

8.0

*IBM Message Service Client for .NET*

**IBM**

**Note**

Before using this information and the product it supports, read the information in [“Notices” on page 243.](#)

This edition applies to version 8 release 0 of IBM® MQ and to all subsequent releases and modifications until otherwise indicated in new editions.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© **Copyright International Business Machines Corporation 2007, 2025.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

<b>Introduction to the IBM Message Service Client for .NET.....</b>	<b>5</b>
Introduction to the IBM Message Service Client for .NET.....	5
Styles of messaging.....	6
The XMS object model.....	7
Attributes and properties of objects.....	8
Administered objects.....	9
The XMS message model.....	10
Preventing applications from using newer XMS version.....	10
Setting up the messaging server environment.....	11
Configuring the queue manager and broker for an application that connects to a WebSphere MQ queue manager.....	11
Configuring a broker for an application that uses a real-time connection to a broker.....	13
Configuring the service integration bus for an application that connects to a WebSphere Service Integration Bus.....	14
Installing Message Service Client for .NET using the installation wizard.....	15
Prerequisites for XMS applications connecting to WebSphere MQ.....	17
Using the XMS sample applications.....	18
The sample applications.....	18
Running the sample applications.....	19
Building the .NET sample applications.....	20
Developing XMS applications.....	20
Writing XMS applications.....	21
Writing XMS .NET applications.....	44
Working with administered objects.....	50
Securing communications for XMS applications.....	64
XMS messages.....	68
Troubleshooting.....	82
Trace configuration for .NET applications.....	82
FFDC configuration for .NET applications.....	86
Tips for troubleshooting.....	86
Message Service Clients for .NET reference.....	87
.NET interfaces.....	87
Properties of XMS objects.....	173
<b>Notices.....</b>	<b>243</b>
Programming interface information.....	244
Trademarks.....	244



# Introduction to the IBM Message Service Client for .NET

---

IBM Message Service Client for .NET provides an application programming interface (API) called XMS that has the same set of interfaces as the Java Message Service (JMS) API. IBM Message Service Client for .NET contains a fully managed implementation of XMS, which can be used by any .NET compliant language.

XMS supports:

- point-to-point messaging
- publish/subscribe messaging
- Synchronous message delivery
- Asynchronous message delivery

An XMS application can exchange messages with the following types of application:

- An XMS application
- An IBM MQ classes for JMS application
- A native IBM MQ application
- A JMS application that is using the IBM MQ default messaging provider

An XMS application can connect to, and use the resources of, any of the following messaging servers:

## **IBM MQ queue manager**

The application can connect in either bindings or client mode.

## **WebSphere® Application Server Service Integration Bus**

The application can use a direct TCP/IP connection, or it can use HTTP over TCP/IP.

## **IBM Integration Bus**

Messages are transported between the application and the broker using WebSphere MQ Real-Time Transport. Messages can be delivered to the application using WebSphere MQ Multicast Transport.

By connecting to an IBM MQ queue manager, an XMS application can use WebSphere MQ Enterprise Transport to communicate with IBM Integration Bus. Alternatively, an XMS application can publish and subscribe by connecting to WebSphere MQ.

## **Related concepts**

[“Styles of messaging” on page 6](#)

[“The XMS object model” on page 7](#)

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

[“The XMS message model” on page 10](#)

The XMS message model is the same as the WebSphere MQ classes for JMS message model.

# Introduction to the IBM Message Service Client for .NET

---

IBM Message Service Client for .NET provides an application programming interface (API) called XMS that has the same set of interfaces as the Java Message Service (JMS) API. IBM Message Service Client for .NET contains a fully managed implementation of XMS, which can be used by any .NET compliant language.

XMS supports:

- point-to-point messaging
- publish/subscribe messaging
- Synchronous message delivery
- Asynchronous message delivery

An XMS application can exchange messages with the following types of application:

- An XMS application
- An IBM MQ classes for JMS application
- A native IBM MQ application
- A JMS application that is using the IBM MQ default messaging provider

An XMS application can connect to, and use the resources of, any of the following messaging servers:

#### **IBM MQ queue manager**

The application can connect in either bindings or client mode.

#### **WebSphere Application Server Service Integration Bus**

The application can use a direct TCP/IP connection, or it can use HTTP over TCP/IP.

#### **IBM Integration Bus**

Messages are transported between the application and the broker using WebSphere MQ Real-Time Transport. Messages can be delivered to the application using WebSphere MQ Multicast Transport.

By connecting to an IBM MQ queue manager, an XMS application can use WebSphere MQ Enterprise Transport to communicate with IBM Integration Bus. Alternatively, an XMS application can publish and subscribe by connecting to WebSphere MQ.

#### **Related concepts**

[“Styles of messaging” on page 6](#)

[“The XMS object model” on page 7](#)

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

[“The XMS message model” on page 10](#)

The XMS message model is the same as the WebSphere MQ classes for JMS message model.

## **Styles of messaging**

---

XMS supports the point-to-point and publish/subscribe 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 IBM Integration Bus, IBM Integration Bus might replicate a message and route copies of the message to different queues. As a result, more than one application can receive the message. IBM Integration Bus might also transform a message and add data to it.

A key characteristic of point-to-point messaging is that an application places a message onto a local queue when it sends a message. The underlying messaging and queuing system determines which destination queue the message is sent to. The receiving application retrieves the message from the destination queue.

### **Publish/subscribe messaging**

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

A *publisher* supplies information in the form of publication 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 creating subscriptions.

The publish/subscribe system receives publications from publishers and subscriptions from subscribers. It routes publications to subscribers. A subscriber receives publications on only those topics to which it subscribed.

A key characteristic of publish/subscribe messaging is that a publisher identifies a topic when it publishes a message. It does not identify the subscribers. 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.

## 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 parameters for a connection. An application uses a `ConnectionFactory` to create a connection. An application can provide the parameters at run time and create a `ConnectionFactory` object. Alternatively, the connection parameters can be stored in a repository of administered objects. An application can retrieve an object from the repository and create a `ConnectionFactory` object from it.

### **Connection**

A `Connection` object encapsulates an active connection from an application to a messaging server. An application uses a connection to create sessions.

### **Destination**

An application sends messages or receives messages using a `Destination` object. In the publish/subscribe domain, a `Destination` object encapsulates a topic and, in the point-to-point domain, a `Destination` object encapsulates a queue. An application can provide the parameters to create a `Destination` object at run time. Alternatively, it can create a `Destination` object from an object definition that is stored in repository of administered objects.

### **Session**

A `Session` object is a single threaded context for sending and receiving messages. An application uses a `Session` object to create `Message`, `MessageProducer`, and `MessageConsumer` objects.

### **Message**

A `Message` object encapsulates the `Message` object that an application sends using a `MessageProducer` object, or receives using a `MessageConsumer` object.

### **MessageProducer**

A `MessageProducer` object is used by an application to send messages to a destination.

### **MessageConsumer**

An `MessageConsumer` object is used by an application to receive messages sent to a destination.

Figure 1 on page 8 shows these objects and their relationships. This diagram shows the main types of XMS object: `ConnectionFactory`, `Connection`, `Session`, `MessageProducer`, `MessageConsumer`, `Message`, and `Destination`. An application uses a connection factory to create a connection, and uses a connection to create sessions. The application can then use a session to create messages, message producers, and message consumers. The application uses a message producer to send messages to a destination, and uses a message consumer to receive messages sent to a destination.

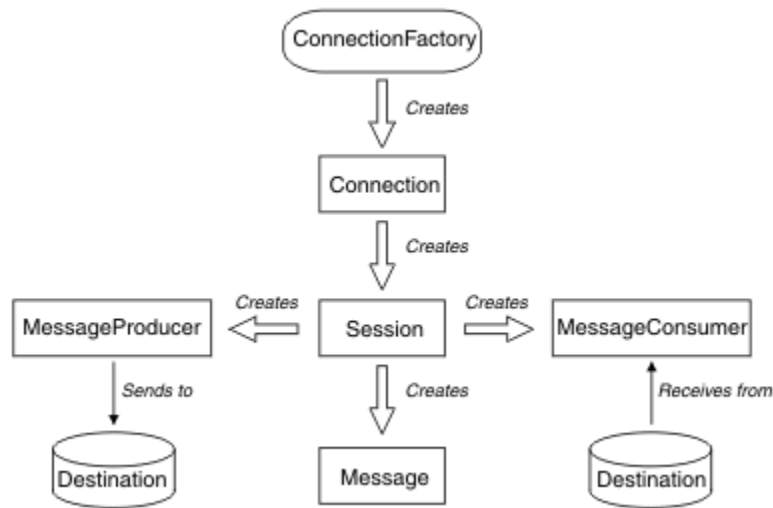


Figure 1. XMS objects and their relationships

In .NET, the XMS classes are defined as a set of .NET interfaces. When you are coding XMS .NET applications, you need only the declared interfaces.

The XMS object model is based on the domain independent interfaces that are described in *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, that are implemented in different ways.

### Attributes

An object characteristic that is always present and occupies storage, even if the attribute does not have a value. In this respect, an attribute is similar to a field in a fixed-length data structure. A distinguishing feature of attributes 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. A property cannot be deleted or its storage recovered after its value is set. You can change its value. XMS provides a set of generic methods for setting and getting property values.

### Related concepts

#### XMS primitive types

XMS provides equivalents of the eight Java primitive types (byte, short, int, long, float, double, char, and boolean). This allows the interchange of messages between XMS and JMS without data becoming lost or corrupted.

#### Implicit conversion of a property value from one data type to another

When an application gets the value of a property, the value can be converted by XMS into another data type. Many rules govern which conversions are supported and how XMS performs the conversions.

### Related reference

[Data types for elements of application data](#)

To ensure that an XMS application can exchange messages with a WebSphere MQ classes for JMS application, both the applications must be able to interpret the application data in the body of a message in the same way.

## Administered objects

Using administered objects, you can administer the connection settings used by client applications to be administered from a central repository. An application retrieves object definitions from the central repository and uses them to create `ConnectionFactory` and `Destination` objects. Using administered objects, you can de-couple applications from the resources that they use at run time.

For example, XMS applications can be written and tested with administered objects that reference a set of connections and destinations in a test environment. When the applications are deployed, the administered objects can be changed to configure the applications to refer to connections and destinations in the production environment.

XMS supports two types of administered object:

- A `ConnectionFactory` object, which is used by applications to make the initial connection to the server.
- A `Destination` object, which is used by applications to specify the destination for messages that are being sent, and the source of messages that are being received. A destination is either a topic or a queue on the server to which an application connects.

The administration tool **JMSAdmin** is supplied with WebSphere MQ. It is used to create and manage administered objects for in a central repository of administered objects.

The administered objects in the repository can be used by WebSphere MQ classes for JMS and XMS applications. XMS applications can use the `ConnectionFactory` and `Destination` objects to connect to a WebSphere MQ queue manager. An administrator can change the object definitions held in the repository without affecting application code.

The following diagram shows how an XMS application typically uses administered objects. The left-hand side of the diagram shows a repository containing `ConnectionFactory` and `Destination` object definitions that are administered using an administration console. The right-hand side of the diagram shows an XMS application that looks up object definitions in the repository, and then uses these object definitions when connecting to a messaging server.

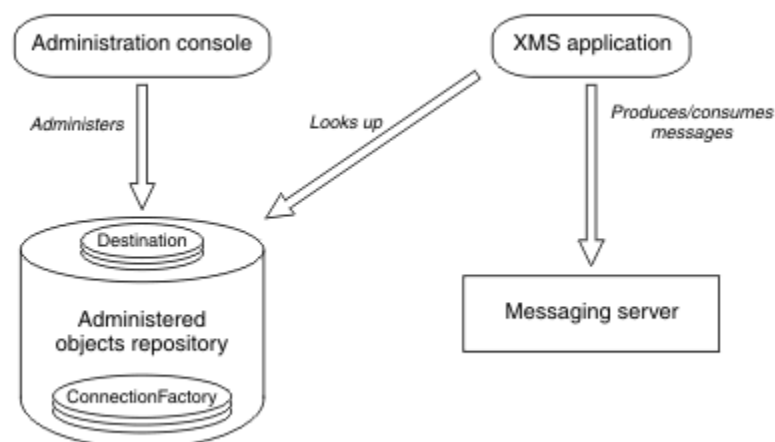


Figure 2. Typical use of administered objects by an XMS application

### Related concepts

[Working with administered objects](#)

This chapter provides information about administered objects. XMS applications can retrieve object definitions from a central administered objects repository, and use them to create connection factories and destinations.

#### Supported types of administered object repository

File System and LDAP administered objects can be used to connect to WebSphere MQ and WebSphere Application Server, whereas COS Naming can be used to connect only to the WebSphere Application Server.

#### **Related tasks**

##### Creating administered objects

The ConnectionFactory and Destination object definitions that XMS applications require to make a connection to a messaging server must be created using the appropriate administrative tools.

## The XMS message model

---

The XMS message model is the same as the WebSphere MQ classes for JMS message model.

In particular, XMS implements the same message header fields and message properties that WebSphere MQ classes for JMS implements:

- JMS header fields. These fields have names that commence with the prefix JMS.
- JMS defined properties. These fields have properties whose names commence with the prefix JMSX.
- IBM defined properties. These fields have properties whose names commence with the prefix JMS\_IBM\_.

As a result, XMS applications can exchange messages with WebSphere MQ classes for JMS applications. In each message, some of the header fields and properties are set by the application and others are set by XMS or WebSphere MQ classes for JMS. Some of the fields set by XMS or WebSphere MQ classes for JMS are set when the message is sent, and others when it is received. Header fields and properties are propagated with a message through a messaging server where appropriate. They are made available to any application that receives the message.

## Preventing applications from using newer XMS version

---

By default, when a newer XMS version is installed, the applications using the previous version automatically switch to the newer version without having to recompile.

### **About this task**

The multiple versions coexistence feature ensures that installation of a newer XMS version does not overwrite the previous XMS version. Instead, multiple instances of similar XMS .NET assemblies coexist in the Global Assembly Cache (GAC), but have different version numbers. Internally, the GAC uses a policy file to route the application calls to the latest version of XMS. Applications run without a need for recompilation and can use new features available in the newer XMS .NET version.

However, if an application is required to use the older XMS version, set the `publisherpolicy` attribute to `no` in the application configuration file.

**Note:** An application configuration file is a file with a name that consists of the name of the executable program to which the file relates, with the suffix `.config`. For example, the application configuration file for `text.exe` would have the name `text.exe.config`.

At any time, however, all the applications of a system use the same version of XMS .NET.

## Setting up the messaging server environment

---

This chapter describes how to set up the messaging server environment to allow XMS applications to connect to a server.

For applications that connect to a WebSphere MQ queue manager, the WebSphere MQ client (or queue manager for bindings mode) is required.

There are currently no prerequisites for applications that use a real-time connection to a broker.

You must set up the messaging server environment before running any XMS applications, including the sample applications provided with XMS.

This chapter contains the following sections:

- [“Configuring the queue manager and broker for an application that connects to a WebSphere MQ queue manager” on page 11](#)
- [“Configuring a broker for an application that uses a real-time connection to a broker” on page 13](#)
- [“Configuring the service integration bus for an application that connects to a WebSphere Service Integration Bus” on page 14](#)

### Related concepts

#### [JNDI Lookup Web service](#)

To access a COS naming directory from XMS, a JNDI Lookup Web service must be deployed on a WebSphere Service Integration Bus server. This Web service translates the Java information from the COS naming service into a form that XMS applications can read.

#### [Using the XMS sample applications](#)

Use the sample applications supplied with XMS to verify your installation and messaging server setup, and to help you build your own applications. The samples provide an overview of the common features of each API.

### Related tasks

#### [Installing Message Service Client for .NET using the installation wizard](#)

The installation uses an InstallShield X/Windows MSI installer. Two setup options are available, so that you can choose either a complete or a custom installation.

### Related reference

#### [Prerequisites for XMS applications connecting to WebSphere MQ](#)

Some prerequisites apply if your XMS application connects to WebSphere MQ.

## Configuring the queue manager and broker for an application that connects to a WebSphere MQ queue manager

This section assumes that you are using WebSphere MQ version 7.0, or later. Before you can run an application that connects to a WebSphere MQ queue manager, you must configure the queue manager. For a publish/subscribe application, some additional configuration is required if you are using Queued publish/subscribe interface.

### Before you begin

XMS works with IBM Integration Bus or WebSphere Message Broker version 6.1 or later

Before starting this task, you perform the following steps:

- Make sure that your application has access to a queue manager that is running.
- If your application is a publish/subscribe application and uses Queued publish/subscribe interface, make sure that "PSMODE" attribute is set to "ENABLED" on the queue manager.
- Make sure that your application uses a connection factory whose properties are set appropriately to connect to the queue manager. If your application is a publish/subscribe application, make sure that the

appropriate connection factory properties are set for using the broker. For more information about the properties of a connection factory, [“Properties of ConnectionFactory” on page 175](#).

## About this task

You configure the queue manager and broker to run XMS applications in the same way that you configure the queue manager and queued publish/subscribe interface to run WebSphere MQ JMS applications. The following steps summarize what you need to do.

## Procedure

1. On the queue manager, create the queues that your application needs.

For an overview of how you create queues, see [Defining queues](#).

If your application is a publish/subscribe application and uses Queued publish/subscribe interface that needs access to WebSphere MQ classes for JMS system queues, wait until [Step 4a](#) before creating the queues.

2. Grant the user ID associated with your application the authority to connect to the queue manager, and the appropriate authority to access the queues.

For an overview about authorization, see [Security](#). If your application connects to the queue manager in client mode, see also [Clients and servers](#).

3. If your application connects to the queue manager in client mode, make sure that a server connection channel is defined at the queue manager and that a listener is started.

You do not need to perform this step for each application that connects to the queue manager. One server connection channel definition and one listener can support all the applications that connect in client mode.

4. If your application is a publish/subscribe application, and uses Queued publish/subscribe interface, perform the following steps.

- a) On the queue manager, create the WebSphere MQ classes for JMS system queues by running the script of MQSC commands that are supplied with WebSphere MQ. Make sure that the user ID associated with the WebSphere Message Broker has the authority to access the queues.

For information about where to find the script and how to run it, see [Using Java](#).

Perform this step only once for the queue manager. The same set of WebSphere MQ classes for JMS system queues can support all XMS and WebSphere MQ classes for JMS applications that connect to the queue manager.

- b) Grant the user ID associated with your application the authority to access the WebSphere MQ classes for JMS system queues.

For information about what authorities the user ID needs, see [Using JMS](#).

- c) For a broker of IBM Integration Bus, create and deploy a message flow to service the queue where applications send messages that they publish.

The basic message flow comprises an MQInput message processing node to read the published messages and a Publication message processing node to publish the messages.

For information about how to create and deploy a message flow, see the [IBM Integration Bus product documentation](#)

You do not need to perform this step if a suitable message flow is already deployed at the broker.

## Results

You can now start your application.

## Related tasks

[Configuring a broker for an application that uses a real-time connection to a broker](#)

Before you can run an application that uses a real-time connection to a broker, you must configure that broker.

#### [Configuring the service integration bus for an application that connects to a WebSphere Service Integration Bus](#)

Before you can run an application that connects to a WebSphere Service Integration Bus, you must configure the service integration bus in the same way that you configure the service integration bus to run JMS applications that use the default messaging provider.

#### [Installing Message Service Client for .NET using the installation wizard](#)

The installation uses an InstallShield X/Windows MSI installer. Two setup options are available, so that you can choose either a complete or a custom installation.

#### **Related reference**

##### [Prerequisites for XMS applications connecting to WebSphere MQ](#)

Some prerequisites apply if your XMS application connects to WebSphere MQ.

## **Configuring a broker for an application that uses a real-time connection to a broker**

Before you can run an application that uses a real-time connection to a broker, you must configure that broker.

### **Before you begin**

Before starting this task, you perform the following steps:

- Make sure that your application has access to a broker that is running.
- Make sure that your application uses a connection factory whose properties are set appropriately for a real-time connection to a broker. For more information about the properties of a connection factory, see [“Properties of ConnectionFactory”](#) on page 175.

### **About this task**

You configure a broker to run XMS applications in the same way that you configure a broker to run WebSphere MQ classes for JMS applications. The following steps summarize what you need to do:

### **Procedure**

1. Create and deploy a message flow to read messages from the TCP/IP port on which a broker is listening and publish the messages.

You can do this in either of the following ways:

- Create a message flow that contains a **Real-timeOptimizedFlow** message processing node.
- Create a message flow that contains a **Real-timeInput** message processing node and a Publication message processing node.

You must configure the **Real-timeOptimizedFlow** or **Real-timeInput** node to listen on the port used for real-time connections. In XMS, the default port number for real-time connections is 1506.

You do not need to perform this step if a suitable message flow is already deployed at the broker.

2. If you require messages to be delivered to your application using WebSphere MQ Multicast Transport, configure the broker to enable multicast. Configure the topics that must be multicast enabled, specifying a reliable quality of service for those topics requiring reliable multicast.
3. If your application supplies a user ID and a password when it connects to the broker, and you want the broker to authenticate your application using this information, configure the user name server and the broker for simple telnet-like password authentication.

## Results

You can now start your application.

### Related tasks

[Configuring the queue manager and broker for an application that connects to a WebSphere MQ queue manager](#)

This section assumes that you are using WebSphere MQ version 7.0, or later. Before you can run an application that connects to a WebSphere MQ queue manager, you must configure the queue manager. For a publish/subscribe application, some additional configuration is required if you are using Queued publish/subscribe interface.

[Configuring the service integration bus for an application that connects to a WebSphere Service Integration Bus](#)

Before you can run an application that connects to a WebSphere Service Integration Bus, you must configure the service integration bus in the same way that you configure the service integration bus to run JMS applications that use the default messaging provider.

[Installing Message Service Client for .NET using the installation wizard](#)

The installation uses an InstallShield X/Windows MSI installer. Two setup options are available, so that you can choose either a complete or a custom installation.

### Related reference

[Prerequisites for XMS applications connecting to WebSphere MQ](#)

Some prerequisites apply if your XMS application connects to WebSphere MQ.

## Configuring the service integration bus for an application that connects to a WebSphere Service Integration Bus

Before you can run an application that connects to a WebSphere Service Integration Bus, you must configure the service integration bus in the same way that you configure the service integration bus to run JMS applications that use the default messaging provider.

### Before you begin

Before starting this task, you must do the following steps:

- Make sure that a messaging bus is created and that your server is added to the bus as a bus member.
- Make sure that your application has access to a service integration bus that contains at least one messaging engine that is running.
- If HTTP operation, is required then an HTTP messaging engine inbound transport channel must be defined. By default, channels for SSL and TCP are defined during the server installation.
- Make sure that your application uses a connection factory whose properties are set appropriately to connect to the service integration bus using a bootstrap server. The minimum information required is:
  - The provider endpoint, which describes the location and protocol to use when negotiating a connection to the messaging server (that is, via the bootstrap server). In its simplest form, for a server installed with default settings, the provide endpoint can be set to the hostname of the server.
  - The name of the bus through which messages are sent.

For more information about the properties of a connection factory, see [“Properties of ConnectionFactory”](#) on page 175.

### About this task

Any queue or topic spaces that you require must be defined. By default a topic space called Default.Topic.Space is defined during the server installation but, if you require further topic spaces, you must create these topic spaces yourself. You do not need to predefine individual topics within a topic space, since the server instantiates these individual topics dynamically as required.

The following steps summarize what you need to do.

## Procedure

1. Create the queues that your application needs for point-to-point messaging.
2. Create any additional topic spaces that your application needs for publish/subscribe messaging.

## Results

You can now start your application.

### Related tasks

[Configuring the queue manager and broker for an application that connects to a WebSphere MQ queue manager](#)

This section assumes that you are using WebSphere MQ version 7.0, or later. Before you can run an application that connects to a WebSphere MQ queue manager, you must configure the queue manager. For a publish/subscribe application, some additional configuration is required if you are using Queued publish/subscribe interface.

[Configuring a broker for an application that uses a real-time connection to a broker](#)

Before you can run an application that uses a real-time connection to a broker, you must configure that broker.

[Installing Message Service Client for .NET using the installation wizard](#)

The installation uses an InstallShield X/Windows MSI installer. Two setup options are available, so that you can choose either a complete or a custom installation.

### Related reference

[Prerequisites for XMS applications connecting to WebSphere MQ](#)

Some prerequisites apply if your XMS application connects to WebSphere MQ.

## Installing Message Service Client for .NET using the installation wizard

The installation uses an InstallShield X/Windows MSI installer. Two setup options are available, so that you can choose either a complete or a custom installation.

### About this task

To install Message Service Client for .NET on Windows, follow this procedure.

### Procedure

1. If you are installing from a SupportPac complete the following steps, otherwise proceed directly to step “2” on page 15.
  - a) On Windows, log on as an administrator.
  - b) Run the dotNETClientsetup.exe installer.
2. Wait for the installation wizard to open and display the following message:

```
Welcome to IBM Message Service Client for
.NET installation wizard
```

Click **Next**.

The wizard might ask you to read the license agreement.

3. If you are asked to read the license agreement and you accept the terms of the license agreement, click **I accept the terms in the license agreement**, and then click **Next**.

The installation wizard asks you to choose the setup type that best suits your needs.

4. Select the type of setup you require:
  - To install all program features, and install them in the default installation directory, click **Complete**.
  - To choose which features you want to install, and specify where they are installed, click **Custom**.
5. Click **Next**.

If you select the complete installation option, the installation wizard displays a message that it is ready to begin installation, as described in step “8” on page 16. If you select the custom installation option, the installation wizard asks you to select the features that you want to install and you must complete step “6” on page 16 and step “7” on page 16 before moving on to step “8” on page 16.

6. For a custom installation only, click an icon in the list of features to specify any changes to how you want the Message Service Client for .NET features to be installed. If you do not want to install Message Service Client for .NET in the directory suggested, choose another directory.

If you choose to install Message Service Client for .NET in a directory that does not currently exist, the installation wizard creates the directory for you.

If you want to develop XMS applications, ensure that the **Development Tools and Samples** feature is selected. This feature provides the sample applications, and the libraries and any other files required to compile .NET applications. If you do not select this feature, only the files required to run XMS applications are installed.

7. If you are using the custom installation option, click **Next** after selecting the options you require as described in step “6” on page 16.

The installation wizard displays a message that it is ready to begin installation.

8. Click **Install** to start the installation.

The installation wizard 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 installation wizard has successfully installed IBM Message Service Client
for .NET. Click Finish to exit the wizard.
```

9. Click **Finish** to close the installation wizard.

## Results

You have successfully installed Message Service Client for .NET, which is ready to use.

## What to do next

Before running any XMS applications, including the sample applications provided with XMS, you must set up the messaging server environment, for details see: [“Setting up the messaging server environment” on page 11.](#)

### Related concepts

#### [JNDI Lookup Web service](#)

To access a COS naming directory from XMS, a JNDI Lookup Web service must be deployed on a WebSphere Service Integration Bus server. This Web service translates the Java information from the COS naming service into a form that XMS applications can read.

#### [Setting up the messaging server environment](#)

This chapter describes how to set up the messaging server environment to allow XMS applications to connect to a server.

#### [Using the XMS sample applications](#)

Use the sample applications supplied with XMS to verify your installation and messaging server setup, and to help you build your own applications. The samples provide an overview of the common features of each API.

### Related tasks

#### [Configuring the queue manager and broker for an application that connects to a WebSphere MQ queue manager](#)

This section assumes that you are using WebSphere MQ version 7.0, or later. Before you can run an application that connects to a WebSphere MQ queue manager, you must configure the queue manager.

For a publish/subscribe application, some additional configuration is required if you are using Queued publish/subscribe interface.

[Configuring a broker for an application that uses a real-time connection to a broker](#)

Before you can run an application that uses a real-time connection to a broker, you must configure that broker.

[Configuring the service integration bus for an application that connects to a WebSphere Service Integration Bus](#)

Before you can run an application that connects to a WebSphere Service Integration Bus, you must configure the service integration bus in the same way that you configure the service integration bus to run JMS applications that use the default messaging provider.

### **Related reference**

[Prerequisites for XMS applications connecting to WebSphere MQ](#)

Some prerequisites apply if your XMS application connects to WebSphere MQ.

## **Prerequisites for XMS applications connecting to WebSphere MQ**

Some prerequisites apply if your XMS application connects to WebSphere MQ.

For applications that connect to a WebSphere MQ queue manager, you must install the appropriate WebSphere MQ client libraries on the machine you use to run the XMS application. These libraries are pre-installed on machines with a local queue manager.

For XMS client for .NET, use the client libraries shipped with WebSphere MQ Version 7.0.1.0 or later. These are the *WebSphere MQ classes for .NET*. They enable client mode connections to WebSphere MQ Version 7.0, WebSphere MQ Version 6.0, and WebSphere MQ Version 5.3 queue managers, and bindings mode connections to a local queue manager, if it is also Version 7.0.1.0 or later.

Microsoft .NET Framework Version 2.0 Redistributable Package must be installed on the computer on which XMS is to be installed. If this package is not available, XMS installation will fail. You then need to quit the installation procedure, install Microsoft .NET Framework Version 2.0 Redistributable Package on your computer, and rerun the installation procedure.

On the Microsoft download site, you need to look for dotnetfx.exe for Microsoft .NET Framework Version 2.0 Redistributable Package (x86) and NetFx64.exe for Microsoft .NET Framework version 2.0 Redistributable Package (x64), whichever is applicable.

### **Related concepts**

[Setting up the messaging server environment](#)

This chapter describes how to set up the messaging server environment to allow XMS applications to connect to a server.

### **Related tasks**

[Configuring the queue manager and broker for an application that connects to a WebSphere MQ queue manager](#)

This section assumes that you are using WebSphere MQ version 7.0, or later. Before you can run an application that connects to a WebSphere MQ queue manager, you must configure the queue manager. For a publish/subscribe application, some additional configuration is required if you are using Queued publish/subscribe interface.

[Configuring a broker for an application that uses a real-time connection to a broker](#)

Before you can run an application that uses a real-time connection to a broker, you must configure that broker.

[Configuring the service integration bus for an application that connects to a WebSphere Service Integration Bus](#)

Before you can run an application that connects to a WebSphere Service Integration Bus, you must configure the service integration bus in the same way that you configure the service integration bus to run JMS applications that use the default messaging provider.

[Installing Message Service Client for .NET using the installation wizard](#)

The installation uses an InstallShield X/Windows MSI installer. Two setup options are available, so that you can choose either a complete or a custom installation.

## Using the XMS sample applications

Use the sample applications supplied with XMS to verify your installation and messaging server setup, and to help you build your own applications. The samples provide an overview of the common features of each API.

### Related concepts

#### [JNDI Lookup Web service](#)

To access a COS naming directory from XMS, a JNDI Lookup Web service must be deployed on a WebSphere Service Integration Bus server. This Web service translates the Java information from the COS naming service into a form that XMS applications can read.

#### [Setting up the messaging server environment](#)

This chapter describes how to set up the messaging server environment to allow XMS applications to connect to a server.

### Related tasks

#### [Installing Message Service Client for .NET using the installation wizard](#)

The installation uses an InstallShield X/Windows MSI installer. Two setup options are available, so that you can choose either a complete or a custom installation.

## The sample applications

The sample applications provide an overview of the common features of each API. You can use them to verify your installation and messaging server setup and to help you build your own applications.

If you need help to create your own applications, you can use the sample applications as a starting point. Both the source and a compiled version are provided for each application. Review the sample source code and identify the key steps to create each required object for your application (ConnectionFactory, Connection, Session, Destination, and a Producer, or a Consumer, or both), and to set any specific properties that are needed to specify how you want your application to work. For more information, see [“Writing XMS applications” on page 21](#). The samples are subject to change in future releases of XMS.

The following table shows the three sets of sample applications (one for each API) that are supplied with XMS.

Name of sample	Description
SampleConsumerCS	A message consumer application that takes messages from a queue or subscribes to a topic.
SampleProducerCS	A message producer application that produces messages to a queue or on a topic.
SampleConfigCS	A configuration application that you can use to create an administered object repository that is file-based. The application contains a connection factory and destination for your particular connection settings. This administered object repository can then be used with each of the sample consumer and producer applications.

The samples that support the same functions in the various APIs have syntactical differences.

- The sample message consumer and producer applications both support the following functions:
  - Connections to WebSphere MQ, IBM Integration Bus (by using a real-time connection to a broker), and a WebSphere Service Integration Bus
  - Administered object repository lookups by using the initial context interface

- Connections to queues (WebSphere MQ and WebSphere Service Integration Bus) and topics (WebSphere MQ, real-time connection to a broker, and WebSphere Service Integration Bus)
- Base, byte, map, object, stream, and text messages
- The sample message consumer application supports synchronous and asynchronous receive modes, and SQL Selector statements.
- The sample message producer application supports persistent and non-persistent delivery modes.

## Operating modes

The samples can operate in one of two modes:

### Simple mode

You can run the samples with the minimum user input.

### Advanced mode

You can customize more finely the way in which the samples operate.

All the samples are compatible and can therefore operate across languages.

### Related concepts

#### Building your own applications

You build your own applications like you build the sample applications.

### Related tasks

#### Running the sample applications

You can run the .NET sample applications interactively in either simple or advanced mode, or noninteractively by using auto-generated or customized response files.

#### Building the .NET sample applications

When you build a sample .NET application, an executable version of your chosen sample is created.

## Running the sample applications

You can run the .NET sample applications interactively in either simple or advanced mode, or noninteractively by using auto-generated or customized response files.

### Before you begin

Before running any of the supplied sample applications, you must first set up the messaging server environment so that the applications can connect to a server. See [“Setting up the messaging server environment” on page 11](#).

### Procedure

To run a .NET sample application, complete the following steps:

**Tip:** When you are running a sample application, type ? at any time to get help about what to do next.

1. Select the mode in which you want to run the sample application.

Type either `Advanced` or `Simple`.

2. Answer the questions.

To select the default value, which is shown in the brackets at the end of the question, press Enter. To select a different value, type the appropriate value, and press Enter.

Here is an example question:

```
Enter connection type [wpm]:
```

In this case, the default value is `wpm` (connection to a WebSphere Service Integration Bus).

## Results

When you run the sample applications, response files are generated automatically in the current working directory. Response file names are in the format *connection\_type-sample\_type.rsp*; for example, *wpm-producer.rsp*. If required, you can use the generated response file to rerun the sample application with the same options, so that you do not have to enter the options again.

### Related concepts

[The sample applications](#)

The sample applications provide an overview of the common features of each API. You can use them to verify your installation and messaging server setup and to help you build your own applications.

### Related tasks

[Building the .NET sample applications](#)

When you build a sample .NET application, an executable version of your chosen sample is created.

## Building the .NET sample applications

When you build a sample .NET application, an executable version of your chosen sample is created.

### Before you begin

Install the appropriate compiler. This task assumes that you have Visual Studio 2012 installed, and that you are familiar with using it.

### Procedure

To build a .NET sample application, complete the following steps:

1. Click the `Samples.sln` solution file provided with the .NET samples.
2. Right-click the solution `Samples` in the Solution Explorer window and select **Build Solution**.

### Results

An executable program is created in the appropriate subfolder of the sample, either `bin/Debug` or `bin/Release`, depending on the configuration that you have chosen. This program has the same name as the folder, with a suffix of `CS`. For example, if you are building the C# version of the message producer sample application, `SampleProducerCS.exe` is created in the `SampleProducer` folder.

### Related concepts

[The sample applications](#)

The sample applications provide an overview of the common features of each API. You can use them to verify your installation and messaging server setup and to help you build your own applications.

[“Building your own applications” on page 43](#)

You build your own applications like you build the sample applications.

### Related tasks

[Running the sample applications](#)

You can run the .NET sample applications interactively in either simple or advanced mode, or noninteractively by using auto-generated or customized response files.

## Developing XMS applications

---

This chapter provides information that you might find useful when writing XMS applications.

For information about writing XMS applications, refer to the following topics:

## Writing XMS applications

This chapter provides information to help you when writing XMS applications.

This chapter contains general concepts for writing XMS applications. See also [“Writing XMS .NET applications” on page 44](#) for information specific to creating .NET applications.

This chapter contains the following sections:

- [“The threading model” on page 21](#)
- [“ConnectionFactories and Connection objects” on page 22](#)
- [“Sessions” on page 25](#)
- [“Destinations” on page 29](#)
- [“Message producers” on page 33](#)
- [“Message consumers” on page 34](#)
- [“Queue browsers” on page 38](#)
- [“Requestors” on page 38](#)
- [“Object Deletion” on page 38](#)
- [“XMS primitive types” on page 39](#)
- [“Implicit conversion of a property value from one data type to another” on page 40](#)
- [“Iterators” on page 43](#)
- [“Coded character set identifiers” on page 43](#)
- [“XMS error and exception codes” on page 43](#)
- [“Building your own applications” on page 43](#)

### Related concepts

Writing XMS .NET applications

This chapter provides information to help you when writing XMS.NET applications.

### Related reference

[.NET interfaces](#)

This section documents the .NET class interfaces and their properties and methods.

## The threading model

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 on only a single thread at any one time.

Exceptions to these rules are indicated by entries labeled "Thread context" in the interface definitions of the methods in the API reference chapters.

### ***Error conditions that can be handled at run time***

Return codes from API calls are error conditions that can be handled at run time. The way in which you deal with this type of error depends on whether you are using the C or C++ API.

### How to detect errors at run time

If an application calls a C API function and the call fails, a response with a return code other than XMS\_OK is returned with an XMS error block containing more information about the reason for the failure.

The C++ API throws an exception when a method is used.

An application uses an exception listener to be notified asynchronously of a problem with a connection. The exception listener is supplied to, and is initialized using, the XMS C or C++ API.

## How to handle errors at run time

Some error conditions are an indication that some resource is unavailable, and the action that an application can take depends on the XMS function that the application is calling. For example, if a connection fails to connect to the server, then the application may wish to retry periodically until a connection is made. An XMS error block or exception might not contain enough information to determine what action to take, and, in these situations, there is often a linked error block or exception that contains more specific diagnostic information.

In the C API, always test for a response with a return code other than XMS\_OK, and always pass an error block on the API call. The action taken usually depends on which API function is the application using.

In the C++ API, always include calls to methods in a try block and, to catch all types of XMS exception, specify the Exception class in the catch construct.

The exception listener is an asynchronous error condition path that can be started at any time. When the exception listener function is started, on its own thread, it is usually an indication of a more severe failure than a normal XMS API error condition. Any appropriate action may be taken, but you must be careful to follow the rules for the XMS threading model as described in [“The threading model” on page 21](#).

## ConnectionFactory and Connection objects

A ConnectionFactory object provides a template that an application uses to create a Connection object. The application uses the Connection object to create a Session object.

For .NET, an XMS application first uses an XMSFactoryFactory object to get a reference to a ConnectionFactory object that is appropriate to the required type of protocol. This ConnectionFactory object can then produce connections only for that protocol type.

An XMS application can create multiple connections, and a multithreaded application can use a single Connection object concurrently on multiple threads. A Connection object encapsulates a communications connection between an application and a messaging server.

A connection serves several purposes:

- When an application creates a connection, the application can be authenticated.
- An application can associate a unique client identifier with a connection. The client identifier is used to support durable subscriptions in the publish/subscribe domain. The client identifier can be set in two ways:

The preferred way of assigning a connections client identifier, is to configure in a client-specific ConnectionFactory object using properties and transparently assign it to the connection it creates.

An alternative way of assigning a client identifier is to use a provider-specific value that is set on the Connection object. This value does not override the identifier that has been administratively configured. It is provided for the case where no administratively specified identifier exists. If an administratively specified identifier does exist, an attempt to override it with a provider-specific value causes an exception to be thrown. If an application explicitly sets an identifier, it must do it immediately after creating the connection and before any other action on the connection is taken; otherwise, an exception is thrown.

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

Creating a connection is relatively expensive in terms of system resources because it involves establishing a communications connection, and it might also involve authenticating the application.

### Related tasks

[Creating administered objects](#)

The ConnectionFactory and Destination object definitions that XMS applications require to make a connection to a messaging server must be created using the appropriate administrative tools.

### **Related reference**

#### ConnectionFactory (for the .NET interface)

An application uses a connection factory to create a connection.

#### Properties of ConnectionFactory

An overview of the properties of the ConnectionFactory object, with links to more detailed reference information.

#### IDestination (for the .NET interface)

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

#### Properties of Destination

An overview of the properties of the Destination object, with links to more detailed reference information.

### ***Connection started and stopped mode***

A connection can operate in either started or stopped mode.

When an application creates a connection, the connection is in stopped mode. When the connection is in stopped mode, the application can initialize sessions, and it can send messages but cannot receive them, either synchronously or asynchronously.

An application can start a connection by calling the `Start Connection` method. When the connection is in started mode, the application can send and receive messages. The application can then stop and restart the connection by calling the `Stop Connection` and `Start Connection` methods.

### **Related concepts**

#### Connection closure

An application closes a connection by calling the `Close Connection` method.

#### Exception Handling

If an application uses a connection only to consume messages asynchronously it learns about a problem with the connection only by using an exception listener.

#### Connection to a WebSphere service integration bus

An XMS application can connect to a WebSphere Service Integration Bus either by using a direct TCP/IP connection or by using HTTP over TCP/IP.

### ***Connection closure***

An application closes a connection by calling the `Close Connection` method.

When an application closes a connection, XMS performs the following actions:

- It closes all the sessions associated with the connection and deletes certain objects associated with these sessions. For more information about which objects are deleted, see [“Object Deletion” on page 38](#). At the same time, XMS rolls back any transactions currently in progress within the sessions.
- It ends the communications connection with the messaging server.
- It releases the memory and other internal resources used by the connection.

XMS does not acknowledge the receipt of any messages that it has failed to acknowledge during a session, prior to closing the connection. For more information about acknowledging the receipt of messages, see [“Message acknowledgement” on page 26](#).

### **Related concepts**

#### Connection started and stopped mode

A connection can operate in either started or stopped mode.

#### Exception Handling

If an application uses a connection only to consume messages asynchronously it learns about a problem with the connection only by using an exception listener.

#### Connection to a WebSphere service integration bus

An XMS application can connect to a WebSphere Service Integration Bus either by using a direct TCP/IP connection or by using HTTP over TCP/IP.

### **Exception Handling**

If an application uses a connection only to consume messages asynchronously it learns about a problem with the connection only by using an exception listener.

XMS .NET exceptions are all derived from System.Exception. For more information, see [“Error handling in .NET”](#) on page 48.

#### **Related concepts**

##### Connection started and stopped mode

A connection can operate in either started or stopped mode.

##### Connection closure

An application closes a connection by calling the Close Connection method.

##### Connection to a WebSphere service integration bus

An XMS application can connect to a WebSphere Service Integration Bus either by using a direct TCP/IP connection or by using HTTP over TCP/IP.

### **Connection to a WebSphere service integration bus**

An XMS application can connect to a WebSphere Service Integration Bus either by using a direct TCP/IP connection or by using HTTP over TCP/IP.

The HTTP protocol can be used in situations where a direct TCP/IP connection is not possible. One common situation is when communicating through a firewall, such as when two enterprises exchange messages. Using HTTP to communicate through a firewall is often referred to as *HTTP tunneling*. HTTP tunneling, however, is inherently slower than using a direct TCP/IP connection because HTTP headers add significantly to the amount of data that is transferred, and because the HTTP protocol requires more communication flows than TCP/IP.

To create a TCP/IP connection, an application can use a connection factory whose `XMSC_WPM_TARGET_TRANSPORT_CHAIN` property is set to `XMSC_WPM_TARGET_TRANSPORT_CHAIN_BASIC`. This is the default value of the property. If the connection is created successfully, the `XMSC_WPM_CONNECTION_PROTOCOL` property of the connection is set to `XMSC_WPM_CP_TCP`.

To create a connection that uses HTTP, an application must use a connection factory whose `XMSC_WPM_TARGET_TRANSPORT_CHAIN` property is set to the name of an inbound transport chain, that is configured to use an HTTP transport channel. If the connection is created successfully, the `XMSC_WPM_CONNECTION_PROTOCOL` property of the connection is set to `XMSC_WPM_CP_HTTP`. For information about how to configure transport chains, see [Configuring transport chains](#) in the WebSphere(r) Application Server product documentation.

An application has a similar choice of communication protocols when connecting to a bootstrap server. The `XMSC_WPM_PROVIDER_ENDPOINTS` property of a connection factory is a sequence of one or more endpoint addresses of bootstrap servers. The bootstrap transport chain component of each endpoint address can be either `XMSC_WPM_BOOTSTRAP_TCP`, for a TCP/IP connection to a bootstrap server or `XMSC_WPM_BOOTSTRAP_HTTP`, for a connection that uses HTTP.

#### **Related concepts**

##### Connection started and stopped mode

A connection can operate in either started or stopped mode.

##### Connection closure

An application closes a connection by calling the Close Connection method.

##### Exception Handling

If an application uses a connection only to consume messages asynchronously it learns about a problem with the connection only by using an exception listener.

### **Related tasks**

#### Creating administered objects

The ConnectionFactory and Destination object definitions that XMS applications require to make a connection to a messaging server must be created using the appropriate administrative tools.

### **Related reference**

#### IConnectionFactory (for the .NET interface)

An application uses a connection factory to create a connection.

#### Properties of ConnectionFactory

An overview of the properties of the ConnectionFactory object, with links to more detailed reference information.

#### IDestination (for the .NET interface)

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

#### Properties of Destination

An overview of the properties of the Destination object, with links to more detailed reference information.

## **Sessions**

A session is a single threaded context for sending and receiving messages.

An application can use a session to create messages, message producers, message consumers, queue browsers, and temporary destinations. An application can also use a session to run local transactions.

An application can create multiple sessions, where each session produces and consumes messages independently of the other sessions. If two message consumers in separate sessions (or even in the same session) subscribe to the same topic, each receives a copy of any message published on that topic.

Unlike a Connection object, a Session object cannot be used concurrently on different threads. Only the Close Session method of a Session object can be called from a thread other than the one that the Session object is using at the time. The Close Session method ends a session and releases any system resources allocated to the session.

If an application must process messages concurrently on more than one thread, the application must create a session on each thread, and then use that session for any send or receive operation within that thread.

### ***Transacted sessions***

XMS applications can run local transactions. A *local transaction* is a transaction that involves changes only to the resources of the queue manager or service integration bus to which the application is connected.

The information in this section is relevant only if an application connects to a WebSphere MQ queue manager or a WebSphere service integration bus. The information is not relevant for a real-time connection to a broker.

To run local transactions, an application must first create a transacted session by calling the Create Session method of a Connection object, specifying as a parameter that the session is transacted. Subsequently, all messages sent and received within the session are grouped into a sequence of transactions. A transaction ends when the application commits or rolls back the messages it has sent and received since the transaction began.

To commit a transaction, an application calls the Commit method of the Session object. When a transaction is committed, all messages sent within the transaction become available for delivery to other applications, and all messages received within the transaction are acknowledged so that the messaging server does not attempt to deliver them to the application again. In the point-to-point domain, the messaging server also removes the received messages from their queues.

To roll back a transaction, an application calls the Rollback method of the Session object. When a transaction is rolled back, all messages sent within the transaction are discarded by the messaging server, and all messages received within the transaction become available for delivery again. In the point-to-point domain, the messages that were received are put back on their queues and become visible to other applications again.

A new transaction starts automatically when an application creates a transacted session or calls the Commit or Rollback method. Therefore, a transacted session always has an active transaction.

When an application closes a transacted session, an implicit rollback occurs. When an application closes a connection, an implicit rollback occurs for all the connection's transacted sessions.

A transaction is wholly contained within a transacted session. A transaction cannot span sessions. This means that it is not possible for an application to send and receive messages in two or more transacted sessions and then commit or roll back all these actions as a single transaction.

### **Related concepts**

#### Message acknowledgement

Every session that is not transacted has an acknowledgement mode that determines how messages received by the application are acknowledged. Three acknowledgement modes are available, and the choice of acknowledgement mode affects the design of the application.

#### Asynchronous message delivery

XMS uses one thread to handle all asynchronous message deliveries for a session. This means that only one message listener function or one `onMessage()` method can run at a time.

#### Synchronous message delivery

Messages are delivered synchronously to an application if the application uses the Receive methods of MessageConsumer objects.

#### Message delivery mode

XMS supports two modes of message delivery.

### ***Message acknowledgement***

Every session that is not transacted has an acknowledgement mode that determines how messages received by the application are acknowledged. Three acknowledgement modes are available, and the choice of acknowledgement mode affects the design of the application.

The information in this section is relevant only if an application connects to a WebSphere MQ queue manager or a WebSphere Service Integration Bus. The information is not relevant for a real-time connection to a broker.

XMS uses the same mechanism for acknowledging the receipt of messages that JMS uses.

If a session is not transacted, the way that messages received by the application are acknowledged is determined by the acknowledgement mode of the session. The three acknowledgement modes are described in the following paragraphs:

#### **XMSC\_AUTO\_ACKNOWLEDGE**

The session automatically acknowledges each message received by the application.

If messages are delivered synchronously to the application, the session acknowledges receipt of a message every time a Receive call completes successfully.

If the application receives a message successfully, but a failure prevents acknowledgement from occurring, the message becomes available for delivery again. The application must therefore be able to handle a message that is redelivered.

#### **XMSC\_DUPS\_OK\_ACKNOWLEDGE**

The session acknowledges the messages received by the application at times it selects.

Using this acknowledgement mode reduces the amount of work the session must do, but a failure that prevents message acknowledgement might result in more than one message becoming available for delivery again. The application must therefore be able to handle messages that are redelivered.

## **XMSC\_CLIENT\_ACKNOWLEDGE**

The application acknowledges the messages it receives by calling the Acknowledge method of the Message class.

The application can acknowledge the receipt of each message individually, or it can receive a batch of messages and call the Acknowledge method only for the last message it receives. When the Acknowledge method is called all messages received since the last time the method was called are acknowledged.

In conjunction with any of these acknowledgement modes, an application can stop and restart the delivery of messages in a session by calling the Recover method of the Session class. Messages whose receipt was previously unacknowledged are redelivered. However, they might not be delivered in the same sequence in which they were previously delivered. In the meantime, higher priority messages might have arrived, and some of the original messages might have expired. In the point-to-point domain, some of the original messages might have been consumed by another application.

An application can determine whether a message is being re-delivered by examining the contents of the JMSRedelivered header field of the message. The application does this by calling the Get JMSRedelivered method of the Message class.

### **Related concepts**

#### Transacted sessions

XMS applications can run local transactions. A *local transaction* is a transaction that involves changes only to the resources of the queue manager or service integration bus to which the application is connected.

#### Asynchronous message delivery

XMS uses one thread to handle all asynchronous message deliveries for a session. This means that only one message listener function or one onMessage () method can run at a time.

#### Synchronous message delivery

Messages are delivered synchronously to an application if the application uses the Receive methods of MessageConsumer objects.

#### Message delivery mode

XMS supports two modes of message delivery.

### ***Asynchronous message delivery***

XMS uses one thread to handle all asynchronous message deliveries for a session. This means that only one message listener function or one onMessage () method can run at a time.

If more than one message consumer in a session is receiving messages asynchronously, and a message listener function or onMessage () method is delivering a message to a message consumer, then any other message consumers that are waiting for the same message must continue to wait. Other messages that are waiting to be delivered to the session must also continue to wait.

If an application requires concurrent delivery of messages, create more than one session so that XMS uses more than one thread to handle asynchronous message delivery. In this way, more than one message listener function or onMessage () method can run concurrently.

A session is not made asynchronous by assigning a message listener to a consumer. A session becomes asynchronous only when the Connection.Start method is called. All synchronous calls are permitted until the Connection.Start method is called. Message delivery to consumers start when the Connection.Start is called.

If synchronous calls, such as creation of a consumer or producer, must be made on an asynchronous session, the Connection.Stop must be called. A session can be resumed by calling the Connection.Start method to start delivery of messages. The only exception to this is the Session message delivery thread, which is the thread that delivers messages to the callback function. This thread can to make any call on session (except a Close call) in the message callback function.

**Note:** In Unmanaged mode, the MQDISC call within a call-back function is not supported by the WMQ.NET client. So, the client application cannot Create or Close sessions within the MessageListener callback in Asynchronous receive mode. Create and dispose the session outside of the MessageListener method.

## Related concepts

### Transacted sessions

XMS applications can run local transactions. A *local transaction* is a transaction that involves changes only to the resources of the queue manager or service integration bus to which the application is connected.

### Message acknowledgement

Every session that is not transacted has an acknowledgement mode that determines how messages received by the application are acknowledged. Three acknowledgement modes are available, and the choice of acknowledgement mode affects the design of the application.

### Synchronous message delivery

Messages are delivered synchronously to an application if the application uses the Receive methods of MessageConsumer objects.

### Message delivery mode

XMS supports two modes of message delivery.

## ***Synchronous message delivery***

Messages are delivered synchronously to an application if the application uses the Receive methods of MessageConsumer objects.

Using the Receive methods, an application can wait a specified period of time for a message, or it can wait indefinitely. Alternatively, if an application does not want to wait for a message, it can use the Receive with No Wait method.

## Related concepts

### Transacted sessions

XMS applications can run local transactions. A *local transaction* is a transaction that involves changes only to the resources of the queue manager or service integration bus to which the application is connected.

### Message acknowledgement

Every session that is not transacted has an acknowledgement mode that determines how messages received by the application are acknowledged. Three acknowledgement modes are available, and the choice of acknowledgement mode affects the design of the application.

### Asynchronous message delivery

XMS uses one thread to handle all asynchronous message deliveries for a session. This means that only one message listener function or one onMessage() method can run at a time.

### Message delivery mode

XMS supports two modes of message delivery.

## ***Message delivery mode***

XMS supports two modes of message delivery.

- *Persistent* messages are delivered once. A messaging server takes special precautions, such as logging the messages, to ensure that persistent messages are not lost in transit, even in the event of a failure.
- *Nonpersistent* messages are delivered no more than once. Nonpersistent messages are less reliable than persistent messages because they can be lost in transit in the event of a failure.

The choice of delivery mode is a trade-off between reliability and performance. Nonpersistent messages are typically transported more quickly than persistent messages.

## Related concepts

### Transacted sessions

XMS applications can run local transactions. A *local transaction* is a transaction that involves changes only to the resources of the queue manager or service integration bus to which the application is connected.

### Message acknowledgement

Every session that is not transacted has an acknowledgement mode that determines how messages received by the application are acknowledged. Three acknowledgement modes are available, and the choice of acknowledgement mode affects the design of the application.

### Asynchronous message delivery

XMS uses one thread to handle all asynchronous message deliveries for a session. This means that only one message listener function or one `onMessage()` method can run at a time.

#### Synchronous message delivery

Messages are delivered synchronously to an application if the application uses the `Receive` methods of `MessageConsumer` objects.

## Destinations

An XMS application uses a `Destination` object to specify the destination of messages that are being sent, and the source of messages that are being received.

An XMS application can either create a `Destination` object at run time, or obtain a predefined destination from the repository of administered objects.

As with a `ConnectionFactory`, the most flexible way for an XMS application to specify a destination is to define it as an administered object. Using this approach, applications written in C, C++, .NET languages, and Java, can share definitions of the destination. The properties of administered `Destination` objects can be changed without changing any code.

For .NET applications, you create a destination by using the `CreateTopic` or `CreateQueue` method. These two methods are available in both the `ISession` and `XMSFactoryFactory` objects in the .NET API. For more information, see [“Destinations in .NET” on page 46](#) and [“IDestination” on page 105](#).

### Related reference

#### IDestination (for the .NET interface)

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

#### Properties of Destination

An overview of the properties of the `Destination` object, with links to more detailed reference information.

### ***Topic uniform resource identifiers***

The topic uniform resource identifier (URI) specifies the name of the topic; it can also specify one or more properties for it.

The URI for a topic begins with the sequence `topic://`, followed by the name of the topic and (optional) a list of name-value pairs that set the remaining topic properties. A topic name cannot be empty.

Here is an example in a fragment of .NET code:

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

For more information about the properties of a topic, including the name and valid values that you can use in a URI, see [“Properties of Destination” on page 181](#).

When specifying a topic URI for use in a subscription, wildcards can be used. The syntax for these wildcards depends on the connection type and broker version; the following options are available:

- WebSphere MQ V7.0 queue manager with Character level wildcard format
- WebSphere MQ V7.0 queue manager with Topic level wild card format
- WebSphere service integration bus

### **WebSphere MQ V7.0 queue manager with Character level wildcard format**

WebSphere MQ V7.0 queue manager with Character level wildcard format uses the following wild card characters:

- \* for 0 or more characters
- ? for 1 character
- % for an escape character

[Table 1 on page 30](#) gives some examples of how to use this wildcard scheme.

Table 1. Example URIs using character level wildcard scheme for WebSphere MQ V7.0 queue manager

Uniform Resource Identifier	Matches	Examples
"topic://Sport*Results"	All topics starting with "Sport" and ending in "Results"	"topic://SportsResults" and "topic://Sport/Hockey/National/Div3/Results"
"topic://Sport?Results"	All topics starting with "Sport" followed by a single character, followed by "Results"	"topic://SportsResults" and "topic://SportXResults"
"topic://Sport/*ball*/Div?/Results*/???"	Topics	"topic://Sport/Football/Div1/Results/2002/Nov" and "topic://Sport/Netball/National/Div3/Results/02/Jan"

## WebSphere MQ V7.0 queue manager with Topic level wild card format

WebSphere MQ V7.0 queue manager with Topic level wild card format uses the following wildcard characters:

- # to match multiple levels
- + to match a single level

Table 2 on page 30 gives some examples of how to use this wildcard scheme.

Table 2. Example URIs using topic level wildcard scheme for WebSphere MQ V7.0 queue manager

Uniform Resource Identifier	Matches	Examples
"topic://Sport+/Results"	All topics with a single hierarchical level name between Sport and Results	"topic://Sport/Football/Results" and "topic://Sport/Ju-Jitsu/Results"
"topic://Sport#/Results"	All topics starting with "Sport/" and ending in "/Results"	"topic://Sport/Football/Results" and "topic://Sport/Hockey/National/Div3/Results"
"topic://Sport/Football/#"	All topics starting with "Sport/Football/"	"topic://Sport/Football/Results" and "topic://Sport/Football/TeamNews/Signings/Managerial"

## WebSphere service integration bus

WebSphere service integration bus uses the following wildcard characters:

- \* to match any characters at one level in the hierarchy
- // to match 0 or more levels
- //. to match 0 or more levels (at the end of a Topic expression)

Table 3 on page 30 gives some examples of how to use this wildcard scheme.

Table 3. Example URIs using wildcard scheme for WebSphere service integration bus

Uniform Resource Identifier	Matches	Examples
"topic://Sport/*ball/Results"	All topics with a single hierarchical level name ending in "ball" between Sport and Results	"topic://Sport/Football/Results" and "topic://Sport/Netball/Results"

Table 3. Example URIs using wildcard scheme for WebSphere service integration bus (continued)

Uniform Resource Identifier	Matches	Examples
"topic://Sport//Results"	All topics starting with "Sport/" and ending in "/Results"	"topic://Sport/Football/Results" and "topic://Sport/Hockey/National/Div3/Results"
"topic://Sport/Football//."	All topics starting with "Sport/Football/"	"topic://Sport/Football/Results" and "topic://Sport/Football/TeamNews/Signings/Managerial"
"topic://Sport/*ball//Results//."	Topics	"topic://Sport/Football/Results" and "topic://Sport/Netball/National/Div3/Results/2002/November"

### Related concepts

#### Queue uniform resource identifiers

The URI for a queue specifies the name of the queue; it can also specify one or more properties of the queue.

#### Temporary destinations

XMS applications can create and use temporary destinations.

#### Destination wildcards

XMS provides support for destination wildcards, ensuring that wildcards can be passed through to the place where they are needed for matching. There is a different wildcard scheme for each server type that XMS can work with.

### Related reference

#### IDestination (for the .NET interface)

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

#### Properties of Destination

An overview of the properties of the Destination object, with links to more detailed reference information.

### Queue uniform resource identifiers

The URI for a queue specifies the name of the queue; it can also specify one or more properties of the queue.

The URI for a queue begins with the sequence queue://, followed by the name of the queue; it might also include a list of name-value pairs that set the remaining queue properties.

For WebSphere MQ queues (but not for WebSphere Application Server default messaging provider queues), the queue manager on which the queue resides may be specified before the queue, with a / separating the queue manager name from the queue name.

If a queue manager is specified, then it must be the one to which XMS is directly connected for the connection using this queue, or it must be accessible from this queue. Remote queue managers are only supported for retrieving messages from queues, not for putting messages onto queues. For full details, refer to the WebSphere MQ queue manager documentation.

If no queue manager is specified, then the extra / separator is optional, and its presence or absence makes no difference to the definition of the queue.

The following queue definitions are all equivalent for a WebSphere MQ queue called QB on a queue manager called QM\_A, to which XMS is directly connected:

```
queue://QB
queue:///QB
queue://QM_A/QB
```

## **Related concepts**

### Topic uniform resource identifiers

The topic uniform resource identifier (URI) specifies the name of the topic; it can also specify one or more properties for it.

### Temporary destinations

XMS applications can create and use temporary destinations.

### Destination wildcards

XMS provides support for destination wildcards, ensuring that wildcards can be passed through to the place where they are needed for matching. There is a different wildcard scheme for each server type that XMS can work with.

## **Related reference**

### IDestination (for the .NET interface)

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

### Properties of Destination

An overview of the properties of the Destination object, with links to more detailed reference information.

## ***Temporary destinations***

XMS applications can create and use temporary destinations.

An application typically uses a temporary destination to receive replies to request messages. To specify the destination where a reply to a request message is to be sent, an application calls the `Set JMSReplyTo` method of the Message object representing the request message. The destination specified on the call can be a temporary destination.

Although a session is used to create a temporary destination, the scope of a temporary destination is actually the connection that was used to create the session. Any of the connection's sessions can create message producers and message consumers for the temporary destination. The temporary destination remains until it is explicitly deleted, or the connection ends, whichever happens first.

When an application creates a temporary queue, a queue is created in the messaging server to which the application is connected. If the application is connected to a queue manager, a dynamic queue is created from the model queue whose name is specified by the `XMSC_WMQ_TEMPORARY_MODEL` property, and the prefix that is used to form the name of the dynamic queue is specified by the `XMSC_WMQ_TEMP_Q_PREFIX` property. If the application is connected to a service integration bus, a temporary queue is created in the bus, and the prefix that is used to form the name of the temporary queue is specified by the `XMSC_WPM_TEMP_Q_PREFIX` property.

When an application that is connected to a service integration bus creates a temporary topic, the prefix that is used to form the name of the temporary topic is specified by the `XMSC_WPM_TEMP_TOPIC_PREFIX` property.

## **Related concepts**

### Topic uniform resource identifiers

The topic uniform resource identifier (URI) specifies the name of the topic; it can also specify one or more properties for it.

### Queue uniform resource identifiers

The URI for a queue specifies the name of the queue; it can also specify one or more properties of the queue.

### Destination wildcards

XMS provides support for destination wildcards, ensuring that wildcards can be passed through to the place where they are needed for matching. There is a different wildcard scheme for each server type that XMS can work with.

## **Related reference**

### IDestination (for the .NET interface)

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

#### Properties of Destination

An overview of the properties of the Destination object, with links to more detailed reference information.

### **Destination wildcards**

XMS provides support for destination wildcards, ensuring that wildcards can be passed through to the place where they are needed for matching. There is a different wildcard scheme for each server type that XMS can work with.

The schemes are:

Type of connection	Wildcard scheme	Description
WebSphere MQ queue manager	*	0 or more characters
	?	1 character
	%	Escape character
Real-time connection to a broker	#	Match multiple levels
	+	Match a single level
WebSphere Service Integration Bus	*	Match any characters at one level in the hierarchy
	//	Match 0 or more levels
	//.	Match 0 or more levels (at the end of a Topic expression)

### **Related concepts**

#### Topic uniform resource identifiers

The topic uniform resource identifier (URI) specifies the name of the topic; it can also specify one or more properties for it.

#### Queue uniform resource identifiers

The URI for a queue specifies the name of the queue; it can also specify one or more properties of the queue.

#### Temporary destinations

XMS applications can create and use temporary destinations.

### **Related reference**

#### IDestination (for the .NET interface)

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

#### Properties of Destination

An overview of the properties of the Destination object, with links to more detailed reference information.

### **Message producers**

In XMS, a message producer can be created either with a valid destination or with no associated destination. When creating a message producer with a null destination, a valid destination needs to be specified when sending a message.

### **Message producers with no associated destination**

In XMS .NET, a message producer can be created with a null destination.

To create a message producer with no associated destination when using the .NET API, NULL must be passed as a parameter into the CreateProducer() method of the ISession object (for example,

`session.CreateProducer(null)`). However a valid destination must be specified when the message is sent.

### ***Message producers with associated destination***

In this scenario, the message producer is created using a valid destination. During the send operation, the destination need not be specified.

## **Message consumers**

Message consumers can be classified as durable and non-durable subscribers and synchronous and asynchronous message consumers.

### ***Durable subscribers***

A durable subscriber is a message consumer that receives all messages published on a topic, including messages published while the subscriber is inactive.

The information in this section is relevant only if an application connects to a WebSphere MQ queue manager or a WebSphere service integration bus. The information is not relevant for a real-time connection to a broker.

To create a durable subscriber for a topic, an application calls the `Create Durable Subscriber` method of a `Session` object, specifying as parameters a name that identifies the durable subscription and a `Destination` object representing the topic. The application can create a durable subscriber with or without a message selector, and it can specify whether the durable subscriber is to receive messages published by its own connection.

The session used to create a durable subscriber must have an associated client identifier. The client identifier is the same as that associated with the connection that is used to create the session; it is specified as described in [“ConnectionFactories and Connection objects”](#) on page 22.

The name that identifies the durable subscription must be unique within the client identifier, and therefore the client identifier forms part of the full, unique identifier of the durable subscription. The messaging server maintains a record of the durable subscription and ensures that all messages published on the topic are retained until they are acknowledged by the durable subscriber or they expire.

The messaging server continues to maintain the record of the durable subscription even after the durable subscriber closes. To reuse a durable subscription that was created previously, an application must create a durable subscriber specifying the same subscription name, and using a session with the same client identifier, as those associated with the durable subscription. Only one session at a time can have a durable subscriber for a particular durable subscription.

The scope of a durable subscription is the messaging server that is maintaining a record of the subscription. If two applications connected to different messaging servers each create a durable subscriber using the same subscription name and client identifier, two completely independent durable subscriptions are created.

To delete a durable subscription, an application calls the `Unsubscribe` method of a `Session` object, specifying as a parameter the name that identifies the durable subscription. The client identifier associated with the session must be the same as that associated with the durable subscription. The messaging server deletes the record of the durable subscription that it is maintaining and does not send any more messages to the durable subscriber.

To change an existing subscription, an application can create a durable subscriber using the same subscription name and client identifier, but specifying a different topic, or message selector (or both). Changing a durable subscription is equivalent to deleting the subscription and creating a new one.

For an application that connects to WebSphere MQ V7.0 queue manager, XMS manages the subscriber queues. Hence the application is not required to specify a subscriber queue. XMS will ignore the subscriber queue if specified.

However for an application that connects to WebSphere MQ V6.0 queue manager, each durable subscriber must have a designated subscriber queue. To specify the name of the subscriber queue for

a topic, set the `XMSC_WMQ_DUR_SUBQ` property of the Destination object representing the topic. The default subscriber queue is `SYSTEM.JMS.D.SUBSCRIBER.QUEUE`.

Durable subscribers connecting to WebSphere MQ V6.0 queue managers can share a single subscriber queue, or each durable subscriber can retrieve its messages from its own exclusive subscriber queue. For a discussion about which approach to adopt for your application, see *WebSphere MQ Using Java*.

Note that you cannot change the subscriber queue for a durable subscription. The only way to change the subscriber queue is to delete the subscription and create a new one.

For an application that connects to a service integration bus, each durable subscriber must have a designated durable subscription home. To specify the durable subscription home for all durable subscribers that use the same connection, set the `XMSC_WPM_DUR_SUB_HOME` property of the ConnectionFactory object that is used to create the connection. To specify the durable subscription home for an individual topic, set the `XMSC_WPM_DUR_SUB_HOME` property of the Destination object representing the topic. A durable subscription home must be specified for a connection before an application can create a durable subscriber that uses the connection. Any value specified for a destination overrides the value specified for the connection.

### ***Non-durable subscribers***

A non-durable subscriber is a message consumer that only receives messages that are published while the subscriber is active. Messages delivered while the subscriber is inactive are lost.

The information in this section is relevant only when you are using publish/subscribe messaging over WebSphere MQ V6.0 queue manager.

If consumer objects are not deleted before or during the closing of the connection, messages can remain on the broker queues for subscribers that are no longer active.

In this situation, the queues can be cleared of these messages using the Cleanup utility provided with WebSphere MQ Classes for JMS. Details of how to use this utility are provided in *WebSphere MQ Using Java*. You may also need to increase the queue depth of the subscriber queue if there are large numbers of messages remaining on this queue.

### ***Synchronous message consumers***

The synchronous message consumer receives the messages from a queue synchronously.

A synchronous message consumer receives one message at a time. When the `Receive(wait interval)` method is used; the call waits only a specified period of time in milliseconds for a message, or until the message consumer is closed.

If the `ReceiveNoWait()` method is used, the synchronous message consumer receives messages without any delay; if the next message is available, it is received immediately, otherwise a pointer to a null Message object is returned.

### ***Asynchronous message consumers***

The asynchronous message consumer receives message from a queue asynchronously. The message listener registered by the application is invoked whenever a new message is available on the queue.

### ***Poison messages in XMS***

A poison message is a message that cannot be processed by a receiving MDB application. If a poison message is encountered, the XMS MessageConsumer object can requeue it according to two queue properties, `BOQUEUE`, and `BOTHRESH`.

In some circumstances, a message delivered to an MDB might be rolled back onto a IBM WebSphere MQ queue. This can happen, for example, if a message is delivered within a unit of work that is subsequently rolled back. A message that is rolled back is generally delivered again, but a badly formatted message might repeatedly cause an MDB to fail and therefore cannot be delivered. Such a message is called a poison message. You can configure IBM WebSphere MQ so that the poison message is automatically transferred to another queue for further investigation or is discarded. For information about how to configure IBM WebSphere MQ in this way, see [Handling poison messages in ASF](#).

Sometimes, a badly formatted message arrives on a queue. In this context, badly formatted means that the receiving application cannot process the message correctly. Such a message can cause the receiving application to fail and to back out this badly formatted message. The message can then be repeatedly delivered to the input queue and repeatedly backed out by the application. These messages are known as poison messages. The XMS MessageConsumer object detects poison messages and reroutes them to an alternative destination.

The IBM WebSphere MQ queue manager keeps a record of the number of times that each message has backed out. When this number reaches a configurable threshold value, the message consumer requeues the message to a named backout queue. If this requeuing fails for any reason, the message is removed from the input queue and either requeued to the dead-letter queue, or discarded.

XMS ConnectionConsumer objects handle poison messages in the same way and using the same queue properties. If multiple connection consumers are monitoring the same queue, it is possible that the poison message may be delivered to an application more times than the threshold value before the requeue occurs. This behavior is due to the way individual connection consumers monitor queues and requeue poison messages.

The threshold value and the name of the back out queue are attributes of a IBM WebSphere MQ queue. The names of the attributes are `BackoutThreshold` and `BackoutRequeueQName`. The queue they apply to is as follows:

- For point-to-point messaging, this is the underlying local queue. This is important when message consumers and connection consumers use queue aliases.
- For publish/subscribe messaging in IBM WebSphere MQ messaging provider normal mode, it is the model queue from which the Topic's managed queue is created.
- For publish/subscribe messaging in IBM WebSphere MQ messaging provider migration mode, it is the CCSUB queue defined on the TopicConnectionFactory object, or the CCDSUB queue defined on the Topic object.

To set the `BackoutThreshold` and `BackoutRequeueQName` attributes, issue the following MQSC command:

```
ALTER QLOCAL(your.queue.name) BOTHRESH(threshold value)
BOQUEUE(your.backout.queue.name)
```

For publish/subscribe messaging, if your system creates a dynamic queue for each subscription, these attribute values are obtained from the WebSphere MQ classes for JMS model queue, `SYSTEM.JMS.MODEL.QUEUE`. To alter these settings, use:

```
ALTER QMODEL(SYSTEM.JMS.MODEL.QUEUE) BOTHRESH(threshold value)
BOQUEUE(your.backout.queue.name)
```

If the backout threshold value is zero, poison message handling is disabled, and poison messages remain on the input queue. Otherwise, when the backout count reaches the threshold value, the message is sent to the named backout queue.

If the backout count reaches the threshold value, but the message cannot go to the backout queue, the message is sent to the dead-letter queue or, if the message is nonpersistent, it is discarded.

This situation occurs if the backout queue is not defined, or if the MessageConsumer object cannot send the message to the backout queue.

## Configuring your system to perform poison message handling

The queue that XMS .NET uses when inquiring the **BOTHRESH** and **BOQNAME** attributes depends on the style of messaging being performed:

- For point-to-point messaging, this is the underlying local queue. This is important when an XMS .NET application is consuming messages from either alias queues or cluster queues.

- For publish/subscribe messaging, a managed queue is created to hold the messages for an application. XMS .NET queries the managed queue to determine the values for the **BOTHRESH** and **BOQNAME** attributes.

The managed queue is created from a model queue associated with the Topic object that the application has subscribed to, and inherits the values of the **BOTHRESH** and **BOQNAME** attributes from the model queue. The model queue that is used depends on whether the receiving application has taken out a durable or non-durable subscription:

- The model queue used for durable subscriptions is specified by the **MDURMDL** attribute of the Topic. The default value of this attribute is `SYSTEM.DURABLE.MODEL.QUEUE`.
- For non-durable subscriptions, the model queue that is used is specified by the **MNDURMDL** attribute. The default value of the **MNDURMDL** attribute is `SYSTEM.NDURABLE.MODEL.QUEUE`.

When inquiring the **BOTHRESH** and **BOQNAME** attributes, XMS .NET:

- Opens the local queue, or the target queue for an alias queue.
- Inquires the **BOTHRESH** and **BOQNAME** attributes.
- Closes the local queue, or the target queue for an alias queue.

The open options that are used when opening a local queue, or the target queue for an alias queue, depend on the version of IBM MQ being used:

- For Version 8.0.0, Fix Pack 13, and earlier, if the local queue, or the target queue for an alias queue, is a cluster queue, then XMS .NET opens the queue with the `MQOO_INPUT_AS_Q_DEF`, `MQOO_INQUIRE` and `MQOO_FAIL_IF QUIESCING` options. This means that the user running the receiving application must have inquire and get access to the local instance of the cluster queue.

XMS .NET opens all other types of local queue with the open options `MQOO_INQUIRE` and `MQOO_FAIL_IF QUIESCING`. In order for XMS .NET to query the values of the attributes, the user running the receiving application must have inquire access on the local queue.

- **V 8.0.0.14** When using XMS .NET from Version 8.0.0, Fix Pack 14, the user running the receiving application must have inquire access on the local queue, regardless of the type of the queue.

To move poison messages to either a backout requeue queue or the queue manager's dead letter queue, you must grant the user running the application `put` and `passall` authorities.

#### *Handling poison messages in ASF*

When you use Application Server Facilities (ASF), the `ConnectionConsumer`, rather than the `MessageConsumer`, processes poison messages. The `ConnectionConsumer` requeues messages according to the `BackoutThreshold` and `BackoutRequeueQName` properties of the queue.

When an application uses `ConnectionConsumers`, the circumstances in which a message is backed out depend on the session that the application server provides:

- When the session is non-transacted, with `AUTO_ACKNOWLEDGE` or `DUPS_OK_ACKNOWLEDGE`, a message is backed out only after a system error, or if the application terminates unexpectedly.
- When the session is non-transacted with `CLIENT_ACKNOWLEDGE`, unacknowledged messages can be backed out by the application server calling `Session.recover()`.

Typically, the client implementation of `MessageListener` or the application server calls `Message.acknowledge()`. `Message.acknowledge()` acknowledges all messages delivered on the session so far.

- When the session is transacted, unacknowledged messages can be backed out by the application server calling `Session.rollback()`.

## Queue browsers

An application uses a queue browser to browse messages on a queue without removing them.

To create a queue browser, an application calls the Create Queue Browser method of an ISession object, specifying as a parameter a Destination object that identifies the queue to be browsed. The application can create a queue browser with or without a message selector.

After creating a queue browser, the application can call the GetEnumerator method of the IQueueBrowser object to get a list of the messages on the queue. The method returns an enumerator that encapsulates a list of Message objects. The order of the Message objects in the list is the same as the order in which the messages would be retrieved from the queue. The application can then use the enumerator to browse each message in turn.

The enumerator is updated dynamically as messages are put on the queue and removed from the queue. Each time the application calls IEnumerator.MoveNext() to browse the next message on the queue, the message reflects the current contents of the queue.

An application can call the GetEnumerator method more than once for a given queue browser. Each call returns a new enumerator. The application can therefore use more than one enumerator to browse the messages on a queue and maintain multiple positions within the queue.

An application can use a queue browser to search for a suitable message to remove from a queue, and then use a message consumer with a message selector to remove the message. The message selector can select the message according to the value of the JMSMessageID header field. For information about this and other JMS message header fields, see [“Header fields in an XMS message” on page 69](#).

## Requestors

An application uses a requestor to send a request message and then to wait for and to receive the reply.

Many messaging applications are based on algorithms that send a request message and then wait for a reply. XMS provides a class called Requestor to help with the development of this style of application.

To create a requestor, an application calls the Create Requestor constructor of the Requestor class, specifying as parameters a Session object and a Destination object that identifies where request messages are to be sent. The session must not be transacted nor have an acknowledgement mode of XMSC\_CLIENT\_ACKNOWLEDGE. The constructor automatically creates a temporary queue or topic where reply messages are to be sent.

After creating a requestor, the application can call the Request method of the Requestor object to send a request message and then wait for, and receive, a reply from the application that receives the request message. The call waits until the reply is received or until the session ends, whichever occurs first. Only one reply is required by the requestor for each request message.

When the application closes the requestor, the temporary queue or topic is deleted. The associated session, however, does not close.

## Object Deletion

When an application deletes an XMS object that it created, XMS releases the internal resources that have been allocated to the object.

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. 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 have no function independent from the object. These objects are shown in [Table 4 on page 39](#).

**Note:** 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.

<i>Table 4. Objects that are deleted automatically</i>		
<b>Deleted object</b>	<b>Method</b>	<b>Dependent objects that are deleted automatically</b>
Connection	Close Connection	ConnectionMetaData and Session objects
Session	Close Session	MessageConsumer, MessageProducer, QueueBrowser, and Requestor objects

## Managed WebSphere MQ XA transactions through XMS

Managed WebSphere MQ XA transactions can be used through XMS.

To use XA transactions through XMS, a transacted session has to be created. When XA transaction is in use, the transaction control is through Distributed Transaction Coordinator (DTC) global transactions and it is not through XMS sessions. When using XA transactions, `Session.commit` or `Session.rollback` cannot be issued on the XMS session. Instead, use the `Transscope.Commit` or `Transscope.Rollback` DTC methods commit or rollback the transactions. If a session is used for XA transaction, the producer or consumer that are created using the session must be a part of the XA transaction. They cannot be used for any operations outside the XA transaction scope. They cannot be used for operations like `Producer.send` or `Consumer.receive` outside the XA transaction.

An `IllegalStateException` exception object is thrown if

- XA transacted session is used for `Session.commit` or `Session.rollback`.
- Producer or consumer objects which are once used in XA transacted session are used outside the XA transaction scope.

The XA transactions are not supported in asynchronous consumers.

### **Note:**

1. A close might be issued on the `Producer`, `Consumer`, `Session`, or `Connection` object before the XA transaction commit. In which cases the messages in the transaction are rolled back. Similarly, if the connection is broken before the XA transaction commit, all the messages in the transaction are rolled back. For a `Producer` object, a rollback means that the messages are not put on the queue. For a `Consumer` object, a rollback means that the messages remain on the queue.
2. If a `Producer` object puts a message with `TimeToLive` in the `TransactionScope` and a commit is issued after the time is elapsed, the message can expire before the commit is issued. In this case, the message is not made available to `Consumer` objects.
3. `Session` objects are not supported across threads. The use of transactions with `Session` objects that are shared across threads is not supported.

## XMS primitive types

XMS provides equivalents of the eight Java primitive types (byte, short, int, long, float, double, char, and boolean). This allows the interchange of messages between XMS and JMS without data becoming lost or corrupted.

Table 5 on page 40 lists the Java equivalent data type, size, and minimum and maximum value of each XMS primitive type.

Table 5. XMS data types and their Java equivalents

XMS data type	Compatible Java data type	Size	Minimum value	Maximum value
System.Boolean	boolean	32 bits	false	true
System.SBYTE	byte	8 bits	$-2^7$ (-128)	$2^7-1$ (127)
System.BYTE	byte	8 bits	$-2^7$ (-128)	$2^7-1$ (127)
System.CHAR	byte	8 bits	$-2^7$ (-128)	$2^7-1$ (127)
System.Int16	short	16 bits	$-2^{15}$ (-32768)	$2^{15}-1$ (32767)
System.Int32	int	32 bits	$-2^{31}$ (-2147483648)	$2^{31}-1$ (2147483647)
System.Int64	long	64 bits	$-2^{63}$ (-9223372036854775808)	$2^{63}-1$ (9223372036854775807)
System.Single	float	32 bits	-3.402823E-38 (to 7-digits precision)	3.402823E+38 (to 7-digits precision)
System.Double	double	64 bits	-1.79769313486231E-308 (to 15-digits precision)	1.79769313486231E+308 (to 15-digits precision)

### Related concepts

#### Attributes and properties of objects

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

#### Implicit conversion of a property value from one data type to another

When an application gets the value of a property, the value can be converted by XMS into another data type. Many rules govern which conversions are supported and how XMS performs the conversions.

### Related reference

#### Data types for elements of application data

To ensure that an XMS application can exchange messages with a WebSphere MQ classes for JMS application, both the applications must be able to interpret the application data in the body of a message in the same way.

## Implicit conversion of a property value from one data type to another

When an application gets the value of a property, the value can be converted by XMS into another data type. Many rules govern which conversions are supported and how XMS performs the conversions.

A property of an object has a name and a value; the value has an associated data type, where the value of a property is also referred to as the *property type*.

An application uses the methods of the PropertyContext class to get and set the properties of objects. In order to get the value of a property, an application calls the method that is appropriate for the property type. For example, to get the value of an integer property, an application typically calls the GetIntProperty method.

However, when an application gets the value of a property, the value can be converted by XMS into another data type. For example, to get the value of an integer property, an application can call the GetStringProperty method, which returns the value of the property as a string. The conversions supported by XMS are shown in [Table 6 on page 41](#).

Property type	Supported target data types
System.String	System.Boolean, System.Double, System.Float, System.Int32, System.Int64, System.SByte, System.Int16
System.Boolean	System.String, System.SByte, System.Int32, System.Int64, System.Int16
System.Char	System.String
System.Double	System.String
System.Float	System.String, System.Double
System.Int32	System.String, System.Int64
System.Int64	System.String
System.SByte	System.String, System.Int32, System.Int64, System.Int16
System.SByte array	System.String
System.Int16	System.String, System.Int32, System.Int64

The following general rules govern the supported conversions:

- Numeric property values can be converted from one data type to another provided no data is lost during the conversion. For example, the value of a property with data type System.Int32 can be converted into a value with data type System.Int64, but it cannot be converted into a value with data type System.Int16.
- A property value of any data type can be converted into a string.
- A string property value can be converted to any other data type provided the string is formatted correctly for the conversion. If an application attempts to convert a string property value that is not formatted correctly, XMS may return errors.
- If an application attempts a conversion that is not supported, XMS may return an error.

The following rules apply when a property value is converted from one data type to another:

- When converting a boolean property value to a string, the value true is converted to the string "true", and the value false is converted to the string "false".
- When converting a boolean property value to a numeric data type, including System.SByte, the value true is converted to 1, and the value false is converted to 0.
- When converting a string property value to a boolean value, the string "true" (not case-sensitive) or "1" is converted to true, and the string "false" (not case-sensitive) or "0" is converted to false. All other strings cannot be converted.
- When converting a string property value to a value with data type System.Int32, System.Int64, System.SByte, or System.Int16, the string must have the following format:

*[blanks][sign]digits*

The string components are defined as follows:

**blanks**

Optional leading blank characters.

**sign**

An optional plus sign (+) or minus sign (-) character.

**digits**

A contiguous sequence of digit characters (0-9). At least one digit character must be present.

After the sequence of digit characters, the string can contain other characters that are not digit characters, but the conversion stops as soon as the first of these characters is reached. The string is assumed to represent a decimal integer.

XMS may return an error if the string is not formatted correctly.

- When converting a string property value to a value with data type System.Double or System.Float, the string must have the following format:

*[blanks][sign][digits][point[d\_digits]][e\_char[e\_sign]e\_digits]*

The string components are defined as follows:

**blanks**

(Optional) Leading blank characters.

**sign**

(Optional) Plus sign (+) or minus sign (-) character.

**digits**

A contiguous sequence of digit characters (0-9). At least one digit character must be present in either *digits* or *d\_digits*.

**point**

(Optional) Decimal point (.).

**d\_digits**

A contiguous sequence of digit characters (0-9). At least one digit character must be present in either *digits* or *d\_digits*.

**e\_char**

An exponent character, which is either *E* or *e*.

**e\_sign**

(Optional) Plus sign (+) or minus sign (-) character for the exponent.

**e\_digits**

A contiguous sequence of digit characters (0-9) for the exponent. At least one digit character must be present if the string contains an exponent character.

After the sequence of digit characters, or the optional characters representing an exponent, the string can contain other characters that are not digit characters, but the conversion stops as soon as the first of these characters is reached. The string is assumed to represent a decimal floating point number with an exponent that is a power of 10.

XMS may return an error if the string is not formatted correctly.

- When converting a numeric property value to a string, including a property value with data type System.SByte, the value is converted to the string representation of the value as a decimal number, not the string containing the ASCII character for that value. For example, the integer 65 is converted to the string "65", not the string "A".
- When converting a byte array property value to a string, each byte is converted to the 2 hexadecimal characters that represent the byte. For example, the byte array {0xF1, 0x12, 0x00, 0xFF} is converted to the string "F11200FF".

Conversions from a property type to other data types are supported by the methods of both the Property and the PropertyContext classes.

## Related concepts

### Attributes and properties of objects

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

### XMS primitive types

XMS provides equivalents of the eight Java primitive types (byte, short, int, long, float, double, char, and boolean). This allows the interchange of messages between XMS and JMS without data becoming lost or corrupted.

## Related reference

### Map messages

The body of a map message contains a set of name-value pairs, where each value has an associated data type.

#### Stream messages

The body of a stream message contains a stream of values, where each value has an associated data type.

#### Data types for elements of application data

To ensure that an XMS application can exchange messages with a WebSphere MQ classes for JMS application, both the applications must be able to interpret the application data in the body of a message in the same way.

## **Iterators**

An iterator encapsulates a list of objects and a cursor that maintains the current position in the list. The concept of an Iterator, as available in Message Service Client for C/C++ is implemented by using IEnumerator interface in Message Service Client for .NET.

When an iterator is created, the position of the cursor is before the first object. An application uses an iterator to retrieve each object in turn.

The Iterator class of Message Service Client for C/C++ is equivalent to the Enumerator class in Java. XMS .NET is similar to Java and uses an IEnumerator interface.

An application can use an IEnumerator to perform the following tasks:

- To get the properties of a message
- To get the name-value pairs in the body of a map message
- To browse the messages on a queue
- To get the names of the JMS defined message properties supported by a connection

## **Coded character set identifiers**

In XMS .NET, all strings are passed using the native .NET string. Because this has a fixed encoding, no further information is required to interpret it. Hence the XMSC\_CLIENT\_CCSID property is not required for XMS .NET applications.

## **XMS error and exception codes**

XMS uses a range of error codes to indicate failures. These error codes are not explicitly listed in this documentation because they may vary from release to release. Only XMS exception codes (in the format XMS\_X\_...) are documented because they remain the same across releases of XMS.

## **Building your own applications**

You build your own applications like you build the sample applications.

Build your .NET application, as described in “Building the .NET sample applications” on page 20. This task also lists the prerequisites you need to build your own .NET applications. For additional guidance on how to build your own applications, use the makefiles provided for each sample application.

**Tip:** To assist with problem diagnosis in the event of a failure, you might find it helpful to compile applications with symbols included.

### **Related concepts**

#### The sample applications

The sample applications provide an overview of the common features of each API. You can use them to verify your installation and messaging server setup and to help you build your own applications.

### **Related reference**

#### .NET interfaces

This section documents the .NET class interfaces and their properties and methods.

#### Properties of XMS objects

This chapter documents the object properties defined by XMS.

### ***Automatic WebSphere MQ Client reconnection through XMS***

Configure your XMS client to reconnect automatically following a network, queue manager, or server failure while using WebSphere MQ V7.1 Client and later, as the message provider.

Use the `WMQ_CONNECTION_NAME_LIST` and `WMQ_CLIENT_RECONNECT_OPTIONS` properties of the `MQConnectionFactory` class to configure a client connection to automatically reconnect. Automatic client reconnection reconnects a client after a connection failure, or as an option after stopping the queue manager. The design of some client applications makes them unsuitable for automatic reconnection.

Automatically reconnectable client connections become reconnectable once the connection is established.

**Note:** The properties `Client Reconnect Options`, `Client Reconnect Timeout`, and `Connection Namelist` can also be set via `Client Channel Definitions Table (CCDT)` or by enabling the client reconnection via the `mqclient.ini` file.

**Note:** If reconnection properties are set on the `ConnectionFactory` object and as well as in the `CCDT`, the precedence rule is as follows. If the default value of the connection name list property is set in the `ConnectionFactory` object, then the `CCDT` takes precedence. If the connection name list is not set to its default value, the property values set in the `ConnectionFactory` object take precedence. The default value of the connection namelist is `localhost(1414)`.

## **Writing XMS .NET applications**

This chapter provides information to help you when writing XMS.NET applications.

The chapter contains the following sections:

- [“Data types for .NET” on page 44](#)
- [“Managed and unmanaged operations in .NET” on page 45](#)
- [“Destinations in .NET” on page 46](#)
- [“Properties in .NET” on page 46](#)
- [“Non-existent properties handling in .NET” on page 47](#)
- [“Error handling in .NET” on page 48](#)
- [“Message and exception listeners in .NET” on page 48](#)

### **Related concepts**

[Writing XMS applications](#)

This chapter provides information to help you when writing XMS applications.

### **Related reference**

[.NET interfaces](#)

This section documents the .NET class interfaces and their properties and methods.

## **Data types for .NET**

XMS .NET supports `System.Boolean`, `System.Byte`, `System.SByte`, `System.Char`, `System.String`, `System.Single`, `System.Double`, `System.Decimal`, `System.Int16`, `System.Int32`, `System.Int64`, `System.UInt16`, `System.UInt32`, `System.UInt64`, and `System.Object`. Data types for XMS .NET are different from data types for XMS C++. You can use this chapter to identify the corresponding data types.

The following table shows the corresponding XMS .NET and XMS C++ data types and briefly describes them.

Table 7. Data types for XMS .NET and XMS C++

XMS .NET type	XMS C++ type	Description
System.SByte	xmsSBYTE xmsINT8	Signed 8-bit value
System.Byte	xmsBYTE xmsUINT8	Unsigned 8-bit value
System.Int16	xmsINT16 xmsSHORT	Signed 16-bit value
System.UInt16	xmsUINT16 xmsUSHORT	Unsigned 16-bit value
System.Int32	xmsINT32 xmsINT	Signed 32-bit value
System.UInt32	xmsUINT32 xmsUINT	Unsigned 32-bit value
System.Int64	xmsLONG xmsINT64	Signed 64-bit value
System.UInt64	xmsULONG xmsUINT64	Unsigned 64-bit value
System.Char	xmsCHAR16	Unsigned 16-bit character (Unicode for .NET)
System.Single	xmsFLOAT	IEEE 32-bit float
System.Double	xmsDOUBLE	IEEE 64-bit float
System.Boolean	xmsBOOL	A True/False value
Not applicable	xmsCHAR	Signed or Unsigned 8-bit value (signed or unsigned depends on platform)
System.Decimal	Not applicable	96-bit signed integer times $10^0$ through $10^{28}$
System.Object	Not applicable	Base of all types
System.String	Not applicable	String type

## Managed and unmanaged operations in .NET

Managed code is executed exclusively within the .NET common language runtime environment and is wholly dependent on the services provided by that runtime. An application is classed as unmanaged if any part of the application runs or calls services outside of the .NET common language runtime environment.

Certain advanced functionality cannot currently be supported within the managed .NET environment.

If your application requires some functionality that is not currently supported in the fully managed environment, then you can change your application to use the unmanaged environment without requiring

substantial change to your application. However, you should note that the XMS stack makes use of unmanaged code when this selection is made.

## Connections to a WebSphere MQ queue manager

Managed connections to WMQ\_CM\_CLIENT will not support non-TCP communications, and channel compression. However, these connections might be supported by using an unmanaged connection (WMQ\_CM\_CLIENT\_UNMANAGED). For more information, see [Developing .NET applications](#).

If you create a connection factory from an administered object in an unmanaged environment, you must manually change the value for the connection mode to XMSC\_WMQ\_CM\_CLIENT\_UNMANAGED.

## Connections to a WebSphere Service Integration Bus messaging engine

Connections to a WebSphere service integration bus messaging engine that require the use of the SSL protocol (including HTTPS) are not currently supported as managed code.

### Related reference

[XMSC\\_WMQ\\_CONNECTION\\_MODE](#)

## Destinations in .NET

In .NET, destinations are created according to protocol type and can be used only on the protocol type for which they are created.

Two functions are provided for creating destinations, one for topics and one for queues:

- `IDestination CreateTopic(String topic);`
- `IDestination CreateQueue(String queue);`

These functions are available on the following two objects in the API:

- `ISession`
- `XMSFactoryFactory`

In both cases these methods can accept a URI style string, which can include parameters, in the following format:

```
"topic://some/topic/name?priority=5"
```

Alternatively, these methods can accept a destination name only, that is, a name without a `topic://` or `queue://` prefix and without parameters.

This means that the following URI style string:

```
CreateTopic("topic://some/topic/name");
```

would produce the same result as the following destination name:

```
CreateTopic("some/topic/name");
```

As for WebSphere Service Integration Bus JMS, topics can also be specified in a shorthand form, which includes both the *topicname* and *topicspace* but cannot include parameters:

```
CreateTopic("topicspace:topicname");
```

## Properties in .NET

A .NET application uses the methods in the `PropertyContext` interface to get and set the properties of objects.

The [PropertyContext](#) interface encapsulates methods that get and set properties. These methods are inherited, directly or indirectly, by the following classes:

- [BytesMessage](#)
- [Connection](#)
- [ConnectionFactory](#)
- [ConnectionMetaData](#)
- [Destination](#)
- [MapMessage](#)
- [Message](#)
- [MessageConsumer](#)
- [MessageProducer](#)
- [ObjectMessage](#)
- [QueueBrowser](#)
- [Session](#)
- [StreamMessage](#)
- [TextMessage](#)

If an application sets the value of a property, the new value replaces any previous value the property had.

For more information about XMS properties, see [“Properties of XMS objects”](#) on page 173.

For ease of use, XMS property names and values in .NET are predefined as public constants in a struct called XMSC. The names of these constants are in the form XMSC.<constant>; for example, XMSC.USERID (a property name constant) and XMSC.DELIVERY\_AS\_APP (a value constant).

Additionally, you can access WebSphere MQ constants by using the IBM.XMS.MQC struct. If the IBM.XMS namespace is already imported, you can access the values for these properties in the form MQC.<constant>. For example, MQC.MQRO\_COA\_WITH\_FULL\_DATA.

Furthermore, if you have a hybrid application that uses both XMS .NET and WebSphere MQ classes for .NET and that imports both IBM.XMS and IBM.WMQ namespaces, then you must fully qualify the MQC struct namespace to ensure that each occurrence is unique.

Some advanced functionality is not currently supported within the managed .NET environment. Refer to [“Managed and unmanaged operations in .NET”](#) on page 45 for more details.

## Non-existent properties handling in .NET

The handling of non-existent properties in the XMS .NET is broadly consistent with the JMS specification, and also with the C and C++ implementations of XMS.

In JMS, accessing a non-existent property can result in a Java system exception when a method tries to convert the non-existent (null) value to the required type. If a property does not exist the following exceptions occur:

- `getStringProperty` and `getObjectProperty` return null
- `getBooleanProperty` returns false because `Boolean.valueOf(null)` returns false
- `getIntProperty` etc. throw `java.lang.NumberFormatException` because `Integer.valueOf(null)` throws the exception

If a property does not exist in XMS .NET the following exceptions occur:

- `GetStringProperty` and `GetObjectProperty` (and `GetBytesProperty`) return null (which is the same as Java)
- `GetBooleanProperty` throws `System.NullReferenceException`
- `GetIntProperty` etc. throws `System.NullReferenceException`

This implementation is different from Java, but it is broadly consistent with the JMS specification, and with the XMS C and C++ interfaces. Like the Java implementation, XMS .NET propagates any exceptions from the `System.Convert` call to the caller. Unlike Java however, XMS explicitly throws

NullReferenceExceptions rather than just using the native behavior of the .NET framework through passing null to system conversion routines. If your application sets a property to a String like "abc" and calls GetIntProperty, the System.FormatException thrown by Convert.ToInt32("abc") is propagated to the caller, which is consistent with Java. MessageFormatException is thrown only if the types used for SetProperty and GetProperty are incompatible. This behavior is also consistent with Java.

## Error handling in .NET

XMS .NET exceptions are all derived from System.Exception. XMS method calls can throw specific XMS exceptions such as MessageFormatException, general XMSExceptions, or system exceptions such as NullReferenceException.

Write applications to catch any of these errors, either in specific catch blocks or in general System.Exception catch blocks, as appropriate to the applications requirements.

## Message and exception listeners in .NET

A .NET application uses a message listener to receive messages asynchronously, and it uses an exception listener to be notified asynchronously of a problem with a connection.

The functionality of both the message and exception listeners is the same for .NET and for C++. However, there are some small implementation differences.

## Message listeners in .NET

To receive messages asynchronously, you must complete the following steps:

1. Define a method that matches the signature of the message listener delegate. The method that you define can be either a static or an instance method and can be defined in any accessible class. The delegate signature is as follows:

```
public delegate void MessageListener(IMessage msg);
```

and so you could define the method as:

```
void SomeMethodName(IMessage msg);
```

2. Instantiate this method as a delegate using something similar to the following:

```
MessageListener OnMsgMethod = new MessageListener(SomeMethodName)
```

3. Register the delegate with one or more consumers by setting it to the MessageListener property of the consumer:

```
consumer.MessageListener = OnMsgMethod;
```

You can remove the delegate by setting the MessageListener back to null:

```
consumer.MessageListener = null;
```

## Exception listeners in .NET

The exception listener works in much the same way as the message listener, but has a different delegate definition and is assigned to the connection rather than the message consumer. This is the same as for C++.

1. Define the method. The delegate signature is as follows:

```
public delegate void ExceptionListener(Exception ex);
```

and so the method defined could be:

```
void SomeMethodName(Exception ex);
```

2. Instantiate this method as a delegate using something similar to:

```
ExceptionListener OnExMethod = new ExceptionListener(SomeMethodName)
```

3. Register the delegate with the connection by setting its ExceptionListener property:

```
connection.ExceptionListener = OnExMethod ;
```

You can remove the delegate by resetting the ExceptionListener to:

```
null: connection.ExceptionListener = null;
```

When no references to them remain, exceptions or messages are deleted automatically by the systems garbage collector.

The following is a sample code:

```
using System;
using System.Threading;
using IBM.XMS;

public class Sample
{
    public static void Main()
    {
        XMSFactoryFactory factoryFactory = XMSFactoryFactory.GetInstance(XMSC.CT_RTT);

        IConnectionFactory connectionFactory = factoryFactory.CreateConnectionFactory();
        connectionFactory.SetStringProperty(XMSC.RTT_HOST_NAME, "localhost");
        connectionFactory.SetStringProperty(XMSC.RTT_PORT, "1506");

        //
        // Create the connection and register an exception listener
        //

        IConnection connection = connectionFactory.CreateConnection();
        connection.ExceptionListener = new ExceptionListener(Sample.OnException);

        ISession session = connection.CreateSession(false, AcknowledgeMode.AutoAcknowledge);
        IDestination topic = session.CreateTopic("topic://xms/sample");

        //
        // Create the consumer and register an async message listener
        //

        IMessageConsumer consumer = session.CreateConsumer(topic);
        consumer.MessageListener = new MessageListener(Sample.OnMessage);

        connection.Start();

        while (true)
        {
            Console.WriteLine("Waiting for messages...");
            Thread.Sleep(1000);
        }
    }

    static void OnMessage(IMessage msg)
    {
        Console.WriteLine(msg);
    }

    static void OnException(Exception ex)
    {
        Console.WriteLine(ex);
    }
}
```

## Working with administered objects

This chapter provides information about administered objects. XMS applications can retrieve object definitions from a central administered objects repository, and use them to create connection factories and destinations.

The chapter contains the following sections:

- [“Supported types of administered object repository” on page 50](#)
- [“Property mapping for administered objects” on page 51](#)
- [“Required properties for administered ConnectionFactory objects” on page 53](#)
- [“Required properties for administered Destination objects” on page 55](#)
- [“Creating administered objects” on page 56](#)
- [“InitialContext objects” on page 58](#)
- [“InitialContext properties” on page 59](#)
- [“URI format for XMS initial contexts” on page 60](#)
- [“JNDI Lookup Web service” on page 62](#)
- [“Retrieval of administered objects” on page 63](#)

### Related concepts

#### Administered objects

Using administered objects, you can administer the connection settings used by client applications to be administered from a central repository. An application retrieves object definitions from the central repository and uses them to create `ConnectionFactory` and `Destination` objects. Using administered objects, you can de-couple applications from the resources that they use at run time.

### Supported types of administered object repository

File System and LDAP administered objects can be used to connect to WebSphere MQ and WebSphere Application Server, whereas COS Naming can be used to connect only to the WebSphere Application Server.

File System object directories take the form of serialized Java and Naming Directory Interface (JNDI) objects. LDAP object directories are directories that contain JNDI objects. File System and LDAP object directories can be administered by using either the JMSAdmin tool, which is provided with WebSphere MQ v6.0, or the WebSphere MQ Explorer, which is provided with WebSphere MQ v7.0 and later. Both the File system and the LDAP object directories can be used to administer client connections by centralizing WebSphere MQ connection factories and destinations. The network administrator can deploy multiple applications that refer to the same central repository, and that are automatically updated to reflect changes to connection settings made in the central repository.

A COS naming directory contains WebSphere Service Integration Bus connection factories and destinations and can be administered by using the WebSphere Application Server administrative console. For an XMS application to retrieve objects from the COS naming directory, a JNDI lookup web service must be deployed. This web service is not available on all WebSphere service integration technologies. Refer to the product documentation for details.

**Note:** Restart application connections for changes to the object directory to take effect.

### Related concepts

#### Property mapping for administered objects

To enable applications to use WebSphere MQ JMS and WebSphere Application Server connection factory and destination object definitions, the properties retrieved from these definitions must be mapped on to the corresponding XMS properties that can be set on XMS connection factories and destinations.

#### InitialContext properties

The parameters of the `InitialContext` constructor include the location of the repository of administered objects, given as a uniform resource indicator (URI). In order for an application to establish a connection

to the repository, it may be necessary to provide more information than the information contained in the URI.

#### URI format for XMS initial contexts

The location of the repository of administered objects is provided as a uniform resource indicator (URI). The format of the URI depends on the context type.

#### JNDI Lookup Web service

To access a COS naming directory from XMS, a JNDI Lookup Web service must be deployed on a WebSphere Service Integration Bus server. This Web service translates the Java information from the COS naming service into a form that XMS applications can read.

#### Retrieval of administered objects

XMS retrieves an administered object from the repository using the address provided when the InitialContext object is created, or in the InitialContext properties.

#### Administered objects

Using administered objects, you can administer the connection settings used by client applications to be administered from a central repository. An application retrieves object definitions from the central repository and uses them to create `ConnectionFactory` and `Destination` objects. Using administered objects, you can de-couple applications from the resources that they use at run time.

### **Related tasks**

#### Creating administered objects

The `ConnectionFactory` and `Destination` object definitions that XMS applications require to make a connection to a messaging server must be created using the appropriate administrative tools.

#### InitialContext objects

An application must create an initial context to be used to make a connection to the administered objects repository to retrieve the required administered objects.

### **Related reference**

#### Required properties for administered `ConnectionFactory` objects

When an application creates a connection factory, a number of properties must be defined to create a connection to a messaging server.

#### Required properties for administered `Destination` objects

An application that is creating a destination must set several properties that the application on an administered `Destination` object.

## **Property mapping for administered objects**

To enable applications to use WebSphere MQ JMS and WebSphere Application Server connection factory and destination object definitions, the properties retrieved from these definitions must be mapped on to the corresponding XMS properties that can be set on XMS connection factories and destinations.

In order to create, for example, an XMS connection factory with properties retrieved from a WebSphere MQ JMS connection factory, the properties must be mapped between the two.

All property mappings are performed automatically.

The following table demonstrates the mappings between some of the most common properties of connection factories and destinations. The properties shown in this table are just a small set of examples, and not all properties shown are relevant to all connection types and servers.

<b>WebSphere MQ JMS property name</b>	<b>XMS property name</b>	<b>WebSphere Service Integration Bus property name</b>
PERSISTENCE (PER)	<u>XMSC_DELIVERY_MODE</u>	
EXPIRY (EXP)	<u>XMSC_TIME_TO_LIVE</u>	
PRIORITY (PRI)	<u>XMSC_PRIORITY</u>	

*Table 8. Examples of name mapping for connection factory and destination properties (continued)*

<b>WebSphere MQ JMS property name</b>	<b>XMS property name</b>	<b>WebSphere Service Integration Bus property name</b>
	<u>XMSC_WPM_HOST_NAME</u>	serverName
	<u>XMSC_WPM_BUS_NAME</u>	busName
	<u>XMSC_WPM_TOPIC_SPACE</u>	topicName

### **Related concepts**

#### Supported types of administered object repository

File System and LDAP administered objects can be used to connect to WebSphere MQ and WebSphere Application Server, whereas COS Naming can be used to connect only to the WebSphere Application Server.

#### InitialContext properties

The parameters of the InitialContext constructor include the location of the repository of administered objects, given as a uniform resource indicator (URI). In order for an application to establish a connection to the repository, it may be necessary to provide more information than the information contained in the URI.

#### URI format for XMS initial contexts

The location of the repository of administered objects is provided as a uniform resource indicator (URI). The format of the URI depends on the context type.

#### JNDI Lookup Web service

To access a COS naming directory from XMS, a JNDI Lookup Web service must be deployed on a WebSphere Service Integration Bus server. This Web service translates the Java information from the COS naming service into a form that XMS applications can read.

#### Retrieval of administered objects

XMS retrieves an administered object from the repository using the address provided when the InitialContext object is created, or in the InitialContext properties.

### **Related tasks**

#### Creating administered objects

The ConnectionFactory and Destination object definitions that XMS applications require to make a connection to a messaging server must be created using the appropriate administrative tools.

#### InitialContext objects

An application must create an initial context to be used to make a connection to the administered objects repository to retrieve the required administered objects.

### **Related reference**

#### Required properties for administered ConnectionFactory objects

When an application creates a connection factory, a number of properties must be defined to create a connection to a messaging server.

#### Required properties for administered Destination objects

An application that is creating a destination must set several properties that the application on an administered Destination object.

#### IDestination (for the .NET interface)

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

#### Properties of Destination

An overview of the properties of the Destination object, with links to more detailed reference information.

#### IConnectionFactory (for the .NET interface)

An application uses a connection factory to create a connection.

#### Properties of ConnectionFactory

An overview of the properties of the ConnectionFactory object, with links to more detailed reference information.

## Required properties for administered ConnectionFactory objects

When an application creates a connection factory, a number of properties must be defined to create a connection to a messaging server.

The properties listed in the following tables are the minimum required for an application to set to create a connection to a messaging server. If you want to customize the way that a connection is created, then your application can set any additional properties of the ConnectionFactory object as necessary. For more information, see “Properties of ConnectionFactory” on page 175. A complete list of available properties is included.

### Connection to a WebSphere MQ queue manager

<i>Table 9. Property settings for administered ConnectionFactory objects for connections to a WebSphere MQ queue manager</i>	
<b>Required XMS</b>	<b>Equivalent WebSphere MQ JMS property required</b>
<u>XMSC_CONNECTION_TYPE</u>	XMS works this out from the connection factory class name and TRANSPORT (TRAN) property.
<u>XMSC_WMQ_HOST_NAME</u>	HOSTNAME (HOST)
<u>XMSC_WMQ_PORT</u>	PORT
<u>XMSC_WMQ_QUEUE_MANAGER</u>	Name of the Queue Manager

### Real-time connection to a broker

<i>Table 10. Property settings for administered ConnectionFactory objects for real-time connections to a broker</i>	
<b>Required XMS</b>	<b>Equivalent WebSphere MQ JMS property required</b>
<u>XMSC_CONNECTION_TYPE</u>	XMS works this out from the connection factory class name and TRANSPORT (TRAN) property.
<u>XMSC_RTT_HOST_NAME</u>	HOSTNAME (HOST)
<u>XMSC_RTT_PORT</u>	PORT

### Connection to a WebSphere Service Integration Bus

<i>Table 11. Property settings for administered ConnectionFactory objects for connections to a WebSphere Service Integration Bus</i>	
<b>XMS property</b>	<b>Description</b>
<u>XMSC_CONNECTION_TYPE</u>	The type of messaging server to which an application connects.. This is determined from the connection factory class name.
<u>XMSC_WPM_BUS_NAME</u>	For a connection factory, the name of the service integration bus that the application connects to or, for a destination, the name of the service integration bus in which the destination exists.

#### Related concepts

[Supported types of administered object repository](#)

File System and LDAP administered objects can be used to connect to WebSphere MQ and WebSphere Application Server, whereas COS Naming can be used to connect only to the WebSphere Application Server.

#### Property mapping for administered objects

To enable applications to use WebSphere MQ JMS and WebSphere Application Server connection factory and destination object definitions, the properties retrieved from these definitions must be mapped on to the corresponding XMS properties that can be set on XMS connection factories and destinations.

#### InitialContext properties

The parameters of the InitialContext constructor include the location of the repository of administered objects, given as a uniform resource indicator (URI). In order for an application to establish a connection to the repository, it may be necessary to provide more information than the information contained in the URI.

#### URI format for XMS initial contexts

The location of the repository of administered objects is provided as a uniform resource indicator (URI). The format of the URI depends on the context type.

#### JNDI Lookup Web service

To access a COS naming directory from XMS, a JNDI Lookup Web service must be deployed on a WebSphere Service Integration Bus server. This Web service translates the Java information from the COS naming service into a form that XMS applications can read.

#### Retrieval of administered objects

XMS retrieves an administered object from the repository using the address provided when the InitialContext object is created, or in the InitialContext properties.

#### Secure connections to a WebSphere MQ queue manager

To enable an XMS .NET application to make secure connections to a WebSphere MQ queue manager, the relevant properties must be defined in the ConnectionFactory object.

#### Secure connections to a WebSphere Service Integration Bus messaging engine

To enable an XMS application to make secure connections to a WebSphere Service Integration Bus messaging engine, the relevant properties must be defined in the ConnectionFactory object.

### **Related tasks**

#### Creating administered objects

The ConnectionFactory and Destination object definitions that XMS applications require to make a connection to a messaging server must be created using the appropriate administrative tools.

#### InitialContext objects

An application must create an initial context to be used to make a connection to the administered objects repository to retrieve the required administered objects.

### **Related reference**

#### Required properties for administered Destination objects

An application that is creating a destination must set several properties that the application on an administered Destination object.

#### IConnectionFactory (for the .NET interface)

An application uses a connection factory to create a connection.

#### Properties of ConnectionFactory

An overview of the properties of the ConnectionFactory object, with links to more detailed reference information.

## Required properties for administered Destination objects

An application that is creating a destination must set several properties that the application on an administered Destination object.

Type of connection	Property	Description
WebSphere MQ queue manager	QUEUE (QU)	The queue that you want to connect to
	TOPIC (TOP)	The topic that the application uses as a destination
Real-time connection to a broker	TOPIC (TOP)	The topic that the application uses as a destination
WebSphere Service Integration Bus	topicName	If your application is connecting to a topic
	queueName	If your application is connecting to a queue

### Related concepts

[Supported types of administered object repository](#)

File System and LDAP administered objects can be used to connect to WebSphere MQ and WebSphere Application Server, whereas COS Naming can be used to connect only to the WebSphere Application Server.

[Property mapping for administered objects](#)

To enable applications to use WebSphere MQ JMS and WebSphere Application Server connection factory and destination object definitions, the properties retrieved from these definitions must be mapped on to the corresponding XMS properties that can be set on XMS connection factories and destinations.

[InitialContext properties](#)

The parameters of the InitialContext constructor include the location of the repository of administered objects, given as a uniform resource indicator (URI). In order for an application to establish a connection to the repository, it may be necessary to provide more information than the information contained in the URI.

[URI format for XMS initial contexts](#)

The location of the repository of administered objects is provided as a uniform resource indicator (URI). The format of the URI depends on the context type.

[JNDI Lookup Web service](#)

To access a COS naming directory from XMS, a JNDI Lookup Web service must be deployed on a WebSphere Service Integration Bus server. This Web service translates the Java information from the COS naming service into a form that XMS applications can read.

[Retrieval of administered objects](#)

XMS retrieves an administered object from the repository using the address provided when the InitialContext object is created, or in the InitialContext properties.

### Related tasks

[Creating administered objects](#)

The ConnectionFactory and Destination object definitions that XMS applications require to make a connection to a messaging server must be created using the appropriate administrative tools.

[InitialContext objects](#)

An application must create an initial context to be used to make a connection to the administered objects repository to retrieve the required administered objects.

### Related reference

[Required properties for administered ConnectionFactory objects](#)

When an application creates a connection factory, a number of properties must be defined to create a connection to a messaging server.

#### IDestination (for the .NET interface)

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

#### Properties of Destination

An overview of the properties of the Destination object, with links to more detailed reference information.

## Creating administered objects

The ConnectionFactory and Destination object definitions that XMS applications require to make a connection to a messaging server must be created using the appropriate administrative tools.

### Before you begin

For further details about the different types of administered object repository that XMS supports, see [“Supported types of administered object repository” on page 50.](#)

### About this task

To create the administered objects for WebSphere MQ use the WebSphere MQ Explorer or WebSphere MQ JMS administration (JMSAdmin) tool.

To create the administered objects for WebSphere MQ or IBM Integration Bus, use the WebSphere MQ JMS administration (JMSAdmin) tool.

To create administered objects for WebSphere Service Integration Bus, use the WebSphere Application Server administrative console.

The following steps summarize what you do to create administered objects.

### Procedure

1. Create a connection factory and define the necessary properties to create a connection from your application to your chosen server.

The minimum properties that XMS requires to make a connection are defined in [“Required properties for administered ConnectionFactory objects” on page 53.](#)

2. Create the required destination on the messaging server, which your application connects to:

- For a connection to a WebSphere MQ queue manager, create a queue or topic.
- For a real-time connection to a broker, create a topic.
- For a connection to a WebSphere Service Integration Bus, create a queue or a topic.

The minimum properties that XMS requires to make a connection are defined in [“Required properties for administered Destination objects” on page 55.](#)

### Related concepts

#### Supported types of administered object repository

File System and LDAP administered objects can be used to connect to WebSphere MQ and WebSphere Application Server, whereas COS Naming can be used to connect only to the WebSphere Application Server.

#### Property mapping for administered objects

To enable applications to use WebSphere MQ JMS and WebSphere Application Server connection factory and destination object definitions, the properties retrieved from these definitions must be mapped on to the corresponding XMS properties that can be set on XMS connection factories and destinations.

#### InitialContext properties

The parameters of the InitialContext constructor include the location of the repository of administered objects, given as a uniform resource indicator (URI). In order for an application to establish a connection

to the repository, it may be necessary to provide more information than the information contained in the URI.

#### URI format for XMS initial contexts

The location of the repository of administered objects is provided as a uniform resource indicator (URI). The format of the URI depends on the context type.

#### JNDI Lookup Web service

To access a COS naming directory from XMS, a JNDI Lookup Web service must be deployed on a WebSphere Service Integration Bus server. This Web service translates the Java information from the COS naming service into a form that XMS applications can read.

#### Retrieval of administered objects

XMS retrieves an administered object from the repository using the address provided when the InitialContext object is created, or in the InitialContext properties.

#### Administered objects

Using administered objects, you can administer the connection settings used by client applications to be administered from a central repository. An application retrieves object definitions from the central repository and uses them to create ConnectionFactory and Destination objects. Using administered objects, you can de-couple applications from the resources that they use at run time.

#### ConnectionFactory and Connection objects

A ConnectionFactory object provides a template that an application uses to create a Connection object. The application uses the Connection object to create a Session object.

#### Connection to a WebSphere service integration bus

An XMS application can connect to a WebSphere Service Integration Bus either by using a direct TCP/IP connection or by using HTTP over TCP/IP.

### **Related tasks**

#### InitialContext objects

An application must create an initial context to be used to make a connection to the administered objects repository to retrieve the required administered objects.

### **Related reference**

#### Required properties for administered ConnectionFactory objects

When an application creates a connection factory, a number of properties must be defined to create a connection to a messaging server.

#### Required properties for administered Destination objects

An application that is creating a destination must set several properties that the application on an administered Destination object.

#### IConnectionFactory (for the .NET interface)

An application uses a connection factory to create a connection.

#### Properties of ConnectionFactory

An overview of the properties of the ConnectionFactory object, with links to more detailed reference information.

#### IDestination (for the .NET interface)

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

#### Properties of Destination

An overview of the properties of the Destination object, with links to more detailed reference information.

## InitialContext objects

An application must create an initial context to be used to make a connection to the administered objects repository to retrieve the required administered objects.

### About this task

An InitialContext object encapsulates a connection to the repository. The XMS API provides methods to perform the following tasks:

- Create an InitialContext object
- Look up an administered object in the administered object repository.

For further details about creating an InitialContext object, see [“InitialContext” on page 108](#) for .NET and [“Properties of InitialContext” on page 184](#).

### Related concepts

#### [Supported types of administered object repository](#)

File System and LDAP administered objects can be used to connect to WebSphere MQ and WebSphere Application Server, whereas COS Naming can be used to connect only to the WebSphere Application Server.

#### [Property mapping for administered objects](#)

To enable applications to use WebSphere MQ JMS and WebSphere Application Server connection factory and destination object definitions, the properties retrieved from these definitions must be mapped on to the corresponding XMS properties that can be set on XMS connection factories and destinations.

#### [InitialContext properties](#)

The parameters of the InitialContext constructor include the location of the repository of administered objects, given as a uniform resource indicator (URI). In order for an application to establish a connection to the repository, it may be necessary to provide more information than the information contained in the URI.

#### [URI format for XMS initial contexts](#)

The location of the repository of administered objects is provided as a uniform resource indicator (URI). The format of the URI depends on the context type.

#### [JNDI Lookup Web service](#)

To access a COS naming directory from XMS, a JNDI Lookup Web service must be deployed on a WebSphere Service Integration Bus server. This Web service translates the Java information from the COS naming service into a form that XMS applications can read.

#### [Retrieval of administered objects](#)

XMS retrieves an administered object from the repository using the address provided when the InitialContext object is created, or in the InitialContext properties.

### Related tasks

#### [Creating administered objects](#)

The ConnectionFactory and Destination object definitions that XMS applications require to make a connection to a messaging server must be created using the appropriate administrative tools.

### Related reference

#### [Required properties for administered ConnectionFactory objects](#)

When an application creates a connection factory, a number of properties must be defined to create a connection to a messaging server.

#### [Required properties for administered Destination objects](#)

An application that is creating a destination must set several properties that the application on an administered Destination object.

#### [InitialContext \(for the .NET interface\)](#)

An application uses an InitialContext object to create objects from object definitions that are retrieved from a repository of administered objects.

#### Properties of InitialContext

An overview of the properties of the InitialContext object, with links to more detailed reference information.

## **InitialContext properties**

The parameters of the InitialContext constructor include the location of the repository of administered objects, given as a uniform resource indicator (URI). In order for an application to establish a connection to the repository, it may be necessary to provide more information than the information contained in the URI.

In JNDI and in the .NET implementation of XMS, the additional information is provided in an environment Hashtable to the constructor.

The location of the administered object repository is defined in the `XMSC_IC_URL` property. This property is typically passed on the Create call, but can be modified to connect to a different naming directory before the lookup. For FileSystem or LDAP contexts, this property defines the address of the directory. For COS naming, this is the address of the Web service that uses these properties to connect to the JNDI directory.

The following properties are passed unmodified to the Web service which will use them to use to connect to the JNDI directory.

- [XMSC\\_IC\\_PROVIDER\\_URL](#)
- [XMSC\\_IC\\_SECURITY\\_CREDENTIALS](#)
- [XMSC\\_IC\\_SECURITY\\_AUTHENTICATION](#)
- [XMSC\\_IC\\_SECURITY\\_PRINCIPAL](#)
- [XMSC\\_IC\\_SECURITY\\_PROTOCOL](#)

### **Related concepts**

#### Supported types of administered object repository

File System and LDAP administered objects can be used to connect to WebSphere MQ and WebSphere Application Server, whereas COS Naming can be used to connect only to the WebSphere Application Server.

#### Property mapping for administered objects

To enable applications to use WebSphere MQ JMS and WebSphere Application Server connection factory and destination object definitions, the properties retrieved from these definitions must be mapped on to the corresponding XMS properties that can be set on XMS connection factories and destinations.

#### URI format for XMS initial contexts

The location of the repository of administered objects is provided as a uniform resource indicator (URI). The format of the URI depends on the context type.

#### JNDI Lookup Web service

To access a COS naming directory from XMS, a JNDI Lookup Web service must be deployed on a WebSphere Service Integration Bus server. This Web service translates the Java information from the COS naming service into a form that XMS applications can read.

#### Retrieval of administered objects

XMS retrieves an administered object from the repository using the address provided when the InitialContext object is created, or in the InitialContext properties.

### **Related tasks**

#### Creating administered objects

The ConnectionFactory and Destination object definitions that XMS applications require to make a connection to a messaging server must be created using the appropriate administrative tools.

#### InitialContext objects

An application must create an initial context to be used to make a connection to the administered objects repository to retrieve the required administered objects.

### **Related reference**

[Required properties for administered ConnectionFactory objects](#)

When an application creates a connection factory, a number of properties must be defined to create a connection to a messaging server.

[Required properties for administered Destination objects](#)

An application that is creating a destination must set several properties that the application on an administered Destination object.

[InitialContext \(for the .NET interface\)](#)

An application uses an InitialContext object to create objects from object definitions that are retrieved from a repository of administered objects.

[Properties of InitialContext](#)

An overview of the properties of the InitialContext object, with links to more detailed reference information.

## **URI format for XMS initial contexts**

The location of the repository of administered objects is provided as a uniform resource indicator (URI). The format of the URI depends on the context type.

### **FileSystem context**

For the FileSystem context, the URL gives the location of the file system based directory. The structure of the URL is as defined by RFC 1738, *Uniform Resource Locators (URL)*: the URL has the prefix `file://`, and the syntax following this prefix is a valid definition of a file that can be opened on the system on which XMS is running.

This syntax can be platform-specific, and can use either '/' separators or '\' separators. If you use '\', then each separator needs to be escaped by using an additional '\'. This prevents the .NET framework from trying to interpret the separator as an escape character for what follows.

These examples illustrate this syntax:

```
file://myBindings
file:///admin/.bindings
file://\admin\.bindings
file://c:/admin/.bindings
file://c:\\admin\\.bindings
file://\\madison\\shared\\admin\\.bindings
file:///usr/admin/.bindings
```

### **LDAP context**

For the LDAP context, the basic structure of the URL is as defined by RFC 2255, *The LDAP URL Format*, with the case-insensitive prefix `ldap://`

The precise syntax is illustrated in the following example:

```
LDAP://[Hostname][:Port]["/" [DistinguishedName]]
```

This syntax is as defined in the RFC but without support for any attributes, scope, filters, or extensions.

Examples of this syntax include:

```
ldap://madison:389/cn=JMSData,dc=IBM,dc=UK
ldap://madison/cn=JMSData,dc=IBM,dc=UK
LDAP:///cn=JMSData,dc=IBM,dc=UK
```

## WSS context

For the WSS context, the URL is in the form of a Web services endpoint, with the prefix `http://`.

Alternatively, you can use the prefix `cosnaming://` or `wsvc://`,

These two prefixes are interpreted as meaning that you are using a WSS context with the URL accessed over `http`, which enables the initial context type to be derived easily directly from the URL.

Examples of this syntax include the following:

```
http://madison.ibm.com:9080/xmsjndi/services/JndiLookup  
cosnaming://madison/jndilookup
```

## Related concepts

### Supported types of administered object repository

File System and LDAP administered objects can be used to connect to WebSphere MQ and WebSphere Application Server, whereas COS Naming can be used to connect only to the WebSphere Application Server.

### Property mapping for administered objects

To enable applications to use WebSphere MQ JMS and WebSphere Application Server connection factory and destination object definitions, the properties retrieved from these definitions must be mapped on to the corresponding XMS properties that can be set on XMS connection factories and destinations.

### InitialContext properties

The parameters of the InitialContext constructor include the location of the repository of administered objects, given as a uniform resource indicator (URI). In order for an application to establish a connection to the repository, it may be necessary to provide more information than the information contained in the URI.

### JNDI Lookup Web service

To access a COS naming directory from XMS, a JNDI Lookup Web service must be deployed on a WebSphere Service Integration Bus server. This Web service translates the Java information from the COS naming service into a form that XMS applications can read.

### Retrieval of administered objects

XMS retrieves an administered object from the repository using the address provided when the InitialContext object is created, or in the InitialContext properties.

## Related tasks

### Creating administered objects

The ConnectionFactory and Destination object definitions that XMS applications require to make a connection to a messaging server must be created using the appropriate administrative tools.

### InitialContext objects

An application must create an initial context to be used to make a connection to the administered objects repository to retrieve the required administered objects.

## Related reference

### Required properties for administered ConnectionFactory objects

When an application creates a connection factory, a number of properties must be defined to create a connection to a messaging server.

### Required properties for administered Destination objects

An application that is creating a destination must set several properties that the application on an administered Destination object.

### InitialContext (for the .NET interface)

An application uses an InitialContext object to create objects from object definitions that are retrieved from a repository of administered objects.

### Properties of InitialContext

An overview of the properties of the InitialContext object, with links to more detailed reference information.

## JNDI Lookup Web service

To access a COS naming directory from XMS, a JNDI Lookup Web service must be deployed on a WebSphere Service Integration Bus server. This Web service translates the Java information from the COS naming service into a form that XMS applications can read.

The Web service is provided in the enterprise archive file SIBXJndiLookupEAR.ear, located within the install directory. For the current release of Message Service Client for .NET, SIBXJndiLookupEAR.ear can be found in the <install\_dir>\java\lib directory. This can be installed within a WebSphere service integration bus server by using either the administrative console or the wsadmin scripting tool. Refer to the product documentation for further information on deploying Web service applications.

To define the Web service within XMS applications, you simply need to set the `XMSC_IC_URL` property of the InitialContext object to the Web service endpoint URL. For example, if the Web service is deployed on a server host called MyServer, an example of a Web service endpoint URL:

```
wsvc://MyHost:9080/SIBXJndiLookup/services/JndiLookup
```

Setting the `XMSC_IC_URL` property allows InitialContext Lookup calls to invoke the Web service at the defined endpoint, which in turn looks up the required administered object from the COS naming service.

.NET applications can use the Web service. The server-side deployment is the same for XMS C, /C++ and XMS .NET. XMS .NET invokes the Web service directly through the Microsoft .NET framework.

### Related concepts

#### Supported types of administered object repository

File System and LDAP administered objects can be used to connect to WebSphere MQ and WebSphere Application Server, whereas COS Naming can be used to connect only to the WebSphere Application Server.

#### Property mapping for administered objects

To enable applications to use WebSphere MQ JMS and WebSphere Application Server connection factory and destination object definitions, the properties retrieved from these definitions must be mapped on to the corresponding XMS properties that can be set on XMS connection factories and destinations.

#### InitialContext properties

The parameters of the InitialContext constructor include the location of the repository of administered objects, given as a uniform resource indicator (URI). In order for an application to establish a connection to the repository, it may be necessary to provide more information than the information contained in the URI.

#### URI format for XMS initial contexts

The location of the repository of administered objects is provided as a uniform resource indicator (URI). The format of the URI depends on the context type.

#### Retrieval of administered objects

XMS retrieves an administered object from the repository using the address provided when the InitialContext object is created, or in the InitialContext properties.

#### Setting up the messaging server environment

This chapter describes how to set up the messaging server environment to allow XMS applications to connect to a server.

#### Using the XMS sample applications

Use the sample applications supplied with XMS to verify your installation and messaging server setup, and to help you build your own applications. The samples provide an overview of the common features of each API.

### Related tasks

#### Creating administered objects

The ConnectionFactory and Destination object definitions that XMS applications require to make a connection to a messaging server must be created using the appropriate administrative tools.

#### InitialContext objects

An application must create an initial context to be used to make a connection to the administered objects repository to retrieve the required administered objects.

#### Installing Message Service Client for .NET using the installation wizard

The installation uses an InstallShield X/Windows MSI installer. Two setup options are available, so that you can choose either a complete or a custom installation.

#### **Related reference**

##### Required properties for administered ConnectionFactory objects

When an application creates a connection factory, a number of properties must be defined to create a connection to a messaging server.

##### Required properties for administered Destination objects

An application that is creating a destination must set several properties that the application on an administered Destination object.

### **Retrieval of administered objects**

XMS retrieves an administered object from the repository using the address provided when the InitialContext object is created, or in the InitialContext properties.

Objects to be retrieved can have the following types of names:

- A simple name describing the Destination object, for example, a queue destination called SalesOrders
- A composite name, which can be made up of SubContexts, separated by '/', and it must end with the object name. An example of a composite name is "Warehouse/PickLists/DispatchQueue2" where Warehouse and PickLists are SubContexts in the naming directory, and DispatchQueue2 is the name of a Destination object.

#### **Related concepts**

##### Supported types of administered object repository

File System and LDAP administered objects can be used to connect to WebSphere MQ and WebSphere Application Server, whereas COS Naming can be used to connect only to the WebSphere Application Server.

##### Property mapping for administered objects

To enable applications to use WebSphere MQ JMS and WebSphere Application Server connection factory and destination object definitions, the properties retrieved from these definitions must be mapped on to the corresponding XMS properties that can be set on XMS connection factories and destinations.

##### InitialContext properties

The parameters of the InitialContext constructor include the location of the repository of administered objects, given as a uniform resource indicator (URI). In order for an application to establish a connection to the repository, it may be necessary to provide more information than the information contained in the URI.

##### URI format for XMS initial contexts

The location of the repository of administered objects is provided as a uniform resource indicator (URI). The format of the URI depends on the context type.

##### JNDI Lookup Web service

To access a COS naming directory from XMS, a JNDI Lookup Web service must be deployed on a WebSphere Service Integration Bus server. This Web service translates the Java information from the COS naming service into a form that XMS applications can read.

#### **Related tasks**

##### Creating administered objects

The ConnectionFactory and Destination object definitions that XMS applications require to make a connection to a messaging server must be created using the appropriate administrative tools.

#### InitialContext objects

An application must create an initial context to be used to make a connection to the administered objects repository to retrieve the required administered objects.

#### **Related reference**

##### Required properties for administered ConnectionFactory objects

When an application creates a connection factory, a number of properties must be defined to create a connection to a messaging server.

##### Required properties for administered Destination objects

An application that is creating a destination must set several properties that the application on an administered Destination object.

##### InitialContext (for the .NET interface)

An application uses an InitialContext object to create objects from object definitions that are retrieved from a repository of administered objects.

##### Properties of InitialContext

An overview of the properties of the InitialContext object, with links to more detailed reference information.

## **Securing communications for XMS applications**

This chapter provides information about setting up secure communications to enable XMS applications to connect via Secure Sockets Layer (SSL) to a WebSphere Service Integration Bus messaging engine or WebSphere MQ queue manager.

The chapter contains the following sections:

- [“Secure connections to a WebSphere MQ queue manager” on page 64](#)
- [“CipherSuite and CipherSpec name mappings for connections to a WebSphere MQ queue manager” on page 65](#)
- [“Secure connections to a WebSphere Service Integration Bus messaging engine” on page 67](#)
- [“CipherSuite and CipherSpec name mappings for connections to a WebSphere Service Integration Bus” on page 68](#)

### **Secure connections to a WebSphere MQ queue manager**

To enable an XMS .NET application to make secure connections to a WebSphere MQ queue manager, the relevant properties must be defined in the ConnectionFactory object.

The protocol used in the encryption negotiation can be either Secure Sockets Layer (SSL) or Transport Layer Security (TLS), depending on which CipherSuite you specify in the ConnectionFactory object.

If you use the WebSphere MQ Version 7.0.0.1 and later client libraries and connect to a WebSphere MQ Version 7 queue manager, then you can create multiple connections to same queue manager in XMS application. However connection to different queue manager is not permitted. If you attempt, you get the MQRC\_SSL\_ALREADY\_INITIALIZED error.

If you use the WebSphere MQ Version 6 and later client libraries, then you can create a SSL connection only if you close any previous SSL connection first. Multiple concurrent SSL connections from the same process to the same or different queue managers are not permitted. If you attempt more than one request, you get the warning MQRC\_SSL\_ALREADY\_INITIALIZED, which might mean that some requested parameters for the SSL connection were ignored.

ConnectionFactory properties for connections via SSL to a IBM WebSphere MQ manager, with a brief description, are shown in the following table:

Name of property	Description
<a href="#">XMSC_WMQ_SSL_CERT_STORES</a>	The locations of the servers that hold the certificate revocation lists (CRLs) to be used on an SSL connection to a queue manager.
<a href="#">XMSC_WMQ_SSL_CIPHER_SPEC</a>	The name of the CipherSpec to be used on a secure connection to a queue manager.
<a href="#">XMSC_WMQ_SSL_CIPHER_SUITE</a>	The name of the CipherSuite to be used on an SSL or TLS connection to a queue manager. The protocol used in negotiating the secure connection depends on the specified CipherSuite.
<a href="#">XMSC_WMQ_SSL_CRYPT_HW</a>	Configuration details for the cryptographic hardware connected to the client system.
<a href="#">XMSC_WMQ_SSL_FIPS_REQUIRED</a>	The value of this property determines whether an application can or cannot use non-FIPS compliant cipher suites. If this property is set to true, only FIPS algorithms are used for the client-server connection.
<a href="#">XMSC_WMQ_SSL_KEY_REPOSITORY</a>	The location of the key database file in which keys and certificates are stored.
<a href="#">XMSC_WMQ_SSL_KEY_RESETCOUNT</a>	The KeyResetCount represents the total number of unencrypted bytes sent and received within an SSL conversation before the secret key is renegotiated.
<a href="#">XMSC_WMQ_SSL_PEER_NAME</a>	The peer name to be used on an SSL connection to a queue manager.

#### Related reference

[IConnectionFactory \(for the .NET interface\)](#)

An application uses a connection factory to create a connection.

[Properties of ConnectionFactory](#)

An overview of the properties of the ConnectionFactory object, with links to more detailed reference information.

[Required properties for administered ConnectionFactory objects](#)

When an application creates a connection factory, a number of properties must be defined to create a connection to a messaging server.

#### ***CipherSuite and CipherSpec name mappings for connections to a WebSphere MQ queue manager***

The InitialContext translates between the JMSAdmin Connection Factory property SSLCIPHERSUITE and the XMS near-equivalent XMSC\_WMQ\_SSL\_CIPHER\_SPEC. A similar translation is necessary if you specify a value for XMSC\_WMQ\_SSL\_CIPHER\_SUITE but omit value for XMSC\_WMQ\_SSL\_CIPHER\_SPEC.

Table 14 on page 65 lists the available CipherSpecs and their JSSE CipherSuite equivalents.

CipherSpec	Equivalent JSSE CipherSuite
DES_SHA_EXPORT	SSL_RSA_WITH_DES_CBC_SHA
DES_SHA_EXPORT1024	SSL_RSA_EXPORT1024_WITH_DES_CBC_SHA
FIPS_WITH_3DES_EDE_CBC_SHA	SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA

<i>Table 14. Available CipherSpecs and their JSSE CipherSuite equivalents (continued)</i>	
<b>CipherSpec</b>	<b>Equivalent JSSE CipherSuite</b>
FIPS_WITH_DES_CBC_SHA	SSL_RSA_FIPS_WITH_DES_CBC_SHA
NULL_MD5	SSL_RSA_WITH_NULL_MD5
NULL_SHA	SSL_RSA_WITH_NULL_SHA
RC4_56_SHA_EXPORT1024	SSL_RSA_EXPORT1024_WITH_RC4_56_SHA
RC4_MD5_EXPORT	SSL_RSA_EXPORT_WITH_RC4_40_MD5
RC4_MD5_US	SSL_RSA_WITH_RC4_128_MD5
RC4_SHA_US	SSL_RSA_WITH_RC4_128_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA	SSL_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA
TRIPLE_DES_SHA_US	SSL_RSA_WITH_3DES_EDE_CBC_SHA

**Note:** A one-to-one mapping for the CipherSuite name SSL\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA or SSL\_RSA\_WITH\_DES\_CBC\_SHA must account for the setting of the property `XMSC_WMQ_SSL_FIPSREQUIRED` and apply a heuristic.

>If you specify SSL\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA or SSL\_RSA\_WITH\_DES\_CBC\_SHA for the property `XMSC_WMQ_SSL_CIPHER_SUITE`, and there is no value for `XMSC_WMQ_SSL_CIPHER_SPEC`, a value for `XMSC_WMQ_SSL_CIPHER_SPEC` is chosen according to the following tables.

The values used for `XMSC_WMQ_SSL_CIPHER_SPEC` when you specify SSL\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA for the `XMSC_WMQ_SSL_CIPHER_SUITE` property are shown in the following table:

<i>Table 15. Values used for XMSC_WMQ_SSL_CIPHER_SPEC when you specify SSL_RSA_WITH_3DES_EDE_CBC_SHA for the XMSC_WMQ_SSL_CIPHER_SUITE property</i>	
<b>Input: XMSC_WMQ_SSL_FIPSREQUIRED value</b>	<b>Output: XMSC_WMQ_SSL_CIPHER_SPEC chosen</b>
false (that is, MQSSL_FIPS_NO)	TRIPLE_DES_SHA_US
true (that is, MQSSL_FIPS_YES)	TLS_RSA_WITH_3DES_EDE_CBC_SHA

**Note:**

- TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA is deprecated. However, it can still be used to transfer up to 32 GB of data before the connection is terminated with error AMQ9288. To avoid this error, you need to either avoid using triple DES, or enable secret key reset when using this CipherSpec.

The values used for `XMSC_WMQ_SSL_CIPHER_SPEC` when you specify SSL\_RSA\_WITH\_DES\_CBC\_SHA for the `XMSC_WMQ_SSL_CIPHER_SUITE` property are shown in the following table:

<i>Table 16. Values used for XMSC_WMQ_SSL_CIPHER_SPEC when you specify SSL_RSA_WITH_DES_CBC_SHA for the XMSC_WMQ_SSL_CIPHER_SUITE property</i>	
<b>Input: XMSC_WMQ_SSL_FIPSREQUIRED value</b>	<b>Output: XMSC_WMQ_SSL_CIPHER_SPEC chosen</b>
false (that is, MQSSL_FIPS_NO)	DES_SHA_EXPORT
true (that is, MQSSL_FIPS_YES)	TLS_RSA_WITH_DES_CBC_SHA

## Secure connections to a WebSphere Service Integration Bus messaging engine

To enable an XMS application to make secure connections to a WebSphere Service Integration Bus messaging engine, the relevant properties must be defined in the ConnectionFactory object.

XMS provides SSL and HTTPS support for connections to a WebSphere Service Integration Bus. SSL and HTTPS provide secure connections for authentication and confidentiality.

Like WebSphere security, XMS security is configured with respect to JSSE security standards and naming conventions, which include the use of CipherSuites to specify the algorithms that are used when negotiating a secure connection. The protocol used in the encryption negotiation can be either SSL or TLS, depending on which CipherSuite you specify in the ConnectionFactory object.

Table 17 on page 67 lists the properties that must be defined in the ConnectionFactory object.

Name of property	Description
<a href="#">XMSC_WPM_SSL_CIPHER_SUITE</a>	The name of the CipherSuite to be used on an SSL or TLS connection to a WebSphere Service Integration Bus messaging engine. The protocol used in negotiating the secure connection depends on the specified CipherSuite.
<a href="#">XMSC_WPM_SSL_KEYRING_LABEL</a>	The certificate to be used when authenticating with the server.

The following is an example of ConnectionFactory properties for secure connections to a WebSphere integration messaging engine:

```
cf.setStringProperty(XMSC_WPM_PROVIDER_ENDPOINTS, host_name:port_number:chain_name);
cf.setStringProperty(XMSC_WPM_SSL_KEY_REPOSITORY, key_repository_pathname);
cf.setStringProperty(XMSC_WPM_TARGET_TRANSPORT_CHAIN, transport_chain);
cf.setStringProperty(XMSC_WPM_SSL_CIPHER_SUITE, cipher_suite);
cf.setStringProperty(XMSC_WPM_SSL_KEYRING_STASH_FILE, stash_file_pathname);
```

Where chain\_name should be set to either BootstrapTunneledSecureMessaging or BootstrapSecureMessaging, and port\_number is the number of the port on which the bootstrap server listens for incoming requests.

The following is an example of ConnectionFactory properties for secure connections to a WebSphere integration messaging engine with sample values inserted:

```
/* CF properties needed for an SSL connection */
cf.setStringProperty(XMSC_WPM_PROVIDER_ENDPOINTS, "localhost:7286:BootstrapSecureMessaging");
cf.setStringProperty(XMSC_WPM_TARGET_TRANSPORT_CHAIN, "InboundSecureMessaging");
cf.setStringProperty(XMSC_WPM_SSL_KEY_REPOSITORY, "C:\\Program Files\\IBM\\gsk7\\bin\\
XMSkey.kdb");
cf.setStringProperty(XMSC_WPM_SSL_KEYRING_STASH_FILE, "C:\\Program Files\\IBM\\gsk7\\bin\\
XMSkey.sth");
cf.setStringProperty(XMSC_WPM_SSL_CIPHER_SUITE, "SSL_RSA_EXPORT_WITH_RC4_40_MD5");
```

### Related reference

[IConnectionFactory \(for the .NET interface\)](#)

An application uses a connection factory to create a connection.

[Properties of ConnectionFactory](#)

An overview of the properties of the ConnectionFactory object, with links to more detailed reference information.

[Required properties for administered ConnectionFactory objects](#)

When an application creates a connection factory, a number of properties must be defined to create a connection to a messaging server.

### ***CipherSuite and CipherSpec name mappings for connections to a WebSphere Service Integration Bus***

Because IBM Global Security Kit (GSKit) uses CipherSpecs rather than CipherSuites, the JSSE-style CipherSuite names specified in the XMSC\_WPM\_SSL\_CIPHER\_SUITE property must be mapped to the GSKit-style CipherSpec names.

Table 18 on page 68 lists the equivalent CipherSpec for each recognized CipherSuite.

<b>CipherSuite</b>	<b>CipherSpec equivalent</b>
SSL_RSA_WITH_NULL_MD5	NULL_MD5
SSL_RSA_EXPORT_WITH_RC4_40_MD5	RC4_MD5_EXPORT
SSL_RSA_WITH_RC4_128_MD5	RC4_MD5_US
SSL_RSA_WITH_NULL_SHA	NULL_SHA
SSL_RSA_EXPORT1024_WITH_RC4_56_SHA	RC4_56_SHA_EXPORT1024
SSL_RSA_WITH_RC4_128_SHA	RC4_SHA_US
SSL_RSA_WITH_DES_CBC_SHA	DES_SHA_EXPORT
SSL_RSA_EXPORT1024_WITH_DES_CBC_SHA	DES_SHA_EXPORT1024
SSL_RSA_FIPS_WITH_DES_CBC_SHA	FIPS_WITH_DES_CBC_SHA
SSL_RSA_WITH_3DES_EDE_CBC_SHA	TRIPLE_DES_SHA_US
SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA	FIPS_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA

**Note:**

- TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA is deprecated. However, it can still be used to transfer up to 32 GB of data before the connection is terminated with error AMQ9288. To avoid this error, you need to either avoid using triple DES, or enable secret key reset when using this CipherSpec.

## **XMS messages**

This chapter describes the structure and content of XMS messages and explains how applications process XMS messages.

This chapter contains the following sections:

- [“Parts of an XMS message” on page 69](#)
- [“Header fields in an XMS message” on page 69](#)
- [“Properties of an XMS message” on page 70](#)
- [“The body of an XMS message” on page 74](#)
- [“Message selectors” on page 79](#)
- [“Mapping XMS messages onto WebSphere MQ messages” on page 80](#)

## Related reference

[IMessage \(for the .NET interface\)](#)

A Message object represents a message that an application sends or receives. IMessage is a superclass for the message classes such as IMapMessage.

## Parts of an XMS message

An XMS message consists of a header, a set of properties, and a body.

### 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”](#) on page 69.

### 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, see [“Properties of an XMS message”](#) on page 70.

### Body

The body of a message contains application data. For more information, see [“The body of an XMS message”](#) on page 74.

An application can select which messages it wants to receive. 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 79.

## Related reference

[Header fields in an XMS message](#)

To allow an XMS application to exchange messages with a WebSphere JMS application, the header of an XMS message contains the JMS message header fields.

[Properties of an XMS message](#)

XMS supports three kinds of message property: JMS defined properties, IBM defined properties, and application-defined properties.

[The body of an XMS message](#)

The body of a message contains application data. However, a message can have no body, and comprise only the header fields and properties.

[Message selectors](#)

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

[Mapping XMS messages onto WebSphere MQ messages](#)

The JMS header fields and properties of an XMS message are mapped onto fields in the header structures of a WebSphere MQ message.

## Header fields in an XMS message

To allow an XMS application to exchange messages with a WebSphere 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 [“IMessage”](#) on page 120. A header field is always readable and writable.

[Table 19 on page 70](#) lists the JMS message header fields and indicates how the value of each field is set for a transmitted message. 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 19. JMS message header fields

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

### Related reference

#### Parts of an XMS message

An XMS message consists of a header, a set of properties, and a body.

#### Properties of an XMS message

XMS supports three kinds of message property: JMS defined properties, IBM defined properties, and application-defined properties.

#### The body of an XMS message

The body of a message contains application data. However, a message can have no body, and comprise only the header fields and properties.

#### Message selectors

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

#### Mapping XMS messages onto WebSphere MQ messages

The JMS header fields and properties of an XMS message are mapped onto fields in the header structures of a WebSphere MQ message.

## Properties of an XMS message

XMS supports three kinds of message property: JMS defined properties, IBM defined properties, and application-defined properties.

An XMS application can exchange messages with a WebSphere JMS application because XMS supports the following predefined properties of a Message object:

- The same JMS-defined properties that WebSphere JMS supports. The names of these properties begin with the prefix JMSX.
- The same IBM-defined properties that WebSphere JMS supports. The names of these properties begin with the prefix JMS\_IBM\_.

Each predefined property has two names:

- A JMS name, for a JMS-defined property, or a WebSphere JMS name, for an IBM-defined property.

This is the name by which the property is known in JMS or WebSphere JMS, and it is also the name that is transmitted with a message that has this property. An XMS application uses this name to identify the property in a message selector expression.

- An XMS name to identify the property in all situations except in a message selector expression. Each XMS name is defined as a named constant in IBM.XMS.XMSC class. The value of the named constant is the corresponding JMS or WebSphere JMS name.

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*.

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.

The received message, when forwarded after clearing up the message properties, will behave in a manner consistent with the behavior of forwarding a standard WMQ XMS for .NET `BytesMessage` with message properties cleared up.

This is, however, not recommended since the following properties will be lost:

- `JMS_IBM_Encoding` property value, implying that the message data cannot be decoded meaningfully.
- `JMS_IBM_Format` property value, implying that the header chaining between the (MQMD or the new MQRFH2) message header and existing headers would be broken.

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 it can use the methods of the `Property` class to retrieve the name, data type, and value of each property.

### **Related reference**

#### Parts of an XMS message

An XMS message consists of a header, a set of properties, and a body.

#### Header fields in an XMS message

To allow an XMS application to exchange messages with a WebSphere JMS application, the header of an XMS message contains the JMS message header fields.

#### The body of an XMS message

The body of a message contains application data. However, a message can have no body, and comprise only the header fields and properties.

#### Message selectors

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

#### Mapping XMS messages onto WebSphere MQ messages

The JMS header fields and properties of an XMS message are mapped onto fields in the header structures of a WebSphere MQ message.

### ***JMS-defined properties of a message***

Several JMS-defined properties of a message are supported by both XMS and WebSphere JMS.

Table 20 on page 72 lists the JMS-defined properties of a message that are supported by both XMS and WebSphere JMS. For a description of the JMS-defined properties, see *Java Message Service Specification, Version 1.1*. The JMS-defined properties are not valid for a real-time connection to a broker.

The table specifies the data type of each property and indicates how the value of the property is set for a transmitted message. 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 20. JMS-defined properties of a message

<b>XMS name of the JMS defined property</b>	<b>JMS name</b>	<b>Data type</b>	<b>How the value is set for a transmitted message (in the format <i>method [class]</i>)</b>
JMSX_APPID	JMSXAppID	System.String	Send [MessageProducer]
JMSX_DELIVERY_COUNT	JMSXDeliveryCount	System.Int32	Receive [MessageConsumer]
JMSX_GROUPID	JMSXGroupID	System.String	Set String Property [PropertyContext]
JMSX_GROUPSEQ	JMSXGroupSeq	System.Int32	Set Integer Property [PropertyContext]
JMSX_USERID	JMSXUserID	System.String	Send [MessageProducer]

### **IBM-defined properties of a message**

Several IBM-defined properties of a message are supported by XMS and WebSphere JMS.

Table 21 on page 72 lists the IBM defined properties of a message that are supported by both XMS and WebSphere JMS. For more information about the IBM-defined properties, see *IBM WebSphere MQ Using Java* or the WebSphere Application Server product documentation.

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

Table 21. IBM-defined properties of a message

<b>XMS name of the IBM defined property</b>	<b>WebSphere JMS name</b>	<b>Data type</b>	<b>How the value is set for a transmitted message (in the format <i>method [class]</i>)</b>
JMS_IBM_CHARACTER_SET	JMS_IBM_Character_Set	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_ENCODING	JMS_IBM_Encoding	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_EXCEPTIONMESSAGE	JMS_IBM_ExceptionMessage	System.String	Receive [MessageConsumer]
JMS_IBM_EXCEPTIONREASON	JMS_IBM_ExceptionReason	System.Int32	Receive [MessageConsumer]
JMS_IBM_EXCEPTIONTIMESTAMP	JMS_IBM_ExceptionTimestamp	System.Int64	Receive [MessageConsumer]
JMS_IBM_EXCEPTIONPROBLEMDESTINATION	JMS_IBM_ExceptionProblemDestination	System.String	Receive [MessageConsumer]
JMS_IBM_FEEDBACK	JMS_IBM_Feedback	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_FORMAT	JMS_IBM_Format	System.String	Set String Property [PropertyContext]

Table 21. IBM-defined properties of a message (continued)

<b>XMS name of the IBM defined property</b>	<b>WebSphere JMS name</b>	<b>Data type</b>	<b>How the value is set for a transmitted message (in the format <i>method [class]</i>)</b>
JMS_IBM_LAST_MSG_IN_GROUP	JMS_IBM_Last_Msg_In_Group	System.Boolean	Set Integer Property [PropertyContext]
JMS_IBM_MSGTYPE	JMS_IBM_MsgType	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_PUTAPPLTYPE	JMS_IBM_PutApplType	System.Int32	Send [MessageProducer]
JMS_IBM_PUTDATE	JMS_IBM_PutDate	System.String	Send [MessageProducer]
JMS_IBM_PUTTIME	JMS_IBM_PutTime	System.String	Send [MessageProducer]
JMS_IBM_REPORT_COA	JMS_IBM_Report_COA	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_COD	JMS_IBM_Report_COD	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_DISCARD_MSG	JMS_IBM_Report_Discard_Msg	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_EXCEPTION	JMS_IBM_Report_Exception	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_EXPIRATION	JMS_IBM_Report_Expiration	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_NAN	JMS_IBM_Report_NAN	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_PAN	JMS_IBM_Report_PAN	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_PASS_CORREL_ID	JMS_IBM_Report_Pass_Correl_ID	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_PASS_MSG_ID	JMS_IBM_Report_Pass_Msg_ID	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_SYSTEM_MESSAGEID	JMS_IBM_System_MessageID	System.String	Send [MessageProducer]

### **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 on the values of these properties.

To allow a WebSphere JMS application to select and process messages sent by an XMS application, the name of an application-defined property must conform to the rules for forming identifiers in message selector expressions, as documented in *WebSphere MQ Using Java*. The value of an application-defined property must have one of the following data types: System.Boolean, System.SByte, System.Int16, System.Int32, System.Int64, System.Float, System.Double, or System.String.

## The body of an XMS message

The body of a message contains application data. However, a message can have no body, and comprise only the header fields and properties.

XMS supports five 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 `IBytesMessage` interface contains the methods to process the body of a bytes message. For more information, see [“Bytes messages” on page 76](#).

### Map

The body contains a set of name-value pairs, where each value has an associated data type. A message with this type of body is called a *map message*. The `IMapMessage` interface contains the methods to process the body of a map message. For more information, see [“Map messages” on page 76](#).

### Object

The body contains a serialized Java or .NET object. A message with this type of body is called an *object message*. The `IObjectMessage` interface contains the methods to process the body of an object message. For more information, see [“Object messages” on page 77](#).

### Stream

The body contains a stream of values, where each value has an associated data type. A message with this type of body is called a *stream message*. The `IStreamMessage` interface contains the methods to process the body of a stream message. For more information, see [“Stream messages” on page 78](#).

### Text

The body contains a string. A message with this type of body is called a *text message*. The `ITextMessage` interface contains the methods to process the body of a text message. For more information, see [“Text messages” on page 79](#).

The `IMessage` interface is the parent of all message objects and can be used in messaging functions to represent any of the XMS message types.

For information about the size and maximum and minimum values of each of these data types, see [Table 5 on page 40](#).

## Related reference

### [Parts of an XMS message](#)

An XMS message consists of a header, a set of properties, and a body.

### [Header fields in an XMS message](#)

To allow an XMS application to exchange messages with a WebSphere JMS application, the header of an XMS message contains the JMS message header fields.

### [Properties of an XMS message](#)

XMS supports three kinds of message property: JMS defined properties, IBM defined properties, and application-defined properties.

### [Message selectors](#)

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

### [Mapping XMS messages onto WebSphere MQ messages](#)

The JMS header fields and properties of an XMS message are mapped onto fields in the header structures of a WebSphere MQ message.

## Data types for elements of application data

To ensure that an XMS application can exchange messages with a WebSphere MQ classes for JMS application, both the applications 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 22 on page 75](#). For each data type, the table shows the

compatible Java data type. XMS provides the methods to write elements of application data only with these data types.

<i>Table 22. XMS data types that are compatible with Java data types</i>		
<b>XMS Data type</b>	<b>Represents</b>	<b>Compatible Java data type</b>
System.Boolean	The boolean value true or false	boolean
System.Char16	Double byte character	char
System.SByte	Signed 8-bit integer	byte
System.Int16	Signed 16-bit integer	short
System.Int32	Signed 32-bit integer	int
System.Int64	Signed 64-bit integer	long
System.Float	Signed floating point number	float
System.Double	Signed double precision floating point number	double
System.String	String of characters	String

For information about the size, maximum value and minimum value of each of these data types, see [“XMS primitive types”](#) on page 39.

### **Related concepts**

#### Attributes and properties of objects

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

#### XMS primitive types

XMS provides equivalents of the eight Java primitive types (byte, short, int, long, float, double, char, and boolean). This allows the interchange of messages between XMS and JMS without data becoming lost or corrupted.

#### Implicit conversion of a property value from one data type to another

When an application gets the value of a property, the value can be converted by XMS into another data type. Many rules govern which conversions are supported and how XMS performs the conversions.

### **Related reference**

#### Bytes messages

The body of a bytes message contains a stream of bytes. The body contains only the actual data, and it is the responsibility of the sending and receiving applications to interpret this data.

#### Map messages

The body of a map message contains a set of name-value pairs, where each value has an associated data type.

#### Object messages

The body of an object message contains a serializedJava or .NET object.

#### Stream messages

The body of a stream message contains a stream of values, where each value has an associated data type.

#### Text messages

The body of a text message contains a string.

### **Bytes messages**

The body of a bytes message contains a stream of bytes. The body contains only the actual data, and it is the responsibility of the sending and receiving applications to interpret this data.

Bytes messages are useful if an XMS application needs to exchange messages with applications that are not using the XMS or JMS application programming interface.

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 `IBytesMessage` interface for .NET. 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 repeatedly.

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 `IBytesMessage` interface 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.

If an application calls the `Reset` method of the `IBytesMessage` interface when the body of a bytes message is writeable, 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 `IMessage` interface for .NET when the body of a bytes message is read-only, the body becomes writeable. The method also clears the body.

### **Related reference**

#### Data types for elements of application data

To ensure that an XMS application can exchange messages with a WebSphere MQ classes for JMS application, both the applications must be able to interpret the application data in the body of a message in the same way.

#### Map messages

The body of a map message contains a set of name-value pairs, where each value has an associated data type.

#### Object messages

The body of an object message contains a `serializedJava` or .NET object.

#### Stream messages

The body of a stream message contains a stream of values, where each value has an associated data type.

#### Text messages

The body of a text message contains a string.

#### IBytesMessage (for the .NET interface)

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

### **Map messages**

The body of a map message contains a set of name-value pairs, where each value has an associated data type.

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 22 on page 75](#). 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.

A .NET application can use the `MapNames` property to get an enumeration of the names in the body of the map message.

When an application gets the value of a name-value pair, the value can be converted by XMS into another data type. For example, to get an integer from the body of a map message, an application can call the `GetString` method of the `MapMessage` class, which returns the integer as a string. The supported conversions are the same as those that are supported when XMS converts a property value from one data type to another. For more information about the supported conversions, see [“Implicit conversion of a property value from one data type to another”](#) on page 40.

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.

### **Related concepts**

#### Implicit conversion of a property value from one data type to another

When an application gets the value of a property, the value can be converted by XMS into another data type. Many rules govern which conversions are supported and how XMS performs the conversions.

### **Related reference**

#### Data types for elements of application data

To ensure that an XMS application can exchange messages with a WebSphere MQ classes for JMS application, both the applications must be able to interpret the application data in the body of a message in the same way.

#### Bytes messages

The body of a bytes message contains a stream of bytes. The body contains only the actual data, and it is the responsibility of the sending and receiving applications to interpret this data.

#### Object messages

The body of an object message contains a `serializedJava` or `.NET` object.

#### Stream messages

The body of a stream message contains a stream of values, where each value has an associated data type.

#### Text messages

The body of a text message contains a string.

#### IMapMessage (for the .NET interface)

A map message is a message whose body comprises a set of name-value pairs, where each value has an associated data type.

### ***Object messages***

The body of an object message contains a `serializedJava` or `.NET` object.

An XMS application can receive an object message, change its header fields and properties, and then send it to another destination. An application can also copy the body of an object message and use it to form another object message. XMS treats the body of an object message as an array of bytes.

After an application creates an object 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 an object message, the body of the message is read-only. If an application calls the `Clear Body` method of the `IMessage` interface for `.NET` when the body of an object message is read-only, the body becomes readable and writable. The method also clears the body.

### **Related reference**

#### Data types for elements of application data

To ensure that an XMS application can exchange messages with a WebSphere MQ classes for JMS application, both the applications must be able to interpret the application data in the body of a message in the same way.

#### Bytes messages

The body of a bytes message contains a stream of bytes. The body contains only the actual data, and it is the responsibility of the sending and receiving applications to interpret this data.

#### Map messages

The body of a map message contains a set of name-value pairs, where each value has an associated data type.

#### Stream messages

The body of a stream message contains a stream of values, where each value has an associated data type.

#### Text messages

The body of a text message contains a string.

#### IObjectMessage (for the .NET interface)

An object message is a message whose body comprises a serialized Java or .NET object.

### ***Stream messages***

The body of a stream message contains a stream of values, where each value has an associated data type.

The data type of a value is one of the XMS data types listed in [Table 22 on page 75](#).

After an application creates a stream message, the body of the message is writable. The application assembles the application data into the body by calling the appropriate write methods of the `IStreamMessage` interface for .NET. Each time the application writes a value to the message stream, the value, and its data type are assembled immediately after the previous value written by the application. XMS maintains an internal cursor to remember the position of the last value 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 stream message, the body of the message is read-only. The application can use the appropriate read methods of the `IStreamMessage` interface for .NET to read the contents of the message stream. The application reads the values in sequence, and XMS maintains an internal cursor to remember the position of the last value that was read.

When an application reads a value from the message stream, the value can be converted by XMS into another data type. For example, to read an integer from the message stream, an application can call the `ReadString` method, which returns the integer as a string. The supported conversions are the same as those that are supported when XMS converts a property value from one data type to another. For more information about the supported conversions, see [“Implicit conversion of a property value from one data type to another” on page 40](#).

If an error occurs while an application is attempting to read a value from the message stream, the cursor is not advanced. The application can recover from the error by attempting to read the value as another data type.

If an application calls the `Reset` method of the `IStreamMessage` interface for .NET when the body of a stream message is write-only, the body becomes read-only. The method also repositions the cursor at the beginning of the message stream.

If an application calls the `Clear Body` method of the `IMessage` interface for .NET when the body of a stream message is read-only, the body becomes write-only. The method also clears the body.

#### **Related concepts**

##### Implicit conversion of a property value from one data type to another

When an application gets the value of a property, the value can be converted by XMS into another data type. Many rules govern which conversions are supported and how XMS performs the conversions.

#### **Related reference**

##### Data types for elements of application data

To ensure that an XMS application can exchange messages with a WebSphere MQ classes for JMS application, both the applications must be able to interpret the application data in the body of a message in the same way.

#### Bytes messages

The body of a bytes message contains a stream of bytes. The body contains only the actual data, and it is the responsibility of the sending and receiving applications to interpret this data.

#### Map messages

The body of a map message contains a set of name-value pairs, where each value has an associated data type.

#### Object messages

The body of an object message contains a serializedJava or .NET object.

#### Text messages

The body of a text message contains a string.

#### IStreamMessage (for the .NET interface)

A stream message is a message whose body comprises a stream of values, where each value has an associated data type. The contents of the body are written and read sequentially.

### ***Text messages***

The body of a text message contains a string.

After an application creates a text 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 text message, the body of the message is read-only. If an application calls the Clear Body method of the IMessage interface for .NET when the body of a text message is read-only, the body becomes readable and writable. The method also clears the body.

### **Related reference**

#### Data types for elements of application data

To ensure that an XMS application can exchange messages with a WebSphere MQ classes for JMS application, both the applications must be able to interpret the application data in the body of a message in the same way.

#### Bytes messages

The body of a bytes message contains a stream of bytes. The body contains only the actual data, and it is the responsibility of the sending and receiving applications to interpret this data.

#### Map messages

The body of a map message contains a set of name-value pairs, where each value has an associated data type.

#### Object messages

The body of an object message contains a serializedJava or .NET object.

#### Stream messages

The body of a stream message contains a stream of values, where each value has an associated data type.

#### ITextMessage (for the .NET interface)

A text message is a message whose body comprises a string.

## **Message selectors**

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

When an application creates a message consumer, it can associate a message selector expression with the consumer. The message selector expression specifies the selection criteria.

When an application is connecting to IBM WebSphere MQ V7.0 queue manager the message selection is done at the queue manager side. XMS does not do any selection and simply delivers the message it received from the queue manager thus providing better performance.

An application can create more than one message consumer, each with its own message selector expression. If an incoming message meets 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 classes for JMS. For information about how to construct a message selector expression, see *WebSphere MQ Using Java*. Note that, in a message selector expression, the names of JMS-defined properties must be the JMS names, and the names of IBM-defined properties must be the WebSphere MQ classes for JMS names. You cannot use the XMS names in a message selector expression.

### Related reference

#### Parts of an XMS message

An XMS message consists of a header, a set of properties, and a body.

#### Header fields in an XMS message

To allow an XMS application to exchange messages with a WebSphere JMS application, the header of an XMS message contains the JMS message header fields.

#### Properties of an XMS message

XMS supports three kinds of message property: JMS defined properties, IBM defined properties, and application-defined properties.

#### The body of an XMS message

The body of a message contains application data. However, a message can have no body, and comprise only the header fields and properties.

#### Mapping XMS messages onto WebSphere MQ messages

The JMS header fields and properties of an XMS message are mapped onto fields in the header structures of a WebSphere MQ message.

## Mapping XMS messages onto WebSphere MQ messages

The JMS header fields and properties of an XMS message are mapped onto fields in the header structures of a WebSphere MQ message.

When an XMS application is connected to a WebSphere MQ queue manager, messages sent to the queue manager are mapped onto WebSphere MQ messages in the same way that WebSphere MQ classes for JMS messages are mapped onto WebSphere MQ messages in similar circumstances.

If the `XMSC_WMQ_TARGET_CLIENT` property of a Destination object is set to `XMSC_WMQ_TARGET_DEST_JMS`, the JMS header fields and properties of a message sent to the destination are mapped onto fields in the MQMD and MQRFH2 header structures of the WebSphere MQ message. Setting the `XMSC_WMQ_TARGET_CLIENT` property in this way assumes that the application

that receives the message can handle an MQRFH2 header. The receiving application might therefore be another XMS application, a WebSphere MQ classes for JMS application, or a native WebSphere MQ application that has been designed to handle an MQRFH2 header.

If the XMSC\_WMQ\_TARGET\_CLIENT property of a Destination object is set to XMSC\_WMQ\_TARGET\_DEST\_MQ instead, the JMS header fields and properties of a message sent to the destination are mapped onto fields in the MQMD header structure of the WebSphere MQ message. The message does not contain an MQRFH2 header, and any JMS header fields and properties that cannot be mapped onto fields in the MQMD header structure are ignored. The application that receives the message can therefore be a native WebSphere MQ that's not designed to handle an MQRFH2 header.

WebSphere MQ messages received from a queue manager are mapped onto XMS messages in the same way that WebSphere MQ messages are mapped onto WebSphere MQ classes for JMS messages in similar circumstances.

If an incoming WebSphere MQ message has an MQRFH2 header, the resulting XMS message has a body whose type is determined by the value of the **Msd** property contained in the mcd folder of the MQRFH2 header. If the **Msd** property is not present in the MQRFH2 header, or if the WebSphere MQ message has no MQRFH2 header, the resulting XMS message has a body whose type is determined by the value of the *Format* field in the MQMD header. If the *Format* field is set to MQFMT\_STRING, the XMS message is a text message. Otherwise, the XMS message is a bytes message. If the WebSphere MQ message has no MQRFH2 header, only those JMS header fields and properties that can be derived from fields in the MQMD header are set.

For more information about mapping WebSphere MQ classes for JMS messages onto WebSphere MQ messages, see *WebSphere MQ Using Java*.

### **Related reference**

#### Parts of an XMS message

An XMS message consists of a header, a set of properties, and a body.

#### Header fields in an XMS message

To allow an XMS application to exchange messages with a WebSphere JMS application, the header of an XMS message contains the JMS message header fields.

#### Properties of an XMS message

XMS supports three kinds of message property: JMS defined properties, IBM defined properties, and application-defined properties.

#### The body of an XMS message

The body of a message contains application data. However, a message can have no body, and comprise only the header fields and properties.

#### Message selectors

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

### ***Reading and writing the message descriptor from a Message Service Client for .NET application***

You can access all the message descriptor (MQMD) fields of a WebSphere MQ message except *StrucId* and *Version*; *BackoutCount* can be read but not written to. This feature is available only when connecting to a WebSphere MQ queue manager Version 6 and later, and is controlled by destination properties described later.

The message attributes provided by the Message Service Client for .NET facilitates XMS applications to set MQMD fields and also to drive WebSphere MQ applications.

Some restrictions apply when using publish/subscribe messaging. For example, MQMD fields like *MsgID* and *CorrelId*, if set, are ignored.

The function described in this topic is unavailable for publish/subscribe messaging when you are either connecting to a WebSphere MQ V6 queue manager. It is also unavailable when the **PROVIDERVERSION** property is set to 6.

## **Accessing IBM WebSphere MQ Message data from a Message Service Client for .NET application**

You can access the complete WebSphere MQ message data including the MQRFH2 header (if present) and any other WebSphere MQ headers (if present) within a Message Service Client for .NET application as the body of a `JMSBytesMessage`.

The function described in this topic is available only when connecting to a WebSphere MQ queue manager at Version 7 or later and the WebSphere MQ messaging provider is in normal mode.

Destination object properties determine how the XMS application accesses the whole of a WebSphere MQ message (including the MQRFH2 header, if present) as the body of a `JMSBytesMessage`.

## **Troubleshooting**

---

This chapter provides information to help you to detect and deal with problems when using Message Service Client for .NET.

This chapter contains the following sections:

- [“Trace configuration for .NET applications” on page 82](#)
- [“FFDC configuration for .NET applications” on page 86](#)
- [“Tips for troubleshooting” on page 86](#)

## **Trace configuration for .NET applications**

For XMS .NET applications, you can configure trace from an application configuration file as well as from the XMS environment variables. You can select the components that you want to trace. Trace is normally used under the guidance of IBM Support.

Tracing for XMS .NET is based on the standard .NET trace infrastructure.

All tracing except for error tracing is disabled by default. You can turn on tracing and configure the trace settings in either of the following ways:

- By using an application configuration file with a name that consists of the name of the executable program to which the file relates, with the suffix `.config`. For example, the application configuration file for `text.exe` would have the name `text.exe.config`. Using an application configuration file is the preferred way of enabling trace for XMS .NET applications. For further details, see [“Trace configuration using an application configuration file” on page 83](#).
- By using XMS environment variables as for XMS C or C++ applications. For further details, see [“Trace configuration using XMS environment variables” on page 85](#).

The active trace file has a name of the format `xms_trace<PID>.log` where `<PID>` represents the process ID of the application. The size of the active trace file is by default limited to 20 MB. When this limit is reached, the file is renamed and archived. Archived files have names of the format `xms_trace<PID>_YY.MM.DD_HH.MM.SS.log`

By default, the number of trace files that are retained is four, that is, one active file and three archived files. These four files are used as a rolling buffer until the application stops, with the oldest file being removed and replaced by the newest file. You can change the number of trace files by specifying a different number in the application configuration file. However, there must be at least two files (one active file and one archived file).

Two trace file formats are available:

- Basic format trace files are human readable, in a WebSphere Application Server format. This format is the default trace file format. The basic format is not compatible with trace analyzer tools.
- Advanced format trace files are compatible with trace analyzer tools. You must specify that you want to produce trace files in advanced format in the application configuration file.

Trace entries contain the following information:

- The date and time that the trace was logged
- The class name
- The trace type
- The trace message

The following example shows an extract from some trace:

```
[09/11/2005 14:33:46:914276]    00000004    IBM.XMS.Comms.IoRequest    >    Allocate    Entry
[09/11/2005 14:33:46:914276]    00000004    IBM.XMS.Comms.IoRequest    >    Initialize    Entry
[09/11/2005 14:33:46:914276]    00000004    IBM.XMS.Comms.IoRequest    <    Initialize    Exit
[09/11/2005 14:33:46:914276]    00000004    IBM.XMS.Comms.IoRequest    <    Allocate    Exit
```

In the previous example, the format is:

[Date Time:Microsecs] or Exit	Thread-id	Classname	Trace-type	Methodname	Entry
----------------------------------	-----------	-----------	------------	------------	-------

where Trace-type is:

- > for Entry
- < for Exit
- d for Debug information

## Trace configuration using an application configuration file

The preferred way of configuring trace for XMS .NET applications is with an application configuration file. The trace section of this file includes parameters that define what is to be traced, the trace file location and maximum allowed size, the number of trace files used, and the trace file format.

To turn on trace using the application configuration file, you simply place the file in the same directory as the executable file for your application.

Trace can be enabled both by component and trace type. It is also possible to turn on trace for an entire trace group. You can turn on trace for components in a hierarchy either individually or collectively. The types of trace available include:

- Debug trace
- Exception trace
- Warnings, informational messages, and error messages
- Method entry and exit trace

The following example shows the trace settings defined in the Trace section of an application configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <configSections>
    <sectionGroup name="IBM.XMS">
      <section name="Trace"
        type="System.Configuration.SingleTagSectionHandler"/>
    </sectionGroup>
  </configSections>

  <IBM.XMS>
    <Trace traceSpecification="*=all=enabled" traceFilePath=""
      traceFileSize="20000000" traceFileNumber="3"
      traceFormat="advanced"/>
  </IBM.XMS>
</configuration>
```

[Table 23 on page 84](#) describes the parameter settings in more detail.

Table 23. Application configuration file trace parameter settings

Parameter	Description
<code>traceSpecification=&lt;ComponentName&gt;=&lt;type&gt;=&lt;state&gt;</code>	<p>&lt;ComponentName&gt; is the name of the class that you want to trace. You can use a * wildcard character in this name. For example, <code>*=all=enabled</code> specifies that you want to trace all classes, and <code>IBM.XMS.impl.*=all=enabled</code> specifies that you require API trace only.</p> <p>&lt;type&gt; can be any of the following trace types:</p> <ul style="list-style-type: none"> <li>• all</li> <li>• debug</li> <li>• event</li> <li>• EntryExit</li> </ul> <p>&lt;state&gt; can be either enabled or disabled.</p> <p>You can string multiple trace elements together by using a ':' (colon) delimiter.</p>
<code>traceFilePath="&lt;filename&gt;"</code>	<p>If you do not specify a traceFilePath, or if the traceFilePath is present but contains an empty string, the trace file is placed in the current directory. To store the trace file in a named directory, specify the directory name in the traceFilePath, for example:</p> <pre style="background-color: #f0f0f0; padding: 5px;">traceFilePath="c:\somepath"</pre>
<code>traceFileSize="&lt;size&gt;"</code>	<p>The maximum allowed size of the trace file. When a file reaches this size, it is archived and renamed. The default maximum is 20 KB, which is specified as <code>traceFileSize="20000000"</code>.</p>
<code>traceFileNumber="&lt;number&gt;"</code>	<p>The number of trace files that are to be retained. The default is 4 (one active file and 3 archive files). The minimum number allowed is 2.</p>
<code>traceFormat="&lt;format&gt;"</code>	<p>The default trace format is basic. Trace files are produced in this format if you specify <code>traceFormat="basic"</code>, or if you do not specify a traceFormat, or if the traceFormat is present but contains an empty string.</p> <p>If you require trace that is compatible with trace analyzer tools, you must specify <code>traceFormat="advanced"</code>.</p>

The trace settings in the application configuration file are dynamic, and are reread every time the file is saved or replaced. If errors are found in the file after it is edited, the trace file settings revert to their default values.

**Related concepts**

[Trace configuration using XMS environment variables](#)

As an alternative to using an application configuration file, you can turn on trace using XMS environment variables. These environment variables are only used if there is no trace specification in the application configuration file.

## Trace configuration using XMS environment variables

As an alternative to using an application configuration file, you can turn on trace using XMS environment variables. These environment variables are only used if there is no trace specification in the application configuration file.

To configure trace for an XMS .NET application, set the following environment variables before running the application:

<i>Table 24. Environment variable settings for .NET trace</i>			
<b>Environment variables</b>	<b>Default</b>	<b>Settings</b>	<b>Meaning</b>
XMS_TRACE_ON	Not applicable	Not applicable: the value of this variable is ignored	If XMS_TRACE_ON is set, all trace is enabled by default.
XMS_TRACE_FILE_PATH	Current working directory	/dirpath/	The directory path that trace and FFDC records are written to.  XMS creates FFDC and trace files in the current working directory, unless you specify an alternative location. You can 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 FFDC and trace files. You must set this environment variable before you start the application that you want to trace. 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 FFDC and trace files.
XMS_TRACE_FORMAT	BASIC	BASIC, ADVANCED	Specifies the required trace format, which can be either BASIC or ADVANCED. The default format is BASIC. The ADVANCED format is compatible with trace analyzer tools.

Table 24. Environment variable settings for .NET trace (continued)

Environment variables	Default	Settings	Meaning
XMS_TRACE_SPECIFICATION	Not applicable	See <a href="#">“Trace configuration using an application configuration file”</a> on page 83	Overrides the trace specification, which follows the format specified in <a href="#">“Trace configuration using an application configuration file”</a> on page 83.

### Related concepts

[Trace configuration using an application configuration file](#)

The preferred way of configuring trace for XMS .NET applications is with an application configuration file. The trace section of this file includes parameters that define what is to be traced, the trace file location and maximum allowed size, the number of trace files used, and the trace file format.

## FFDC configuration for .NET applications

For the .NET implementation of XMS, one FFDC file is produced for each FFDC.

First Failure Data Capture (FFDC) files are stored in human readable text files. These files have names of the form `xmsffdc<processID>_<Date>T<Timestamp>.txt`. An example of a file name is `xmsffdc264_2006.01.06T13.18.52.990955.txt`. The timestamp contains microseconds resolution.

Files start with the date and time that the exception occurred, followed by the exception type. The files include a unique short probeId, which can be used to locate where this FFDC occurred.

You do not need to carry out any configuration to turn on FFDC. By default, all FFDC files are written to the current directory. However, if required, you can specify a different directory by changing `ffdcDirectory` in the Trace section of the application configuration file. In the following example, all trace files are logged to the directory `c:\client\ffdc`:

```
<IBM.XMS>
  <Trace ffdc=true ffdcDirectory="c:\client\ffdc"/>
</IBM.XMS>
```

You can disable trace by setting FFDC to false in the Trace section of the application configuration file.

If you are not using an application configuration file, FFDC is on and trace is off.

## Tips for troubleshooting

Use these tips to help you troubleshoot problems with using XMS.

### An XMS application cannot connect to a queue manager (MQRC\_NOT\_AUTHORIZED)

The XMS .NET client may have different behavior from the behavior of the WebSphere MQ JMS client. Therefore, you may find that your XMS application cannot connect to your queue manager, although your JMS application can.

- A simple solution to this problem is to try using a user ID that is no more than 12 characters long and is authorized completely in the queue manager's authority list. If this solution is not ideal, a different but more complex approach would be to use security exits. If you need further help on this issue, contact IBM Support for assistance.
- If you set the `XMSC_USERID` property of the connection factory, it must match the user ID and password of the logged on user. If you do not set this property, the queue manager uses the user ID of the logged on user by default.

- User authentication for WebSphere MQ is performed by using the details of the user currently logged on and not the information provided in the XMSC.USERID and XMSC.PASSWORD fields. This is designed to maintain consistency with WebSphere MQ. For more information on authentication, refer to *Authentication Information* in the online IBM WebSphere MQ product documentation.

### Connection redirected to the messaging engine

When you connect to a WebSphere Application Server Version 6.0.2 service integration bus, all connections may be redirected from the original provider endpoint to the messaging engine that the bus chooses for that client connection. When doing so, it will always redirect the connection to a host server specified by the hostname, rather than by an IP address. Therefore, you may experience connection problems if the hostname cannot be resolved.

To successfully connect to WebSphere Application Server Version 6.0.2 service integration bus, you may need to provide a mapping between the hostnames and IP addresses on your client host machine. For example you can specify the mapping in a local hosts table on your client host machine.

### Support for telnet-like password authentication

The XMS .NET Real Time Transport protocol supports only simple telnet-like password authentication. The XMS .NET Real Time Transport protocol does not support Quality Of Protection.

### Setting values for property type double

On a Windows 64-bit platform, the SetDoubleProperty() or GetDoubleProperty() methods may not work correctly when setting or getting values for the property type double, if the values are smaller than Double.Epsilon.

For example, if you try to set a value of 4.9E-324 for a property with type double, the Windows 64-bit platforms treat it as 0.0. So, in a distributed messaging environment, if a JMS or another application sets the value for a double property as 4.9E-324 on any Unix or Windows 32-bit machine, and XMS .Net runs on a 64-bit machine, the value returned by GetDoubleProperty() is 0.0. This is a known issue with Microsoft .NET 2.0 Framework.

## Message Service Clients for .NET reference

This reference section provides information to help you with using Message Service Client for .NET. This information helps you carry out the tasks involved in programming with XMS.

### .NET interfaces

This section documents the .NET class interfaces and their properties and methods.

The following table summarizes all the interfaces, which are defined within the IBM.XMS namespace.

Interface	Description
<a href="#">“IBytesMessage” on page 90</a>	A bytes message is a message whose body comprises a stream of bytes.
<a href="#">“IConnection” on page 100</a>	A Connection object represents the active connection of the application to a messaging server.
<a href="#">“IConnectionFactory” on page 103</a>	An application uses a connection factory to create a connection.
<a href="#">“IConnectionMetaData” on page 105</a>	A ConnectionMetaData object provides information about a connection.

Table 25. Summary of the .NET class interfaces (continued)

Interface	Description
<a href="#">“IDestination” on page 105</a>	A destination is where an application sends messages, or it is a source from which an application receives messages, or both.
<a href="#">“ExceptionListener” on page 107</a>	An application uses an exception listener to be notified asynchronously of a problem with a connection.
<a href="#">“IllegalStateException” on page 107</a>	XMS throws this exception if an application calls a method at an incorrect or inappropriate time, or if XMS is not in an appropriate state for the requested operation.
<a href="#">“InitialContext” on page 108</a>	An application uses an InitialContext object to create objects from object definitions that are retrieved from a repository of administered objects.
<a href="#">“InvalidClientIDException” on page 110</a>	XMS throws this exception if an application attempts to set a client identifier for a connection, but the client identifier is not valid or is already in use.
<a href="#">“InvalidDestinationException” on page 110</a>	XMS throws this exception if an application specifies a destination that is not valid.
<a href="#">“InvalidSelectorException” on page 111</a>	XMS throws this exception if an application provides a message selector expression whose syntax is not valid.
<a href="#">“IMapMessage” on page 111</a>	A map message is a message whose body comprises a set of name-value pairs, where each value has an associated data type.
<a href="#">“IMessage” on page 120</a>	A Message object represents a message that an application sends or receives. IMessage is a superclass for the message classes such as IMapMessage.
<a href="#">“IMessageConsumer” on page 126</a>	An application uses a message consumer to receive messages sent to a destination.
<a href="#">“MessageEOFException” on page 129</a>	XMS throws this exception if XMS encounters the end of a bytes message stream when an application is reading the body of a bytes message.
<a href="#">“MessageFormatException” on page 129</a>	XMS throws this exception if XMS encounters a message with a format that is not valid.
<a href="#">“IMessageListener (delegate)” on page 129</a>	An application uses a message listener to receive messages asynchronously.
<a href="#">“MessageNotReadableException” on page 130</a>	XMS throws this exception if an application attempts to read the body of a message that is write only.
<a href="#">“MessageNotWritableException” on page 130</a>	XMS throws this exception if an application attempts to write to the body of a message that is read-only.

Table 25. Summary of the .NET class interfaces (continued)

Interface	Description
<a href="#">“IMessageProducer” on page 130</a>	An application uses a message producer to send messages to a destination.
<a href="#">“IObjectMessage” on page 136</a>	An object message is a message whose body comprises a serialized Java or .NET object.
<a href="#">“IPropertyContext” on page 137</a>	IPropertyContext is an abstract superclass that contains methods that get and set properties. These methods are inherited by other classes.
<a href="#">“IQueueBrowser” on page 146</a>	An application uses a queue browser to browse messages on a queue without removing them.
<a href="#">“Requestor” on page 148</a>	An application uses a requestor to send a request message and then wait for, and receive, the reply.
<a href="#">“ResourceAllocationException” on page 149</a>	XMS throws this exception if XMS cannot allocate the resources required by a method.
<a href="#">“SecurityException” on page 149</a>	XMS throws this exception if the user identifier and password provided to authenticate an application are rejected. XMS also throws this exception if an authority check fails and prevents a method from completing.
<a href="#">“ISession” on page 150</a>	A session is a single threaded context for sending and receiving messages.
<a href="#">“IStreamMessage” on page 160</a>	A stream message is a message whose body comprises a stream of values, where each value has an associated data type.
<a href="#">“ITextMessage” on page 169</a>	A text message is a message whose body comprises a string.
<a href="#">“TransactionInProgressException” on page 170</a>	XMS throws this exception if an application requests an operation that is not valid because a transaction is in progress.
<a href="#">“TransactionRolledBackException” on page 170</a>	XMS throws this exception if an application calls <code>Session.commit()</code> to commit the current transaction, but the transaction is then rolled back.
XMSC	For .NET, XMS property names and values are defined in this class as public constants. For further details, see <a href="#">“Properties of XMS objects” on page 173</a> .
<a href="#">“XMSException” on page 171</a>	<p>If XMS detects an error while processing a call to a .NET method, XMS throws an exception. An exception is an object that encapsulates information about the error.</p> <p>There are different types of XMS exception, and an <code>XMSException</code> object is just one type of exception. However, the <code>XMSException</code> class is a superclass of the other XMS exception classes. XMS throws an <code>XMSException</code> object in situations where none of the other types of exception are appropriate.</p>

Table 25. Summary of the .NET class interfaces (continued)

Interface	Description
<a href="#">“XMSFactoryFactory” on page 171</a>	If an application is not using administered objects, use this class to create connection factories, queues, and topics.

The definition of each method lists the exception codes that XMS might return if it detects an error while processing a call to the method. Each exception code is represented by its named constant, which has a corresponding exception.

### Related concepts

[Building your own applications](#)

You build your own applications like you build the sample applications.

[Writing XMS applications](#)

This chapter provides information to help you when writing XMS applications.

[Writing XMS .NET applications](#)

This chapter provides information to help you when writing XMS.NET applications.

### Related reference

[Properties of XMS objects](#)

This chapter documents the object properties defined by XMS.

## IBytesMessage

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

### Inheritance hierarchy:

```

IBM.XMS.IPropertyContext
|
+----IBM.XMS.IMessage
      |
      +----IBM.XMS.IBytesMessage
  
```

### Related reference

[Bytes messages](#)

The body of a bytes message contains a stream of bytes. The body contains only the actual data, and it is the responsibility of the sending and receiving applications to interpret this data.

## .NET properties

*BodyLength* - Get Body Length

### Interface:

```

Int64 BodyLength
{
    get;
}
  
```

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

The value returned is the length of the whole body regardless of where the cursor for reading the message is currently positioned.

### Exceptions:

- XMSException
- MessageNotReadableException

## Methods

*ReadBoolean - Read Boolean Value*

### Interface:

```
Boolean ReadBoolean();
```

Read a boolean value from the bytes message stream.

### Parameters:

None

### Returns:

The boolean value that is read.

### Exceptions:

- XMSEException
- MessageNotReadableException
- MessageEOFException

*ReadSignedByte - Read Byte*

### Interface:

```
Int16 ReadSignedByte();
```

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

### Parameters:

None

### Returns:

The byte that is read.

### Exceptions:

- XMSEException
- MessageNotReadableException
- MessageEOFException

*ReadBytes - Read Bytes*

### Interface:

```
Int32 ReadBytes(Byte[] array);  
Int32 ReadBytes(Byte[] array, Int32 length);
```

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

### Parameters:

#### **array (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.

If you specify a null pointer on input, the method skips over the bytes without reading them.

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 number of bytes skipped is equal to the length of the buffer. Otherwise, all the remaining bytes are skipped. The cursor remains at the next position to read in the byte message stream.

**length (input)**

The length of the buffer in bytes

**Returns:**

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`.

If you specify a null pointer on input, the method returns no value.

**Exceptions:**

- `XMSEException`
- `MessageNotReadableException`

*ReadChar - Read Character*

**Interface:**

```
Char ReadChar();
```

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

**Parameters:**

None

**Returns:**

The character that is read.

**Exceptions:**

- `XMSEException`
- `MessageNotReadableException`
- `MessageEOFException`

*ReadDouble - Read Double Precision Floating Point Number*

**Interface:**

```
Double ReadDouble();
```

Read the next 8 bytes from the bytes message stream as a double precision floating point number.

**Parameters:**

None

**Returns:**

The double precision floating point number that is read.

**Exceptions:**

- `XMSEException`
- `MessageNotReadableException`
- `MessageEOFException`

### *ReadFloat - Read Floating Point Number*

#### **Interface:**

```
Single ReadFloat();
```

Read the next 4 bytes from the bytes message stream as a floating point number.

#### **Parameters:**

None

#### **Returns:**

The floating point number that is read.

#### **Exceptions:**

- XMSEException
- MessageNotReadableException
- MessageEOFException

### *ReadInt - Read Integer*

#### **Interface:**

```
Int32 ReadInt();
```

Read the next 4 bytes from the bytes message stream as a signed 32-bit integer.

#### **Parameters:**

None

#### **Returns:**

The integer that is read.

#### **Exceptions:**

- XMSEException
- MessageNotReadableException
- MessageEOFException

### *ReadLong - Read Long Integer*

#### **Interface:**

```
Int64 ReadLong();
```

Read the next 8 bytes from the bytes message stream as a signed 64-bit integer.

#### **Parameters:**

None

#### **Returns:**

The long integer that is read.

#### **Exceptions:**

- XMSEException
- MessageNotReadableException
- MessageEOFException

### *ReadShort - Read Short Integer*

#### **Interface:**

```
Int16 ReadShort();
```

Read the next 2 bytes from the bytes message stream as a signed 16-bit integer.

#### **Parameters:**

None

#### **Returns:**

The short integer that is read.

#### **Exceptions:**

- XMSEException
- MessageNotReadableException
- MessageEOFException

### *ReadByte - Read Unsigned Byte*

#### **Interface:**

```
Byte ReadByte();
```

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

#### **Parameters:**

None

#### **Returns:**

The byte that is read.

#### **Exceptions:**

- XMSEException
- MessageNotReadableException
- MessageEOFException

### *ReadUnsignedShort - Read Unsigned Short Integer*

#### **Interface:**

```
Int32 ReadUnsignedShort();
```

Read the next 2 bytes from the bytes message stream as an unsigned 16-bit integer.

#### **Parameters:**

None

#### **Returns:**

The unsigned short integer that is read.

#### **Exceptions:**

- XMSEException
- MessageNotReadableException
- MessageEOFException

### *ReadUTF - Read UTF String*

#### **Interface:**

```
String ReadUTF();
```

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

**Note:** Before calling ReadUTF(), ensure that the cursor of the buffer is pointing to beginning of the byte message stream.

#### **Parameters:**

None

#### **Returns:**

A String object encapsulating the string that is read.

#### **Exceptions:**

- XMSEException
- MessageNotReadableException
- MessageEOFException

### *Reset - Reset*

#### **Interface:**

```
void Reset();
```

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

#### **Parameters:**

None

#### **Returns:**

Void

#### **Exceptions:**

- XMSEException
- MessageNotReadableException

### *WriteBoolean - Write Boolean Value*

#### **Interface:**

```
void WriteBoolean(Boolean value);
```

Write a boolean value to the bytes message stream.

#### **Parameters:**

##### **value (input)**

The boolean value to be written.

#### **Returns:**

Void

#### **Exceptions:**

- XMSEException
- MessageNotWritableException

### *WriteByte - Write Byte*

#### **Interface:**

```
void WriteByte(Byte value);  
void WriteSignedByte(Int16 value);
```

Write a byte to the bytes message stream.

#### **Parameters:**

**value (input)**

The byte to be written.

#### **Returns:**

Void

#### **Exceptions:**

- XMSEException
- MessageNotWritableException

### *WriteBytes - Write Bytes*

#### **Interface:**

```
void WriteBytes(Byte[] value);
```

Write an array of bytes to the bytes message stream.

