHDest destination ,
                          xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID setJMSReplyTo(const Destination & destination);
```

Set the destination where a reply to the message is to be sent.

Parameters:

message (input)
The handle for the message.

destination (input)
The handle for the destination where a reply to the message is to be sent. A null handle means that no reply is expected.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set JMSTimestamp

C interface:
```
xmsRC xmsMsgSetJMSTimestamp(xmsHMsg message,
                            xmsLONG timestamp,
                            xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID setJMSTimestamp(const xmsLONG timestamp);
```

Set the time when the message is sent.

A timestamp set by this method before the message is sent is ignored and replaced by the Send call when the message is sent. However, you can use this method to change the timestamp of a message that has been received.

Parameters:

message (input)
The handle for the message.

timestamp (input)
The time when the message is sent expressed in milliseconds since 00:00:00 GMT on the 1 January 1970.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set JMSType

C interface:
```
xmsRC xmsMsgSetJMSType(xmsHMsg message,
                       xmsCHAR *type,
                       xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID setJMSType(const String & type);
```

Set the type of the message.

Parameters:

message (input)
The handle for the message.

type (input)
The type of the message in the format of a null terminated string.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Long Integer Property

C interface:
```
xmsRC xmsMsgSetLongProperty(xmsHMsg message,
                            xmsCHAR *propertyName,
                            xmsLONG propertyValue,
                            xmsHErrorBlock errorBlock);
```

C++ interface:
Inherited from the PropertyContext class

Set the value of the long integer property identified by name.

Parameters:

message (input)
The handle for the message.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue (input)
The value of the property.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_WRITABLE_EXCEPTION

Set Property

C interface:
```
xmsRC xmsMsgSetProperty(xmsHMsg message,
                        xmsHProperty property,
                        xmsHErrorBlock errorBlock);
```

C++ interface:
```
virtual xmsVOID setProperty(const Property & property);
```

Set the value of a property using a Property object.

Parameters:

message (input)
The handle for the message.

property (input)
The handle for the Property object.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_WRITABLE_EXCEPTION

Set Short Integer Property

C interface:

```
xmsRC xmsMsgSetShortProperty(xmsHMsg message,
                             xmsCHAR *propertyNname,
                             xmsSHORT propertyValue,
                             xmsHErrorBlock errorBlock);
```

C++ interface:
Inherited from the PropertyContext class

Set the value of the short integer property identified by name.

Parameters:

message (input)
The handle for the message.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue (input)
The value of the property.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_WRITABLE_EXCEPTION

Set String Property

C interface:

```
xmsRC xmsMsgSetStringProperty(xmsHMsg message,
                              xmsCHAR *propertyName,
                              xmsCHAR *propertyValue,
                              xmsSIZE length,
                              xmsHErrorBlock errorBlock);
```

C++ interface:
Inherited from the PropertyContext class

Set the value of the string property identified by name.

Parameters:

message (input)
The handle for the message.

propertyName (input)
The name of the property in the format of a null terminated string.

propertyValue (input)
The value of the property as a character array.

length (input)
The length of the value of the property in bytes.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_NOT_WRITABLE_EXCEPTION

MessageConsumer

An application uses a message consumer to receive messages sent to a destination.

For a list of the XMS defined properties of a MessageConsumer object, see “Properties of MessageConsumer” on page 174.

Methods

Close Message Consumer

C interface:
```
xmsRC xmsMsgConsumerClose(xmsHMsgConsumer *consumer,
                          xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID close();
```

Close the message consumer.

If an application tries to close a message consumer that is already closed, the call is ignored.

Parameters:

consumer (input/output)
On input, the handle for the message consumer. On output, the call returns a null handle.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Message Listener

C interface:
```
xmsRC xmsMsgConsumerGetMessageListener(xmsHMsgConsumer consumer,
                                       fpXMS_MESSAGE_CALLBACK *lsr,
                                       xmsCONTEXT *context,
                                       xmsHErrorBlock errorBlock);
```

C++ interface:
```
MessageListener * getMessageListener() const;
```

Get pointers to the message listener function and context data that are registered with the message consumer.

For more information about using message listeners and context data, see “Listeners and callbacks” on page 27.

Parameters:

consumer (input)
The handle for the message consumer.

lsr (output)
A pointer to the message listener function. If no message listener function is registered with the message consumer, the call returns a null pointer.

context (output)
A pointer to the context data. If no message listener function is registered with the connection, the call returns a null pointer.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Message Selector

C interface:
```
xmsRC xmsMsgConsumerGetMessageSelector(xmsHMsgConsumer consumer,
                                       xmsCHAR *messageSelector,
                                       xmsSIZE length,
                                       xmsSIZE *actualLength,
                                       xmsHErrorBlock errorBlock);
```

C++ interface:
```
String getMessageSelector() const;
```

Get the message selector for the message consumer.

For more information about how to use this function in a C application, see “C functions that return a string or byte array by value” on page 28.

Parameters:

consumer (input)
The handle for the message consumer.

messageSelector (output)
The buffer to contain the message selector expression.

length (input)
The length of the buffer in bytes. If you specify a length of zero, the message selector expression is not returned, but its length is returned in the actualLength parameter.

actualLength (output)
The length of the message selector expression in bytes.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Property

C interface:
```
xmsRC xmsMsgConsumerGetProperty(xmsHMsgConsumer consumer,
                                xmsCHAR *propertyName,
                                xmsHProperty *property,
                                xmsHErrorBlock errorBlock);
```

C++ interface:
```
virtual Property getProperty(const String & propertyName) const;
```

Get a Property object for the property identified by name.

Parameters:

consumer (input)
The handle for the message consumer.

propertyName (input)
The name of the property in the format of a null terminated string.

property (output)
The handle for the Property object.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Receive

C interface:

```
xmsRC xmsMsgConsumerReceive(xmsHMsgConsumer consumer,
                            xmsHMsg *message,
                            xmsHErrorBlock errorBlock);
```

C++ interface:

```
Message * receive() const;
```

Receive the next message for the message consumer. The call waits indefinitely for a message, or until the message consumer is closed.

Parameters:

consumer (input)
The handle for the message consumer.

message (output)
The handle for the message. If the message consumer is closed while the call is waiting for a message, the call returns a null handle.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Receive (with a wait interval)

C interface:

```
xmsRC xmsMsgConsumerReceiveWithWait(xmsHMsgConsumer consumer,
                                    xmsLONG waitInterval,
                                    xmsHMsg *message,
                                    xmsHErrorBlock errorBlock);
```

C++ interface:

```
Message * receive(const xmsLONG waitInterval) const;
```

Receive the next message for the message consumer. The call waits only a specified period of time for a message, or until the message consumer is closed.

Parameters:

consumer (input)
The handle for the message consumer.

waitInterval (input)
The time, in milliseconds, that the call waits for a message. If you specify a wait interval of zero, the call waits indefinitely for a message.

message (output)
The handle for the message. If no message arrives during the wait interval, or if the message consumer is closed while the call is waiting for a message, the call returns a null handle.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Receive with No Wait

C interface:

```
xmsRC xmsMsgConsumerReceiveNoWait(xmsHMsgConsumer consumer,
                                  xmsHMsg *message,
                                  xmsHErrorBlock errorBlock);
```

C++ interface:

```
Message * receiveNoWait() const;
```

Receive the next message for the message consumer if one is available immediately.

Parameters:

consumer (input)
The handle for the message consumer.

message (output)
The handle for the message. If no message is available immediately, the call returns a null handle.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Message Listener

C interface:

```
xmsRC xmsMsgConsumerGetMessageListener(xmsHMsgConsumer consumer,
                                       fpXMS_MESSAGE_CALLBACK *lsr,
                                       xmsCONTEXT *context,
                                       xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsVOID setMessageListener(const MessageListener * lsr);
```

Register a message listener function and context data with the message consumer.

For more information about using message listeners and context data, see "Listeners and callbacks" on page 27.

Parameters:

consumer (input)
: The handle for the message consumer.

lsr (input)
: A pointer to the message listener function. If a message listener function is already registered with the message consumer, you can cancel the registration by specifying a null pointer instead.

context (input)
: A pointer to the context data.

errorBlock (input)
: The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Property

C interface:
```
xmsRC xmsMsgConsumerSetProperty(xmsHMsgConsumer consumer,
                                xmsHProperty property,
                                xmsHErrorBlock errorBlock);
```

C++ interface:
```
virtual xmsVOID setProperty(const Property & property);
```

Set the value of a property using a Property object.

Parameters:

consumer (input)
: The handle for the message consumer.

property (input)
: The handle for the Property object.

errorBlock (input)
: The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

MessageListener

An application uses a message listener to receive messages asynchronously.

Methods

On Message

C interface:

```
xmsVOID onMessage(xmsCONTEXT context,
                  xmsHMsg message);
```

C++ interface:

```
virtual xmsVOID onMessage(const Message * message);
```

Deliver a message asynchronously to the message consumer.

For a C application, onMessage() is the message listener function that is registered with the message consumer. For a C++ application, onMessage() is a method of the message listener that is registered with the message consumer.

For more information about using message listeners, see "Listeners and callbacks" on page 27.

Parameters:

context (input)
A pointer to the context data that is registered with the message consumer.

message (input)
The handle for the message.

MessageProducer

An application uses a message producer to send messages to a destination.

Methods

Close Message Producer

C interface:

```
xmsRC xmsMsgProducerClose(xmsHMsgProducer *producer,
                          xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsVOID close();
```

Close the message producer.

If an application tries to close a message producer that is already closed, the call is ignored.

Parameters:

producer (input/output)
On input, the handle for the message producer. On output, the call returns a null handle.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Default Delivery Mode

C interface:

```
xmsRC xmsMsgProducerGetDeliveryMode(xmsHMsgProducer producer,
                                    xmsDELIVERY_MODE *deliveryMode,
                                    xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsDELIVERY_MODE getDeliveryMode() const;
```

Get the default delivery mode for messages sent by the message producer.

Parameters:

producer (input)
The handle for the message producer.

deliveryMode (output)
The default delivery mode. If the connection uses WebSphere MQ Real-Time Transport, the value is always `XMSC_NON_PERSISTENT`.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Default Priority

C interface:

```
xmsRC xmsMsgProducerGetPriority(xmsHMsgProducer producer,
                                xmsINT *priority,
                                xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsINT getPriority() const;
```

Get the default priority for messages sent by the message producer.

Parameters:

producer (input)
: The handle for the message producer.

priority (output)
: The default message priority. The value is an integer in the range `0`, the lowest priority, to `9`, the highest priority.

errorBlock (input)
: The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Default Time to Live

C interface:

```
xmsRC xmsMsgProducerGetTimeToLive(xmsHMsgProducer producer,
                                  xmsLONG *timeToLive,
                                  xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsLONG getTimeToLive() const;
```

Get the default length of time that a message exists before it expires. The time is measured from when the message producer sends the message.

Parameters:

producer (input)
: The handle for the message producer.

timeToLive (output)
: The default time to live in milliseconds. A value of zero means that a message never expires. If the connection uses WebSphere MQ Real-Time Transport, the value is always zero.

errorBlock (input)
: The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Destination

C interface:
```
xmsRC xmsMsgProducerGetDestination(xmsHMsgProducer producer,
                                   xmsHDest *destination,
                                   xmsHErrorBlock errorBlock);
```

C++ interface:
```
Destination getDestination() const;
```

Get the destination for the message producer.

Parameters:

producer (input)
The handle for the message producer.

destination (output)
The handle for the destination. If the message producer does not have a destination, the call returns a null handle.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Disable Message ID Flag

C interface:
```
xmsRC xmsMsgProducerGetDisableMsgID(xmsHMsgProducer producer,
                                    xmsBOOL *msgIDDisabled,
                                    xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsBOOL getDisabledMessageID() const;
```

Get an indication of whether a receiving application uses message identifiers that are included in messages sent by the message producer.

Parameters:

producer (input)
The handle for the message producer.

msgIDDisabled (output)
If the value is `xmsTRUE`, a receiving application does not use message identifiers included in messages sent by the message producer. If the value is `xmsFALSE`, a receiving application does use message identifiers.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Disable Timestamp Flag

C interface:

```
xmsRC xmsMsgProducerGetDisableMsgTS(xmsHMsgProducer producer,
                                    xmsBOOL *timestampDisabled,
                                    xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsBOOL getDisableMessageTimestamp() const;
```

Get an indication of whether a receiving application uses timestamps that are included in messages sent by the message producer.

Parameters:

producer (input)
The handle for the message producer.

timestampDisabled (output)
If the value is `xmsTRUE`, a receiving application does not use timestamps included in messages sent by the message producer. If the value is `xmsFALSE`, a receiving application does use timestamps.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Property

C interface:

```
xmsRC xmsMsgProducerGetProperty(xmsHMsgProducer producer,
                                xmsCHAR *propertyName,
                                xmsHProperty *propertyValue,
                                xmsHErrorBlock errorBlock);
```

C++ interface:

```
virtual Property getProperty(const String & propertyName) const;
```

Get a Property object for the property identified by name.

Parameters:

producer (input)
The handle for the message producer.

propertyName (input)
The name of the property in the format of a null terminated string.

property (output)
The handle for the Property object.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Send

C interface:

```
xmsRC xmsMsgProducerSend(xmsHMsgProducer producer,
                         xmsHMsg message,
                         xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsVOID send(const Message & message) const;
```

Send a message to the destination that was specified when the message producer was created. Send the message using the message producer's default delivery mode, priority, and time to live.

Parameters:

producer (input)
The handle for the message producer.

message (input)
The handle for the message.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_FORMAT_EXCEPTION
- XMS_X_INVALID_DESTINATION_EXCEPTION

Send (specifying a delivery mode, priority, and time to live)

C interface:

```
xmsRC xmsMsgProducerSendWithAttr(xmsHMsgProducer producer,
                                 xmsHMsg message,
                                 xmsDELIVERY_MODE deliveryMode,
                                 xmsINT priority,
                                 xmsLONG timeToLive,
                                 xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsVOID send(const Message & message,
             const xmsDELIVERY_MODE deliveryMode,
             const xmsINT priority,
             const xmsLONG timeToLive) const;
```

Send a message to the destination that was specified when the message producer was created. Send the message using the specified delivery mode, priority, and time to live.

Parameters:

producer (input)
The handle for the message producer.

message (input)
The handle for the message.

deliveryMode (input)
The delivery mode for the message. If the connection uses WebSphere MQ Real-Time Transport, the value must be `XMSC_NON_PERSISTENT`.

priority (input)
: The priority of the message. The value can be an integer in the range `0`, for the lowest priority, to `9`, for the highest priority. If the connection uses WebSphere MQ Real-Time Transport, the value is ignored.

timeToLive (input)
: The time to live for the message in milliseconds. A value of zero means that the message never expires. If the connection uses WebSphere MQ Real-Time Transport, the value must be zero.

errorBlock (input)
: The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_FORMAT_EXCEPTION
- XMS_X_INVALID_DESTINATION_EXCEPTION
- XMS_X_ILLEGAL_STATE_EXCEPTION

Send (to a specified destination)

C interface:

```
xmsRC xmsMsgProducerSendDest(xmsHMsgProducer producer,
                             xmsHDest destination,
                             xmsHMsg message,
                             xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsVOID send(const Destination & destination,
             const Message & message) const;
```

Send a message to a specified destination if you are using a message producer for which no destination was specified when the message producer was created. Send the message using the message producer's default delivery mode, priority, and time to live.

Typically, you specify a destination when you create a message producer but, if you do not, you must specify a destination every time you send a message.

Parameters:

producer (input)
: The handle for the message producer.

destination (input)
: The handle for the destination.

message (input)
: The handle for the message.

errorBlock (input)
: The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_FORMAT_EXCEPTION
- XMS_X_INVALID_DESTINATION_EXCEPTION

Send (to a specified destination, specifying a delivery mode, priority, and time to live)

C interface:

```
xmsRC xmsMsgProducerSendWithAttr(xmsHMsgProducer producer,
                                 xmsHDest destination,
                                 xmsHMsg message,
                                 xmsDELIVERY_MODE deliveryMode,
                                 xmsINT priority,
                                 xmsLONG timeToLive,
                                 xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsVOID send(const Destination & destination,
             const Message & message,
             const xmsDELIVERY_MODE deliveryMode,
             const xmsINT priority,
             const xmsLONG timeToLive) const;
```

Send a message to a specified destination if you are using a message producer for which no destination was specified when the message producer was created. Send the message using the specified delivery mode, priority, and time to live.

Typically, you specify a destination when you create a message producer but, if you do not, you must specify a destination every time you send a message.

Parameters:

producer (input)
The handle for the message producer.

destination (input)
The handle for the destination.

message (input)
The handle for the message.

deliveryMode (input)
The delivery mode for the message. If the connection uses WebSphere MQ Real-Time Transport, the value must be `XMSC_NON_PERSISTENT`.

priority (input)
The priority of the message. The value can be an integer in the range `0`, for the lowest priority, to `9`, for the highest priority. If the connection uses WebSphere MQ Real-Time Transport, the value is ignored.

timeToLive (input)
The time to live for the message in milliseconds. A value of zero means that the message never expires. If the connection uses WebSphere MQ Real-Time Transport, the value must be zero.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_MESSAGE_FORMAT_EXCEPTION
- XMS_X_INVALID_DESTINATION_EXCEPTION
- XMS_X_ILLEGAL_STATE_EXCEPTION

Set Default Delivery Mode

C interface:
```
xmsRC xmsMsgProducerSetDeliveryMode(xmsHMsgProducer producer,
                                    xmsDELIVERY_MODE deliveryMode,
                                    xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID setDeliveryMode(const xmsDELIVERY_MODE deliveryMode);
```

Set the default delivery mode for messages sent by the message producer.

Parameters:

producer (input)
The handle for the message producer.

deliveryMode (input)
The default delivery mode. If the connection uses WebSphere MQ Real-Time Transport, the value must be `XMSC_NON_PERSISTENT`, which is the default value in this case.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Default Priority

C interface:
```
xmsRC xmsMsgProducerSetPriority(xmsHMsgProducer producer,
                                xmsINT priority,
                                xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID setPriority(const xmsINT priority);
```

Set the default priority for messages sent by the message producer.

If the connection uses WebSphere MQ Real-Time Transport, the priority of a message is ignored.

Parameters:

producer (input)
The handle for the message producer.

priority (input)
The default message priority. The value can be an integer in the range `0`, for the lowest priority, to `9`, for the highest priority. The default value is `4`.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Default Time to Live

C interface:

```
xmsRC xmsMsgProducerSetTimeToLive(xmsHMsgProducer producer,
                                  xmsLONG timeToLive,
                                  xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsVOID setTimeToLive(const xmsLONG timeToLive);
```

Set the default length of time that a message exists before it expires. The time is measured from when the message producer sends the message.

Parameters:

producer (input)
The handle for the message producer.

timeToLive (input)
The default time to live in milliseconds. A value of zero means that a message never expires. If the connection uses WebSphere MQ Real-Time Transport, the value must be zero, which is the default value in this case.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Disable Message ID Flag

C interface:

```
xmsRC xmsMsgProducerSetDisableMsgID(xmsHMsgProducer producer,
                                    xmsBOOL msgIDDisabled,
                                    xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsVOID setDisableMessageID(const xmsBOOL msgIDDisabled);
```

Indicate whether a receiving application uses message identifiers that are included in messages sent by the message producer.

Parameters:

producer (input)
The handle for the message producer.

msgIDDisabled (input)
The value `xmsTRUE` means that a receiving application does not use message identifiers included in messages sent by the message producer. The value `xmsFALSE` means that a receiving application does use message identifiers. The default value is `xmsFALSE`.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Disable Timestamp Flag

C interface:
```
xmsRC xmsMsgProducerSetDisableMsgTS(xmsHMsgProducer producer,
                                    xmsBOOL timestampDisabled,
                                    xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID setDisableMessageTimestamp(const xmsBOOL timestampDisabled);
```

Indicate whether a receiving application uses timestamps that are included in messages sent by the message producer.

Parameters:

producer (input)
The handle for the message producer.

timestampDisabled (input)
The value `xmsTRUE` means that a receiving application does not use timestamps included in messages sent by the message producer. The value `xmsFALSE` means that a receiving application does use timestamps. The default value is `xmsFALSE`.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Property

C interface:
```
xmsRC xmsMsgProducerSetProperty(xmsHMsgProducer producer,
                                xmsHProperty property,
                                xmsHErrorBlock errorBlock);
```

C++ interface:
```
virtual xmsVOID setProperty(const Property & property);
```

Set the value of a property using a Property object.

Parameters:

producer (input)
The handle for the message producer.

property (input)
The handle for the Property object.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Property

A Property object represents a property of an object. A Property object has three attributes:

Property name
The name of the property

Property value
The value of the property

Property type
The data type of the value of the property

Methods

Check Property Type

C interface:
```
xmsRC xmsPropertyIsType(xmsHProperty property,
                        xmsPROPERTY_TYPE propertyType,
                        xmsBOOL *isType,
                        xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Check whether the Property object has the specified property type.

Parameters:

property (input)
The handle for the Property object.

propertyType (input)
The property type, which must be one of the following values:

XMS_PROPERTY_TYPE_UNKNOWN

XMS_PROPERTY_TYPE_BOOL

XMS_PROPERTY_TYPE_BYTE

XMS_PROPERTY_TYPE_BYTEARRAY

XMS_PROPERTY_TYPE_CHAR

XMS_PROPERTY_TYPE_STRING

XMS_PROPERTY_TYPE_SHORT

XMS_PROPERTY_TYPE_INT

XMS_PROPERTY_TYPE_LONG

XMS_PROPERTY_TYPE_FLOAT

XMS_PROPERTY_TYPE_DOUBLE

isType (output)
If the value is xmsTRUE, the Property object has the specified property type. If the value is xmsFALSE, the Property object does not have the specified property type.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Copy Property

C interface:
```
xmsRC xmsPropertyDuplicate(xmsHProperty property,
                           xmsHProperty *copiedProperty,
                           xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Copy the Property object.

Parameters:

property (input)
The handle for the Property object.

copiedProperty (output)
The handle for the copy of the Property object.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Create Property

C interface:
```
xmsRC xmsPropertyCreate(xmsCHAR *propertyName,
                        xmsHProperty *property,
                        xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Create a Property object. The new Property object has no property type or property value.

Parameters:

propertyName (input)
The property name.

property (output)
The handle for the Property object.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Delete Property

C interface:

```
xmsRC xmsPropertyDispose(xmsHProperty *property,
                         xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Delete the Property object.

If an application tries to delete a Property object that is already deleted, the call is ignored.

Parameters:

property (input/output)
On input, the handle for the Property object. On output the call returns a null handle.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Boolean Property Value

C interface:

```
xmsRC xmsPropertyGetBoolean(xmsHProperty property,
                            xmsBOOL *propertyValue,
                            xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Get the boolean property value from the Property object.

Parameters:

property (input)
The handle for the Property object.

propertyValue (output)
The boolean property value.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Byte Array Property Value

C interface:

```
xmsRC xmsPropertyGetByteArray(xmsHProperty property,
                              xmsBYTE *propertyValue,
                              xmsSIZE length,
                              xmsSIZE *actualLength,
                              xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Get the byte array property value from the Property object.

For more information about how to use this function, see "C functions that return a string or byte array by value" on page 28.

Parameters:

property (input)
The handle for the Property object.

propertyValue (output)
The buffer to contain the byte array property value.

length (input)
The length of the buffer in bytes. If you specify a length of zero, the property value is not returned, but its length is returned in the actualLength parameter.

actualLength (output)
The length of the property value in bytes.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Byte Array Property Value by Reference

C interface:

```
xmsRC xmsPropertyGetByteArrayByRef(xmsHProperty property,
                                   xmsBYTE **propertyValue,
                                   xmsSIZE *length,
                                   xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Get a pointer to the byte array property value in the Property object.

For more information about how to use this function, see "C functions that return a string or byte array by reference" on page 29.