#### **Parameters:**

**value (input)**

The array of bytes to be written.

#### **Returns:**

Void

#### **Exceptions:**

- XMSEException
- MessageNotWritableException

### *WriteBytes - Write Partial Bytes Array*

#### **Interface:**

```
void WriteBytes(Byte[] value, int offset, int length);
```

Write a partial array of bytes to the bytes message stream, as defined by the specified length.

#### **Parameters:**

**value (input)**

The array of bytes to be written.

**offset (input)**

The starting point for the array of bytes to be written.

**length (input)**

The number of bytes to write.

**Returns:**

Void

**Exceptions:**

- XMSEException
- MessageNotWritableException

*WriteChar - Write Character***Interface:**

```
void WriteChar(Char value);
```

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

**Parameters:****value (input)**

The character to be written.

**Returns:**

Void

**Exceptions:**

- XMSEException
- MessageNotWritableException

*WriteDouble - Write Double Precision Floating Point Number***Interface:**

```
void WriteDouble(Double 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:****value (input)**

The double precision floating point number to be written.

**Returns:**

Void

**Exceptions:**

- XMSEException
- MessageNotWritableException

*WriteFloat - Write Floating Point Number***Interface:**

```
void WriteFloat(Single 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:****value (input)**

The floating point number to be written.

**Returns:**

Void

**Exceptions:**

- XMSEException
- MessageNotWritableException

*WriteInt - Write Integer***Interface:**

```
void WriteInt(Int32 value);
```

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

**Parameters:****value (input)**

The integer to be written.

**Returns:**

Void

**Exceptions:**

- XMSEException
- MessageNotWritableException

*WriteLong - Write Long Integer***Interface:**

```
void WriteLong(Int64 value);
```

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

**Parameters:****value (input)**

The long integer to be written.

**Returns:**

Void

**Exceptions:**

- XMSEException
- MessageNotWritableException

*WriteObject - Write Object***Interface:**

```
void WriteObject(Object value);
```

Write the specified object into the byte message stream.

**Parameters:****value (input)**

The object to be written, which must be a reference to a primitive type.

**Returns:**

Void

**Exceptions:**

- XMSEException
- MessageNotWritableException

*WriteShort - Write Short Integer*

**Interface:**

```
void WriteShort(Int16 value);
```

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

**Parameters:****value (input)**

The short integer to be written.

**Returns:**

Void

**Exceptions:**

- XMSEException
- MessageNotWritableException

*WriteUTF - Write UTF String*

**Interface:**

```
void WriteUTF(String value);
```

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

**Parameters:****value (input)**

A String object encapsulating the string to be written.

**Returns:**

Void

**Exceptions:**

- XMSEException
- MessageNotWritableException

***Inherited properties and methods***

The following properties are inherited from the [IMessage](#) interface:

[JMSCorrelationID](#), [JMSDeliveryMode](#), [JMSDestination](#), [JMSExpiration](#), [JMSMessageID](#), [JMSPriority](#), [JMSRedelivered](#), [JMSReplyTo](#), [JMSTimestamp](#), [JMSType](#), [Properties](#)

The following methods are inherited from the [IMessage](#) interface:

[clearBody](#), [clearProperties](#), [PropertyExists](#)

The following methods are inherited from the `IPropertyContext` interface:

[GetBooleanProperty](#), [GetByteProperty](#), [GetBytesProperty](#), [GetCharProperty](#), [GetDoubleProperty](#), [GetFloatProperty](#), [GetIntProperty](#), [GetLongProperty](#), [GetObjectProperty](#), [GetShortProperty](#), [GetStringProperty](#), [SetBooleanProperty](#), [SetByteProperty](#), [SetBytesProperty](#), [SetCharProperty](#), [SetDoubleProperty](#), [SetFloatProperty](#), [SetIntProperty](#), [SetLongProperty](#), [SetObjectProperty](#), [SetShortProperty](#), [SetStringProperty](#)

## IConnection

A Connection object represents the active connection of the application to a messaging server.

### Inheritance hierarchy:

```
IBM.XMS.IPropertyContext
|
+----IBM.XMS.IConnection
```

For a list of the XMS defined properties of a Connection object, see [“Properties of Connection”](#) on page 174.

## .NET properties

*ClientID - Get and Set Client ID*

### Interface:

```
String ClientID
{
    get;
    set;
}
```

Get and set the client identifier for the connection.

The client identifier can either be preconfigured by the administrator in a `ConnectionFactory`, or assigned by setting `ClientID`.

A client identifier is used only to support durable subscriptions in the publish/subscribe domain, and is ignored in the point-to-point domain.

If an application sets a client identifier for a connection, the application must do so immediately after creating the connection, and before performing any other operation on the connection. If the application tries to set a client identifier after this point, the call throws exception `IllegalStateException`.

This property is not valid for a real-time connection to a broker.

### Exceptions:

- `XMSEException`
- `IllegalStateException`
- `InvalidClientIDException`

*ExceptionListener - Get and Set Exception Listener*

### Interface:

```
ExceptionListener ExceptionListener
{
    get;
    set;
}
```

Get the exception listener that is registered with the connection, and register an exception listener with the connection.

If no exception listener is registered with the connection, the method returns null. If an exception listener is already registered with the connection, you can cancel the registration by specifying a null instead of the exception listener.

For more information about using exception listeners, see [“Message and exception listeners in .NET” on page 48](#).

**Exceptions:**

- XMSEException

*Metadata - Get Metadata*

**Interface:**

```
IConnectionMetaData MetaData
{
    get;
}
```

Get the metadata for the connection.

**Exceptions:**

- XMSEException

**Methods**

*Close - Close Connection*

**Interface:**

```
void Close();
```

Close the connection.

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

**Parameters:**

None

**Returns:**

Void

**Exceptions:**

- XMSEException

*CreateSession - Create Session*

**Interface:**

```
ISession CreateSession(Boolean transacted,
                        AcknowledgeMode acknowledgeMode);
```

Create a session.

**Parameters:****transacted (input)**

The value `True` means that the session is transacted. The value `False` means that the session is not transacted.

For a real-time connection to a broker, the value must be `False`.

**acknowledgeMode (input)**

Indicates that how messages received by an application are acknowledged. The value must be one of the following from the `AcknowledgeMode` enumerator:

`AcknowledgeMode.AutoAcknowledge`

`AcknowledgeMode.ClientAcknowledge`

`AcknowledgeMode.DupsOkAcknowledge`

For a real-time connection to a broker, the value must be

`AcknowledgeMode.AutoAcknowledge` or `AcknowledgeMode.DupsOkAcknowledge`

This parameter is ignored if the session is transacted. For more information about acknowledgement modes, see [“Message acknowledgement”](#) on page 26.

**Returns:**

The `Session` object

**Exceptions:**

- `XMSEException`

*Start - Start Connection***Interface:**

```
void Start();
```

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

**Parameters:**

None

**Returns:**

Void

**Exceptions:**

- `XMSEException`

*Stop - Stop Connection***Interface:**

```
void Stop();
```

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

**Parameters:**

None

**Returns:**

Void

## Exceptions:

- `XMSEException`

## *Inherited properties and methods*

The following methods are inherited from the `IPropertyContext` interface:

`GetBooleanProperty`, `GetByteProperty`, `GetBytesProperty`, `GetCharProperty`, `GetDoubleProperty`, `GetFloatProperty`, `GetIntProperty`, `GetLongProperty`, `GetObjectProperty`, `GetShortProperty`, `GetStringProperty`, `SetBooleanProperty`, `SetByteProperty`, `SetBytesProperty`, `SetCharProperty`, `SetDoubleProperty`, `SetFloatProperty`, `SetIntProperty`, `SetLongProperty`, `SetObjectProperty`, `SetShortProperty`, `SetStringProperty`

## **IConnectionFactory**

An application uses a connection factory to create a connection.

### **Inheritance hierarchy:**

```
IBM.XMS.IPropertyContext
|
+---- IBM.XMS.IConnectionFactory
```

For a list of the XMS defined properties of a `ConnectionFactory` object, see [“Properties of ConnectionFactory”](#) on page 175.

### **Related concepts**

[ConnectionFactory](#) and [Connection](#) objects

A `ConnectionFactory` object provides a template that an application uses to create a `Connection` object. The application uses the `Connection` object to create a `Session` object.

[Connection to a WebSphere service integration bus](#)

An XMS application can connect to a WebSphere Service Integration Bus either by using a direct TCP/IP connection or by using HTTP over TCP/IP.

[Secure connections to a WebSphere MQ queue manager](#)

To enable an XMS .NET application to make secure connections to a WebSphere MQ queue manager, the relevant properties must be defined in the `ConnectionFactory` object.

[Secure connections to a WebSphere Service Integration Bus messaging engine](#)

To enable an XMS application to make secure connections to a WebSphere Service Integration Bus messaging engine, the relevant properties must be defined in the `ConnectionFactory` object.

[Property mapping for administered objects](#)

To enable applications to use WebSphere MQ JMS and WebSphere Application Server connection factory and destination object definitions, the properties retrieved from these definitions must be mapped on to the corresponding XMS properties that can be set on XMS connection factories and destinations.

### **Related tasks**

[Creating administered objects](#)

The `ConnectionFactory` and `Destination` object definitions that XMS applications require to make a connection to a messaging server must be created using the appropriate administrative tools.

### **Related reference**

[Required properties for administered ConnectionFactory objects](#)

When an application creates a connection factory, a number of properties must be defined to create a connection to a messaging server.

## **Methods**

*CreateConnection - Create Connection Factory (using the default user identity)*

**Interface:**

```
IConnection CreateConnection();
```

Create a connection factory with the default properties.

If you are connecting to WebSphere MQ and XMSC\_USERID is not set, then the queue manager uses the userID of the logged on user by default. If you require further connection-level authentication of individual users you can write a client authentication exit which is configured in WebSphere MQ.

**Parameters:**

None

**Exceptions:**

- XMSEException

*CreateConnection - Create Connection (using a specified user identity)*

**Interface:**

```
IConnection CreateConnection(String userId, String password);
```

Create a connection using a specified user identity.

If you are connecting to WebSphere MQ and XMSC\_USERID is not set, then the queue manager uses the userID of the logged on user by default. If you require further connection-level authentication of individual users you can write a client authentication exit which is configured in WebSphere MQ.

The connection is created in stopped mode. No messages are delivered until the application calls **Connection.start()**.

**Parameters:**

**userID (input)**

A String object encapsulating the user identifier to be used to authenticate the application. If you provide a null, an attempt is made to create the connection without authentication.

**password (input)**

A String object encapsulating the password to be used to authenticate the application. If you provide a null, an attempt is made to create the connection without authentication.

**Returns:**

The Connection object.

**Exceptions:**

- XMSEException
- XMS\_X\_SECURITY\_EXCEPTION

***Inherited properties and methods***

The following methods are inherited from the IPropertyContext interface:

GetBooleanProperty, GetByteProperty, GetBytesProperty, GetCharProperty, GetDoubleProperty, GetFloatProperty, GetIntProperty, GetLongProperty, GetObjectProperty, GetShortProperty, GetStringProperty, SetBooleanProperty, SetByteProperty, SetBytesProperty, SetCharProperty, SetDoubleProperty, SetFloatProperty, SetIntProperty, SetLongProperty, SetObjectProperty, SetShortProperty, SetStringProperty

## IConnectionMetaData

A ConnectionMetaData object provides information about a connection.

### Inheritance hierarchy:

```
IBM.XMS.IPropertyContext
|
+---- IBM.XMS.IConnectionMetaData
```

For a list of the XMS defined properties of a ConnectionMetaData object, see [“Properties of ConnectionMetaData”](#) on page 181.

## .NET properties

*JMSXPropertyNames* - Get JMS Defined Message Properties

### Interface:

```
System.Collections.IEnumerator JMSXPropertyNames
{
    get;
}
```

Return an enumeration of the names of the JMS defined message properties supported by the connection.

JMS defined message properties are not supported by a real-time connection to a broker.

### Exceptions:

- XMSEException

## Inherited properties and methods

The following methods are inherited from the [IPropertyContext](#) interface:

[GetBooleanProperty](#), [GetByteProperty](#), [GetBytesProperty](#), [GetCharProperty](#), [GetDoubleProperty](#), [GetFloatProperty](#), [GetIntProperty](#), [GetLongProperty](#), [GetObjectProperty](#), [GetShortProperty](#), [GetStringProperty](#), [SetBooleanProperty](#), [SetByteProperty](#), [SetBytesProperty](#), [SetCharProperty](#), [SetDoubleProperty](#), [SetFloatProperty](#), [SetIntProperty](#), [SetLongProperty](#), [SetObjectProperty](#), [SetShortProperty](#), [SetStringProperty](#)

## IDestination

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

### Inheritance hierarchy:

```
IBM.XMS.IPropertyContext
|
+---- IBM.XMS.IDestination
```

For a list of the XMS defined properties of a Destination object, see [“Properties of Destination”](#) on page 181.

### Related concepts

[ConnectionFactory](#) and [Connection](#) objects

A [ConnectionFactory](#) object provides a template that an application uses to create a [Connection](#) object. The application uses the [Connection](#) object to create a [Session](#) object.

[Connection to a WebSphere service integration bus](#)

An XMS application can connect to a WebSphere Service Integration Bus either by using a direct TCP/IP connection or by using HTTP over TCP/IP.

#### Destinations

An XMS application uses a Destination object to specify the destination of messages that are being sent, and the source of messages that are being received.

#### Destination wildcards

XMS provides support for destination wildcards, ensuring that wildcards can be passed through to the place where they are needed for matching. There is a different wildcard scheme for each server type that XMS can work with.

#### Topic uniform resource identifiers

The topic uniform resource identifier (URI) specifies the name of the topic; it can also specify one or more properties for it.

#### Queue uniform resource identifiers

The URI for a queue specifies the name of the queue; it can also specify one or more properties of the queue.

#### Temporary destinations

XMS applications can create and use temporary destinations.

#### Property mapping for administered objects

To enable applications to use WebSphere MQ JMS and WebSphere Application Server connection factory and destination object definitions, the properties retrieved from these definitions must be mapped on to the corresponding XMS properties that can be set on XMS connection factories and destinations.

### **Related tasks**

#### Creating administered objects

The ConnectionFactory and Destination object definitions that XMS applications require to make a connection to a messaging server must be created using the appropriate administrative tools.

### **Related reference**

#### Required properties for administered Destination objects

An application that is creating a destination must set several properties that the application on an administered Destination object.

## ***.NET properties***

*Name - Get Destination Name*

#### **Interface:**

```
String Name
{
    get;
}
```

Get the name of the destination. The name is a string encapsulating either the name of a queue or the name of a topic.

#### **Exceptions:**

- XMSException

*TypeId - Get Destination Type*

#### **Interface:**

```
DestinationType TypeId
{
    get;
}
```

Get the type of the destination. The type of the destination is one of the following values:

`DestinationType.Queue`  
`DestinationType.Topic`

**Exceptions:**

- `XMSEException`

### ***Inherited properties and methods***

The following methods are inherited from the `IPropertyContext` interface:

[GetBooleanProperty](#), [GetByteProperty](#), [GetBytesProperty](#), [GetCharProperty](#), [GetDoubleProperty](#), [GetFloatProperty](#), [GetIntProperty](#), [GetLongProperty](#), [GetObjectProperty](#), [GetShortProperty](#), [GetStringProperty](#), [SetBooleanProperty](#), [SetByteProperty](#), [SetBytesProperty](#), [SetCharProperty](#), [SetDoubleProperty](#), [SetFloatProperty](#), [SetIntProperty](#), [SetLongProperty](#), [SetObjectProperty](#), [SetShortProperty](#), [SetStringProperty](#)

## **ExceptionListener**

**Inheritance hierarchy:**

None

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. In other situations, an exception listener can provide a more immediate way of learning about a problem with a connection than waiting until the next synchronous call to XMS.

### ***Delegate***

*ExceptionListener - Exception Listener*

**Interface:**

```
public delegate void ExceptionListener(Exception ex)
```

Notify the application of a problem with a connection.

Methods that implement this delegate can be registered with the connection.

For more information about using exception listeners, see [“Message and exception listeners in .NET” on page 48](#).

**Parameters:**

**exception (input)**

A pointer to an exception created by XMS.

**Returns:**

Void

## **IllegalStateException**

**Inheritance hierarchy:**

```
IBM.XMS.XMSEException
|
+----IBM.XMS.Exception
|
+----IBM.XMS.IllegalStateException
```

XMS throws this exception if an application calls a method at an incorrect or inappropriate time, or if XMS is not in an appropriate state for the requested operation.

### ***Inherited properties and methods***

The following methods are inherited from the [XMSEException](#) interface:

[GetErrorCode](#), [GetLinkedException](#)

### **InitialContext**

An application uses an InitialContext object to create objects from object definitions that are retrieved from a repository of administered objects.

#### **Inheritance hierarchy:**

None

#### **Related concepts**

##### InitialContext properties

The parameters of the InitialContext constructor include the location of the repository of administered objects, given as a uniform resource indicator (URI). In order for an application to establish a connection to the repository, it may be necessary to provide more information than the information contained in the URI.

##### URI format for XMS initial contexts

The location of the repository of administered objects is provided as a uniform resource indicator (URI). The format of the URI depends on the context type.

##### Retrieval of administered objects

XMS retrieves an administered object from the repository using the address provided when the InitialContext object is created, or in the InitialContext properties.

#### **Related tasks**

##### InitialContext objects

An application must create an initial context to be used to make a connection to the administered objects repository to retrieve the required administered objects.

### ***.NET properties***

*Environment - Get the environment*

#### **Interface:**

```
Hashtable Environment
{
    get;
}
```

Get the environment.

#### **Exceptions:**

- Exceptions are specific to the directory service being used.

### ***Constructors***

*InitialContext - Create Initial Context*

#### **Interface:**

```
InitialContext(Hashtable env);
```

Create an InitialContext object.

**Parameters:**

The information required to establish a connection to the repository of administered objects is provided to the constructor in an environment Hashtable.

**Exceptions:**

- XMSEException

**Methods**

*AddToEnvironment - Add a New Property to the Environment*

**Interface:**

```
Object AddToEnvironment(String propName, Object propVal);
```

Add a new property to the environment.

**Parameters:****propName (input)**

A String object encapsulating the name of the property to be added.

**propVal (input)**

The value of the property to be added.

**Returns:**

The old value of the property.

**Exceptions:**

- Exceptions are specific to the directory service being used.

*Close - Close this context*

**Interface:**

```
void Close();
```

Close this context.

**Parameters:**

None

**Returns:**

None

**Exceptions:**

- Exceptions are specific to the directory service being used.

*Lookup - Look Up Object in Initial Context*

**Interface:**

```
Object Lookup(String name);
```

Create an object from an object definition that is retrieved from the repository of administered objects.

**Parameters:****name (input)**

A String object encapsulating the name of the administered object to be retrieved. The name can be either a simple name or a complex name. For further details, see [“Retrieval of administered objects”](#) on page 63.

**Returns:**

Either an IConnectionFactory or an IDestination, depending on the type of object being retrieved. If the function can access the directory, but cannot find the required object, a null is returned.

**Exceptions:**

- Exceptions are specific to the directory service being used.

*RemoveFromEnvironment - Remove a Property from the Environment*

**Interface:**

```
Object RemoveFromEnvironment(String propName);
```

Remove a property from the environment.

**Parameters:****propName (input)**

A String object encapsulating the name of the property to be removed.

**Returns:**

The object that was removed.

**Exceptions:**

- Exceptions are specific to the directory service being used.

**InvalidClientIDException****Inheritance hierarchy:**

```
IBM.XMS.XMSEException
|
+----IBM.XMS.XMSEException
      |
      +----IBM.XMS.InvalidClientIDException
```

XMS throws this exception if an application attempts to set a client identifier for a connection, but the client identifier is not valid or is already in use.

***Inherited properties and methods***

The following methods are inherited from the [XMSEException](#) interface:

[GetErrorCode](#), [GetLinkedException](#)

**InvalidDestinationException****Inheritance hierarchy:**

```
IBM.XMS.XMSEException
|
+----IBM.XMS.XMSEException
      |
      +----IBM.XMS.InvalidDestinationException
```

XMS throws this exception if an application specifies a destination that is not valid.

***Inherited properties and methods***

The following methods are inherited from the [XMSEException](#) interface:

[GetErrorCode](#), [GetLinkedException](#)

## InvalidSelectorException

### Inheritance hierarchy:

```
IBM.XMS.XMSException
|
+----IBM.XMS.XMSException
      |
      +----IBM.XMS.InvalidSelectorException
```

XMS throws this exception if an application provides a message selector expression whose syntax is not valid.

### Inherited properties and methods

The following methods are inherited from the [XMSException](#) interface:

[GetErrorCode](#), [GetLinkedException](#)

## IMapMessage

A map message is a message whose body comprises a set of name-value pairs, where each value has an associated data type.

### Inheritance hierarchy:

```
IBM.XMS.IPropertyContext
|
+----IBM.XMS.IMessage
      |
      +----IBM.XMS.IMapMessage
```

When an application gets the value of name-value pair, the value can be converted by XMS into another data type. For more information about this form of implicit conversion, see [“Map messages” on page 76](#).

### Related reference

#### Map messages

The body of a map message contains a set of name-value pairs, where each value has an associated data type.

## .NET properties

*MapNames* - Get Map Names

### Interface:

```
System.Collections.IEnumerator MapNames
{
    get;
}
```

Get an enumeration of the names in the body of the map message.

### Exceptions:

- [XMSException](#)

## Methods

*GetBoolean* - Get Boolean Value

### Interface:

```
Boolean GetBoolean(String name);
```

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

**Parameters:**

**name (input)**

A String object encapsulating the name that identifies the boolean value.

**Returns:**

The boolean value retrieved from the body of the map message.

**Exceptions:**

- XMSEException

*GetByte - Get Byte*

**Interface:**

```
Byte    GetByte(String name);  
Int16   GetSignedByte(String name);
```

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

**Parameters:**

**name (input)**

A String object encapsulating the name that identifies the byte.

**Returns:**

The byte retrieved from the body of the map message. No data conversion is performed on the byte.

**Exceptions:**

- XMSEException

*GetBytes - Get Bytes*

**Interface:**

```
Byte[]  GetBytes(String name);
```

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

**Parameters:**

**name (input)**

A String object encapsulating the name that identifies the array of bytes.

**Returns:**

The number of bytes in the array.

**Exceptions:**

- XMSEException

*GetChar - Get Character*

**Interface:**

```
Char    GetChar(String name);
```

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

**Parameters:****name (input)**

A String object encapsulating the name that identifies the character.

**Returns:**

The character retrieved from the body of the map message.

**Exceptions:**

- XMSEException

*GetDouble - Get Double Precision Floating Point Number***Interface:**

```
Double GetDouble(String name);
```

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

**Parameters:****name (input)**

A String object encapsulating the name that identifies the double precision floating point number.

**Returns:**

The double precision floating point number retrieved from the body of the map message.

**Exceptions:**

- XMSEException

*GetFloat - Get Floating Point Number***Interface:**

```
Single GetFloat(String name);
```

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

**Parameters:****name (input)**

A String object encapsulating the name that identifies the floating point number.

**Returns:**

The floating point number retrieved from the body of the map message.

**Exceptions:**

- XMSEException

*GetInt - Get Integer***Interface:**

```
Int32 GetInt(String name);
```

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

**Parameters:****name (input)**

A String object encapsulating the name that identifies the integer.

**Returns:**

The integer retrieved from the body of the map message.

**Exceptions:**

- XMSEException

*GetLong - Get Long Integer***Interface:**

```
Int64 GetLong(String name);
```

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

**Parameters:****name (input)**

A String object encapsulating the name that identifies the long integer.

**Returns:**

The long integer retrieved from the body of the map message.

**Exceptions:**

- XMSEException

*GetObject - Get Object***Interface:**

```
Object GetObject(String name);
```

Get a reference to the value of a name-value pair, from the body of the map message. The name-value pair is identified by name.

**Parameters:****name (input)**

A String object encapsulating the name of the name-value pair.

**Returns:**

The value, which is one of the following object types:

- Boolean
- Byte
- Byte[]
- Char
- Double
- Single
- Int32
- Int64
- Int16
- String

**Exceptions:**

XMSEException

### *GetShort - Get Short Integer*

#### **Interface:**

```
Int16 GetShort(String name);
```

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

#### **Parameters:**

##### **name (input)**

A String object encapsulating the name that identifies the short integer.

#### **Returns:**

The short integer retrieved from the body of the map message.

#### **Exceptions:**

- XMSEException

### *GetString - Get String*

#### **Interface:**

```
String GetString(String name);
```

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

#### **Parameters:**

##### **name (input)**

A String object encapsulating the name that identifies the string in the body of the map message.

#### **Returns:**

A String object encapsulating the string retrieved from the body of the map message. If data conversion is required, this value is the string after conversion.

#### **Exceptions:**

- XMSEException

### *ItemExists - Check Name-Value Pair Exists*

#### **Interface:**

```
Boolean ItemExists(String name);
```

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

#### **Parameters:**

##### **name (input)**

A String object encapsulating the name of the name-value pair.

#### **Returns:**

- True, if the body of the map message contains a name-value pair with the specified name.
- False, if the body of the map message does not contain a name-value pair with the specified name.

#### **Exceptions:**

- XMSEException

## *SetBoolean - Set Boolean Value*

### **Interface:**

```
void SetBoolean(String name, Boolean value);
```

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

### **Parameters:**

#### **name (input)**

A String object encapsulating the name to identify the boolean value in the body of the map message.

#### **value (input)**

The boolean value to be set.

### **Returns:**

Void

### **Exceptions:**

- XMSEException

## *SetByte - Set Byte*

### **Interface:**

```
void SetByte(String name, Byte value);  
void SetSignedByte(String name, Int16 value);
```

Set a byte in the body of the map message.

### **Parameters:**

#### **name (input)**

A String object encapsulating the name to identify the byte in the body of the map message.

#### **value (input)**

The byte to be set.

### **Returns:**

Void

### **Exceptions:**

- XMSEException

## *SetBytes - Set Bytes*

### **Interface:**

```
void SetBytes(String name, Byte[] value);
```

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

### **Parameters:**

#### **name (input)**

A String object encapsulating the name to identify the array of bytes in the body of the map message.

#### **value (input)**

The array of bytes to be set.

### **Returns:**

Void

**Exceptions:**

- XMSEException

*SetChar - Set Character***Interface:**

```
void SetChar(String name, Char value);
```

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

**Parameters:****name (input)**

A String object encapsulating the name to identify the character in the body of the map message.

**value (input)**

The character to be set.

**Returns:**

Void

**Exceptions:**

- XMSEException

*SetDouble - Set Double Precision Floating Point Number***Interface:**

```
void SetDouble(String name, Double value);
```

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

**Parameters:****name (input)**

A String object encapsulating the name to identify the double precision floating point number in the body of the map message.

**value (input)**

The double precision floating point number to be set.

**Returns:**

Void

**Exceptions:**

- XMSEException

*SetFloat - Set Floating Point Number***Interface:**

```
void SetFloat(String name, Single value);
```

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

**Parameters:****name (input)**

A String object encapsulating the name to identify the floating point number in the body of the map message.

**value (input)**

The floating point number to be set.

**Returns:**

Void

**Exceptions:**

- XMSEException

*SetInt - Set Integer***Interface:**

```
void SetInt(String name, Int32 value);
```

Set an integer in the body of the map message.

**Parameters:****name (input)**

A String object encapsulating the name to identify the integer in the body of the map message.

**value (input)**

The integer to be set.

**Returns:**

Void

**Exceptions:**

- XMSEException

*SetLong - Set Long Integer***Interface:**

```
void SetLong(String name, Int64 value);
```

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

**Parameters:****name (input)**

A String object encapsulating the name to identify the long integer in the body of the map message.

**value (input)**

The long integer to be set.

**Returns:**

Void

**Exceptions:**

- XMSEException

*SetObject - Set Object***Interface:**

```
void SetObject(String name, Object value);
```

Set a value, which must be an XMS primitive type, in the body of the map message.

**Parameters:****name (input)**

A String object encapsulating the name to identify the value in the body of the map message.

**value (input)**

An array of bytes containing the value to be set.

**Returns:**

Void

**Exceptions:**

- XMSEException

*SetShort - Set Short Integer***Interface:**

```
void SetShort(String name, Int16 value);
```

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

**Parameters:****name (input)**

A String object encapsulating the name to identify the short integer in the body of the map message.

**value (input)**

The short integer to be set.

**Returns:**

Void

**Exceptions:**

- XMSEException

*SetString - Set String***Interface:**

```
void SetString(String name, String value);
```

Set a string in the body of the map message.

**Parameters:****name (input)**

A String object encapsulating the name to identify the string in the body of the map message.

**value (input)**

A String object encapsulating the string to be set.

**Returns:**

Void

**Exceptions:**

- XMSEException

***Inherited properties and methods***

The following properties are inherited from the [IMessage](#) interface:

[JMSCorrelationID](#), [JMSDeliveryMode](#), [JMSDestination](#), [JMSExpiration](#), [JMSMessageID](#), [JMSPriority](#), [JMSRedelivered](#), [JMSReplyTo](#), [JMSTimestamp](#), [JMSType](#), [Properties](#)

The following methods are inherited from the [IMessage](#) interface:

[clearBody](#), [clearProperties](#), [PropertyExists](#)

The following methods are inherited from the [IPropertyContext](#) interface:

[GetBooleanProperty](#), [GetByteProperty](#), [GetBytesProperty](#), [GetCharProperty](#), [GetDoubleProperty](#), [GetFloatProperty](#), [GetIntProperty](#), [GetLongProperty](#), [GetObjectProperty](#), [GetShortProperty](#), [GetStringProperty](#), [SetBooleanProperty](#), [SetByteProperty](#), [SetBytesProperty](#), [SetCharProperty](#), [SetDoubleProperty](#), [SetFloatProperty](#), [SetIntProperty](#), [SetLongProperty](#), [SetObjectProperty](#), [SetShortProperty](#), [SetStringProperty](#)

## IMessage

A Message object represents a message that an application sends or receives. IMessage is a superclass for the message classes such as IMessage.

### Inheritance hierarchy:

```
IBM.XMS.IPropertyContext
|
+---- IBM.XMS.IMessage
```

For a list of the JMS message header fields in a Message object, see “[Header fields in an XMS message](#)” on page 69. For a list of the JMS defined properties of a Message object, see “[JMS-defined properties of a message](#)” on page 71. For a list of the IBM defined properties of a Message object, see “[IBM-defined properties of a message](#)” on page 72. For a list of JMS\_IBM\_MQMD\* properties for the Message object, see “[JMS\\_IBM\\_MQMD\\* properties](#)” on page 186

Messages are deleted by the garbage collector. When a message is deleted, this frees the resources it was using.

### Related reference

[XMS messages](#)

This chapter describes the structure and content of XMS messages and explains how applications process XMS messages.

## .NET properties

*GetJMSCorrelationID - Get and Set JMSCorrelationID*

### Interface:

```
String JMSCorrelationID
{
    get;
    set;
}
```

Get and set the correlation identifier of the message as a String object.

### Exceptions:

- XMSException

*JMSDeliveryMode - Get and Set JMSDeliveryMode*

### Interface:

```
DeliveryMode JMSDeliveryMode
{
    get;
```

```
    set;
}
```

Get and set the delivery mode of the message.

The delivery mode of the message is one of the following values:

```
    DeliveryMode.Persistent
    DeliveryMode.NonPersistent
```

For a newly created message that was not sent, the delivery mode is `DeliveryMode.Persistent`, except for a real-time connection to a broker for which the delivery mode is `DeliveryMode.NonPersistent`. For a message that is received, the method returns the delivery mode that was set by the `IMessageProducer.send()` call when the message was sent unless the receiving application changes the delivery mode by setting `JMSDeliveryMode`.

**Exceptions:**

- `XMSEException`

*JMSDestination - Get and Set JMSDestination*

**Interface:**

```
IDestination JMSDestination
{
    get;
    set;
}
```

Get and set the destination of the message.

The destination is set by the `IMessageProducer.send()` call when the message is sent. The value of `JMSDestination` is ignored. However, you can use `JMSDestination` to change the destination of a message that was received.

For a newly created message that was not sent, the method returns a null `Destination` object, unless the sending application sets a destination by setting `JMSDestination`. For a message that was received, the method returns a `Destination` object for the destination that was set by the `IMessageProducer.send()` call when the message was sent unless the receiving application changes the destination by setting `JMSDestination`.

**Exceptions:**

- `XMSEException`

*JMSExpiration - Get and Set JMSExpiration*

**Interface:**

```
Int64 JMSExpiration
{
    get;
    set;
}
```

Get and set the expiration time of the message.

The expiration time is set by the `IMessageProducer.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.

For a newly created message that was not sent, the expiration time is 0 unless the sending application sets a different expiration time by setting `JMSExpiration`. For a message that was received, the method

returns the expiration time that was set by the `IMessageProducer.send()` call when the message was sent unless the receiving application changes the expiration time by setting `JMSExpiration`.

If the time to live is 0, the `IMessageProducer.send()` call sets the expiration time to 0 to indicate that the message does not expire.

XMS discards expired messages and does not deliver them to applications.

**Exceptions:**

- `XMSException`

*JMSMessageID - Get and Set JMSMessageID*

**Interface:**

```
String JMSMessageID
{
    get;
    set;
}
```

Get and set the message identifier of the message as a string object encapsulating the message identifier.

The message identifier is set by the `IMessageProducer.send()` call when the message is sent. For a message that was received, the method returns the message identifier that was set by the `IMessageProducer.send()` call when the message was sent unless the receiving application changes the message identifier by setting `JMSMessageID`.

If the message has no message identifier, the method returns a null.

**Exceptions:**

- `XMSException`

*JMSPriority - Get and Set JMSPriority*

**Interface:**

```
Int32 JMSPriority
{
    get;
    set;
}
```

Get and set the priority of the message.

The priority is set by the `IMessageProducer.send()` call when the message is sent. The value is an integer in the range 0, the lowest priority, to 9, the highest priority.

For a newly created message that was not sent, the priority is 4 unless the sending application sets a different priority by setting `JMSPriority`. For a message that was received, the method returns the priority that was set by the `IMessageProducer.send()` call when the message was sent unless the receiving application changes the priority by setting `JMSPriority`.

**Exceptions:**

- `XMSException`

## *JMSRedelivered - Get and Set JMSRedelivered*

### **Interface:**

```
Boolean JMSRedelivered
{
    get;
    set;
}
```

Get an indication of whether the message is being redelivered, and indicate whether the message is being redelivered. The indication is set by the `IMessageConsumer.receive()` call when the message is received.

This property has the following values:

- `True`, if the message is being redelivered.
- `False`, if the message is not being redelivered.

For a real-time connection to a broker, the value is always `False`.

An indication of redelivery set by `JMSRedelivered` before the message is sent is ignored by the `IMessageProducer.send()` call when the message is sent, and is ignored and replaced by the `IMessageConsumer.receive()` call when the message is received. However, you can use `JMSRedelivered` to change the indication for a message that was received.

### **Exceptions:**

- `XMSEException`

## *JMSReplyTo - Get and Set JMSReplyTo*

### **Interface:**

```
IDestination JMSReplyTo
{
    get;
    set;
}
```

Get and set the destination where a reply to the message is to be sent.

The value of this property is a `Destination` object for the destination where a reply to the message is to be sent. A null `Destination` object means that no reply is expected.

### **Exceptions:**

- `XMSEException`

## *JMSTimestamp - Get and Set JMSTimestamp*

### **Interface:**

```
Int64 JMSTimestamp
{
    get;
    set;
}
```

Get and set the time when the message was sent.

The time stamp is set by the `IMessageProducer.send()` call when the message is sent and is expressed in milliseconds since 00:00:00 GMT on the 1 January 1970.

For a newly created message that was not sent, the time stamp is 0 unless the sending application sets a different time stamp by setting `JMSTimestamp`. For a message that was received, the method returns

the time stamp that was set by the `IMessageProducer.send()` call when the message was sent unless the receiving application changes the time stamp by setting `JMSTimestamp`.

**Exceptions:**

- `XMSEException`

**Notes:**

1. If the time stamp is undefined, the method returns 0 but throws no exception.

*JMSType - Get and Set JMSType*

**Interface:**

```
String JMSType
{
    get;
    set;
}
```

Get and set the type of the message.

The value of `JMSType` is a string encapsulating the type of the message. If data conversion is required, this value is the type after conversion.

**Exceptions:**

- `XMSEException`

*PropertyNames - Get Properties*

**Interface:**

```
System.Collections.IEnumerator PropertyNames
{
    get;
}
```

Get an enumeration of the names properties of the message.

**Exceptions:**

- `XMSEException`

**Methods**

*Acknowledge - Acknowledge*

**Interface:**

```
void Acknowledge();
```

Acknowledge this message and all previously unacknowledged messages received by the session.

An application can call this method if the acknowledgement mode of the session is `AcknowledgeMode.ClientAcknowledge`. Calls to the method are ignored if the session has any other acknowledgement mode or is transacted.

Messages that were received but not acknowledged might be redelivered.

For more information about acknowledging messages, see [“Message acknowledgement” on page 26](#).

**Parameters:**

None

**Returns:**

Void

**Exceptions:**

- XMSEException
- IllegalStateException

*ClearBody - Clear Body***Interface:**

```
void ClearBody();
```

Clear the body of the message. The header fields and message properties are not cleared.

If an application clears a message body, the body remains in the same state as an empty body in a newly created message. The state of an empty body in a newly created message depends on the type of message body. For more information, see [“The body of an XMS message” on page 74](#).

An application can clear a message body at any time, no matter what state the body is in. If a message body is read-only, the only way that an application can write to the body is for the application to clear the body first.

**Parameters:**

None

**Returns:**

Void

**Exceptions:**

- XMSEException

*ClearProperties - Clear Properties***Interface:**

```
void ClearProperties();
```

Clear the properties of the message. The header fields and the message body are not cleared.

If an application clears the properties of a message, the properties become readable and writable.

An application can clear the properties of a message at any time, no matter what state the properties are in. If the properties of a message are read-only, the only way that the properties can become writable is for the application to clear the properties first.

**Parameters:**

None

**Returns:**

Void

**Exceptions:**

- XMSEException

## *PropertyExists - Check Property Exists*

### **Interface:**

```
Boolean PropertyExists(String propertyName);
```

Check whether the message has a property with the specified name.

### **Parameters:**

#### **propertyName (input)**

A String object encapsulating the name of the property.

### **Returns:**

- True, if the message has a property with the specified name.
- False, if the message does not have a property with the specified name.

### **Exceptions:**

- XMSEException

## ***Inherited properties and methods***

The following methods are inherited from the [IPropertyContext](#) interface:

[GetBooleanProperty](#), [GetByteProperty](#), [GetBytesProperty](#), [GetCharProperty](#), [GetDoubleProperty](#), [GetFloatProperty](#), [GetIntProperty](#), [GetLongProperty](#), [GetObjectProperty](#), [GetShortProperty](#), [GetStringProperty](#), [SetBooleanProperty](#), [SetByteProperty](#), [SetBytesProperty](#), [SetCharProperty](#), [SetDoubleProperty](#), [SetFloatProperty](#), [SetIntProperty](#), [SetLongProperty](#), [SetObjectProperty](#), [SetShortProperty](#), [SetStringProperty](#)

## **IMessageConsumer**

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

### **Inheritance hierarchy:**

```
IBM.XMS.IPropertyContext
|
+----IBM.XMS.IMessageConsumer
```

For a list of the XMS defined properties of a MessageConsumer object, see [“Properties of MessageConsumer”](#) on page 190.

## ***.NET properties***

### *MessageListener - Get and Set Message Listener*

### **Interface:**

```
MessageListener MessageListener
{
    get;
    set;
}
```

Get the message listener that is registered with the message consumer, and register a message listener with the message consumer.

If no message listener is registered with the message consumer, MessageListener is null. If a message listener is already registered with the message consumer, you can cancel the registration by specifying a null instead.

For more information about using message listeners, see [“Message and exception listeners in .NET”](#) on page 48.

**Exceptions:**

- XMSEException

*MessageSelector - Get Message Selector*

**Interface:**

```
String MessageSelector
{
    get;
}
```

Get the message selector for the message consumer. The return value is a String object encapsulating the message selector expression. If data conversion is required, this value is the message selector expression after conversion. If the message consumer does not have a message selector, the value of MessageSelector is a null String object.

**Exceptions:**

- XMSEException

**Methods**

*Close - Close Message Consumer*

**Interface:**

```
void Close();
```

Close the message consumer.

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

**Parameters:**

None

**Returns:**

Void

**Exceptions:**

- XMSEException

*Receive - Receive*

**Interface:**

```
IMessage Receive();
```

Receive the next message for the message consumer. The call waits indefinitely for a message, or until the message consumer is closed.

**Parameters:**

None

**Returns:**

A pointer to the Message object. If the message consumer is closed while the call is waiting for a message, the method returns a pointer to a null Message object.

**Exceptions:**

- XMSEException

*Receive - Receive (with a wait interval)*

**Interface:**

```
IMessage Receive(Int64 delay);
```

Receive the next message for the message consumer. The call waits only a specified period for a message, or until the message consumer is closed.

**Parameters:****delay (input)**

The time, in milliseconds, that the call waits for a message. If you specify a wait interval of 0, the call waits indefinitely for a message.

**Returns:**

A pointer to the Message object. If no message arrives during the wait interval, or if the message consumer is closed while the call is waiting for a message, the method returns a pointer to a null Message object but throws no exception.

**Exceptions:**

- XMSEException

*ReceiveNoWait - Receive with No Wait*

**Interface:**

```
IMessage ReceiveNoWait();
```

Receive the next message for the message consumer if one is available immediately.

**Parameters:**

None

**Returns:**

A pointer to a Message object. If no message is available immediately, the method returns a pointer to a null Message object.

**Exceptions:**

- XMSEException

***Inherited properties and methods***

The following methods are inherited from the [IPropertyContext](#) interface:

[GetBooleanProperty](#), [GetByteProperty](#), [GetBytesProperty](#), [GetCharProperty](#), [GetDoubleProperty](#), [GetFloatProperty](#), [GetIntProperty](#), [GetLongProperty](#), [GetObjectProperty](#), [GetShortProperty](#), [GetStringProperty](#), [SetBooleanProperty](#), [SetByteProperty](#), [SetBytesProperty](#), [SetCharProperty](#), [SetDoubleProperty](#), [SetFloatProperty](#), [SetIntProperty](#), [SetLongProperty](#), [SetObjectProperty](#), [SetShortProperty](#), [SetStringProperty](#)

## MessageEOFException

### Inheritance hierarchy:

```
IBM.XMS.XMSEException
|
+----IBM.XMS.XMSEException
|
+----IBM.XMS.MessageEOFException
```

XMS throws this exception if XMS encounters the end of a bytes message stream when an application is reading the body of a bytes message.

### ***Inherited properties and methods***

The following methods are inherited from the [XMSEException](#) interface:

[GetErrorCode](#), [GetLinkedException](#)

## MessageFormatException

### Inheritance hierarchy:

```
IBM.XMS.XMSEException
|
+----IBM.XMS.XMSEException
|
+----IBM.XMS.MessageFormatException
```

XMS throws this exception if XMS encounters a message with a format that is not valid.

### ***Inherited properties and methods***

The following methods are inherited from the [XMSEException](#) interface:

[GetErrorCode](#), [GetLinkedException](#)

## IMessageListener (delegate)

### Inheritance hierarchy:

None

An application uses a message listener to receive messages asynchronously.

### ***Delegate***

*MessageListener - Message Listener*

### Interface:

```
public delegate void MessageListener(IMessage msg);
```

Deliver a message asynchronously to the message consumer.

Methods that implement this delegate can be registered with the connection.

For more information about using message listeners, see [“Message and exception listeners in .NET” on page 48](#).

### Parameters:

#### **mesg (input)**

The Message object.

### Returns:

Void

## MessageNotReadableException

### Inheritance hierarchy:

```
IBM.XMS.XMSEException
|
+----IBM.XMS.XMSEException
|
+----IBM.XMS.MessageNotReadableException
```

XMS throws this exception if an application attempts to read the body of a message that is write only.

### *Inherited properties and methods*

The following methods are inherited from the [XMSEException](#) interface:

[GetErrorCode](#), [GetLinkedException](#)

## MessageNotWritableException

### Inheritance hierarchy:

```
IBM.XMS.XMSEException
|
+----IBM.XMS.XMSEException
|
+----IBM.XMS.MessageNotWritableException
```

XMS throws this exception if an application attempts to write to the body of a message that is read-only.

### *Inherited properties and methods*

The following methods are inherited from the [XMSEException](#) interface:

[GetErrorCode](#), [GetLinkedException](#)

## IMessageProducer

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

### Inheritance hierarchy:

```
IBM.XMS.IPropertyContext
|
+----IBM.XMS.IMessageProducer
```

For a list of the XMS defined properties of a MessageProducer object, see [“Properties of MessageProducer”](#) on page 190.

### *.NET properties*

*DeliveryMode - Get and Set Default Delivery Mode*

#### Interface:

```
DeliveryMode DeliveryMode
{
    get;
    set;
}
```

Get and set the default delivery mode for messages sent by the message producer.

The default delivery mode has one of the following values:

`DeliveryMode.Persistent`  
`DeliveryMode.NonPersistent`

For a real-time connection to a broker, the value must be `DeliveryMode.NonPersistent`.

The default value is `DeliveryMode.Persistent`, except for a real-time connection to a broker for which the default value is `DeliveryMode.NonPersistent`.

**Exceptions:**

- `XMSEException`

*Destination - Get Destination*

**Interface:**

```
IDestination Destination
{
    get;
}
```

Get the destination for the message producer.

**Parameters:**

None

**Returns:**

The `Destination` object. If the message producer does not have a destination, the method returns a null `Destination` object.

**Exceptions:**

- `XMSEException`

*DisableMsgID - Get and Set Disable Message ID Flag*

**Interface:**

```
Boolean DisableMessageID
{
    get;
    set;
}
```

Get an indication of whether a receiving application requires message identifiers to be included in messages sent by the message producer, and indicate whether a receiving application requires message identifiers to be included in messages sent by the message producer.

On a connection to a queue manager, or on a real-time connection to a broker, this flag is ignored. On a connection to a service integration bus, the flag is honored.

`DisabledMsgID` has the following values:

- `True`, if a receiving application does not require message identifiers to be included in messages sent by the message producer.
- `False`, if a receiving application does require message identifiers to be included in messages sent by the message producer.

**Exceptions:**

- `XMSEException`

## *DisableMsgTS - Get and Set Disable Time Stamp Flag*

### **Interface:**

```
Boolean DisableMessageTimestamp
{
    get;
    set;
}
```

Get an indication of whether a receiving application requires time stamps to be included in messages sent by the message producer, and indicate whether a receiving application requires time stamps to be included in messages sent by the message producer.

On a real-time connection to a broker, this flag is ignored. On a connection to a queue manager, or on a connection to a service integration bus, the flag is honored.

DisableMsgTS has the following values:

- `True`, if a receiving application does not require time stamps to be included in messages sent by the message producer.
- `False`, if a receiving application does require time stamps to be included in messages sent by the message producer.

### **Returns:**

### **Exceptions:**

- `XMSEException`

## *Priority - Get and Set Default Priority*

### **Interface:**

```
Int32 Priority
{
    get;
    set;
}
```

Get and set the default priority for messages sent by the message producer.

The value of the default message priority is an integer in the range 0, the lowest priority, to 9, the highest priority.

On a real-time connection to a broker, the priority of a message is ignored.

### **Exceptions:**

- `XMSEException`

## *TimeToLive - Get and Set Default Time to Live*

### **Interface:**

```
Int64 TimeToLive
{
    get;
    set;
}
```

Get and 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 and is the default time to live in milliseconds. A value of 0 means that a message never expires.

For a real-time connection to a broker, this value is always 0.

**Exceptions:**

- XMSEException

**Methods**

*Close - Close Message Producer*

**Interface:**

```
void Close();
```

Close the message producer.

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

**Parameters:**

None

**Returns:**

Void

**Exceptions:**

- XMSEException

*Send - Send*

**Interface:**

```
void Send(IMessage msg) ;
```

Send a message to the destination that was specified when the message producer was created. Send the message using the message producer default delivery mode, priority, and time to live.

**Parameters:****msg (input)**

The Message object.

**Returns:**

Void

**Exceptions:**

- XMSEException
- MessageFormatException
- InvalidDestinationException

*Send - Send (specifying a delivery mode, priority, and time to live)*

**Interface:**

```
void Send(IMessage msg,  
          DeliveryMode deliveryMode,  
          Int32 priority,  
          Int64 timeToLive);
```

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:****msg (input)**

The Message object.

**deliveryMode (input)**

The delivery mode for the message, which must be one of the following values:

`DeliveryMode.Persistent`

`DeliveryMode.NonPersistent`

For a real-time connection to a broker, the value must be `DeliveryMode.NonPersistent`.

**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. On a real-time connection to a broker, the value is ignored.

**timeToLive (input)**

The time to live for the message in milliseconds. A value of 0 means that the message never expires. For a real-time connection to a broker, the value must be 0.

**Returns:**

Void

**Exceptions:**

- `XMSEException`
- `MessageFormatException`
- `InvalidDestinationException`
- `IllegalStateException`

*Send - Send (to a specified destination)*

**Interface:**

```
void Send(IDestination dest, IMessage msg) ;
```

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 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:****dest (input)**

The Destination object.

**msg (input)**

The Message object.

**Returns:**

Void

**Exceptions:**

- `XMSEException`
- `MessageFormatException`
- `InvalidDestinationException`

*Send - Send (to a specified destination, specifying a delivery mode, priority, and time to live)*

**Interface:**

```
void Send(IDestination dest,
          IMessage msg,
          DeliveryMode deliveryMode,
          Int32 priority,
          Int64 timeToLive) ;
```

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:**

**dest (input)**

The Destination object.

**msg (input)**

The Message object.

**deliveryMode (input)**

The delivery mode for the message, which must be one of the following values:

`DeliveryMode.Persistent`  
`DeliveryMode.NonPersistent`

For a real-time connection to a broker, the value must be `DeliveryMode.NonPersistent`.

**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. On a real-time connection to a broker, the value is ignored.

**timeToLive (input)**

The time to live for the message in milliseconds. A value of 0 means that the message never expires. For a real-time connection to a broker, the value must be 0.

**Returns:**

Void

**Exceptions:**

- `XMSEException`
- `MessageFormatException`
- `InvalidDestinationException`
- `IllegalStateException`

***Inherited properties and methods***

The following methods are inherited from the `IPropertyContext` interface:

[GetBooleanProperty](#), [GetByteProperty](#), [GetBytesProperty](#), [GetCharProperty](#), [GetDoubleProperty](#), [GetFloatProperty](#), [GetIntProperty](#), [GetLongProperty](#), [GetObjectProperty](#), [GetShortProperty](#), [GetStringProperty](#), [SetBooleanProperty](#), [SetByteProperty](#), [SetBytesProperty](#), [SetCharProperty](#), [SetDoubleProperty](#), [SetFloatProperty](#), [SetIntProperty](#), [SetLongProperty](#), [SetObjectProperty](#), [SetShortProperty](#), [SetStringProperty](#)

## IOBJECTMESSAGE

An object message is a message whose body comprises a serialized Java or .NET object.

### Inheritance hierarchy:

```
IBM.XMS.IPropertyContext
|
+----IBM.XMS.IMessage
|
+----IBM.XMS.IObjectMessage
```

### Related reference

[Object messages](#)

The body of an object message contains a serializedJava or .NET object.

### .NET properties

*Object - Get and Set Object as Bytes*

#### Interface:

```
System.Object Object
{
    get;
    set;
}

Byte[] GetObject();
```

Get and set the object that forms the body of the object message.

#### Exceptions:

- [XMSEException](#)
- [MessageNotReadableException](#)
- [MessageEOFException](#)
- [MessageNotWritableException](#)

### Inherited properties and methods

The following properties are inherited from the [IMessage](#) interface:

[JMSCorrelationID](#), [JMSDeliveryMode](#), [JMSDestination](#), [JMSExpiration](#), [JMSMessageID](#), [JMSPriority](#), [JMSRedelivered](#), [JMSReplyTo](#), [JMSTimestamp](#), [JMSType](#), [Properties](#)

The following methods are inherited from the [IMessage](#) interface:

[clearBody](#), [clearProperties](#), [PropertyExists](#)

The following methods are inherited from the [IPropertyContext](#) interface:

[GetBooleanProperty](#), [GetByteProperty](#), [GetBytesProperty](#), [GetCharProperty](#), [GetDoubleProperty](#), [GetFloatProperty](#), [GetIntProperty](#), [GetLongProperty](#), [GetObjectProperty](#), [GetShortProperty](#), [GetStringProperty](#), [SetBooleanProperty](#), [SetByteProperty](#), [SetBytesProperty](#), [SetCharProperty](#), [SetDoubleProperty](#), [SetFloatProperty](#), [SetIntProperty](#), [SetLongProperty](#), [SetObjectProperty](#), [SetShortProperty](#), [SetStringProperty](#)

## IPropertyContext

IPropertyContext is an abstract superclass that contains methods that get and set properties. These methods are inherited by other classes.

### Inheritance hierarchy:

None

### Methods

*GetBooleanProperty - Get Boolean Property*

#### Interface:

```
Boolean GetBooleanProperty(String property_name);
```

Get the value of the boolean property with the specified name.

#### Parameters:

**property\_name (input)**

A String object encapsulating the name of the property.

#### Returns:

The value of the property.

#### Thread context:

Determined by the subclass

#### Exceptions:

- XMSEException

*GetByteProperty - Get Byte Property*

#### Interface:

```
Byte    GetByteProperty(String property_name) ;  
Int16   GetSignedByteProperty(String property_name) ;
```

Get the value of the byte property identified by name.

#### Parameters:

**property\_name (input)**

A String object encapsulating the name of the property.

#### Returns:

The value of the property.

#### Thread context:

Determined by the subclass

#### Exceptions:

- XMSEException

*GetBytesProperty - Get Byte Array Property*

#### Interface:

```
Byte[]  GetBytesProperty(String property_name) ;
```

Get the value of the byte array property identified by name.

**Parameters:****property\_name (input)**

A String object encapsulating the name of the property.

**Returns:**

The number of bytes in the array.

**Thread context:**

Determined by the subclass

**Exceptions:**

- XMSEException

*GetCharProperty - Get Character Property***Interface:**

```
Char GetCharProperty(String property_name) ;
```

Get the value of the 2-byte character property identified by name.

**Parameters:****property\_name (input)**

A String object encapsulating the name of the property.

**Returns:**

The value of the property.

**Thread context:**

Determined by the subclass

**Exceptions:**

- XMSEException

*GetDoubleProperty - Get Double Precision Floating Point Property***Interface:**

```
Double GetDoubleProperty(String property_name) ;
```

Get the value of the double precision floating point property identified by name.

**Parameters:****property\_name (input)**

A String object encapsulating the name of the property.

**Returns:**

The value of the property.

**Thread context:**

Determined by the subclass

**Exceptions:**

- XMSEException

### *GetFloatProperty - Get Floating Point Property*

#### **Interface:**

```
Single GetFloatProperty(String property_name) ;
```

Get the value of the floating point property identified by name.

#### **Parameters:**

##### **property\_name (input)**

A String object encapsulating the name of the property.

#### **Returns:**

The value of the property.

#### **Thread context:**

Determined by the subclass

#### **Exceptions:**

- XMSEException

### *GetIntProperty - GetIntProperty*

#### **Interface:**

```
Int32 GetIntProperty(String property_name) ;
```

Get the value of the integer property identified by name.

#### **Parameters:**

##### **property\_name (input)**

A String object encapsulating the name of the property.

#### **Returns:**

The value of the property.

#### **Thread context:**

Determined by the subclass

#### **Exceptions:**

- XMSEException

### *GetLongProperty - Get Long Integer Property*

#### **Interface:**

```
Int64 GetLongProperty(String property_name) ;
```

Get the value of the long integer property identified by name.

#### **Parameters:**

##### **property\_name (input)**

A String object encapsulating the name of the property.

#### **Returns:**

The value of the property.

#### **Thread context:**

Determined by the subclass

#### **Exceptions:**

- XMSEException

### *GetObjectProperty - Get Object Property*

#### **Interface:**

```
Object GetObjectProperty( String property_name) ;
```

Get the value and data type of the property identified by name.

#### **Parameters:**

##### **property\_name (input)**

A String object encapsulating the name of the property.

#### **Returns:**

The value of the property, which is one of the following object types:

- Boolean
- Byte
- Byte[]
- Char
- Double
- Single
- Int32
- Int64
- Int16
- String

#### **Thread context:**

Determined by the subclass

#### **Exceptions:**

- XMSEException

### *GetShortProperty - Get Short Integer Property*

#### **Interface:**

```
Int16 GetShortProperty(String property_name) ;
```

Get the value of the short integer property identified by name.

#### **Parameters:**

##### **property\_name (input)**

A String object encapsulating the name of the property.

#### **Returns:**

The value of the property.

#### **Thread context:**

Determined by the subclass

#### **Exceptions:**

- XMSEException

## *GetStringProperty - GetStringProperty*

### **Interface:**

```
String GetStringProperty(String property_name) ;
```

Get the value of the string property identified by name.

### **Parameters:**

#### **property\_name (input)**

A String object encapsulating the name of the property.

### **Returns:**

A String object encapsulating the string that is the value of the property. If data conversion is required, this value is the string after conversion.

### **Thread context:**

Determined by the subclass

### **Exceptions:**

- XMSEException

## *SetBooleanProperty - Set Boolean Property*

### **Interface:**

```
void SetBooleanProperty( String property_name, Boolean value) ;
```

Set the value of the boolean property identified by name.

### **Parameters:**

#### **property\_name (input)**

A String object encapsulating the name of the property.

#### **value (input)**

The value of the property.

### **Returns:**

Void

### **Thread context:**

Determined by the subclass

### **Exceptions:**

- XMSEException
- MessageNotWritableException

## *SetByteProperty - Set Byte Property*

### **Interface:**

```
void SetByteProperty( String property_name, Byte value) ;  
void SetSignedByteProperty( String property_name, Int16 value) ;
```

Set the value of the byte property identified by name.

### **Parameters:**

#### **property\_name (input)**

A String object encapsulating the name of the property.

#### **value (input)**

The value of the property.

**Returns:**

Void

**Thread context:**

Determined by the subclass

**Exceptions:**

- XMSEException
- MessageNotWritableException

*SetBytesProperty - Set Byte Array Property***Interface:**

```
void SetBytesProperty( String property_name, Byte[] value ) ;
```

Set the value of the byte array property identified by name.

**Parameters:****property\_name (input)**

A String object encapsulating the name of the property.

**value (input)**

The value of the property, which is an array of bytes.

**Returns:**

Void

**Thread context:**

Determined by the subclass

**Exceptions:**

- XMSEException
- MessageNotWritableException

*SetCharProperty - Set Character Property***Interface:**

```
void SetCharProperty( String property_name, Char value) ;
```

Set the value of the 2-byte character property identified by name.

**Parameters:****property\_name (input)**

A String object encapsulating the name of the property.

**value (input)**

The value of the property.

**Returns:**

Void

**Thread context:**

Determined by the subclass

**Exceptions:**

- XMSEException
- MessageNotWritableException

### *SetDoubleProperty - Set Double Precision Floating Point Property*

#### **Interface:**

```
void SetDoubleProperty( String property_name, Double value) ;
```

Set the value of the double precision floating point property identified by name.

#### **Parameters:**

**property\_name (input)**

A String object encapsulating the name of the property.

**value (input)**

The value of the property.

#### **Returns:**

Void

#### **Thread context:**

Determined by the subclass

#### **Exceptions:**

- XMSEException
- MessageNotWritableException

### *SetFloatProperty - Set Floating Point Property*

#### **Interface:**

```
void SetFloatProperty( String property_name, Single value) ;
```

Set the value of the floating point property identified by name.

#### **Parameters:**

**property\_name (input)**

A String object encapsulating the name of the property.

**value (input)**

The value of the property.

#### **Returns:**

Void

#### **Thread context:**

Determined by the subclass

#### **Exceptions:**

- XMSEException
- MessageNotWritableException

### *SetIntProperty - Set Integer Property*

#### **Interface:**

```
void SetIntProperty( String property_name, Int32 value) ;
```

Set the value of the integer property identified by name.

#### **Parameters:**

**property\_name (input)**

A String object encapsulating the name of the property.

**value (input)**

The value of the property.

**Returns:**

Void

**Thread context:**

Determined by the subclass

**Exceptions:**

- XMSEException
- MessageNotWritableException

*SetLongProperty - Set Long Integer Property***Interface:**

```
void SetLongProperty( String property_name, Int64 value) ;
```

Set the value of the long integer property identified by name.

**Parameters:****property\_name (input)**

A String object encapsulating the name of the property.

**value (input)**

The value of the property.

**Returns:**

Void

**Thread context:**

Determined by the subclass

**Exceptions:**

- XMSEException
- MessageNotWritableException

*SetObjectProperty - Set Object Property***Interface:**

```
void SetObjectProperty( String property_name, Object value) ;
```

Set the value and data type of a property identified by name.

**Parameters:****property\_name (input)**

A String object encapsulating the name of the property.

**objectType (input)**

The value of the property, which must be one of the following object types:

Boolean  
Byte  
Byte[]  
Char  
Double  
Single  
Int32

Int64  
Int16  
String

**value (input)**

The value of the property as an array of bytes.

**length (input)**

The number of bytes in the array.

**Returns:**

Void

**Thread context:**

Determined by the subclass

**Exceptions:**

- XMSEException
- MessageNotWritableException

*SetShortProperty - Set Short Integer Property*

**Interface:**

```
void SetShortProperty( String property_name, Int16 value) ;
```

Set the value of the short integer property identified by name.

**Parameters:**

**property\_name (input)**

A String object encapsulating the name of the property.

**value (input)**

The value of the property.

**Returns:**

Void

**Thread context:**

Determined by the subclass

**Exceptions:**

- XMSEException
- MessageNotWritableException

*SetStringProperty - Set String Property*

**Interface:**

```
void SetStringProperty( String property_name, String value);
```

Set the value of the string property identified by name.

**Parameters:**

**property\_name (input)**

A String object encapsulating the name of the property.

**value (input)**

A String object encapsulating the string that is the value of the property.

**Returns:**

Void

**Thread context:**

Determined by the subclass

**Exceptions:**

- XMSEException
- MessageNotWritableException

**IQueueBrowser**

An application uses a queue browser to browse messages on a queue without removing them.

**Inheritance hierarchy:**

```

IBM.XMS.IPropertyContext
System.Collections.IEnumerable
|
+---- IBM.XMS.IQueueBrowser

```

**.NET Properties**

*MessageSelector* - Get Message Selector

**Interface:**

```

String MessageSelector
{
    get;
}

```

Get the message selector for the queue browser.

The message selector is a String object encapsulating the message selector expression. If data conversion is required, this value is the message selector expression after conversion. If the queue browser does not have a message selector, the method returns a null String object.

**Exceptions:**

- XMSEException

*Queue* - Get Queue

**Interface:**

```

IDestination Queue
{
    get;
}

```

Get the queue associated with the queue browser as a destination object representing the queue.

**Exceptions:**

- XMSEException

**Methods**

## Close - Close Queue Browser

### Interface:

```
void Close();
```

Close the queue browser.

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

### Parameters:

None

### Returns:

Void

### Exceptions:

- XMSEException

## GetEnumerator - Get Messages

### Interface:

```
IEnumerator GetEnumerator();
```

Get a list of the messages on the queue.

The method returns an enumerator that encapsulates a list of Message objects. The order of the Message objects is the same as the order in which the messages would be retrieved from the queue. The application can then use the enumerator to browse each message in turn.

The enumerator is updated dynamically as messages are put on the queue and removed from the queue. Each time the application calls `IEnumerator.MoveNext()` to browse the next message on the queue, the message reflects the current contents of the queue.

If an application calls this method more than once for a queue browser, each call returns a new enumerator. The application can therefore use more than one enumerator to browse the messages on a queue and maintain multiple positions within the queue.

### Parameters:

None

### Returns:

The Iterator object.

### Exceptions:

- XMSEException

## ***Inherited properties and methods***

The following methods are inherited from the [IPropertyContext](#) interface:

[GetBooleanProperty](#), [GetByteProperty](#), [GetBytesProperty](#), [GetCharProperty](#), [GetDoubleProperty](#), [GetFloatProperty](#), [GetIntProperty](#), [GetLongProperty](#), [GetObjectProperty](#), [GetShortProperty](#), [GetStringProperty](#), [SetBooleanProperty](#), [SetByteProperty](#), [SetBytesProperty](#), [SetCharProperty](#), [SetDoubleProperty](#), [SetFloatProperty](#), [SetIntProperty](#), [SetLongProperty](#), [SetObjectProperty](#), [SetShortProperty](#), [SetStringProperty](#)

## Requestor

An application uses a requestor to send a request message and then wait for, and receive, the reply.

### Inheritance hierarchy:

None

### Constructors

*Requestor - Create Requestor*

#### Interface:

```
Requestor(ISession sess, IDestination dest);
```

Create a requestor.

#### Parameters:

##### **sess (input)**

A Session object. The session must not be transacted and must have one of the following acknowledgement modes:

`AcknowledgeMode.AutoAcknowledge`

`AcknowledgeMode.DupsOkAcknowledge`

##### **dest (input)**

A Destination object representing the destination where the application can send request messages.

#### Thread context:

The session associated with the requestor

#### Exceptions:

- `XMSEException`

### Methods

*Close - Close Requestor*

#### Interface:

```
void Close();
```

Close the requestor.

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

**Note:** When an application closes a requestor, the associated session does not close as well. In this respect, XMS behaves differently compared to JMS.

#### Parameters:

None

#### Returns:

Void

#### Thread context:

Any

#### Exceptions:

- `XMSEException`

## Request - Request Response

### Interface:

```
IMessage Request(IMessage requestMessage);
```

Send a request message and then wait for, and receive, a reply from the application that receives the request message.

A call to this method blocks until a reply is received or until the session ends, whichever is the sooner.

### Parameters:

#### **requestMessage (input)**

The Message object encapsulating the request message.

### Returns:

A pointer to the Message object encapsulating the reply message.

### Thread context:

The session associated with the requestor

### Exceptions:

- XMSEException

## ResourceAllocationException

### Inheritance hierarchy:

```
IBM.XMS.XMSEException
|
+----IBM.XMS.XMSEException
|
+----IBM.XMS.ResourceAllocationException
```

XMS throws this exception if XMS cannot allocate the resources required by a method.

### ***Inherited properties and methods***

The following methods are inherited from the [XMSEException](#) interface:

[GetErrorCode](#), [GetLinkedException](#)

## SecurityException

### Inheritance hierarchy:

```
IBM.XMS.XMSEException
|
+----IBM.XMS.XMSEException
|
+----IBM.XMS.SecurityException
```

XMS throws this exception if the user identifier and password provided to authenticate an application are rejected. XMS also throws this exception if an authority check fails and prevents a method from completing.

### ***Inherited properties and methods***

The following methods are inherited from the [XMSEException](#) interface:

[GetErrorCode](#), [GetLinkedException](#)

## ISession

A session is a single threaded context for sending and receiving messages.

### Inheritance hierarchy:

```
IBM.XMS.IPropertyContext
|
+---- IBM.XMS.ISession
```

For a list of the XMS defined properties of a Session object, see [“Properties of Session” on page 190](#).

### .NET properties

*AcknowledgeMode - Get Acknowledgement Mode*

#### Interface:

```
AcknowledgeMode AcknowledgeMode
{
    get;
}
```

Get the acknowledgement mode for the session.

The acknowledgement mode is specified when the session is created.

Provided the session is not transacted, the acknowledgement mode is one of the following values:

```
AcknowledgeMode.AutoAcknowledge
AcknowledgeMode.ClientAcknowledge
AcknowledgeMode.DupsOkAcknowledge
```

For more information about acknowledgement modes, see [“Message acknowledgement” on page 26](#).

A session that is transacted has no acknowledgement mode. If the session is transacted, the method returns `AcknowledgeMode.SessionTransacted` instead.

#### Exceptions:

- `XMSEException`

*Transacted - Determine Whether Transacted*

#### Interface:

```
Boolean Transacted
{
    get;
}
```

Determine whether the session is transacted.

The transacted stated is:

- `True`, if the session is transacted.
- `False`, if the session is not transacted.

For a real-time connection to a broker, the method always returns `False`.

#### Exceptions:

- `XMSEException`

## Methods

### *Close - Close Session*

#### **Interface:**

```
void Close();
```

Close the session. If the session is transacted, any transaction in progress is rolled back.

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

#### **Parameters:**

None

#### **Returns:**

Void

#### **Thread context:**

Any

#### **Exceptions:**

- XMSEException

### *Commit - Commit*

#### **Interface:**

```
void Commit();
```

Commit all messages processed in the current transaction.

The session must be a transacted session.

#### **Parameters:**

None

#### **Returns:**

Void

#### **Exceptions:**

- XMSEException
- IllegalStateException
- TransactionRolledBackException

### *CreateBrowser - Create Queue Browser*

#### **Interface:**

```
IQueueBrowser CreateBrowser(IDestination queue) ;
```

Create a queue browser for the specified queue.

#### **Parameters:**

##### **queue (input)**

A Destination object representing the queue.

#### **Returns:**

The QueueBrowser object.

**Exceptions:**

- XMSEException
- InvalidDestinationException

*CreateBrowser - Create Queue Browser (with message selector)*

**Interface:**

```
IQueueBrowser CreateBrowser(IDestination queue, String selector) ;
```

Create a queue browser for the specified queue using a message selector.

**Parameters:****queue (input)**

A Destination object representing the queue.

**selector (input)**

A String object encapsulating a message selector expression. Only those messages with properties that match the message selector expression are delivered to the queue browser.

A null String object means that there is no message selector for the queue browser.

**Returns:**

The QueueBrowser object.

**Exceptions:**

- XMSEException
- InvalidDestinationException
- InvalidSelectorException

*CreateBytesMessage - Create Bytes Message*

**Interface:**

```
IBytesMessage CreateBytesMessage();
```

Create a bytes message.

**Parameters:**

None

**Returns:**

The BytesMessage object.

**Exceptions:**

- XMSEException
- IllegalStateException (The session is closed)

*CreateConsumer - Create Consumer*

**Interface:**

```
IMessageConsumer CreateConsumer(IDestination dest) ;
```

Create a message consumer for the specified destination.

**Parameters:****dest (input)**

The Destination object.

**Returns:**

The MessageConsumer object.

**Exceptions:**

- XMSEException
- InvalidDestinationException

*CreateConsumer - Create Consumer (with message selector)*

**Interface:**

```
IMessageConsumer CreateConsumer(IDestination dest,  
                                String selector) ;
```

Create a message consumer for the specified destination using a message selector.

**Parameters:****dest (input)**

The Destination object.

**selector (input)**

A String object encapsulating a message selector expression. Only those messages with properties that match the message selector expression are delivered to the message consumer.

A null String object means that there is no message selector for the message consumer.

**Returns:**

The MessageConsumer object.

**Exceptions:**

- XMSEException
- InvalidDestinationException
- InvalidSelectorException

*CreateConsumer - Create Consumer (with message selector and local message flag)*

**Interface:**

```
IMessageConsumer CreateConsumer(IDestination dest,  
                                String selector,  
                                Boolean noLocal) ;
```

Create a message consumer for the specified destination using a message selector and, if the destination is a topic, specifying whether the message consumer receives the messages published by its own connection.

**Parameters:****dest (input)**

The Destination object.

**selector (input)**

A String object encapsulating a message selector expression. Only those messages with properties that match the message selector expression are delivered to the message consumer.

A null String object means that there is no message selector for the message consumer.

**noLocal (input)**

The value `True` means that the message consumer does not receive the messages published by its own connection. The value `False` means that the message consumer does receive the messages published by its own connection. The default value is `False`.

**Returns:**

The `MessageConsumer` object.

**Exceptions:**

- `XMSEException`
- `InvalidDestinationException`
- `InvalidSelectorException`

*CreateDurableSubscriber - Create Durable Subscriber***Interface:**

```
IMessageConsumer CreateDurableSubscriber(IDestination dest,  
                                         String subscription) ;
```

Create a durable subscriber for the specified topic.

This method is not valid for a real-time connection to a broker.

For more information about durable subscribers, see [“Durable subscribers” on page 34](#).

**Parameters:****dest (input)**

A `Destination` object representing the topic. The topic must not be a temporary topic.

**subscription (input)**

A `String` object encapsulating a name that identifies the durable subscription. The name must be unique within the client identifier for the connection.

**Returns:**

The `MessageConsumer` object representing the durable subscriber.

**Exceptions:**

- `XMSEException`
- `InvalidDestinationException`

*CreateDurableSubscriber - Create Durable Subscriber (with message selector and local message flag)***Interface:**

```
IMessageConsumer CreateDurableSubscriber(IDestination dest,  
                                         String subscription,  
                                         String selector,  
                                         Boolean noLocal) ;
```

Create a durable subscriber for the specified topic using a message selector and specifying whether the durable subscriber receives the messages published by its own connection.

This method is not valid for a real-time connection to a broker.

For more information about durable subscribers, see [“Durable subscribers” on page 34](#).

**Parameters:****dest (input)**

A `Destination` object representing the topic. The topic must not be a temporary topic.

**subscription (input)**

A String object encapsulating a name that identifies the durable subscription. The name must be unique within the client identifier for the connection.

**selector (input)**

A String object encapsulating a message selector expression. Only those messages with properties that match the message selector expression are delivered to the durable subscriber.

A null String object means that there is no message selector for the durable subscriber.

**noLocal (input)**

The value `True` means that the durable subscriber does not receive the messages published by its own connection. The value `False` means that the durable subscriber does receive the messages published by its own connection. The default value is `False`.

**Returns:**

The `MessageConsumer` object representing the durable subscriber.

**Exceptions:**

- `XMSEException`
- `InvalidDestinationException`
- `InvalidSelectorException`

*CreateMapMessage - Create Map Message***Interface:**

```
IMapMessage CreateMapMessage();
```

Create a map message.

**Parameters:**

None

**Returns:**

The `MapMessage` object.

**Exceptions:**

- `XMSEException`
- `IllegalStateException` (The session is closed)

*CreateMessage - Create Message***Interface:**

```
IMessage CreateMessage();
```

Create a message that has no body.

**Parameters:**

None

**Returns:**

The `Message` object.

**Exceptions:**

- `XMSEException`
- `IllegalStateException` (The session is closed)

## *CreateObjectMessage - Create Object Message*

### **Interface:**

```
IObjectMessage CreateObjectMessage();
```

Create an object message.

### **Parameters:**

None

### **Returns:**

The ObjectMessage object.

### **Exceptions:**

- XMSEException
- IllegalStateException (The session is closed)

## *CreateProducer - Create Producer*

### **Interface:**

```
IMessageProducer CreateProducer(IDestination dest) ;
```

Create a message producer to send messages to the specified destination.

### **Parameters:**

#### **dest (input)**

The Destination object.

If you specify a null Destination object, the message producer is created without a destination. In this case, the application must specify a destination every time it uses the message producer to send a message.

### **Returns:**

The MessageProducer object.

### **Exceptions:**

- XMSEException
- InvalidDestinationException

## *CreateQueue - Create Queue*

### **Interface:**

```
IDestination CreateQueue(String queue) ;
```

Create a Destination object to represent a queue in the messaging server.

This method does not create the queue in the messaging server. You must create the queue before an application can call this method.

### **Parameters:**

#### **queue (input)**

A String object encapsulating the name of the queue, or encapsulating a uniform resource identifier (URI) that identifies the queue.

### **Returns:**

The Destination object representing the queue.

**Exceptions:**

- XMSEException

*CreateStreamMessage - Create Stream Message***Interface:**

```
IStreamMessage CreateStreamMessage();
```

Create a stream message.

**Parameters:**

None

**Returns:**

The StreamMessage object.

**Exceptions:**

- XMSEException
- XMS\_ILLEGAL\_STATE\_EXCEPTION

*CreateTemporaryQueue - Create Temporary Queue***Interface:**

```
IDestination CreateTemporaryQueue() ;
```

Create a temporary queue.

The scope of the temporary queue is the connection. Only the sessions created by the connection can use the temporary queue.

The temporary queue remains until it is explicitly deleted, or the connection ends, whichever is the sooner.

For more information about temporary queues, see [“Temporary destinations”](#) on page 32.

**Parameters:**

None

**Returns:**

The Destination object representing the temporary queue.

**Exceptions:**

- XMSEException

*CreateTemporaryTopic - Create Temporary Topic***Interface:**

```
IDestination CreateTemporaryTopic() ;
```

Create a temporary topic.

The scope of the temporary topic is the connection. Only the sessions created by the connection can use the temporary topic.

The temporary topic remains until it is explicitly deleted, or the connection ends, whichever is the sooner.

For more information about temporary topics, see [“Temporary destinations”](#) on page 32.

**Parameters:**

None

**Returns:**

The Destination object representing the temporary topic.

**Exceptions:**

- XMSEException

*CreateTextMessage - Create Text Message*

**Interface:**

```
ITextMessage CreateTextMessage();
```

Create a text message with an empty body.

**Parameters:**

None

**Returns:**

The TextMessage object.

**Exceptions:**

- XMSEException

*CreateTextMessage - Create Text Message (initialized)*

**Interface:**

```
ITextMessage CreateTextMessage(String initialValue);
```

Create a text message whose body is initialized with the specified text.

**Parameters:****initialValue (input)**

A String object encapsulating the text to initialize the body of the text message.

None

**Returns:**

The TextMessage object.

**Exceptions:**

- XMSEException

*CreateTopic - Create Topic*

**Interface:**

```
IDestination CreateTopic(String topic) ;
```

Create a Destination object to represent a topic.

**Parameters:****topic (input)**

A String object encapsulating the name of the topic, or encapsulating a uniform resource identifier (URI) that identifies the topic.

**Returns:**

The Destination object representing the topic.

**Exceptions:**

- XMSEException

*Recover - Recover***Interface:**

```
void Recover();
```

Recover the session. Message delivery is stopped and then restarted with the oldest unacknowledged message.

The session must not be a transacted session.

For more information about recovering a session, see [“Message acknowledgement” on page 26](#).

**Parameters:**

None

**Returns:**

Void

**Exceptions:**

- XMSEException
- IllegalStateException

*Rollback - Rollback***Interface:**

```
void Rollback();
```

Rollback all messages processed in the current transaction.

The session must be a transacted session.

**Parameters:**

None

**Returns:**

Void

**Exceptions:**

- XMSEException
- IllegalStateException

*Unsubscribe - Unsubscribe***Interface:**

```
void Unsubscribe(String subscription);
```

Delete a durable subscription. The messaging server deletes the record of the durable subscription that it is maintaining and does not send any more messages to the durable subscriber.

An application cannot delete a durable subscription in any of the following circumstances:

- While there is an active message consumer for the durable subscription
- While a consumed message is part of a pending transaction
- While a consumed message was not acknowledged

This method is not valid for a real-time connection to a broker.

**Parameters:**

**subscription (input)**

A String object encapsulating the name that identifies the durable subscription.

**Returns:**

Void

**Exceptions:**

- XMSException
- InvalidDestinationException
- IllegalStateException

***Inherited properties and methods***

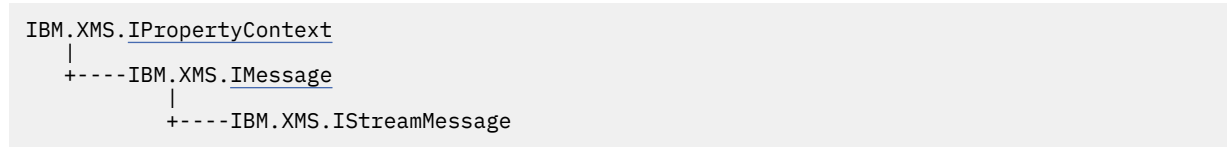
The following methods are inherited from the IPROPERTYContext interface:

GetBooleanProperty, GetByteProperty, GetBytesProperty, GetCharProperty, GetDoubleProperty, GetFloatProperty, GetIntProperty, GetLongProperty, GetObjectProperty, GetShortProperty, GetStringProperty, SetBooleanProperty, SetByteProperty, SetBytesProperty, SetCharProperty, SetDoubleProperty, SetFloatProperty, SetIntProperty, SetLongProperty, SetObjectProperty, SetShortProperty, SetStringProperty

**IStreamMessage**

A stream message is a message whose body comprises a stream of values, where each value has an associated data type. The contents of the body are written and read sequentially.

**Inheritance hierarchy:**



When an application reads a value from the message stream, the value can be converted by XMS into another data type. For more information about this form of implicit conversion, see “Stream messages” on page 78.

**Related reference**

Stream messages

The body of a stream message contains a stream of values, where each value has an associated data type.

**Methods**

*ReadBoolean - Read Boolean Value*

**Interface:**

```
Boolean ReadBoolean();
```

Read a boolean value from the message stream.

**Parameters:**

None

**Returns:**

The boolean value that is read.

**Exceptions:**

- XMSEException
- MessageNotReadableException
- MessageEOFException

*ReadByte - Read Byte***Interface:**

```
Int16  ReadSignedByte();  
Byte   ReadByte();
```

Read a signed 8-bit integer from the message stream.

**Parameters:**

None

**Returns:**

The byte that is read.

**Exceptions:**

- XMSEException
- MessageNotReadableException
- MessageEOFException

*ReadBytes - Read Bytes***Interface:**

```
Int32  ReadBytes(Byte[] array);
```

Read an array of bytes from the message stream.

**Parameters:****array (input)**

The buffer containing the array of bytes that is read and the length of the buffer in bytes.

If the number of bytes in the array is less than or equal to the length of the buffer, the whole array is read into the buffer. If the number of bytes in the array is greater than the length of the buffer, the buffer is filled with part of the array, and an internal cursor marks the position of the next byte to be read. A subsequent call to `readBytes()` reads bytes from the array starting from the current position of the cursor.

If you specify a null pointer on input, the call skips over the array of bytes without reading it.

**Returns:**

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 in the array remaining to be read. If there are no bytes remaining to be read from the array before the call, the value is `XMSC_END_OF_BYTEARRAY`.

If you specify a null pointer on input, the method returns no value.

**Exceptions:**

- XMSEException

- MessageNotReadableException
- MessageEOFException

#### *ReadChar - Read Character*

##### **Interface:**

```
Char ReadChar();
```

Read a 2-byte character from the message stream.

##### **Parameters:**

None

##### **Returns:**

The character that is read.

##### **Exceptions:**

- XMSEException
- MessageNotReadableException
- MessageEOFException

#### *ReadDouble - Read Double Precision Floating Point Number*

##### **Interface:**

```
Double ReadDouble();
```

Read an 8-byte double precision floating point number from the message stream.

##### **Parameters:**

None

##### **Returns:**

The double precision floating point number that is read.

##### **Exceptions:**

- XMSEException
- MessageNotReadableException
- MessageEOFException

#### *ReadFloat - Read Floating Point Number*

##### **Interface:**

```
Single ReadFloat();
```

Read a 4-byte floating point number from the message stream.

##### **Parameters:**

None

##### **Returns:**

The floating point number that is read.

##### **Exceptions:**

- XMSEException
- MessageNotReadableException

- MessageEOFException

#### *ReadInt - Read Integer*

##### **Interface:**

```
Int32 ReadInt();
```

Read a signed 32-bit integer from the message stream.

##### **Parameters:**

None

##### **Returns:**

The integer that is read.

##### **Exceptions:**

- XMSEException
- MessageNotReadableException
- MessageEOFException

#### *ReadLong - Read Long Integer*

##### **Interface:**

```
Int64 ReadLong();
```

Read a signed 64-bit integer from the message stream.

##### **Parameters:**

None

##### **Returns:**

The long integer that is read.

##### **Exceptions:**

- XMSEException
- MessageNotReadableException
- MessageEOFException

#### *ReadObject - Read Object*

##### **Interface:**

```
Object ReadObject();
```

Read a value from the message stream, and return its data type.

##### **Parameters:**

None

##### **Returns:**

The value, which is one of the following object types:

- Boolean
- Byte
- Byte[]
- Char

Double  
Single  
Int32  
Int64  
Int16  
String

**Exceptions:**

XMSEException

*ReadShort - Read Short Integer*

**Interface:**

```
Int16 ReadShort();
```

Read a signed 16-bit integer from the message stream.

**Parameters:**

None

**Returns:**

The short integer that is read.

**Exceptions:**

- XMSEException
- MessageNotReadableException
- MessageEOFException

*ReadString - Read String*

**Interface:**

```
String ReadString();
```

Read a string from the message stream. If required, XMS converts the characters in the string into the local code page.

**Parameters:**

None

**Returns:**

A String object encapsulating the string that is read. If data conversion is required, this is the string after conversion.

**Exceptions:**

- XMSEException
- MessageNotReadableException
- MessageEOFException

*Reset - Reset*

**Interface:**

```
void Reset();
```

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

**Parameters:**

None

**Returns:**

Void

**Exceptions:**

- XMSEException
- MessageNotReadableException
- MessageEOFException

*WriteBoolean - Write Boolean Value*

**Interface:**

```
void WriteBoolean(Boolean value);
```

Write a boolean value to the message stream.

**Parameters:**

**value (input)**

The boolean value to be written.

**Returns:**

Void

**Exceptions:**

- XMSEException
- MessageNotWritableException

*WriteByte - Write Byte*

**Interface:**

```
void WriteByte(Byte value);  
void WriteSignedByte(Int16 value);
```

Write a byte to the message stream.

**Parameters:**

**value (input)**

The byte to be written.

**Returns:**

Void

**Exceptions:**

- XMSEException
- MessageNotWritableException

### *WriteBytes - Write Bytes*

#### **Interface:**

```
void WriteBytes(Byte[] value);
```

Write an array of bytes to the message stream.

#### **Parameters:**

**value (input)**

The array of bytes to be written.

**length (input)**

The number of bytes in the array.

#### **Returns:**

Void

#### **Exceptions:**

- XMSEException
- MessageNotWritableException

### *WriteChar - Write Character*

#### **Interface:**

```
void WriteChar(Char value);
```

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

#### **Parameters:**

**value (input)**

The character to be written.

#### **Returns:**

Void

#### **Exceptions:**

- XMSEException
- MessageNotWritableException

### *WriteDouble - Write Double Precision Floating Point Number*

#### **Interface:**

```
void WriteDouble(Double value);
```

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

#### **Parameters:**

**value (input)**

The double precision floating point number to be written.

#### **Returns:**

Void

#### **Exceptions:**

- XMSEException

- MessageNotWritableException

#### *WriteFloat - Write Floating Point Number*

##### **Interface:**

```
void WriteFloat(Single value);
```

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

##### **Parameters:**

###### **value (input)**

The floating point number to be written.

##### **Returns:**

Void

##### **Exceptions:**

- XMSEException
- MessageNotWritableException

#### *WriteInt - Write Integer*

##### **Interface:**

```
void WriteInt(Int32 value);
```

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

##### **Parameters:**

###### **value (input)**

The integer to be written.

##### **Returns:**

Void

##### **Exceptions:**

- XMSEException
- MessageNotWritableException

#### *WriteLong - Write Long Integer*

##### **Interface:**

```
void WriteLong(Int64 value);
```

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

##### **Parameters:**

###### **value (input)**

The long integer to be written.

##### **Returns:**

Void

##### **Exceptions:**

- XMSEException

- MessageNotWritableException

### *WriteObject - Write Object*

#### **Interface:**

```
void WriteObject(Object value);
```

Write a value, with a specified data type, to the message stream.

#### **Parameters:**

##### **objectType (input)**

The value, which must be one of the following object types:

- Boolean
- Byte
- Byte[]
- Char
- Double
- Single
- Int32
- Int64
- Int16
- String

##### **value (input)**

An array of bytes containing the value to be written.

##### **length (input)**

The number of bytes in the array.

#### **Returns:**

Void

#### **Exceptions:**

- XMSEException

### *WriteShort - Write Short Integer*

#### **Interface:**

```
void WriteShort(Int16 value);
```

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

#### **Parameters:**

##### **value (input)**

The short integer to be written.

#### **Returns:**

Void

#### **Exceptions:**

- XMSEException
- MessageNotWritableException

*WriteString - Write String*

**Interface:**

```
void WriteString(String value);
```

Write a string to the message stream.

**Parameters:**

**value (input)**  
A String object encapsulating the string to be written.

**Returns:**

Void

**Exceptions:**

- XMSException
- MessageNotWritableException

***Inherited properties and methods***

The following properties are inherited from the [IMessage](#) interface:

[JMSCorrelationID](#), [JMSDeliveryMode](#), [JMSDestination](#), [JMSExpiration](#), [JMSMessageID](#), [JMSPriority](#), [JMSRedelivered](#), [JMSReplyTo](#), [JMSTimestamp](#), [JMSType](#), [Properties](#)

The following methods are inherited from the [IMessage](#) interface:

[clearBody](#), [clearProperties](#), [PropertyExists](#)

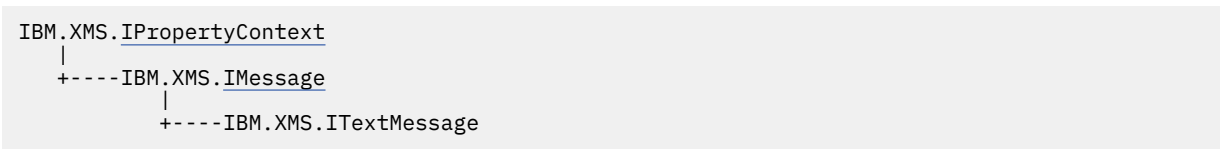
The following methods are inherited from the [IPropertyContext](#) interface:

[GetBooleanProperty](#), [GetByteProperty](#), [GetBytesProperty](#), [GetCharProperty](#), [GetDoubleProperty](#), [GetFloatProperty](#), [GetIntProperty](#), [GetLongProperty](#), [GetObjectProperty](#), [GetShortProperty](#), [GetStringProperty](#), [SetBooleanProperty](#), [SetByteProperty](#), [SetBytesProperty](#), [SetCharProperty](#), [SetDoubleProperty](#), [SetFloatProperty](#), [SetIntProperty](#), [SetLongProperty](#), [SetObjectProperty](#), [SetShortProperty](#), [SetStringProperty](#)

**ITextMessage**

A text message is a message whose body comprises a string.

**Inheritance hierarchy:**



**Related reference**

[Text messages](#)

The body of a text message contains a string.

***.NET properties***

*Text - Get and Set Text*

**Interface:**

```
String Text
{
    get;
    set;
}
```

Get and set the string that forms the body of the text message.

If required, XMS converts the characters in the string into the local code page.

**Exceptions:**

- `XMSEException`
- `MessageNotReadableException`
- `MessageNotWritableException`
- `MessageEOFException`

***Inherited properties and methods***

The following properties are inherited from the `IMessage` interface:

`JMSCorrelationID`, `JMSDeliveryMode`, `JMSDestination`, `JMSExpiration`, `JMSMessageID`, `JMSPriority`, `JMSRedelivered`, `JMSReplyTo`, `JMSTimestamp`, `JMSType`, `Properties`

The following methods are inherited from the `IMessage` interface:

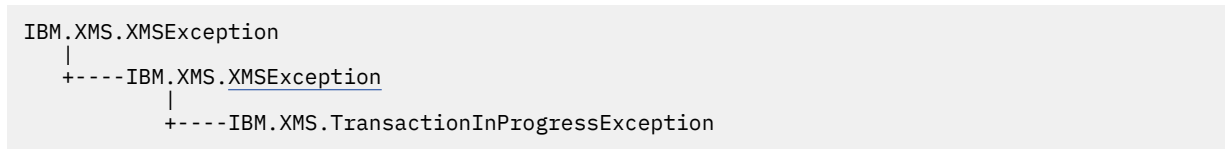
`clearBody`, `clearProperties`, `PropertyExists`

The following methods are inherited from the `IPropertyContext` interface:

`GetBooleanProperty`, `GetByteProperty`, `GetBytesProperty`, `GetCharProperty`, `GetDoubleProperty`, `GetFloatProperty`, `GetIntProperty`, `GetLongProperty`, `GetObjectProperty`, `GetShortProperty`, `GetStringProperty`, `SetBooleanProperty`, `SetByteProperty`, `SetBytesProperty`, `SetCharProperty`, `SetDoubleProperty`, `SetFloatProperty`, `SetIntProperty`, `SetLongProperty`, `SetObjectProperty`, `SetShortProperty`, `SetStringProperty`

**TransactionInProgressException**

**Inheritance hierarchy:**



XMS throws this exception if an application requests an operation that is not valid because a transaction is in progress.

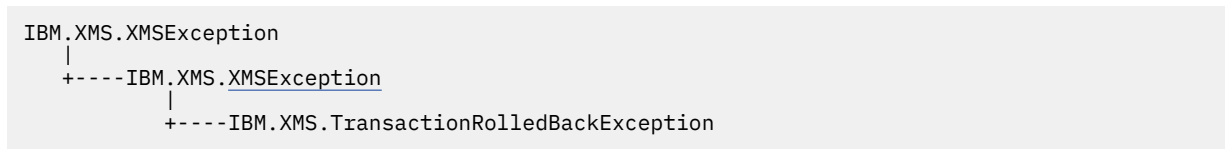
***Inherited properties and methods***

The following methods are inherited from the `XMSEException` interface:

`GetErrorCode`, `GetLinkedException`

**TransactionRolledBackException**

**Inheritance hierarchy:**



XMS throws this exception if an application calls `Session.commit()` to commit the current transaction, but the transaction is then rolled back.

## ***Inherited properties and methods***

The following methods are inherited from the [XMSEException](#) interface:

[GetErrorCode](#), [GetLinkedException](#)

## **XMSEException**

If XMS detects an error while processing a call to a .NET method, XMS throws an exception. An exception is an object that encapsulates information about the error.

### **Inheritance hierarchy:**

```
System.Exception
|
+----IBM.XMS.XMSEException
```

There are different types of XMS exception, and an XMSEException object is just one type of exception. However, the XMSEException class is a superclass of the other XMS exception classes. XMS throws an XMSEException object in situations where none of the other types of exception are appropriate.

## ***.NET properties***

*ErrorCode* - Get Error Code

### **Interface:**

```
public String ErrorCode
{
    get {return errorCode_;}
}
```

Get the error code.

### **Exceptions:**

- XMSEException

*LinkedException* - Get Linked Exception

### **Interface:**

```
public Exception LinkedException
{
    get { return linkedException_;}
    set { linkedException_ = value;}
}
```

Get the next exception in the chain of exceptions.

The method returns a null if there are no more exceptions in the chain.

### **Exceptions:**

- XMSEException

## **XMSFactoryFactory**

If an application is not using administered objects, use this class to create connection factories, queues, and topics.

### **Inheritance hierarchy:**

None

## **.NET properties**

*Metadata - Retrieve metadata*

### **Interface:**

```
IConnectionMetaData MetaData
```

Get the metadata that is appropriate to the connection type of the XMSFactoryFactory object.

### **Exceptions:**

None

## **Methods**

*CreateConnectionFactory - Create Connection Factory*

### **Interface:**

```
IConnectionFactory CreateConnectionFactory();
```

Create a ConnectionFactory object of the declared type.

### **Parameters:**

None

### **Returns:**

The ConnectionFactory object.

### **Exceptions:**

- XMSEException

*CreateQueue - Create Queue*

### **Interface:**

```
IDestination CreateQueue(String name);
```

Create a Destination object to represent a queue in the messaging server.

This method does not create the queue in the messaging server. You must create the queue before an application can call this method.

### **Parameters:**

#### **name (input)**

A String object encapsulating the name of the queue, or encapsulating a uniform resource identifier (URI) that identifies the queue.

### **Returns:**

The Destination object representing the queue.

### **Exceptions:**

- XMSEException

## *CreateTopic - Create Topic*

### **Interface:**

```
IDestination CreateTopic(String name);
```

Create a Destination object to represent a topic.

### **Parameters:**

#### **name (input)**

A String object encapsulating the name of the topic, or encapsulating a uniform resource identifier (URI) that identifies the topic.

### **Returns:**

The Destination object representing the topic.