Parameters:

property (input)
The handle for the Property object.

propertyValue (output)
A pointer to the byte array property value.

length (output)
The length of the property value in bytes.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Byte Property Value

C interface:
```
xmsRC xmsPropertyGetByte(xmsHProperty property,
                         xmsBYTE *propertyValue,
                         xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Get the byte property value from the Property object.

Parameters:

property (input)
The handle for the Property object.

propertyValue (output)
The byte property value.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Character Property Value

C interface:
```
xmsRC xmsPropertyGetChar(xmsHProperty property,
                         xmsCHAR16 *propertyValue,
                         xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Get the 2-byte character property value from the Property object.

Parameters:

property (input)
The handle for the Property object.

propertyValue (output)
The 2-byte character property value.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Double Precision Floating Point Property Value

C interface:
```
xmsRC xmsPropertyGetDouble(xmsHProperty property,
                           xmsDOUBLE *propertyValue,
                           xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Get the double precision floating point property value from the Property object.

Parameters:

property (input)
The handle for the Property object.

propertyValue (output)
The double precision floating point property value.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Floating Point Property Value

C interface:
```
xmsRC xmsPropertyGetFloat(xmsHProperty property,
                          xmsFLOAT *propertyValue,
                          xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Get the floating point property value from the Property object.

Parameters:

property (input)
The handle for the Property object.

propertyValue (output)
The floating point property value.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Integer Property Value

C interface:
```
xmsRC xmsPropertyGetInt(xmsHProperty property,
                        xmsINT *propertyValue,
                        xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Get the integer property value from the Property object.

Parameters:

property (input)
The handle for the Property object.

propertyValue (output)
The integer property value.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Long Integer Property Value

C interface:
```
xmsRC xmsPropertyGetLong(xmsHProperty property,
                         xmsLONG *propertyValue,
                         xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Get the long integer property value from the Property object.

Parameters:

property (input)
The handle for the Property object.

propertyValue (output)
The long integer property value.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Property Name

C interface:

```
xmsRC xmsPropertyGetName(xmsHProperty property,
                         xmsCHAR *propertyName,
                         xmsSIZE length,
                         xmsSIZE *actualLength,
                         xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Get the property name from the Property object.

For more information about how to use this function, see "C functions that return a string or byte array by value" on page 28.

Parameters:

property (input)
The handle for the Property object.

propertyName (output)
The buffer to contain the property name.

length (input)
The length of the buffer in bytes. If you specify a length of zero, the property name is not returned, but its length is returned in the actualLength parameter.

actualLength (output)
The length of the property name in bytes.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Property Type

C interface:

```
xmsRC xmsPropertyGetType(xmsHProperty property,
                         xmsPROPERTY_TYPE *propertyType,
                         xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Get the property type from the Property object.

Parameters:

property (input)
The handle for the Property object.

propertyType (output)
The property type, which is one of the following values:

```
XMS_PROPERTY_TYPE_UNKNOWN
XMS_PROPERTY_TYPE_BOOL
XMS_PROPERTY_TYPE_BYTE
XMS_PROPERTY_TYPE_BYTEARRAY
XMS_PROPERTY_TYPE_CHAR
XMS_PROPERTY_TYPE_STRING
XMS_PROPERTY_TYPE_SHORT
XMS_PROPERTY_TYPE_INT
XMS_PROPERTY_TYPE_LONG
XMS_PROPERTY_TYPE_FLOAT
XMS_PROPERTY_TYPE_DOUBLE
```

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get Short Integer Property Value

C interface:

```
xmsRC xmsPropertyGetShort(xmsHProperty property,
                          xmsSHORT *propertyValue,
                          xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Get the short integer property value from the Property object.

Parameters:

property (input)
The handle for the Property object.

propertyValue (output)
The short integer property value.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get String Property Value

C interface:

```
xmsRC xmsPropertyGetString(xmsHProperty property,
                           xmsCHAR *propertyValue,
                           xmsSIZE length,
                           xmsSIZE *actualLength,
                           xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Get the string property value from the Property object.

For more information about how to use this function, see "C functions that return a string or byte array by value" on page 28.

Parameters:

property (input)
The handle for the Property object.

propertyValue (output)
The buffer to contain the string property value.

length (input)
The length of the buffer in bytes. If you specify a length of zero, the property value is not returned, but its length is returned in the actualLength parameter.

actualLength (output)
The length of the property value in bytes.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Get String Property Value by Reference

C interface:

```
xmsRC xmsPropertyGetStringByRef(xmsHProperty property,
                                xmsCHAR **propertyValue,
                                xmsSIZE *length,
                                xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Get a pointer to the string property value in the Property object.

For more information about how to use this function, see "C functions that return a string or byte array by reference" on page 29.

Parameters:

property (input)
The handle for the Property object.

propertyValue (output)
A pointer to the string property value.

length (output)
The length of the property value in bytes.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Boolean Property Value

C interface:

```
xmsRC xmsPropertySetBoolean(xmsHProperty property,
                            xmsBOOL propertyValue,
                            xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Set a boolean property value in the Property object.

Parameters:

property (input)
The handle for the Property object.

propertyValue (input)
The boolean property value.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Byte Array Property Value

C interface:

```
xmsRC xmsPropertySetByteArray(xmsHProperty property,
                              xmsBYTE *propertyValue,
                              xmsSIZE length,
                              xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Set a byte array property value in the Property object.

Parameters:

property (input)
The handle for the Property object.

propertyValue (input)
The byte array property value.

length (input)
The length of the property value in bytes.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Byte Property Value

C interface:

```
xmsRC xmsPropertySetByte(xmsHProperty property,
                         xmsBYTE propertyValue,
                         xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Set a byte property value in the Property object.

Parameters:

property (input)
The handle for the Property object.

propertyValue (input)
The byte property value.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Character Property Value

C interface:

```
xmsRC xmsPropertySetChar(xmsHProperty Property,
                         xmsCHAR16 Value,
                         xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Set a 2-byte character property value in the Property object.

Parameters:

property (input)
The handle for the Property object.

propertyValue (input)
The 2-byte character property value.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Double Precision Floating Point Property Value

C interface:

```
xmsRC xmsPropertySetDouble(xmsHProperty property,
                           xmsDOUBLE propertyValue,
                           xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Set a double precision floating point property value in the Property object.