### **Exceptions:**

- XMSEException

## *GetInstance - Get an instance of XMSFactoryFactory*

### **Interface:**

```
static XMSFactoryFactory GetInstance(int connectionType);
```

Create an instance of XMSFactoryFactory. An XMS application uses an XMSFactoryFactory object to get a reference to a ConnectionFactory object that is appropriate to the required type of protocol. This ConnectionFactory object can then produce connections for that protocol type only.

### **Parameters:**

#### **connectionType (input)**

The type of connection for which the ConnectionFactory object produces connections:

- XMSC.CT\_WPM
- XMSC.CT\_RTT
- XMSC.CT\_WMQ

### **Returns:**

The XMSFactoryFactory object dedicated to the declared connection type.

### **Exceptions:**

- NotSupportedException

## **Properties of XMS objects**

This chapter documents the object properties defined by XMS.

The chapter contains the following sections:

- [“Properties of Connection” on page 174](#)
- [“Properties of ConnectionFactory” on page 175](#)
- [“Properties of ConnectionMetaData” on page 181](#)
- [“Properties of Destination” on page 181](#)
- [“Properties of InitialContext” on page 184](#)
- [“Properties of Message” on page 185](#)
- [“Properties of MessageConsumer” on page 190](#)
- [“Properties of MessageProducer” on page 190](#)
- [“Properties of Session” on page 190](#)

Each section lists the properties of an object of the specified type and provides a short description of each property.

This chapter also contains the [“Property definitions” on page 190](#) section, which provides a definition of each property.

If an application defines its own properties of the objects described in this chapter, it does not cause an error, but it might cause unpredictable results.

**Note:** The property names and values in this section are shown in the form `XMSC.NAME`, which is the form used for C and C++. However, in .NET, the form of the property name can be either `XMSC.NAME` or `XMSC_NAME`, depending on how you are using it:

- If you are specifying a property, the property name must be in the form `XMSC.NAME` as shown in the following example:

```
cf.SetStringProperty(XMSC.WMQ_CHANNEL, "DOTNET.SVRCONN");
```

- If you are specifying a string, the property name must be in the form `XMSC_NAME` as shown in the following example:

```
cf.SetStringProperty("XMSC_WMQ_CHANNEL", "DOTNET.SVRCONN");
```

In .NET, property names and values are provided as constants in the XMSC class. These constants identify strings and would be used by any XMS .NET application. If you are using these predefined constants, the property names and values are in the form `XMSC.NAME`, so, for example, you would use `XMSC.USERID`, rather than `XMSC_USERID`.

The data types are also in the form used for C/C++. You can find the corresponding values for .NET in [“Data types for .NET” on page 44](#).

### Related concepts

[Building your own applications](#)

You build your own applications like you build the sample applications.

### Related reference

[.NET interfaces](#)

This section documents the .NET class interfaces and their properties and methods.

## Properties of Connection

An overview of the properties of the Connection object, with links to more detailed reference information.

Name of property	Description
<a href="#">“XMSC_WMQ_RESOLVED_QUEUE_MANAGER” on page 222</a>	This property is used to obtain the name of the queue manager to which it is connected.
<a href="#">“XMSC_WMQ_RESOLVED_QUEUE_MANAGER_ID” on page 222</a>	This property is populated with the ID of the queue manager after the connection.
<a href="#">XMSC_WPM_CONNECTION_PROTOCOL</a>	The communications protocol used for the connection to the messaging engine. This property is read-only.
<a href="#">XMSC_WPM_HOST_NAME</a>	The host name or IP address of the system that contains the messaging engine to which the application is connected. This property is read-only.
<a href="#">XMSC_WPM_ME_NAME</a>	The name of the messaging engine to which the application is connected. This property is read-only.

Table 26. Properties of Connection (continued)

Name of property	Description
<a href="#">XMSC_WPM_PORT</a>	The number of the port listened on by the messaging engine to which the application is connected. This property is read-only.

A Connection object also has read-only properties that are derived from the properties of the connection factory that was used to create the connection. These properties are derived not only from the connection factory properties that were set at the time the connection was created, but also from the default values of the properties that were not set. The properties include only the ones that are relevant for the type of messaging server that the application is connected to. The names of the properties are the same as the names of the connection factory properties.

## Properties of ConnectionFactory

An overview of the properties of the ConnectionFactory object, with links to more detailed reference information.

Table 27. Properties of ConnectionFactory

Name of property	Description
<a href="#">“XMSC_ASYNC_EXCEPTIONS” on page 200</a>	This property determines whether XMS informs an ExceptionListener only when a connection is broken, or when any exception occurs asynchronously to an XMS API call. This property applies to all Connections created from this ConnectionFactory that have an ExceptionListener registered.
<a href="#">XMSC_CLIENT_ID</a>	The client identifier for a connection.
<a href="#">XMSC_CONNECTION_TYPE</a>	The type of messaging server to which an application connects.
<a href="#">XMSC_PASSWORD</a>	A password that can be used to authenticate the application when it attempts to connect to a messaging server.
<a href="#">“XMSC_RTT_BROKER_PING_INTERVAL” on page 205</a>	The time interval, in milliseconds, after which XMS .NET checks the connection to a Real Time messaging server to detect any activity.
<a href="#">XMSC_RTT_CONNECTION_PROTOCOL</a>	The communications protocol used for a real-time connection to a broker.
<a href="#">XMSC_RTT_HOST_NAME</a>	The host name or IP address of the system on which a broker runs.
<a href="#">XMSC_RTT_LOCAL_ADDRESS</a>	The host name or IP address of the local network interface to be used for a real-time connection to a broker.
<a href="#">XMSC_RTT_MULTICAST</a>	The multicast setting for a connection factory or destination.
<a href="#">XMSC_RTT_PORT</a>	The number of the port on which a broker listens for incoming requests.
<a href="#">XMSC_USERID</a>	A user identifier that can be used to authenticate the application when it attempts to connect to a messaging server.

Table 27. Properties of ConnectionFactory (continued)

Name of property	Description
<a href="#">XMSC_WMQ_BROKER_CONTROLQ</a>	<p>The name of the control queue used by a broker.</p> <p><b>Note:</b> This property can be used with Version 2.0 of IBM Message Service Client for .NET but has no effect for an application connected to a IBM WebSphere MQ 7.0 queue manager unless the XMSC_WMQ_PROVIDER_VERSION property of the connection factory is set to a version number less than 7.</p>
<a href="#">XMSC_WMQ_BROKER_PUBQ</a>	<p>The name of the queue monitored by a broker where applications send messages that they publish.</p> <p><b>Note:</b> This property can be used with Version 2.0 of IBM Message Service Client for .NET but has no effect for an application connected to a IBM WebSphere MQ 7.0 queue manager unless the XMSC_WMQ_PROVIDER_VERSION property of the connection factory is set to a version number less than 7.</p>
<a href="#">XMSC_WMQ_BROKER_QMGR</a>	<p>The name of the queue manager to which a broker is connected.</p> <p><b>Note:</b> This property can be used with Version 2.0 of IBM Message Service Client for .NET but has no effect for an application connected to a IBM WebSphere MQ 7.0 queue manager unless the XMSC_WMQ_PROVIDER_VERSION property of the connection factory is set to a version number less than 7.</p>
<a href="#">XMSC_WMQ_BROKER_SUBQ</a>	<p>The name of the subscriber queue for a nondurable message consumer.</p> <p><b>Note:</b> This property can be used with Version 2.0 of IBM Message Service Client for .NET but has no effect for an application connected to a IBM WebSphere MQ 7.0 queue manager unless the XMSC_WMQ_PROVIDER_VERSION property of the connection factory is set to a version number less than 7.</p>
<a href="#">XMSC_WMQ_BROKER_VERSION</a>	<p>The type of broker used by the application for a connection or for the destination.</p> <p><b>Note:</b> This property can be used with Version 2.0 of IBM Message Service Client for .NET but has no effect for an application connected to a IBM WebSphere MQ 7.0 queue manager unless the XMSC_WMQ_PROVIDER_VERSION property of the connection factory is set to a version number less than 7.</p>
<a href="#">“XMSC_WMQ_CCDTURL” on page 209</a>	<p>A Uniform Resource Locator (URL) that identifies the name and location of the file that contains the client channel definition table and also specifies how the file can be accessed.</p>
<a href="#">XMSC_WMQ_CHANNEL</a>	<p>The name of the channel to be used for a connection.</p>
<a href="#">“XMSC_WMQ_CLIENT_RECONNECT_OPTIONS” on page 210</a>	<p>This property specifies the client reconnect options for new connections created by this factory</p>

Table 27. Properties of ConnectionFactory (continued)

Name of property	Description
<a href="#">“XMSC_WMQ_CLIENT_RECONNECT_TIMEOUT” on page 210</a>	This property specifies the duration of time, in seconds, that a client connection attempts to reconnect.
<a href="#">XMSC_WMQ_CONNECTION_MODE</a>	The mode by which an application connects to a queue manager.
<a href="#">“XMSC_WMQ_CONNECTION_NAME_LIST” on page 211</a>	This property specifies the hosts to which the client attempts to reconnect to after its connection are broken.
<a href="#">XMSC_WMQ_FAIL_IF QUIESCE</a>	Whether calls to certain methods fail if the queue manager to which the application is connected is in a quiescing state.
<a href="#">XMSC_WMQ_HOST_NAME</a>	The host name or IP address of the system on which a queue manager runs.
<a href="#">XMSC_WMQ_LOCAL_ADDRESS</a>	For a connection to a queue manager, this property specifies the local network interface to be used, or the local port or range of local ports to be used, or both.
<a href="#">XMSC_WMQ_MESSAGE_SELECTION</a>	<p>Determines whether message selection is done by the XMS client or by the broker.</p> <p><b>Note:</b> This property can be used with Version 2.0 of IBM Message Service Client for .NET but has no effect for an application connected to a IBM WebSphere MQ 7.0 queue manager unless the XMSC_WMQ_PROVIDER_VERSION property of the connection factory is set to a version number less than 7.</p>
<a href="#">XMSC_WMQ_MSG_BATCH_SIZE</a>	<p>The maximum number of messages to be retrieved from a queue in one batch when using asynchronous message delivery.</p> <p><b>Note:</b> This property can be used with Version 2.0 of IBM Message Service Client for .NET but has no effect for an application connected to a IBM WebSphere MQ 7.0 queue manager unless the XMSC_WMQ_PROVIDER_VERSION property of the connection factory is set to a version number less than 7.</p>
<a href="#">XMSC_WMQ_POLLING_INTERVAL</a>	<p>If each message listener within a session has no suitable message on its queue, this value is the maximum interval, in milliseconds, that elapses before each message listener tries again to get a message from its queue.</p> <p><b>Note:</b> This property can be used with Version 2.0 of IBM Message Service Client for .NET but has no effect for an application connected to a IBM WebSphere MQ 7.0 queue manager unless the XMSC_WMQ_PROVIDER_VERSION property of the connection factory is set to a version number less than 7.</p>
<a href="#">“XMSC_WMQ_PROVIDER_VERSION” on page 219</a>	The version, release, modification level and fix pack of the queue manager to which the application intends to connect.
<a href="#">XMSC_WMQ_PORT</a>	The number of the port on which a queue manager listens for incoming requests.

Table 27. Properties of ConnectionFactory (continued)

Name of property	Description
<a href="#">XMSC_WMQ_PUB_ACK_INTERVAL</a>	<p>The number of messages published by a publisher before the XMS client requests an acknowledgement from the broker.</p> <p><b>Note:</b> This property can be used with Version 2.0 of IBM Message Service Client for .NET but has no effect for an application connected to a IBM WebSphere MQ 7.0 queue manager unless the XMSC_WMQ_PROVIDER_VERSION property of the connection factory is set to a version number less than 7.</p>
<a href="#">“XMSC_WMQ_PUT_ASYNC_ALLOWED” on page 215</a>	<p>This property determines whether message producers are allowed to use asynchronous puts to send messages to this destination.</p>
<a href="#">XMSC_WMQ_QMGR_CCSID</a>	<p>The identifier (CCSID) of the coded character set, or code page, in which fields of character data defined in the Message Queue Interface (MQI) are exchanged between the XMS client and the WebSphere MQ client.</p>
<a href="#">XMSC_WMQ_QUEUE_MANAGER</a>	<p>The name of the queue manager to connect to.</p>
<a href="#">XMSC_WMQ_RECEIVE_EXIT</a>	<p>Identifies a channel receive exit to be run.</p>
<a href="#">XMSC_WMQ_RECEIVE_EXIT_INIT</a>	<p>The user data that is passed to a channel receive exit when it is called.</p>
<a href="#">XMSC_WMQ_SECURITY_EXIT</a>	<p>Identifies a channel security exit.</p>
<a href="#">XMSC_WMQ_SECURITY_EXIT_INIT</a>	<p>The user data that is passed to a channel security exit when it is called.</p>
<a href="#">“XMSC_WMQ_SEND_CHECK_COUNT” on page 224</a>	<p>The number of send calls to allow between checking for asynchronous put errors, within a single non-transacted XMS session.</p>
<a href="#">XMSC_WMQ_SEND_EXIT</a>	<p>Identifies a channel send exit.</p>
<a href="#">XMSC_WMQ_SEND_EXIT_INIT</a>	<p>The user data that is passed to channel send exits when they are called.</p>
<a href="#">“XMSC_WMQ_SHARE_CONV_ALLOWED” on page 224</a>	<p>Whether a client connection can share its socket with other top-level XMS connections from the same process to the same queue manager, if the channel definitions match. This property is provided to allow complete isolation of Connections in separate sockets if required for application development, maintenance, or operational reasons.</p>
<a href="#">XMSC_WMQ_SSL_CERT_STORES</a>	<p>The locations of the servers that hold the certificate revocation lists (CRLs) to be used on an SSL connection to a queue manager.</p>
<a href="#">XMSC_WMQ_SSL_CIPHER_SPEC</a>	<p>The name of the CipherSpec to be used on a secure connection to a queue manager.</p>
<a href="#">XMSC_WMQ_SSL_CIPHER_SUITE</a>	<p>The name of the CipherSuite to be used on an SSL or TLS connection to a queue manager. The protocol used in negotiating the secure connection depends on the specified CipherSuite.</p>

Table 27. Properties of ConnectionFactory (continued)

Name of property	Description
<a href="#">XMSC_WMQ_SSL_CRYPTO_HW</a>	Configuration details for the cryptographic hardware connected to the client system.
<a href="#">XMSC_WMQ_SSL_FIPS_REQUIRED</a>	The value of this property determines whether an application can or cannot use non-FIPS compliant cipher suites. If this property is set to true, only FIPS algorithms are used for the client-server connection.
<a href="#">XMSC_WMQ_SSL_KEY_REPOSITORY</a>	The location of the key database file in which keys and certificates are stored.
<a href="#">XMSC_WMQ_SSL_KEY_RESETCOUNT</a>	The KeyResetCount represents the total number of unencrypted bytes sent and received within an SSL conversation before the secret key is renegotiated.
<a href="#">XMSC_WMQ_SSL_PEER_NAME</a>	The peer name to be used on an SSL connection to a queue manager.
<a href="#">XMSC_WMQ_SYNCPOINT_ALL_GETS</a>	Whether all messages must be retrieved from queues within sync point control.
"XMSC_WMQ_TARGET_CLIENT" on page 231	
<a href="#">XMSC_WMQ_TEMP_Q_PREFIX</a>	The prefix used to form the name of the WebSphere MQ dynamic queue that is created when the application creates an XMS temporary queue.
<a href="#">XMSC_WMQ_TEMP_TOPIC_PREFIX</a>	When creating temporary topics, XMS generates a topic string of the form "TEMP/TEMPTOPICPREFIX/unique_id", or if this property contains the default value, then this string, "TEMP/unique_id", is generated. Specifying a non-empty value allows specific model queues to be defined for creating the managed queues for subscribers to temporary topics created under this connection.
<a href="#">XMSC_WMQ_TEMPORARY_MODEL</a>	The name of the WebSphere MQ model queue from which a dynamic queue is created when the application creates an XMS temporary queue.
<a href="#">XMSC_WPM_BUS_NAME</a>	For a connection factory, the name of the service integration bus that the application connects to or, for a destination, the name of the service integration bus in which the destination exists.
<a href="#">XMSC_WPM_CONNECTION_PROXIMITY</a>	The connection proximity setting for the connection.
<a href="#">XMSC_WPM_DUR_SUB_HOME</a>	The name of the messaging engine where all durable subscriptions for a connection or a destination are managed.
<a href="#">XMSC_WPM_LOCAL_ADDRESS</a>	For a connection to a service integration bus, this property specifies the local network interface to be used, or the local port or range of local ports to be used, or both.
<a href="#">XMSC_WPM_NON_PERSISTENT_MAP</a>	The reliability level of nonpersistent messages that are sent using the connection.

Table 27. Properties of ConnectionFactory (continued)

Name of property	Description
<a href="#">XMSC_WPM_PERSISTENT_MAP</a>	The reliability level of persistent messages that are sent using the connection.
<a href="#">XMSC_WPM_PROVIDER_ENDPOINTS</a>	A sequence of one or more endpoint addresses of bootstrap servers.
<a href="#">XMSC_WPM_TARGET_GROUP</a>	The name of a target group of messaging engines.
<a href="#">XMSC_WPM_TARGET_SIGNIFICANCE</a>	The significance of the target group of messaging engines.
<a href="#">XMSC_WPM_TARGET_TRANSPORT_CHAIN</a>	The name of the inbound transport chain that the application must use to connect to a messaging engine.
<a href="#">XMSC_WPM_TARGET_TYPE</a>	The type of the target group of messaging engines.
<a href="#">XMSC_WPM_TEMP_Q_PREFIX</a>	The prefix used to form the name of the temporary queue that is created in the service integration bus when the application creates an XMS temporary queue.
<a href="#">XMSC_WPM_TEMP_TOPIC_PREFIX</a>	The prefix used to form the name of a temporary topic that is created by the application.

### Related concepts

#### [ConnectionFactory and Connection objects](#)

A ConnectionFactory object provides a template that an application uses to create a Connection object. The application uses the Connection object to create a Session object.

#### [Connection to a WebSphere service integration bus](#)

An XMS application can connect to a WebSphere Service Integration Bus either by using a direct TCP/IP connection or by using HTTP over TCP/IP.

#### [Secure connections to a WebSphere MQ queue manager](#)

To enable an XMS .NET application to make secure connections to a WebSphere MQ queue manager, the relevant properties must be defined in the ConnectionFactory object.

#### [Secure connections to a WebSphere Service Integration Bus messaging engine](#)

To enable an XMS application to make secure connections to a WebSphere Service Integration Bus messaging engine, the relevant properties must be defined in the ConnectionFactory object.

#### [Property mapping for administered objects](#)

To enable applications to use WebSphere MQ JMS and WebSphere Application Server connection factory and destination object definitions, the properties retrieved from these definitions must be mapped on to the corresponding XMS properties that can be set on XMS connection factories and destinations.

### Related tasks

#### [Creating administered objects](#)

The ConnectionFactory and Destination object definitions that XMS applications require to make a connection to a messaging server must be created using the appropriate administrative tools.

### Related reference

#### [Required properties for administered ConnectionFactory objects](#)

When an application creates a connection factory, a number of properties must be defined to create a connection to a messaging server.

## Properties of ConnectionMetaData

An overview of the properties of the ConnectionMetaData object, with links to more detailed reference information.

Name of property	Description
<a href="#">XMSC_JMS_MAJOR_VERSION</a>	The major version number of the JMS specification upon which XMS is based. This property is read-only.
<a href="#">XMSC_JMS_MINOR_VERSION</a>	The minor version number of the JMS specification upon which XMS is based. This property is read-only.
<a href="#">XMSC_JMS_VERSION</a>	The version identifier of the JMS specification upon which XMS is based. This property is read-only.
<a href="#">XMSC_MAJOR_VERSION</a>	The version number of the XMS client. This property is read-only.
<a href="#">XMSC_MINOR_VERSION</a>	The release number of the XMS client. This property is read-only.
<a href="#">XMSC_PROVIDER_NAME</a>	The provider of the XMS client. This property is read-only.
<a href="#">XMSC_VERSION</a>	The version identifier of the XMS client. This property is read-only.

## Properties of Destination

An overview of the properties of the Destination object, with links to more detailed reference information.

Name of property	Description
<a href="#">XMSC_DELIVERY_MODE</a>	The delivery mode of messages sent to the destination.
<a href="#">XMSC_PRIORITY</a>	The priority of messages sent to the destination.
<a href="#">XMSC_RTT_MULTICAST</a>	The multicast setting for a connection factory or destination.
<a href="#">XMSC_TIME_TO_LIVE</a>	The time to live for messages sent to the destination.
<a href="#">XMSC_WMQ_BROKER_VERSION</a>	The type of broker used by the application for a connection or for the destination.
<a href="#">XMSC_WMQ_CCSD</a>	The identifier (CCSID) of the coded character set, or code page, that the strings of character data in the body of a message are in when the XMS client forwards the message to the destination.

Table 29. Properties of Destination (continued)

Name of property	Description
<a href="#">XMSC_WMQ_DUR_SUBQ</a>	<p>The name of the subscriber queue for a durable subscriber that is receiving messages from the destination.</p> <p><b>Note:</b> This property can be used with Version 2.0 of IBM Message Service Client for .NET but has no effect for an application connected to a IBM WebSphere MQ 7.0 queue manager unless the XMSC_WMQ_PROVIDER_VERSION property of the connection factory is set to a version number less than 7.</p>
<a href="#">XMSC_WMQ_ENCODING</a>	<p>How numerical data in the body of a message is represented when the XMS client forwards the message to the destination.</p>
<a href="#">XMSC_WMQ_FAIL_IF QUIESCE</a>	<p>Whether calls to certain methods fail if the queue manager to which the application is connected is in a quiescing state.</p>
<a href="#">“XMSC_WMQ_MESSAGE_BODY” on page 213</a>	<p>This property determines whether an XMS application processes the MQRFH2 of a IBM WebSphere MQ message as part of the message payload (that is, as part of the message body).</p>
<a href="#">“XMSC_WMQ_MQMD_MESSAGE_CONTEXT” on page 214</a>	<p>Determines what level of message context is to be set by the XMS application. The application must be running with appropriate context authority for this property to take effect.</p>
<a href="#">“XMSC_WMQ_MQMD_READ_ENABLED” on page 214</a>	<p>This property determines whether an XMS application can extract the values of MQMD fields or not.</p>
<a href="#">“XMSC_WMQ_MQMD_WRITE_ENABLED” on page 215</a>	<p>This property determines whether an XMS application can the values of MQMD fields or not.</p>
<a href="#">“XMSC_WMQ_READ_AHEAD_ALLOWED” on page 216</a>	<p>This property determines whether message consumers and queue browsers are allowed to use read ahead to get non-persistent, non-transactional messages from this destination into an internal buffer before receiving them.</p>
<a href="#">“XMSC_WMQ_READ_AHEAD_CLOSE_POLICY” on page 216</a>	<p>This property determines, for messages being delivered to an asynchronous message listener, what happens to messages in the internal read ahead buffer when the message consumer is closed.</p>
<a href="#">“XMSC_WMQ_RECEIVE_CCSD” on page 221</a>	<p>Destination property that sets the target CCSID for queue manager message conversion. The value is ignored unless XMSC_WMQ_RECEIVE_CONVERSION is set to WMQ_RECEIVE_CONVERSION_QMGR.</p>
<a href="#">“XMSC_WMQ_RECEIVE_CONVERSION” on page 221</a>	<p>Destination property that determines whether data conversion is going to be performed by the queue manager.</p>
<a href="#">XMSC_WMQ_TARGET_CLIENT</a>	<p>Whether messages sent to the destination contain an MQRFH2 header.</p>

Table 29. Properties of Destination (continued)

Name of property	Description
<u>XMSC_WMQ_TEMP_TOPIC_PREFIX</u>	When creating temporary topics, XMS generates a topic string of the form "TEMP/TEMPTOPICPREFIX/unique_id", or if this property contains the default value, then this string, "TEMP/unique_id", is generated. Specifying a non-empty value allows specific model queues to be defined for creating the managed queues for subscribers to temporary topics created under this connection.
<u>XMSC_WPM_BUS_NAME</u>	For a connection factory, the name of the service integration bus that the application connects to or, for a destination, the name of the service integration bus in which the destination exists.
<u>XMSC_WPM_TOPIC_SPACE</u>	The name of the topic space that contains the topic.

**Related concepts**

ConnectionFactory and Connection objects

A ConnectionFactory object provides a template that an application uses to create a Connection object. The application uses the Connection object to create a Session object.

Connection to a WebSphere service integration bus

An XMS application can connect to a WebSphere Service Integration Bus either by using a direct TCP/IP connection or by using HTTP over TCP/IP.

Destinations

An XMS application uses a Destination object to specify the destination of messages that are being sent, and the source of messages that are being received.

Destination wildcards

XMS provides support for destination wildcards, ensuring that wildcards can be passed through to the place where they are needed for matching. There is a different wildcard scheme for each server type that XMS can work with.

Topic uniform resource identifiers

The topic uniform resource identifier (URI) specifies the name of the topic; it can also specify one or more properties for it.

Queue uniform resource identifiers

The URI for a queue specifies the name of the queue; it can also specify one or more properties of the queue.

Temporary destinations

XMS applications can create and use temporary destinations.

Property mapping for administered objects

To enable applications to use WebSphere MQ JMS and WebSphere Application Server connection factory and destination object definitions, the properties retrieved from these definitions must be mapped on to the corresponding XMS properties that can be set on XMS connection factories and destinations.

**Related tasks**

Creating administered objects

The ConnectionFactory and Destination object definitions that XMS applications require to make a connection to a messaging server must be created using the appropriate administrative tools.

**Related reference**

Required properties for administered Destination objects

An application that is creating a destination must set several properties that the application on an administered Destination object.

## Properties of InitialContext

An overview of the properties of the InitialContext object, with links to more detailed reference information.

Name of property	Description
<a href="#">XMSC_IC_PROVIDER_URL</a>	Used to locate the JNDI naming directory so that the COS naming service does not need to be on the same server as the web service.
<a href="#">XMSC_IC_SECURITY_AUTHENTICATION</a>	Based on the Java Context interface SECURITY_AUTHENTICATION. This property is only applicable to the COS naming context.
<a href="#">XMSC_IC_SECURITY_CREDENTIALS</a>	Based on the Java Context interface SECURITY_CREDENTIALS. This property is only applicable to the COS naming context.
<a href="#">XMSC_IC_SECURITY_PRINCIPAL</a>	Based on the Java Context interface SECURITY_PRINCIPAL. This property is only applicable to the COS naming context.
<a href="#">XMSC_IC_SECURITY_PROTOCOL</a>	Based on the Java Context interface SECURITY_PROTOCOL. This property is only applicable to the COS naming context.
<a href="#">XMSC_IC_URL</a>	For LDAP and FileSystem contexts, the address of the repository containing administered objects. For COS naming contexts, the address of the web service that looks up the objects in the directory.

### Related concepts

#### [InitialContext properties](#)

The parameters of the InitialContext constructor include the location of the repository of administered objects, given as a uniform resource indicator (URI). In order for an application to establish a connection to the repository, it may be necessary to provide more information than the information contained in the URI.

#### [URI format for XMS initial contexts](#)

The location of the repository of administered objects is provided as a uniform resource indicator (URI). The format of the URI depends on the context type.

#### [Retrieval of administered objects](#)

XMS retrieves an administered object from the repository using the address provided when the InitialContext object is created, or in the InitialContext properties.

### Related tasks

#### [InitialContext objects](#)

An application must create an initial context to be used to make a connection to the administered objects repository to retrieve the required administered objects.

## Properties of Message

An overview of the properties of the Message object, with links to more detailed reference information.

Name of property	Description
<a href="#">JMS_IBM_CHARACTER_SET</a>	The identifier (CCSID) of the coded character set, or code page, that the strings of character data in the body of the message is in when the XMS client forwards the message to its intended destination. In XMS this property has a numeric value and maps to CCSID. However, this property is based on a JMS property so has a string type value and maps to the Java character set that represents this numeric CCSID.
<a href="#">JMS_IBM_ENCODING</a>	How numerical data in the body of the message is represented when the XMS client forwards the message to its intended destination.
<a href="#">JMS_IBM_EXCEPTIONMESSAGE</a>	Text that describes why the message was sent to the exception destination. This property is read-only.
<a href="#">JMS_IBM_EXCEPTIONPROBLEMDESTINATION</a>	The name of the destination that the message was at before the message was sent to the exception destination.
<a href="#">JMS_IBM_EXCEPTIONREASON</a>	A reason code indicating the reason why the message was sent to the exception destination.
<a href="#">JMS_IBM_EXCEPTIONTIMESTAMP</a>	The time when the message was sent to the exception destination.
<a href="#">JMS_IBM_FEEDBACK</a>	A code that indicates the nature of a report message.
<a href="#">JMS_IBM_FORMAT</a>	The nature the application data in the message.
<a href="#">JMS_IBM_LAST_MSG_IN_GROUP</a>	Indicate whether the message is the last message in a message group.
<a href="#">JMS_IBM_MSGTYPE</a>	The type of the message.
<a href="#">JMS_IBM_PUTAPPLTYPE</a>	The type of application that sent the message.
<a href="#">JMS_IBM_PUTDATE</a>	The date when the message was sent.
<a href="#">JMS_IBM_PUTTIME</a>	The time when the message was sent.
<a href="#">JMS_IBM_REPORT_COA</a>	Request 'confirm on arrival' report messages, specifying how much application data from the original message must be included in a report message.
<a href="#">JMS_IBM_REPORT_COD</a>	Request 'confirm on delivery' report messages, specifying how much application data from the original message must be included in a report message.
<a href="#">JMS_IBM_REPORT_DISCARD_MSG</a>	Request that the message is discarded if it cannot be delivered to its intended destination.
<a href="#">JMS_IBM_REPORT_EXCEPTION</a>	Request exception report messages, specifying how much application data from the original message must be included in a report message.

<i>Table 31. Properties of Message (continued)</i>	
<b>Name of property</b>	<b>Description</b>
<u>JMS_IBM_REPORT_EXPIRATION</u>	Request expiration report messages, specifying how much application data from the original message must be included in a report message.
<u>JMS_IBM_REPORT_NAN</u>	Request negative action notification report messages.
<u>JMS_IBM_REPORT_PAN</u>	Request positive action notification report messages.
<u>JMS_IBM_REPORT_PASS_CORREL_ID</u>	Request that the correlation identifier of any report or reply message is the same as the correlation identifier of the original message.
<u>JMS_IBM_REPORT_PASS_MSG_ID</u>	Request that the message identifier of any report or reply message is the same as the message identifier of the original message.
<u>JMS_IBM_RETAIN</u>	Setting this property indicates to the queue manager to treat a message as Retained Publication.
<u>JMS_IBM_SYSTEM_MESSAGEID</u>	An identifier that identifies the message uniquely within the service integration bus. This property is read-only.
<u>JMSX_APPID</u>	The name of the application that sent the message.
<u>JMSX_DELIVERY_COUNT</u>	The number of attempts to deliver the message.
<u>JMSX_GROUPID</u>	The identifier of the message group to which the message belongs.
<u>JMSX_GROUPSEQ</u>	The sequence number of the message within a message group.
<u>JMSX_USERID</u>	The user identifier associated with the application that sent the message.

### **JMS\_IBM\_MQMD\* properties**

IBM Message Service Client for .NET enables client applications to read/write MQMD fields using APIs. It also allows access to MQ message data. By default access to MQMD is disabled and must be enabled explicitly by the application using Destination properties XMSC\_WMQ\_MQMD\_WRITE\_ENABLED and XMSC\_WMQ\_MQMD\_READ\_ENABLED. These two properties are independent of each other.

All MQMD fields except StrucId and Version are exposed as additional Message object properties and are prefixed JMS\_IBM\_MQMD.

JMS\_IBM\_MQMD\* properties take higher precedence over other properties like JMS\_IBM\* described in the previous table.

### **Sending messages**

All MQMD fields except StrucId and Version are represented. These properties refer only to the MQMD fields; where a property occurs both in the MQMD and in the MQRFH2 header, the version in the MQRFH2 is not set or extracted. Any of these properties can be set, except JMS\_IBM\_MQMD\_BackoutCount. Any value set for JMS\_IBM\_MQMD\_BackoutCount is ignored.

If a property has a maximum length and you supply a value that is too long, the value is truncated.

For certain properties, you must also set the XMSC\_WMQ\_MQMD\_MESSAGE\_CONTEXT property on the Destination object. The application must be running with appropriate context authority for this property to take effect. If you do not set XMSC\_WMQ\_MQMD\_MESSAGE\_CONTEXT to an appropriate value, the

property value is ignored. If you set XMSC\_WMQ\_MQMD\_MESSAGE\_CONTEXT to an appropriate value but you do not have sufficient context authority for the queue manager, an exception is issued. Properties requiring specific values of XMSC\_WMQ\_MQMD\_MESSAGE\_CONTEXT are as follows.

The following properties require XMSC\_WMQ\_MQMD\_MESSAGE\_CONTEXT to be set to XMSC\_WMQ\_MDCTX\_SET\_IDENTITY\_CONTEXT or XMSC\_WMQ\_MDCTX\_SET\_ALL\_CONTEXT:

- JMS\_IBM\_MQMD\_UserIdentifier
- JMS\_IBM\_MQMD\_AccountingToken
- JMS\_IBM\_MQMD\_ApplIdentityData

The following properties require XMSC\_WMQ\_MQMD\_MESSAGE\_CONTEXT to be set to XMSC\_WMQ\_MDCTX\_SET\_ALL\_CONTEXT:

- JMS\_IBM\_MQMD\_PutApplType
- JMS\_IBM\_MQMD\_PutApplName
- JMS\_IBM\_MQMD\_PutDate
- JMS\_IBM\_MQMD\_PutTime
- JMS\_IBM\_MQMD\_ApplOriginData

## Receiving messages

All these properties are available on a received message if XMSC\_WMQ\_MQMD\_READ\_ENABLED property is set to true, irrespective of the actual properties that the producing application set. An application cannot modify the properties of a received message unless all properties are cleared first, according to the JMS specification. The received message can be forwarded without modifying the properties.

**Note:** If your application receives a message from a destination with XMSC\_WMQ\_MQMD\_READ\_ENABLED property set to true, and forwards it to a destination with XMSC\_WMQ\_MQMD\_WRITE\_ENABLED set to true, this results in all the MQMD field values of the received message being copied into the forwarded message. Table of properties

Property	Description	Type
JMS_IBM_MQMD_REPORT	Options for report messages	System.Int32
JMS_IBM_MQMD_MSGTYPE	Message type	System.Int32
JMS_IBM_MQMD_EXPIRY	message lifetime	System.Int32
JMS_IBM_MQMD_FEEDBACK	Feedback or reason code	System.Int32
JMS_IBM_MQMD_ENCODING	Numeric encoding of message data	System.Int32
JMS_IBM_MQMD_CODEDCHARSETID	Character set identifier of message data	System.Int32
JMS_IBM_MQMD_FORMAT	Format name of message data	System.String
JMS_IBM_MQMD_PRIORITY	Message priority	System.Int32
	<b>Note:</b> If you assign a value to JMS_IBM_MQMD_PRIORITY that is not within the range 0-9, this value violates the JMS specification.	
JMS_IBM_MQMD_PERSISTENCE	Message persistence	System.Int32

Table 32. Properties of the Message object representing the MQMD fields (continued)

Property	Description	Type
JMS_IBM_MQMD_MSGID <b>Note:</b> The JMS specification states that the message ID must be set by the JMS provider and that it must either be unique or null. If you assign a value to JMS_IBM_MQMD_MSGID, this value is copied to the JMSMessageID. Thus it is not set by the JMS provider and might not be unique: this value violates the JMS specification.	Message identifier	Byte Array <b>Note:</b> The use of byte array properties on a message violates the JMS specification.
JMS_IBM_MQMD_CORRELID <b>Note:</b> If you assign a value to JMS_IBM_MQMD_CORRELID that starts with the string 'ID:', this value violates the JMS specification.	Correlation identifier	Byte Array <b>Note:</b> The use of byte array properties on a message violates the JMS specification.
JMS_IBM_MQMD_BACKOUTCOUNT	Backout counter	System.Int32
JMS_IBM_MQMD_REPLYTOQ	Name of reply queue	System.String
JMS_IBM_MQMD_REPLYTOQMGR	Name of reply queue manager	System.String
JMS_IBM_MQMD_USERIDENTIFIER	User identifier	System.String
JMS_IBM_MQMD_ACCOUNTINGTOKEN	Accounting token	Byte Array <b>Note:</b> The use of byte array properties on a message violates the JMS specification.
JMS_IBM_MQMD_APPLIDENTITYDATA	Application data relating to identity	System.String
JMS_IBM_MQMD_PUTAPPLTYPE	Type of application that put the message	System.Int32
JMS_IBM_MQMD_PUTAPPLNAME	Name of the application that put the message	System.String
JMS_IBM_MQMD_PUTDATE	Date when message was put	System.String
JMS_IBM_MQMD_PUTTIME	Time when message was put	System.String
JMS_IBM_MQMD_APPLORIGINDATA	Application data relating to origin	System.String
JMS_IBM_MQMD_GROUPID	Group identifier	Byte Array <b>Note:</b> The use of byte array properties on a message violates the JMS specification.
JMS_IBM_MQMD_MSGSEQNUMBER	Sequence number of local message within group	System.Int32
JMS_IBM_MQMD_OFFSET	Offset of data in physical message from start of logical message	System.Int32

Table 32. Properties of the Message object representing the MQMD fields (continued)

Property	Description	Type
JMS_IBM_MQMD_MSGFLAGS	Message flags	System.Int32
JMS_IBM_MQMD_ORIGINALLENGTH	Length of original message	System.Int32

See [MQMD](#) for further details.

## Examples

This example results in a message being put to a queue or topic with MQMD.UserIdentifier set to "JoeBloggs".

```
// Create a ConnectionFactory, connection, session, producer, message
// ...

// Create a destination
// ...

// Enable MQMD write
dest.setBooleanProperty(XMSC_WMQ_MQMD_WRITE_ENABLED,
    XMSC_WMQ_MQMD_WRITE_ENABLED_YES);

// Optionally, set a message context if applicable for this MD field
dest.setIntProperty(XMSC_WMQ_MQMD_MESSAGE_CONTEXT,
    XMSC_WMQ_MDCTX_SET_IDENTITY_CONTEXT);

// On the message, set property to provide custom UserId
msg.setStringProperty(JMS_IBM_MQMD_USERIDENTIFIER, "JoeBloggs");

// Send the message
// ...
```

It is necessary to set XMSC\_WMQ\_MQMD\_MESSAGE\_CONTEXT before setting JMS\_IBM\_MQMD\_USERIDENTIFIER. For more information about the use of XMSC\_WMQ\_MQMD\_MESSAGE\_CONTEXT, see Message object properties.

Similarly, you can extract the contents of the MQMD fields by setting XMSC\_WMQ\_MQMD\_READ\_ENABLED to true before receiving a message and then using the get methods of the message, such as getStringProperty. Any properties received are read-only.

This example results in the value field holding the value of the MQMD.ApplIdentityData field of a message got from a queue or a topic.

```
// Create a ConnectionFactory, connection, session, consumer
// ...

// Create a destination
// ...

// Enable MQMD read
dest.setBooleanProperty(XMSC_WMQ_MQMD_READ_ENABLED, XMSC_WMQ_MQMD_READ_ENABLED_YES);

// Receive a message
// ...

// Get desired MQMD field value using a property
System.String value = rcvMsg.getStringProperty(JMS_IBM_MQMD_APPLIDENTITYDATA);
```

## Properties of MessageConsumer

An overview of the properties of the MessageConsumer object, with links to more detailed reference information.

Name of property	Description
<code>XMSC_IS_SUBSCRIPTION_MULTICAST</code>	Indicates whether messages are being delivered to the message consumer using WebSphere MQ Multicast Transport. This property is read-only.
<code>XMSC_IS_SUBSCRIPTION_RELIABLE_MULTICAST</code>	Indicates whether messages are being delivered to the message consumer using WebSphere MQ Multicast Transport with a reliable quality of service. This property is read-only.

Refer to [.NET properties of IMessageConsumer](#) for more details.

## Properties of MessageProducer

An overview of the properties of the MessageProducer object, with links to more detailed reference information.

See [.NET properties of IMessageProducer](#) for more details.

## Properties of Session

An overview of the properties of the Session object, with links to more detailed reference information.

See [.NET properties of ISession](#) for more details.

## Property definitions

This section provides a definition of each object property.

Each property definition includes the following information:

- The data type of the property
- The types of object that have the property
- For a property of Destination, the name that can be used in a uniform resource identifier (URI)
- A more detailed description of the property
- The valid values of the property
- The default value of the property

Properties whose names commence with one of the following prefixes are relevant only for the specified type of connection:

### **XMSC\_RTT**

The properties are relevant only for a real-time connection to a broker. The names of the properties are defined as named constants in the header file `xmsc_rtt.h`.

### **XMSC\_WMQ**

The properties are relevant only when an application connects to a WebSphere MQ queue manager. The names of the properties are defined as named constants in the header file `xmsc_wmq.h`.

### **XMSC\_WPM**

The properties are relevant only when an application connects to a WebSphere service integration bus. The names of the properties are defined as named constants in the header file `xmsc_wpm.h`.

Unless stated otherwise in their definitions, the remaining properties are relevant for all types of connection. The names of the properties are defined as named constants in the header file `xmsc.h`.

Properties whose names commence with the prefix JMSX are JMS defined properties of a message, and properties whose names commence with the prefix JMS\_IBM are IBM defined properties of a message. For more information about the properties of messages, see [“Properties of an XMS message” on page 70](#).

Unless stated otherwise in its definition, each property is relevant in both the point-to-point and publish/subscribe domains.

An application can get and set the value of any property, unless the property is designated as read-only.

### **JMS\_IBM\_CHARACTER\_SET**

**Data type:**

System.Int32

**Property of:**

Message

The identifier (CCSID) of the coded character set, or code page, that the strings of character data in the body of the message is in when the XMS client forwards the message to its intended destination. In XMS this property has a numeric value and maps to CCSID. However, this property is based on a JMS property so has a string type value and maps to the Java character set that represents this numeric CCSID. This property overrides any CCSID specified for the destination by the [XMSC\\_WMQ\\_CCSID](#) property.

By default, the property is not set.

This property is not relevant when an application connects to a service integration bus.

### **JMS\_IBM\_ENCODING**

**Data type:**

System.Int32

**Property of:**

Message

How numerical data in the body of the message is represented when the XMS client forwards the message to its intended destination. This property overrides any encoding specified for the destination by the [XMSC\\_WMQ\\_ENCODING](#) property. The property specifies the representation of binary integers, packed decimal integers, and floating point numbers.

The valid values of the property are the same as the values that can be specified in the [Encoding](#) field of a message descriptor.

An application can use the following named constants to set the property:

<b>Named constant</b>	<b>Meaning</b>
MQENC_INTEGER_NORMAL	Normal integer encoding
MQENC_INTEGER_REVERSED	Reversed integer encoding
MQENC_DECIMAL_NORMAL	Normal packed decimal encoding
MQENC_DECIMAL_REVERSED	Reversed packed decimal encoding
MQENC_FLOAT_IEEE_NORMAL	Normal IEEE floating point encoding
MQENC_FLOAT_IEEE_REVERSED	Reversed IEEE floating point encoding
MQENC_FLOAT_S390	z/OS® architecture floating point encoding
MQENC_NATIVE	Native machine encoding

To form a value for the property, the application can add three of these constants as follows:

- A constant whose name commences with MQENC\_INTEGER, to specify the representation of binary integers

- A constant whose name commences with MQENC\_DECIMAL, to specify the representation of packed decimal integers
- A constant whose name commences with MQENC\_FLOAT, to specify the representation of floating point numbers

Alternatively, the application can set the property to MQENC\_NATIVE, whose value is environment-dependent.

By default, the property is not set.

This property is not relevant when an application connects to a service integration bus.

### ***JMS\_IBM\_EXCEPTIONMESSAGE***

**Data type:**

String

**Property of:**

Message

Text that describes why the message was sent to the exception destination. This property is read-only.

This property is relevant only when an application connects to a service integration bus and receives a message from an exception destination.

### ***JMS\_IBM\_EXCEPTIONPROBLEMDESTINATION***

**Data type:**

String

**Property of:**

Message

The name of the destination that the message was at before the message was sent to the exception destination.

This property is relevant only when an application connects to a service integration bus and receives a message from an exception destination.

### ***JMS\_IBM\_EXCEPTIONREASON***

**Data type:**

System.Int32

**Property of:**

Message

A reason code indicating the reason why the message was sent to the exception destination.

This property is relevant only when an application connects to a service integration bus and receives a message from an exception destination.

### ***JMS\_IBM\_EXCEPTIONTIMESTAMP***

**Data type:**

System.Int64

**Property of:**

Message

The time when the message was sent to the exception destination.

The time is expressed in milliseconds since 00:00:00 GMT on the 1 January 1970.

This property is relevant only when an application connects to a service integration bus and receives a message from an exception destination.

## ***JMS\_IBM\_FEEDBACK***

**Data type:**  
System.Int32

**Property of:**  
Message

A code that indicates the nature of a report message.

The valid values of the property are the feedback codes and reason codes that can be specified in the **Feedback** field of a message descriptor.

By default, the property is not set.

## ***JMS\_IBM\_FORMAT***

**Data type:**  
String

**Property of:**  
Message

The nature the application data in the message.

The valid values of the property are the same as the values that can be specified in the **Format** field of a message descriptor.

By default, the property is not set.

This property is not relevant when an application connects to a service integration bus.

## ***JMS\_IBM\_LAST\_MSG\_IN\_GROUP***

**Data type:**  
System.Boolean

**Property of:**  
Message

Indicate whether the message is the last message in a message group.

Set the property to true if the message is the last message in a message group. Otherwise, set the property to false, or do not set the property. By default, the property is not set.

The value true corresponds to the status flag MQMF\_LAST\_MSG\_IN\_GROUP, which can be specified in the **MsgFlags** field of a message descriptor. .

This property is ignored in the publish/subscribe domain and is not relevant when an application connects to a service integration bus.

## ***JMS\_IBM\_MSGTYPE***

**Data type:**  
System.Int32

**Property of:**  
Message

The type of the message.

The valid values of the property are as follows:

<b>Valid value</b>	<b>Meaning</b>
MQMT_DATAGRAM	The message is one that does not require a reply.
MQMT_REQUEST	The message is one that requires a reply.
MQMT_REPLY	The message is a reply message.

**Valid value**

MQMT\_REPORT

**Meaning**

The message is a report message.

These values correspond to the message types that can be specified in the **MsgType** field of a message descriptor.

By default, the property is not set.

This property is not relevant when an application connects to a service integration bus.

***JMS\_IBM\_PUTAPPLTYPE*****Data type:**

System.Int32

**Property of:**

Message

The type of application that sent the message.

The valid values of the property are the application types that can be specified in the **PutApp1Type** field of a message descriptor.

By default, the property is not set.

This property is not relevant when an application connects to a service integration bus.

***JMS\_IBM\_PUTDATE*****Data type:**

String

**Property of:**

Message

The date when the message was sent.

The valid values of the property are the same as the values that can be specified in the **PutDate** field of a message descriptor.

By default, the property is not set.

This property is not relevant when an application connects to a service integration bus.

***JMS\_IBM\_PUTTIME*****Data type:**

String

**Property of:**

Message

The time when the message was sent.

The valid values of the property are the same as the values that can be specified in the **PutTime** field of a message descriptor.

By default, the property is not set.

This property is not relevant when an application connects to a service integration bus.

***JMS\_IBM\_REPORT\_COA*****Data type:**

System.Int32

**Property of:**

Message

Request 'confirm on arrival' report messages, specifying how much application data from the original message must be included in a report message.

The valid values of the property are as follows:

<b>Valid value</b>	<b>Meaning</b>
MQRO_COA	Request 'confirm on arrival' report messages, with no application data from the original message included in a report message.
MQRO_COA_WITH_DATA	Request 'confirm on arrival' report messages, with the first 100 bytes of application data from the original message included in a report message.
MQRO_COA_WITH_FULL_DATA	Request 'confirm on arrival' report messages, with all the application data from the original message included in a report message.

These values correspond to report options that can be specified in the **Report** field of a message descriptor. For more information about these options, see [Report \(MQLONG\)](#).

By default, the property is not set.