Parameters:

property (input)
The handle for the Property object.

propertyValue (input)
The double precision floating point property value.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Floating Point Property Value

C interface:

```
xmsRC xmsPropertySetFloat(xmsHProperty property,
                          xmsFLOAT propertyValue,
                          xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Set a floating point property value in the Property object.

Parameters:

property (input)
The handle for the Property object.

propertyValue (input)
The floating point property value.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Integer Property Value

C interface:
```
xmsRC xmsPropertySetInt(xmsHProperty property,
                        xmsINT propertyValue,
                        xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Set an integer property value in the Property object.

Parameters:

property (input)
The handle for the Property object.

propertyValue (input)
The integer property value.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Long Integer Property Value

C interface:
```
xmsRC xmsPropertySetLong(xmsHProperty property,
                         xmsLONG propertyValue,
                         xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Set a long integer property value in the Property object.

Parameters:

property (input)
The handle for the Property object.

propertyValue (input)
The long integer property value.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Short Integer Property Value

C interface:
```
xmsRC xmsPropertySetShort(xmsHProperty property,
                          xmsSHORT propertyValue,
                          xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Set a short integer property value in the Property object.

Parameters:

property (input)
The handle for the Property object.

propertyValue (input)
The short integer property value.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set String Property Value

C interface:
```
xmsRC xmsPropertySetString(xmsHProperty property,
                           xmsCHAR *propertyValue,
                           xmsSIZE length,
                           xmsHErrorBlock errorBlock);
```

C++ interface:
Not applicable

Set a string property value in the Property object.

Parameters:

property (input)
The handle for the Property object.

propertyValue (input)
The string property value as a character array.

length (input)
The length of the property value in bytes.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Session

A session is a single threaded context for sending and receiving messages.

Methods

Close Session

C interface:
```
xmsRC xmsSessClose(xmsHSess *session,
                   xmsHErrorBlock errorBlock);
```

C++ interface:
```
xmsVOID close();
```

Close the session.

All objects dependent on the session are deleted. For information about which objects are deleted, see "Deleting objects" on page 24.

If an application tries to close a session that is already closed, the call is ignored.

Parameters:

session (input/output)
On input, the handle for the session. On output, the call returns a null handle.

errorBlock (input)
The handle for an error block or a null handle.

Thread context:
Any

Exceptions:
XMS_X_GENERAL_EXCEPTION

Create Bytes Message

C interface:
```
xmsRC xmsSessCreateBytesMessage(xmsHSess session,
                                xmsHMsg *message,
                                xmsHErrorBlock errorBlock);
```

C++ interface:
```
BytesMessage createBytesMessage() const;
```

Create a bytes message.

Parameters:

session (input)
The handle for the session.

message (output)
The handle for the bytes message.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Create Consumer

C interface:

```
xmsRC xmsSessCreateConsumer(xmsHSess session,
                            xmsHDest destination,
                            xmsHMsgConsumer *consumer,
                            xmsHErrorBlock errorBlock);
```

C++ interface:

```
MessageConsumer createConsumer(const Destination & destination) const;
```

Create a message consumer for the specified destination.

Parameters:

session (input)
The handle for the session.

destination (input)
The handle for the destination.

consumer (output)
The handle for the message consumer.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_INVALID_DESTINATION_EXCEPTION

Create Consumer (with message selector)

C interface:

```
xmsRC xmsSessCreateConsumerSelector(xmsHSess session,
                                    xmsHDest destination,
                                    xmsCHAR *messageSelector,
                                    xmsSIZE length,
                                    xmsHMsgConsumer *consumer,
                                    xmsHErrorBlock errorBlock);
```

C++ interface:

```
MessageConsumer createConsumer(const Destination & destination,
                               const String & messageSelector) const;
```

Create a message consumer for the specified destination using a message selector.

Parameters:

session (input)
The handle for the session.

destination (input)
The handle for the destination.

messageSelector (input)
A message selector expression as a character array. Only those messages with properties that match the message selector expression are delivered to the message consumer. A value of null or an empty string indicates that there is no message selector for the message consumer.

length (input)
The length of the message selector expression in bytes.

consumer (output)
The handle for the message consumer.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_INVALID_DESTINATION_EXCEPTION
- XMS_X_INVALID_SELECTOR_EXCEPTION

Create Consumer (with message selector and local message flag)

C interface:

```
xmsRC xmsSessCreateConsumerSelectorLocal(xmsHSess session,
                                         xmsHDest destination,
                                         xmsCHAR *messageSelector,
                                         xmsSIZE length,
                                         xmsBOOL noLocal,
                                         xmsHMsgConsumer *consumer,
                                         xmsHErrorBlock errorBlock);
```

C++ interface:

```
MessageConsumer createConsumer(const Destination & destination,
                               const String & messageSelector,
                               const xmsBOOL noLocal) const;
```

Create a message consumer for the specified destination using a message selector, and specifying whether the message consumer can receive messages published by its own connection.

Parameters:

session (input)
The handle for the session.

destination (input)
The handle for the destination.

messageSelector (input)
A message selector expression as a character array. Only those messages with properties that match the message selector expression are delivered to the message consumer. A value of null or an empty string indicates that there is no message selector for the message consumer.

length (input)
The length of the message selector expression in bytes.

noLocal (input)
If the value is `xmsTRUE`, the message consumer does not receive the messages published by its own connection. By default, a message consumer receives messages published by its own connection.

consumer (output)
The handle for the message consumer.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
- XMS_X_GENERAL_EXCEPTION
- XMS_X_INVALID_DESTINATION_EXCEPTION
- XMS_X_INVALID_SELECTOR_EXCEPTION

Create Map Message

C interface:
```
xmsRC xmsSessCreateMapMessage(xmsHSess session,
                              xmsHMsg *message,
                              xmsHErrorBlock errorBlock);
```

C++ interface:
```
MapMessage createMapMessage() const;
```

Create a map message.

Parameters:

session (input)
The handle for the session.

message (output)
The handle for the map message.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Create Message

C interface:
```
xmsRC xmsSessCreateMessage(xmsHSess session,
                           xmsHMsg *message,
                           xmsHErrorBlock errorBlock);
```

C++ interface:
```
Message createMessage() const;
```

Create a message that has no body and therefore contains no application data.

Parameters:

session (input)
The handle for the session.

message (output)
The handle for the message.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Create Producer

C interface:

```
xmsRC xmsSessCreateProducer(xmsHSess session,
                            xmsHDest destination,
                            xmsHMsgProducer *producer,
                            xmsHErrorBlock errorBlock);
```

C++ interface:

```
MessageProducer createProducer(const Destination & destination) const;
```

Create a message producer to send messages to the specified destination.

Parameters:

session (input)
The handle for the session.

destination (input)
The handle for the destination. If you specify a null handle, the message producer is created without a destination.

producer (output)
The handle for the message producer.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

- XMS_X_GENERAL_EXCEPTION
- XMS_X_INVALID_DESTINATION_EXCEPTION

Create Temporary Topic

C interface:

```
xmsRC xmsSessCreateTemporaryTopic(xmsHSess session,
                                  xmsHDest *topic,
                                  xmsHErrorBlock errorBlock);
```

C++ interface:

```
Destination createTemporaryTopic() const;
```

Create a temporary topic. The temporary topic remains until the connection ends or the topic is explicitly deleted, whichever is the sooner.

Parameters:

session (input)
The handle for the session.

topic (output)
The handle for the temporary topic.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

XMS_X_GENERAL_EXCEPTION

Create Topic

C interface:

Not implemented. Use one of the following methods instead:

- "Create Destination (using a URI)" on page 73
- "Create Destination (specifying a type and name)" on page 73

C++ interface:

```
Destination createTopic(const String & topicName) const;
```

Create a topic.

Parameter:

topicName (input)
The name of the topic in the format of a null terminated string.

Exceptions:

XMS_X_GENERAL_EXCEPTION

Get Acknowledgement Mode

C interface:

```
xmsRC xmsSessGetAcknowledgeMode(xmsHSess session,
                                xmsACKNOWLEDGE_MODE *acknowledgeMode,
                                xmsHErrorBlock errorBlock);
```

C++ interface:

```
xmsACKNOWLEDGE_MODE getAcknowledgeMode() const;
```

Get the acknowledgement mode for the session. The acknowledgement mode is specified when the session is created.

Parameters:

session (input)
The handle for the session.

acknowledgeMode (output)
The acknowledgement mode, which is `XMSC_AUTO_ACKNOWLEDGE`.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:

XMS_X_GENERAL_EXCEPTION

Get Property

C interface:

```
xmsRC xmsSessGetProperty(xmsHSess session,
                         xmsCHAR *propertyName,
                         xmsHProperty *property,
                         xmsHErrorBlock errorBlock);
```

C++ interface:

```
virtual Property getProperty(const String & propertyName) const;
```

Get a Property object for the property identified by name.

Parameters:

session (input)
The handle for the session.

propertyName (input)
The name of the property in the format of a null terminated string.

property (output)
The handle for the Property object.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Set Property

C interface:
```
xmsRC xmsSessSetProperty(xmsHSess session,
                         xmsHProperty property,
                         xmsHErrorBlock errorBlock);
```

C++ interface:
```
virtual xmsVOID setProperty(const Property & property);
```

Set the value of a property using a Property object.

Parameters:

session (input)
The handle for the session.

property (input)
The handle for the Property object.

errorBlock (input)
The handle for an error block or a null handle.

Exceptions:
XMS_X_GENERAL_EXCEPTION

Chapter 8. Properties of XMS objects

This chapter describes the properties of objects that are defined by XMS. For the properties of a Message object, however, see Chapter 6, "XMS messages," on page 31.

The name of each property is defined as a named constant in the header file, xmsc.h, and so an application can use the name to identify the property.

If an application defines its own properties of the objects discussed in this chapter, it does not cause an error but it might cause unpredictable results.

The chapter contains the following sections:

Properties of Connection

Table 14 on page 172 lists the XMS defined properties of a Connection object. For each property, the table provides its name, data type, description, and valid values, including its default value.

Table 13. Properties of Connection

Name of property	Data type	Description	Valid values (default value in bold)
XMSC_CLIENT_CCSID[1]	xmsINT	The identifier of the coded character set that the application uses for the connection.	• **XMSC_USE_PROCESS_CCSID** • XMSC_DEFAULT_RTT_CCSID • A coded character set identifier

Notes:

1. For a description of each of the valid values of this property, see Table 14 on page 172.

Properties of ConnectionFactory

Table 14 on page 172 lists the XMS defined properties of a ConnectionFactory object. For each property, the table provides its name, data type, description, and valid values, including its default value.

Properties of XMS objects

Table 14. Properties of ConnectionFactory

Name of property	Data type	Description	Valid values (default value in bold)
XMSC_CLIENT_CCSID	xmsINT	The identifier of the coded character set that the application uses for a connection.	• **XMSC_USE_PROCESS_CCSID** - The application uses the same character set as the current process. On Windows, XMS calls the console function GetConsoleCP() to determine the character set of the current process. • XMSC_DEFAULT_RTT_CCSID - The application uses the same character set as the broker to which it is connected. If the connection uses WebSphere MQ Real-Time Transport or WebSphere MQ Multicast Transport, no code page conversion is attempted for messages sent to, or received from, the broker. • A coded character set identifier - The application uses the specified character set.
XMSC_HOST_NAME	String[1]	The host name or IP address of the system on which the broker resides.	• **Not set** • localhost • A host name • An IP address
XMSC_MULTICAST[2]	xmsINT	The multicast setting for the connection factory. This setting is used only when the multicast setting for a destination is XMSC_MULTICAST_ASCF.	• **XMSC_MULTICAST_DISABLED** • XMSC_MULTICAST_NOT_RELIABLE • XMSC_MULTICAST_RELIABLE • XMSC_MULTICAST_ENABLED
XMSC_PASSWORD	Byte array[3]	A password that can be used to authenticate the application when it attempts to create a connection. The password is used in conjunction with the value of XMSC_USERID.	• **Not set** • An array of bytes
XMSC_PORT	xmsINT	The port number on which the broker listens for published messages and subscription requests.	• **Not set** • A port number
XMSC_TRANSPORT_TYPE	xmsINT	The means by which an application connects to a broker.	**XMSC_TP_DIRECT_TCPIP** - The connection uses WebSphere MQ Real-Time Transport or WebSphere MQ Multicast Transport.
XMSC_USERID	String[1]	A user identifier that can be used to authenticate the application when it attempts to create a connection. The user identifier is used in conjunction with the value of XMSC_PASSWORD.	• **Not set** • A null terminated string

Notes:

1. This is the data type if you are using C++. If you are programming in C, it is a character array.
2. For a full description of this property, and a description of each of its valid values, see Table 16 on page 174.
3. A value with this data type can be set using the Set Byte Array Property Value method of a Property object.

Properties of ConnectionMetaData

Table 15 lists the XMS defined properties of a ConnectionMetaData object. For each property, the table provides its name, data type, description, and value.

Table 15. Properties of ConnectionMetaData

Name of property	Data type	Description	Value
XMSC_JMS_MAJOR_VERSION	xmsINT	The major version number of the JMS specification upon which XMS is based.	1
XMSC_JMS_MINOR_VERSION	xmsINT	The minor version number of the JMS specification upon which XMS is based.	1
XMSC_JMS_VERSION	String[1]	The version identifier of the JMS specification upon which XMS is based.	“1.1”
XMSC_MAJOR_VERSION	xmsINT	The version number of the XMS client.	1
XMSC_MINOR_VERSION	xmsINT	The release number of the XMS client.	0
XMSC_PROVIDER_NAME	String[1]	The provider of the XMS client.	“IBM”
XMSC_VERSION	String[1]	The version identifier of the XMS client.	“1.0”

Notes:

1. This is the data type if you are using C++. If you are programming in C, it is a character array.

Properties of Destination

Table 16 on page 174 lists the XMS defined properties of a Destination object. For each property, the table provides its name, data type, description, and valid values, including its default value.

Table 16. Properties of Destination

Name of property	Data type	Description	Valid values (default value in bold)
XMSC_MULTICAST	xmsINT	The multicast setting for the destination. The setting determines how messages are delivered to a message consumer that receives messages from the destination. Only a destination that is a topic can have this property. The property has no effect on how a message producer sends messages to the destination.	• **XMSC_MULTICAST_ASCF** - Messages are delivered to the message consumer according to the multicast setting for the connection factory associated with the message consumer. The multicast setting for the connection factory is noted at the time the message consumer is created. • XMSC_MULTICAST_DISABLED - Messages are not delivered to the message consumer using WebSphere MQ Multicast Transport. • XMSC_MULTICAST_NOT_RELIABLE - If the topic is configured for multicast in the broker, messages are delivered to the message consumer using WebSphere MQ Multicast Transport. A reliable quality of service is not used even if the topic is configured for reliable multicast. • XMSC_MULTICAST_RELIABLE - If the topic is configured for reliable multicast in the broker, messages are delivered to the message consumer using WebSphere MQ Multicast Transport with a reliable quality of service. If the topic is not configured for reliable multicast, you cannot create a message consumer for the topic. • XMSC_MULTICAST_ENABLED - If the topic is configured for multicast in the broker, messages are delivered to a message consumer using WebSphere MQ Multicast Transport. A reliable quality of service is used if the topic is configured for reliable multicast.
XMSC_PRIORITY[1]	xmsINT	The priority of messages sent to the destination.	• **XMSC_PROPERTY_AS_APP** - A message has the priority specified by the Send call. If the Send call specifies no priority, the default priority of the message producer is used instead. • An integer in the range 0, for the lowest priority, to 9, for the highest priority - A message has the priority specified for the destination. The default priority of the message producer and any priority specified on the Send call are ignored.

Notes:

1. WebSphere MQ Real-Time Transport and WebSphere MQ Multicast Transport take no action based upon the priority of a message. The priority of a message is honoured only if the message is eventually put on a WebSphere MQ queue and the `MsgDeliverySequence` attribute of the queue is MQMDS_PRIORITY.

Properties of MessageConsumer

Table 17 on page 175 lists the XMS defined properties of a MessageConsumer object. For each property, the table provides its name, data type, description, and valid values.

Table 17. Properties of MessageConsumer

Name of property	Data type	Description	Valid values
XMSC_IS_SUBSCRIPTION_ MULTICAST	xmsBOOL	Indicates whether messages are being delivered to the message consumer using WebSphere MQ Multicast Transport.	• xmsTRUE - Messages are being delivered using WebSphere MQ Multicast Transport. • xmsFALSE - Messages are not being delivered using WebSphere MQ Multicast Transport.
XMSC_IS_SUBSCRIPTION_ RELIABLE_MULTICAST	xmsBOOL	Indicates whether messages are being delivered to the message consumer using WebSphere MQ Multicast Transport with a reliable quality of service.	• xmsTRUE - Messages are being delivered using WebSphere MQ Multicast Transport with a reliable quality of service. • xmsFALSE - Messages are not being delivered using WebSphere MQ Multicast Transport with a reliable quality of service.

Properties of Session

Table 18 lists the XMS defined properties of a Session object. For each property, the table provides its name, data type, description, and valid values, including its default value.

Table 18. Properties of Session

Name of property	Data type	Description	Valid values (default value in bold)
XMSC_CLIENT_CCSID[1]	xmsINT	The identifier of the coded character set that the application uses for the session.	• **XMSC_USE_PROCESS_CCSID** • XMSC_DEFAULT_RTT_CCSID • A coded character set identifier

Notes:

1. For a description of each of the valid values of this property, see Table 14 on page 172.

Appendix. Notices

This information was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Trademarks

Index

Sending your comments to IBM

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book.

Please limit your comments to the information in this book and the way in which the information is presented.

To make comments about the functions of IBM products or systems, talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

You can send your comments to IBM in any of the following ways:

- By mail, to this address:

  User Technologies Department (MP095)
  IBM United Kingdom Laboratories
  Hursley Park
  WINCHESTER,
  Hampshire
  SO21 2JN
  United Kingdom
- By fax:
  - From outside the U.K., after your international access code use 44–1962–816151
  - From within the U.K., use 01962–816151
- Electronically, use the appropriate network ID:
  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink™: HURSLEY(IDRCF)
  - Internet: idrcf@hursley.ibm.com

Whichever method you use, ensure that you include:

- The publication title and order number
- The topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

Printed in USA

SC34-6363-00