### ***JMS\_IBM\_REPORT\_COD***

**Data type:**  
System.Int32

**Property of:**  
Message

Request 'confirm on delivery' report messages, specifying how much application data from the original message must be included in a report message.

The valid values of the property are as follows:

<b>Valid value</b>	<b>Meaning</b>
MQRO_COD	Request 'confirm on delivery' report messages, with no application data from the original message included in a report message.
MQRO_COD_WITH_DATA	Request 'confirm on delivery' report messages, with the first 100 bytes of application data from the original message included in a report message.
MQRO_COD_WITH_FULL_DATA	Request 'confirm on delivery' report messages, with all the application data from the original message included in a report message.

These values correspond to report options that can be specified in the **Report** field of a message descriptor.

By default, the property is not set.

### ***JMS\_IBM\_REPORT\_DISCARD\_MSG***

**Data type:**  
System.Int32

**Property of:**  
Message

Request that the message is discarded if it cannot be delivered to its intended destination.

Set the property to MQRO\_DISCARD\_MSG to request that the message is discarded if it cannot be delivered to its intended destination. If you require the message to be put on a dead letter queue instead, or sent to an exception destination, do not set the property. By default, the property is not set.

The value MQRO\_DISCARD\_MSG corresponds to a report option that can be specified in the **Report** field of a message descriptor.

### **JMS\_IBM\_REPORT\_EXCEPTION**

**Data type:**  
System.Int32

**Property of:**  
Message

Request exception report messages, specifying how much application data from the original message must be included in a report message.

The valid values of the property are as follows:

<b>Valid value</b>	<b>Meaning</b>
MQRO_EXCEPTION	Request exception report messages, with no application data from the original message included in a report message.
MQRO_EXCEPTION_WITH_DATA	Request exception report messages, with the first 100 bytes of application data from the original message included in a report message.
MQRO_EXCEPTION_WITH_FULL_DATA	Request exception report messages, with all the application data from the original message included in a report message.

These values correspond to report options that can be specified in the **Report** field of a message descriptor.

By default, the property is not set.

### **JMS\_IBM\_REPORT\_EXPIRATION**

**Data type:**  
System.Int32

**Property of:**  
Message

Request expiration report messages, specifying how much application data from the original message must be included in a report message.

The valid values of the property are as follows:

<b>Valid value</b>	<b>Meaning</b>
MQRO_EXPIRATION	Request expiration report messages, with no application data from the original message included in a report message.
MQRO_EXPIRATION_WITH_DATA	Request expiration report messages, with the first 100 bytes of application data from the original message included in a report message.
MQRO_EXPIRATION_WITH_FULL_DATA	Request expiration report messages, with all the application data from the original message included in a report message.

These values correspond to report options that can be specified in the **Report** field of a message descriptor.

By default, the property is not set.

### ***JMS\_IBM\_REPORT\_NAN***

**Data type:**  
System.Int32

**Property of:**  
Message

Request negative action notification report messages.

Set the property to MQRO\_NAN to request negative action notification report messages. If you do not require negative action notification report messages, do not set the property. By default, the property is not set.

The value MQRO\_NAN corresponds to a report option that can be specified in the **Report** field of a message descriptor.

### ***JMS\_IBM\_REPORT\_PAN***

**Data type:**  
System.Int32

**Property of:**  
Message

Request positive action notification report messages.

Set the property to MQRO\_PAN to request positive action notification report messages. If you do not require positive action notification report messages, do not set the property. By default, the property is not set.

The value MQRO\_PAN corresponds to a report option that can be specified in the **Report** field of a message descriptor.

### ***JMS\_IBM\_REPORT\_PASS\_CORREL\_ID***

**Data type:**  
System.Int32

**Property of:**  
Message

Request that the correlation identifier of any report or reply message is the same as the correlation identifier of the original message.

The valid values of the property are as follows:

<b>Valid value</b>	<b>Meaning</b>
MQRO_PASS_CORREL_ID	Request that the correlation identifier of any report or reply message is the same as the correlation identifier of the original message.
MQRO_COPY_MSG_ID_TO_CORREL_ID	Request that the correlation identifier of any report or reply message is the same as the message identifier of the original message.

These values correspond to report options that can be specified in the **Report** field of a message descriptor. .

The default value of the property is MQRO\_COPY\_MSG\_ID\_TO\_CORREL\_ID.

## ***JMS\_IBM\_REPORT\_PASS\_MSG\_ID***

**Data type:**

System.Int32

**Property of:**

Message

Request that the message identifier of any report or reply message is the same as the message identifier of the original message.

The valid values of the property are as follows:

<b>Valid value</b>	<b>Meaning</b>
MQRO_PASS_MSG_ID	Request that the message identifier of any report or reply message is the same as the message identifier of the original message.
MQRO_NEW_MSG_ID	Request that a new message identifier is generated for each report or reply message.

These values correspond to report options that can be specified in the [Report](#) field of a message descriptor.

The default value of the property is MQRO\_NEW\_MSG\_ID.

## ***JMS\_IBM\_RETAIN***

**Data type:**

System.Int32

**Property of:**

Message

Setting this property indicates to the queue manager to treat a message as Retained Publication. When a subscriber receives messages from topics, it might receive additional messages immediately after subscribing, beyond the messages received in previous releases. These messages are the optional retained publications for the topics subscribed. For each topic matching the subscription, if there is a retained publication the publication is made available for delivery to the subscribing message consumer.

RETAIN\_PUBLICATION is the only valid value for this property. By default this property is not set.

**Note:** This property is relevant only in publish/subscribe domain only

## ***JMS\_IBM\_SYSTEM\_MESSAGEID***

**Data type:**

String

**Property of:**

Message

An identifier that identifies the message uniquely within the service integration bus. This property is read-only.

This property is relevant only when an application connects to a service integration bus.

## ***JMSX\_APPID***

**Data type:**

String

**Property of:**

Message

The name of the application that sent the message.

This property is the JMS defined property with the JMS name JMSXAppID. For more information about the property, see the *Java Message Service Specification, Version 1.1*.

By default, the property is not set.

This property is not valid for a real-time connection to a broker.

### **JMSX\_DELIVERY\_COUNT**

**Data type:**

System.Int32

**Property of:**

Message

The number of attempts to deliver the message.

This property is the JMS defined property with the JMS name JMSXDeliveryCount. For more information about the property, see the *Java Message Service Specification, Version 1.1*.

By default, the property is not set.

This property is not valid for a real-time connection to a broker.

### **JMSX\_GROUPID**

**Data type:**

String

**Property of:**

Message

The identifier of the message group to which the message belongs.

This property is the JMS defined property with the JMS name JMSXGroupID. For more information about the property, see the *Java Message Service Specification, Version 1.1*.

By default, the property is not set.

This property is not valid for a real-time connection to a broker.

### **JMSX\_GROUPSEQ**

**Data type:**

System.Int32

**Property of:**

Message

The sequence number of the message within a message group.

This property is the JMS defined property with the JMS name JMSXGroupSeq. For more information about the property, see the *Java Message Service Specification, Version 1.1*.

By default, the property is not set.

This property is not valid for a real-time connection to a broker.

### **JMSX\_USERID**

**Data type:**

String

**Property of:**

Message

The user identifier associated with the application that sent the message.

This property is the JMS defined property with the JMS name JMSXUserID. For more information about the property, see the *Java Message Service Specification, Version 1.1*.

By default, the property is not set.

This property is not valid for a real-time connection to a broker.

### ***XMSC\_ASYNC\_EXCEPTIONS***

**Data type:**

System.Int32

**Property of:**

ConnectionFactory

This property determines whether XMS informs an ExceptionListener only when a connection is broken, or when any exception occurs asynchronously to an XMS API call. This property applies to all Connections created from this ConnectionFactory that have an ExceptionListener registered.

Valid values for this property are:

#### ***XMSC\_ASYNC\_EXCEPTIONS\_ALL***

Any exception detected asynchronously, outside the scope of a synchronous API call, and all connection broken exceptions are sent to the ExceptionListener.

#### ***XMSC\_ASYNC\_EXCEPTIONS\_CONNECTIONBROKEN***

Only exceptions indicating a broken connection are sent to the ExceptionListener. Any other exceptions occurring during asynchronous processing are not reported to the ExceptionListener, and hence the application is not informed of these exceptions.

By default this property is set to XMSC\_ASYNC\_EXCEPTIONS\_ALL.

### ***XMSC\_CLIENT\_ID***

**Data type:**

String

**Property of:**

ConnectionFactory

The client identifier for a connection.

A client identifier is used only to support durable subscriptions in the publish/subscribe domain, and is ignored in the point-to-point domain. For more information about setting client identifiers, see [“ConnectionFactory and Connection objects” on page 22](#).

This property is not relevant for a real-time connection to a broker.

### ***XMSC\_CONNECTION\_TYPE***

**Data type:**

System.Int32

**Property of:**

ConnectionFactory

The type of messaging server to which an application connects.

The valid values of the property are as follows:

<b>Valid value</b>	<b>Meaning</b>
XMSC_CT_RTT	A real-time connection to a broker.
XMSC_CT_WMQ	A connection to a WebSphere MQ queue manager.
XMSC_CT_WPM	A connection to a WebSphere service integration bus.

By default, the property is not set.

## ***XMSC\_DELIVERY\_MODE***

**Data type:**

System.Int32

**Property of:**

Destination

**Name used in a URI:**

persistence (for a WebSphere MQ destination)

deliveryMode (for a WebSphere default messaging provider destination)

The delivery mode of messages sent to the destination.

The valid values of the property are as follows:

<b>Valid value</b>	<b>Meaning</b>
XMSC_DELIVERY_NOT_PERSISTENT	A message sent to the destination is nonpersistent. The default delivery mode of the message producer, or any delivery mode specified on the Send call, is ignored. If the destination is a WebSphere MQ queue, the value of the queue attribute <i>DefPersistence</i> is also ignored.
XMSC_DELIVERY_PERSISTENT	A message sent to the destination is persistent. The default delivery mode of the message producer, or any delivery mode specified on the Send call, is ignored. If the destination is a WebSphere MQ queue, the value of the queue attribute <i>DefPersistence</i> is also ignored.
XMSC_DELIVERY_AS_APP	A message sent to the destination has the delivery mode specified on the Send call. If the Send call specifies no delivery mode, the default delivery mode of the message producer is used instead. If the destination is a WebSphere MQ queue, the value of the queue attribute <i>DefPersistence</i> is ignored.
XMSC_DELIVERY_AS_DEST	If the destination is a WebSphere MQ queue, a message put on the queue has the delivery mode specified by the value of the queue attribute <i>DefPersistence</i> . The default delivery mode of the message producer, or any delivery mode specified on the Send call, is ignored.  If the destination is not a WebSphere MQ queue, the meaning is the same as that of XMSC_DELIVERY_AS_APP.

The default value is XMSC\_DELIVERY\_AS\_APP.

## ***XMSC\_IC\_PROVIDER\_URL***

**Data type:**

String

**Property of:**

InitialContext

Used to locate the JNDI naming directory so that the COS naming service does not need to be on the same server as the web service.

## ***XMSC\_IC\_SECURITY\_AUTHENTICATION***

**Data type:**

String

**Property of:**

InitialContext

Based on the Java Context interface SECURITY\_AUTHENTICATION. This property is only applicable to the COS naming context.

## ***XMSC\_IC\_SECURITY\_CREDENTIALS***

**Data type:**

String

**Property of:**

InitialContext

Based on the Java Context interface SECURITY\_CREDENTIALS. This property is only applicable to the COS naming context.

## ***XMSC\_IC\_SECURITY\_PRINCIPAL***

**Data type:**

String

**Property of:**

InitialContext

Based on the Java Context interface SECURITY\_PRINCIPAL. This property is only applicable to the COS naming context.

## ***XMSC\_IC\_SECURITY\_PROTOCOL***

**Data type:**

String

**Property of:**

InitialContext

Based on the Java Context interface SECURITY\_PROTOCOL This property is only applicable to the COS naming context.

## ***XMSC\_IC\_URL***

**Data type:**

String

**Property of:**

InitialContext

For LDAP and FileSystem contexts, the address of the repository containing administered objects.

For COS naming contexts, the address of the web service that looks up the objects in the directory.

## ***XMSC\_IS\_SUBSCRIPTION\_MULTICAST***

**Data type:**

System.Boolean

**Property of:**

MessageConsumer

Indicates whether messages are being delivered to the message consumer using WebSphere MQ Multicast Transport. This property is read-only.

The value of the property is true if messages are being delivered to the message consumer using WebSphere MQ Multicast Transport. Otherwise, the value is false.

This property is relevant only for a real-time connection to a broker.

### ***XMSC\_IS\_SUBSCRIPTION\_RELIABLE\_MULTICAST***

**Data type:**

System.Boolean

**Property of:**

MessageConsumer

Indicates whether messages are being delivered to the message consumer using WebSphere MQ Multicast Transport with a reliable quality of service. This property is read-only.

The value of the property is true if messages are being delivered to the message consumer using WebSphere MQ Multicast Transport with a reliable quality of service. Otherwise, the value is false.

This property is relevant only for a real-time connection to a broker.

### ***XMSC\_JMS\_MAJOR\_VERSION***

**Data type:**

System.Int32

**Property of:**

ConnectionMetaData

The major version number of the JMS specification upon which XMS is based. This property is read-only.

### ***XMSC\_JMS\_MINOR\_VERSION***

**Data type:**

System.Int32

**Property of:**

ConnectionMetaData

The minor version number of the JMS specification upon which XMS is based. This property is read-only.

### ***XMSC\_JMS\_VERSION***

**Data type:**

String

**Property of:**

ConnectionMetaData

The version identifier of the JMS specification upon which XMS is based. This property is read-only.

### ***XMSC\_MAJOR\_VERSION***

**Data type:**

System.Int32

**Property of:**

ConnectionMetaData

The version number of the XMS client. This property is read-only.

### ***XMSC\_MINOR\_VERSION***

**Data type:**

System.Int32

**Property of:**

ConnectionMetaData

The release number of the XMS client. This property is read-only.

## ***XMSC\_PASSWORD***

**Data type:**

Byte array

**Property of:**

ConnectionFactory

A password that can be used to authenticate the application when it attempts to connect to a messaging server. The password is used with the [XMSC\\_USERID](#) property.

By default, the property is not set.

If you are connecting to WebSphere MQ on distributed platforms, and you set the XMSC\_USERID property of the connection factory, it must match the **userid** of the logged on user. If you do not set these properties, the queue manager uses the **userid** of the logged on user by default. If you require further connection-level authentication of individual users you can write a client authentication exit which is configured in WebSphere MQ. You can learn more about creating a client authentication exit in the Authentication topic in the WebSphere MQ Clients manual.

To authenticate the user when connecting to WebSphere MQ on z/OS you need to use a security exit.

## ***XMSC\_PRIORITY***

**Data type:**

System.Int32

**Property of:**

Destination

**Name used in a URI:**

priority

The priority of messages sent to the destination.

The valid values of the property are as follows:

<b>Valid value</b>	<b>Meaning</b>
An integer in the range 0, the lowest priority, to 9, the highest priority	A message sent to the destination has the specified priority. The default priority of the message producer, or any priority specified on the Send call, is ignored. If the destination is a WebSphere MQ queue, the value of the queue attribute <b>DefPriority</b> is also ignored.
XMSC_PRIORITY_AS_APP	A message sent to the destination has the priority specified on the Send call. If the Send call specifies no priority, the default priority of the message producer is used instead. If the destination is a WebSphere MQ queue, the value of the queue attribute <b>DefPriority</b> is ignored.
XMSC_PRIORITY_AS_DEST	If the destination is a WebSphere MQ queue, a message put on the queue has the priority specified by the value of the queue attribute <b>DefPriority</b> . The default priority of the message producer, or any priority specified on the Send call, is ignored.  If the destination is not a WebSphere MQ queue, the meaning is the same as that of XMSC_PRIORITY_AS_APP.

The default value is XMSC\_PRIORITY\_AS\_APP.

WebSphere MQ Real-Time Transport and WebSphere MQ Multicast Transport take no action based upon the priority of a message.

## ***XMSC\_PROVIDER\_NAME***

**Data type:**

String

**Property of:**

ConnectionMetaData

The provider of the XMS client. This property is read-only.

## ***XMSC\_RTT\_BROKER\_PING\_INTERVAL***

**Data type:**

System.Int32

**Property of:**

ConnectionFactory

The time interval, in milliseconds, after which XMS .NET checks the connection to a Real Time messaging server to detect any activity. If no activity is detected, the client initiates a ping; the connection is closed if no response is detected to the ping.

The default value of the property is 30000.

## ***XMSC\_RTT\_CONNECTION\_PROTOCOL***

**Data type:**

System.Int32

**Property of:**

ConnectionFactory

The communications protocol used for a real-time connection to a broker.

The value of the property must be `XMSC_RTT_CP_TCP`, which means a real-time connection to a broker over TCP/IP. The default value is `XMSC_RTT_CP_TCP`.

## ***XMSC\_RTT\_HOST\_NAME***

**Data type:**

String

**Property of:**

ConnectionFactory

The host name or IP address of the system on which a broker runs.

This property is used with the `XMSC_RTT_PORT` property to identify the broker.

By default, the property is not set.

## ***XMSC\_RTT\_LOCAL\_ADDRESS***

**Data type:**

String

**Property of:**

ConnectionFactory

The host name or IP address of the local network interface to be used for a real-time connection to a broker.

This property is useful only if the system on which the application is running has two or more network interfaces and you need to be able to specify which interface must be used for a real-time connection. If the system has only one network interface, only that interface can be used. If the system has two or more network interfaces and the property is not set, the interface is selected at random.

By default, the property is not set.

## ***XMSC\_RTT\_MULTICAST***

**Data type:**

System.Int32

**Property of:**

ConnectionFactory and Destination

**Name used in a URI:**

multicast

The multicast setting for a connection factory or destination. Only a destination that is a topic can have this property.

An application uses this property to enable multicast in association with a real-time connection to a broker and, if multicast is enabled, to specify the precise way in which multicast is used to deliver messages from the broker to a message consumer. The property has no effect on how a message producer sends messages to the broker.

The valid values of the property are as follows:

**Valid value**

XMSC\_RTT\_MULTICAST\_DISABLED

**Meaning**

Messages are not delivered to a message consumer using WebSphere MQ Multicast Transport. This value is the default value for a ConnectionFactory object.

XMSC\_RTT\_MULTICAST\_ASCF

Messages are delivered to a 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 that the connection is created. This value is valid only for a Destination object, and is the default value for a Destination object.

XMSC\_RTT\_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\_RTT\_MULTICAST\_RELIABLE

If the topic is configured for reliable multicast in the broker, messages are delivered to a 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\_RTT\_MULTICAST\_NOT\_RELIABLE

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 not used even if the topic is configured for reliable multicast.

## ***XMSC\_RTT\_PORT***

**Data type:**

System.Int32

**Property of:**

ConnectionFactory

The number of the port on which a broker listens for incoming requests. On the broker, you must configure a Real-timeInput or Real-timeOptimizedFlow message processing node to listen on this port.

This property is used with the `XMSC_RTT_HOST_NAME` property to identify the broker.

The default value of the property is `XMSC_RTT_DEFAULT_PORT`, or 1506.

***XMSC\_TIME\_TO\_LIVE*****Data type:**

System.Int32

**Property of:**

Destination

**Name used in a URI:**

expiry (for a WebSphere MQ destination)

timeToLive (for a WebSphere default messaging provider destination)

The time to live for messages sent to the destination.

The valid values of the property are as follows:

Valid value	Meaning
0	A message sent to the destination never expires.
A positive integer	A message sent to the destination has the specified time to live in milliseconds. The default time to live of the message producer, or any time to live specified on the Send call, is ignored.
<code>XMSC_TIME_TO_LIVE_AS_APP</code>	A message sent to the destination has the time to live specified on the Send call. If the Send call specifies no time to live, the default time to live of the message producer is used instead.

The default value is `XMSC_TIME_TO_LIVE_AS_APP`.

***XMSC\_USERID*****Data type:**

String

**Property of:**

ConnectionFactory

A user identifier that can be used to authenticate the application when it attempts to connect to a messaging server. The user identifier is used with the `XMSC_PASSWORD` property.

By default, the property is not set.

If you are connecting to WebSphere MQ distributed platforms, and you set the `XMSC_USERID` property of the connection factory, it must match the **userid** of the logged on user. If you do not set these properties, the queue manager uses the **userid** of the logged on user by default. If you require further connection-level authentication of individual users you can write a client authentication exit which is configured in WebSphere MQ.

To authenticate the user when connecting to WebSphere MQ on z/OS you need to use a security exit.

***XMSC\_VERSION*****Data type:**

String

**Property of:**

ConnectionMetaData

The version identifier of the XMS client. This property is read-only.

***XMSC\_WMQ\_BROKER\_CONTROLQ*****Data type:**

String

**Property of:**

ConnectionFactory

The name of the control queue used by a broker.

The default value of the property is SYSTEM.BROKER.CONTROL.QUEUE.

This property is relevant only in the publish/subscribe domain.

***XMSC\_WMQ\_BROKER\_PUBQ*****Data type:**

String

**Property of:**

ConnectionFactory

The name of the queue monitored by a broker where applications send messages that they publish.

The default value of the property is SYSTEM.BROKER.DEFAULT.STREAM.

This property is relevant only in the publish/subscribe domain.

***XMSC\_WMQ\_BROKER\_QMGR*****Data type:**

String

**Property of:**

ConnectionFactory

The name of the queue manager to which a broker is connected.

By default, the property is not set.

This property is relevant only in the publish/subscribe domain.

***XMSC\_WMQ\_BROKER\_SUBQ*****Data type:**

String

**Property of:**

ConnectionFactory

The name of the subscriber queue for a nondurable message consumer.

The name of the subscriber queue must start with the following characters:

SYSTEM.JMS.ND.

If you want all nondurable message consumers to share a subscriber queue, specify the complete name of the shared queue. A queue with the specified name must exist before an application can create a nondurable message consumer.

If you want each nondurable message consumer to retrieve messages from its own exclusive subscriber queue, specify a queue name that ends with an asterisk (\*). Then, when an application creates a nondurable message consumer, the XMS client creates a dynamic queue for exclusive use by the message consumer. The XMS client uses the value of the property to set the contents of the **DynamicQName** field in the object descriptor that is used to create the dynamic queue.

The default value of the property is SYSTEM.JMS.ND.SUBSCRIBER.QUEUE, which means that XMS uses the shared queue approach by default.

This property is relevant only in the publish/subscribe domain.

### ***XMSC\_WMQ\_BROKER\_VERSION***

**Data type:**

System.Int32

**Property of:**

ConnectionFactory and Destination

**Name used in a URI:**

brokerVersion

The type of broker used by the application for a connection or for the destination. Only a destination that is a topic can have this property.

The valid values of the property are as follows:

<b>Valid value</b>	<b>Meaning</b>
XMSC_WMQ_BROKER_V1	The application is using a WebSphere MQ Publish/Subscribe broker.  The application can also use this value if you migrate from WebSphere MQ Publish/Subscribe to WebSphere Message Broker but did not change the application.
XMSC_WMQ_BROKER_V2	The application is using a broker of IBM Integration Bus.
XMSC_WMQ_BROKER_UNSPECIFIED	After the broker is migrated, set this property so that RFH2 headers are no longer used. After migration, this property is no longer relevant.

The default value for a connectionfactory is XMSC\_WMQ\_BROKER\_UNSPECIFIED but, by default, the property is not set for a destination. Setting the property for a destination overrides any value specified by the connection factory property.

### ***XMSC\_WMQ\_CCDTURL***

**Data type:**

System.String

**Property of:**

ConnectionFactory

A Uniform Resource Locator (URL) that identifies the name and location of the file that contains the client channel definition table and also specifies how the file can be accessed.

By default, this property is not set.

### ***XMSC\_WMQ\_CCSID***

**Data type:**

System.Int32

**Property of:**

Destination

**Name used in a URI:**

CCSID

The identifier (CCSID) of the coded character set, or code page, that the strings of character data in the body of a message are in when the XMS client forwards the message to the destination. If set for

an individual message, the `JMS_IBM_CHARACTER_SET` property overrides the CCSID specified for the destination by this property.

The default value of the property is 1208.

This property is relevant only to messages sent to the destination, not to messages received from the destination.

### ***XMSC\_WMQ\_CHANNEL***

**Data type:**

String

**Property of:**

ConnectionFactory

The name of the channel to be used for a connection.

By default, the property is not set.

This property is relevant only when an application connects to a queue manager in client mode.

### ***XMSC\_WMQ\_CLIENT\_RECONNECT\_OPTIONS***

**Data type:**

String

**Property of:**

ConnectionFactory

This property specifies the client reconnect options for new connections created by this factory. It is found in XMSC, and is one of:

- `WMQ_CLIENT_RECONNECT_AS_DEF` (default). Use the value specified in the `mqclient.ini` file. Set the value by using the **DefRecon** property within the Channels stanza. It can be set to one of:
  1. YES. Behaves as the `WMQ_CLIENT_RECONNECT` option
  2. NO. Default. Does not specify any reconnection options
  3. QMGR. Behaves as the `WMQ_CLIENT_RECONNECT_Q_MGR` option
  4. DISABLED. Behaves as the `WMQ_CLIENT_RECONNECT_DISABLED` option
- `WMQ_CLIENT_RECONNECT`. Reconnect to any of the queue managers specified in the connection name list.
- `WMQ_CLIENT_RECONNECT_Q_MGR`. Reconnects to the same queue manager that it is originally connected to. It returns `MQRC_RECONNECT_QMID_MISMATCH` if the queue manager it tries to connect to (specified in the connection name list) has a different QMID to the queue manager originally connected to.
- `WMQ_CLIENT_RECONNECT_DISABLED`. Reconnection is disabled.

### ***XMSC\_WMQ\_CLIENT\_RECONNECT\_TIMEOUT***

**Data type:**

String

**Property of:**

ConnectionFactory

The `XMSC_WMQ_CLIENT_RECONNECT_TIMEOUT` property is valid only for the Managed XMS .NET client.

This property specifies the duration of time, in seconds, that a client connection attempts to reconnect.

After attempting to reconnect for this duration of time, the client will fail with `MQRC_RECONNECT_FAILED`. The default setting for this property is `XMSC.WMQ_CLIENT_RECONNECT_TIMEOUT_DEFAULT`.

The default value of this property is 1800.

## ***XMSC\_WMQ\_CONNECTION\_MODE***

**Data type:**

System.Int32

**Property of:**

ConnectionFactory

The mode by which an application connects to a queue manager.

The valid values of the property are as follows:

<b>Valid value</b>	<b>Meaning</b>
XMSC_WMQ_CM_BINDINGS	A connection to a queue manager in bindings mode, for optimal performance. This value is the default value for C/C++.
XMSC_WMQ_CM_CLIENT	A connection to a queue manager in client mode, to ensure a fully managed stack. This value is the default value for .NET.
XMSC_WMQ_CM_CLIENT_UNMANAGED (for .NET only)	A connection to a queue manager which forces an unmanaged client stack.

### **Related concepts**

[Managed and unmanaged operations in .NET](#)

Managed code is executed exclusively within the .NET common language runtime environment and is wholly dependent on the services provided by that runtime. An application is classed as unmanaged if any part of the application runs or calls services outside of the .NET common language runtime environment.

## ***XMSC\_WMQ\_CONNECTION\_NAME\_LIST***

**Data type:**

String

**Property of:**

ConnectionFactory

This property specifies the hosts to which the client attempts to reconnect to after its connection are broken.

The connection name list is a comma-separated list of host/ip port pairs. The default setting for this property is WMQ\_CONNECTION\_NAME\_LIST\_DEFAULT.

For example, 127.0.0.1(1414), host2.example.com(1400)

The default setting of this property is localhost(1414).

## ***XMSC\_WMQ\_DUR\_SUBQ***

**Data type:**

String

**Property of:**

Destination

The name of the subscriber queue for a durable subscriber that is receiving messages from the destination. Only a destination that is a topic can have this property.

The name of the subscriber queue must start with the following characters:

SYSTEM.JMS.D.

If you want all durable subscribers to share a subscriber queue, specify the complete name of the shared queue. A queue with the specified name must exist before an application can create a durable subscriber.

If you want each durable subscriber to retrieve messages from its own exclusive subscriber queue, specify a queue name that ends with an asterisk (\*). Then, when an application creates a durable

subscriber, the XMS client creates a dynamic queue for exclusive use by the durable subscriber. The XMS client uses the value of the property to set the contents of the **DynamicQName** field in the object descriptor that is used to create the dynamic queue.

The default value of the property is SYSTEM.JMS.D.SUBSCRIBER.QUEUE, which means that XMS uses the shared queue approach by default.

This property is relevant only in the publish/subscribe domain.

## ***XMSC\_WMQ\_ENCODING***

### **Data type:**

System.Int32

### **Property of:**

Destination

How numerical data in the body of a message is represented when the XMS client forwards the message to the destination. If set for an individual message, the JMS\_IBM\_ENCODING property overrides the encoding specified for the destination by this property. The property specifies the representation of binary integers, packed decimal integers, and floating point numbers.

The valid values of the property are the same as the values that can be specified in the **Encoding** field of a message descriptor.

An application can use the following named constants to set the property:

<b>Named constant</b>	<b>Meaning</b>
MQENC_INTEGER_NORMAL	Normal integer encoding
MQENC_INTEGER_REVERSED	Reversed integer encoding
MQENC_DECIMAL_NORMAL	Normal packed decimal encoding
MQENC_DECIMAL_REVERSED	Reversed packed decimal encoding
MQENC_FLOAT_IEEE_NORMAL	Normal IEEE floating point encoding
MQENC_FLOAT_IEEE_REVERSED	Reversed IEEE floating point encoding
MQENC_FLOAT_S390	z/OS architecture floating point encoding
MQENC_NATIVE	Native machine encoding

To form a value for the property, the application can add three of these constants as follows:

- A constant whose name commences with MQENC\_INTEGER, to specify the representation of binary integers
- A constant whose name commences with MQENC\_DECIMAL, to specify the representation of packed decimal integers
- A constant whose name commences with MQENC\_FLOAT, to specify the representation of floating point numbers

Alternatively, the application can set the property to MQENC\_NATIVE, whose value is environment-dependent.

The default value of the property is MQENC\_NATIVE.

This property is relevant only to messages sent to the destination, not to messages received from the destination.

## ***XMSC\_WMQ\_FAIL\_IF\_QUIESCE***

### **Data type:**

System.Int32

**Property of:**

ConnectionFactory and Destination

**Name used in a URI:**

failIfQuiesce

Whether calls to certain methods fail if the queue manager to which the application is connected is in a quiescing state.

The valid values of the property are as follows:

Valid value	Meaning
XMSC_WMQ_FIQ_YES	Calls to certain methods fail if the queue manager is in a quiescing state. When the application detects that the queue manager is quiescing, the application can complete its immediate task and close the connection, allowing the queue manager to stop.
XMSC_WMQ_FIQ_NO	No method calls fail because the queue manager is in a quiescing state. If you specify this value, the application cannot detect that the queue manager is quiescing. The application might continue to perform operations against the queue manager and therefore prevent the queue manager from stopping.

The default value for a connection factory is XMSC\_WMQ\_FIQ\_YES but, by default, the property is not set for a destination. Setting the property for a destination overrides any value specified by the connection factory property.

***XMSC\_WMQ\_MESSAGE\_BODY*****Data type:**

System.Int32

**Property of:**

Destination

This property determines whether an XMS application processes the MQRFH2 of a IBM WebSphere MQ message as part of the message payload (that is, as part of the message body).

**Note:** When sending messages to a destination, XMSC\_WMQ\_MESSAGE\_BODY property supersedes existing XMS Destination property XMSC\_WMQ\_TARGET\_CLIENT.

Valid values for this property are:

**XMSC\_WMQ\_MESSAGE\_BODY\_JMS**

**Receive:** The inbound XMS message type and body are determined by the contents of the MQRFH2 (if present) or the MQMD (if there is no MQRFH2) in the received IBM WebSphere MQ message.

**Send:** The outbound XMS message body contains a prepended and auto-generated MQRFH2 header based on XMS Message properties and header fields.

**XMSC\_WMQ\_MESSAGE\_BODY\_MQ**

**Receive:** The inbound XMS message type is always ByteMessage, irrespective of the contents of received IBM WebSphere MQ message or the format field of the received MQMD. The XMS message body is the unaltered message data returned by the underlying messaging provider API call. The character set and encoding of the data in the message body is determined by the CodedCharSetId and Encoding fields of the MQMD. The format of the data in the message body is determined by the Format field of the MQMD.

**Send:** The outbound XMS message body contains the application payload as-is; and no auto-generated IBM WebSphere MQ header is added to the body.

**XMSC\_WMQ\_MESSAGE\_BODY\_UNSPECIFIED**

**Receive:** The XMS client determines a suitable value for this property. On receive path, this value is the WMQ\_MESSAGE\_BODY\_JMS property value.

**Send:** The XMS client determines a suitable value for this property. On send path, this value is the XMSC\_WMQ\_TARGET\_CLIENT property value.

By default this property is set to XMSC\_WMQ\_MESSAGE\_BODY\_UNSPECIFIED.

### ***XMSC\_WMQ\_MQMD\_MESSAGE\_CONTEXT***

**Data type:**  
System.Int32

**Property of:**  
Destination

Determines what level of message context is to be set by the XMS application. The application must be running with appropriate context authority for this property to take effect.

The valid values for this property are:

#### **XMSC\_WMQ\_MDCTX\_DEFAULT**

For outbound messages, the MQOPEN API call and the MQPMO structure specifies no explicit message context options.

#### **XMSC\_WMQ\_MDCTX\_SET\_IDENTITY\_CONTEXT**

The MQOPEN API call specifies the message context option MQOO\_SET\_IDENTITY\_CONTEXT and the MQPMO structure specifies MQPMO\_SET\_IDENTITY\_CONTEXT.

#### **XMSC\_WMQ\_MDCTX\_SET\_ALL\_CONTEXT**

The MQOPEN API call specifies the message context option MQOO\_SET\_ALL\_CONTEXT and the MQPMO structure specifies MQPMO\_SET\_ALL\_CONTEXT.

By default this property is set to XMSC\_WMQ\_MDCTX\_DEFAULT.

**Note:** This property is not relevant when an application connects to System Integration Bus.

The following properties require XMSC\_WMQ\_MQMD\_MESSAGE\_CONTEXT property to be set to XMSC\_WMQ\_MDCTX\_SET\_IDENTITY\_CONTEXT property value or XMSC\_WMQ\_MDCTX\_SET\_ALL\_CONTEXT property value when sending a message for in order to have wanted effect:

- JMS\_IBM\_MQMD\_USERIDENTIFIER
- JMS\_IBM\_MQMD\_ACCOUNTINGTOKEN
- JMS\_IBM\_MQMD\_APPLIDENTITYDATA

Following properties require XMSC\_WMQ\_MQMD\_MESSAGE\_CONTEXT property to be set to XMSC\_WMQ\_MDCTX\_SET\_ALL\_CONTEXT property value when sending a message for in order to have wanted effect:

- JMS\_IBM\_MQMD\_PUTAPPLTYPE
- JMS\_IBM\_MQMD\_PUTAPPLNAME
- JMS\_IBM\_MQMD\_PUTDATE
- JMS\_IBM\_MQMD\_PUTTIME
- JMS\_IBM\_MQMD\_APPLORIGINDATA

### ***XMSC\_WMQ\_MQMD\_READ\_ENABLED***

**Data type:**  
System.Int32

**Property of:**  
Destination

This property determines whether an XMS application can extract the values of MQMD fields or not.

The valid values for this property are:

**XMSC\_WMQ\_READ\_ENABLED\_NO**

When sending messages, the JMS\_IBM\_MQMD\* properties on a sent message are not updated to reflect the updated field values in the MQMD.

When receiving messages, none of the JMS\_IBM\_MQMD\* properties are available on a received message, even if some or all of them are set by the sender.

**XMSC\_WMQ\_READ\_ENABLED\_YES**

When sending messages, all of the JMS\_IBM\_MQMD\* properties on a sent message are updated to reflect the updated field values in the MQMD, including those properties that the sender did not set explicitly.

When receiving messages, all of the JMS\_IBM\_MQMD\* properties are available on a received message, including those properties that the sender did not set explicitly.

By default this property is set to XMSC\_WMQ\_READ\_ENABLED\_NO.

***XMSC\_WMQ\_MQMD\_WRITE\_ENABLED*****Data type:**

System.Int32

**Property of:**

Destination

This property determines whether an XMS application can the values of MQMD fields or not.

The valid values for this property are:

**XMSC\_WMQ\_WRITE\_ENABLED\_NO**

All JMS\_IBM\_MQMD\* properties are ignored and their values are not copied into the underlying MQMD structure.

**XMSC\_WMQ\_WRITE\_ENABLED\_YES**

JMS\_IBM\_MQMD\* properties are processed. Their values are copied into the underlying MQMD structure.

By default this property is set to XMSC\_WMQ\_WRITE\_ENABLED\_NO.

***XMSC\_WMQ\_PUT\_ASYNC\_ALLOWED*****Data type:**

System.Int32

**Property of:**

Destination

This property determines whether message producers are allowed to use asynchronous puts to send messages to this destination.

The valid values for this property are:

**XMSC\_WMQ\_PUT\_ASYNC\_ALLOWED\_AS\_DEST**

Determine whether asynchronous puts are allowed by referring to the queue or topic definition.

**XMSC\_WMQ\_PUT\_ASYNC\_ALLOWED\_AS\_Q\_DEF**

Determine whether asynchronous puts are allowed by referring to the queue definition.

**XMSC\_WMQ\_PUT\_ASYNC\_ALLOWED\_AS\_TOPIC\_DEF**

Determine whether asynchronous puts are allowed by referring to the topic definition.

**XMSC\_WMQ\_PUT\_ASYNC\_ALLOWED\_DISABLED**

Asynchronous puts are not allowed.

**XMSC\_WMQ\_PUT\_ASYNC\_ALLOWED\_ENABLED**

Asynchronous puts are allowed.

By default this property is set to XMSC\_WMQ\_PUT\_ASYNC\_ALLOWED\_AS\_DEST.

**Note:** This property is not relevant when an application is connecting to System Integration Bus.

**XMSC\_WMQ\_READ\_AHEAD\_ALLOWED****Data type:**

System.Int32

**Property of:**

Destination

This property determines whether message consumers and queue browsers are allowed to use read ahead to get non-persistent, non-transactional messages from this destination into an internal buffer before receiving them.

The valid values for this property are:

**XMSC\_WMQ\_READ\_AHEAD\_ALLOWED\_AS\_Q\_DEF**

Determine whether read ahead is allowed by referring to the queue definition.

**XMSC\_WMQ\_READ\_AHEAD\_ALLOWED\_AS\_TOPIC\_DEF**

Determine whether read ahead is allowed by referring to the topic definition.

**XMSC\_WMQ\_READ\_AHEAD\_ALLOWED\_AS\_DEST**

Determine whether read ahead is allowed by referring to the queue or topic definition.

**XMSC\_WMQ\_READ\_AHEAD\_ALLOWED\_DISABLED**

Read ahead is not allowed while consuming or browsing messages.

**XMSC\_WMQ\_READ\_AHEAD\_ALLOWED\_ENABLED**

Read ahead is allowed.

By default this property is set to XMSC\_WMQ\_READ\_AHEAD\_ALLOWED\_AS\_DEST.

**XMSC\_WMQ\_READ\_AHEAD\_CLOSE\_POLICY****Data type:**

System.Int32

**Property of:**

Destination

This property determines, for messages being delivered to an asynchronous message listener, what happens to messages in the internal read ahead buffer when the message consumer is closed.

This property is applicable in specifying closing queue options when consuming messages from a destination and not applicable when sending messages to a destination.

This property is ignored for Queue Browsers since during browse the messages are still available in the queues.

The valid values for this property are:

**XMSC\_WMQ\_READ\_AHEAD\_CLOSE\_POLICY\_DELIVER\_CURRENT**

Only the current message listener invocation completes before returning, potentially leaving messages in the internal read ahead buffer, which are then discarded.

**XMSC\_WMQ\_READ\_AHEAD\_CLOSE\_POLICY\_DELIVER\_ALL**

All messages in the internal read ahead buffer are delivered to the application message listener before returning.

By default this property is set to `XMSC_WMQ_READ_AHEAD_CLOSE_POLICY_DELIVER_CURRENT`.

**Note:**

- **Abnormal application termination**

All the messages in the read ahead buffer are lost when an XMS application terminates abruptly.

- **Implications on Transactions**

The read ahead is disabled when the applications use transaction. So, the application is not seeing any difference in the behavior when they use transacted sessions.

- **Implications of Session Acknowledgement modes**

The read ahead is enabled on a non-transacted session when the acknowledgement modes are either `XMSC_AUTO_ACKNOWLEDGE` or `XMSC_DUPS_OK_ACKNOWLEDGE`. The read ahead is disabled if the session acknowledgement mode is `XMSC_CLIENT_ACKNOWLEDGE` irrespective of transacted or non-transacted sessions.

- **Implications on Queue Browsers and Queue Browser Selectors**

The Queue Browsers and Queue Browser Selectors, used in XMS applications, get the performance advantage from read ahead. Closing the Queue Browser does not degrade performance, because the message is still available in the queue for any further operations. There are no any other implication on queue browsers and queue browser selectors apart from performance benefits of read ahead.

## ***XMSC\_WMQ\_HOST\_NAME***

**Data type:**

String

**Property of:**

ConnectionFactory

The host name or IP address of the system on which a queue manager runs.

This property is used only when an application connects to a queue manager in client mode. The property is used with the `XMSC_WMQ_PORT` property to identify the queue manager.

The default value of the property is `localhost`.

## ***XMSC\_WMQ\_LOCAL\_ADDRESS***

**Data type:**

String

**Property of:**

ConnectionFactory

For a connection to a queue manager, this property specifies the local network interface to be used, or the local port or range of local ports to be used, or both.

The value of the property is a string with the following format:

`[host_name][(low_port)[,high_port]]`

The meanings of the variables are as follows:

***host\_name***

The host name or IP address of the local network interface to be used for the connection.

Providing this information is necessary only if the system on which the application is running has two or more network interfaces and you need to be able to specify which interface must be used for the connection. If the system has only one network interface, only that interface can be used. If the system has two or more network interfaces and you do not specify which interface must be used, the interface is selected at random.

***low\_port***

The number of the local port to be used for the connection.

If *high\_port* is also specified, *low\_port* is interpreted the lowest port number in a range of port numbers.

**high\_port**

The highest port number in a range of port numbers. One of the ports in the specified range must be used for the connection.

The maximum length of the string is 48 characters.

Here are some examples of valid values of the property:

JUPITER  
9.20.4.98  
JUPITER(1000)  
9.20.4.98(1000,2000)  
(1000)  
(1000,2000)

By default, the property is not set.

This property is relevant only when an application connects to a queue manager in client mode.

***XMSC\_WMQ\_MESSAGE\_SELECTION***

**Data type:**

System.Int32

**Property of:**

ConnectionFactory

Determines whether message selection is done by the XMS client or by the broker.

The valid values of the property are as follows:

<b>Valid value</b>	<b>Meaning</b>
XMSC_WMQ_MSEL_CLIENT	Message selection is done by the XMS client.
XMSC_WMQ_MSEL_BROKER	Message selection is done by the broker.

The default value is XMSC\_WMQ\_MSEL\_CLIENT.

This property is relevant only in the publish/subscribe domain. Message selection by the broker is not supported if the XMSC\_WMQ\_BROKER\_VERSION property is set to XMSC\_WMQ\_BROKER\_V1.

***XMSC\_WMQ\_MSG\_BATCH\_SIZE***

**Data type:**

System.Int32

**Property of:**

ConnectionFactory

The maximum number of messages to be retrieved from a queue in one batch when using asynchronous message delivery.

When an application is using asynchronous message delivery, under certain conditions, the XMS client retrieves a batch of messages from a queue before forwarding each message individually to the application. This property specifies the maximum number of messages that can be in the batch.

The value of the property is a positive integer, and the default value is 10. Consider setting the property to a different value only if you have a specific performance problem that you need to address.

If an application is connected to a queue manager over a network, raising the value of this property can reduce network overheads and response times, but increase the amount of memory required to store the messages on the client system. Conversely, lowering the value of this property might increase network overheads and response times, but reduce the amount of memory required to store the messages.

## ***XMSC\_WMQ\_POLLING\_INTERVAL***

**Data type:**

System.Int32

**Property of:**

ConnectionFactory

If each message listener within a session has no suitable message on its queue, this value is the maximum interval, in milliseconds, that elapses before each message listener tries again to get a message from its queue.

If it frequently happens that no suitable message is available for any of the message listeners in a session, consider increasing the value of this property.

The value of the property is a positive integer. The default value is 5000.

## ***XMSC\_WMQ\_PORT***

**Data type:**

System.Int32

**Property of:**

ConnectionFactory

The number of the port on which a queue manager listens for incoming requests.

This property is used only when an application connects to a queue manager in client mode. The property is used with the `XMSC_WMQ_HOST_NAME` property to identify the queue manager.

The default value of the property is `XMSC_WMQ_DEFAULT_CLIENT_PORT`, or 1414.

## ***XMSC\_WMQ\_PROVIDER\_VERSION***

**Data type:**

String

**Property of:**

ConnectionFactory

The version, release, modification level and fix pack of the queue manager to which the application intends to connect. Valid values for this property are:

- Unspecified

Or a string in one of the following formats

- V.R.M.F
- V.R.M
- V.R
- V

Where V, R, M and F are integer values greater than or equal to zero.

A value of 7 or greater indicates that this version is intended as a IBM WebSphere MQ 7.0 ConnectionFactory for connections to a IBM WebSphere MQ 7.0 queue manager. A value earlier than 7 (for example "6.0.2.0"), indicates that it is intended for use with queue managers earlier than Version 7.0. The default value, unspecified, allows connections to any level of queue manager, determining the applicable properties and functionality available based on the queue manager's capabilities.

By default this property is set to "unspecified".

**Note:**

- No socket sharing happens if `XMSC_WMQ_PROVIDER_VERSION` is set to 6. 2.

- Connection fails if XMSC\_WMQ\_PROVIDER\_VERSION is set to 7 and on the server SHARECNV for the channel is set to 0.
- IBM WebSphere MQ 7.0 specific features are disabled if XMSC\_WMQ\_PROVIDER\_VERSION is set to UNSPECIFIED and SHARECNV is set to 0.

The version of IBM WebSphere MQ Client also plays major role in whether an XMS client application can use IBM WebSphere MQ 7.0 specific features. The following table describes the behavior.

**Note:** A system property XMSC\_WMQ\_OVERRIDEPROVIDERVERSION overrides the XMSC\_WMQ\_PROVIDER\_VERSION property. This property can be used if you are unable to change connection factory setting.

<i>Table 34. XMS client - Ability to use IBM WebSphere MQ 7.0 specific features.</i>			
#	XMSC_WMQ_PROVIDER_VERSION	IBM WebSphere MQ Client Version	IBM WebSphere MQ 7.0 features
1	unspecified	7	ON
2	unspecified	6	OFF
3	7	7	ON
4	7	6	Exception
5	6	6	OFF
6	6	7	OFF

### ***XMSC\_WMQ\_PUB\_ACK\_INTERVAL***

**Data type:**

System.Int32

**Property of:**

ConnectionFactory

The number of messages published by a publisher before the XMS client requests an acknowledgement from the broker.

If you decrease the value of this property, the client requests acknowledgements more often, and therefore the performance of the publisher decreases. If you raise the value, the client takes a longer time to throw an exception if the broker fails.

The value of the property is a positive integer. The default value is 25.

### ***XMSC\_WMQ\_QMGR\_CCSID***

**Data type:**

System.Int32

**Property of:**

ConnectionFactory

The identifier (CCSID) of the coded character set, or code page, in which fields of character data defined in the Message Queue Interface (MQI) are exchanged between the XMS client and the WebSphere MQ client. This property does not apply to the strings of character data in the bodies of messages.

When an XMS application connects to a queue manager in client mode, the XMS client links to the WebSphere MQ client. The information exchanged between the two clients contains fields of character data that are defined in the MQI. Under normal circumstances, the WebSphere MQ client assumes that these fields are in the code page of the system on which the clients are running. If the XMS client provides and expects to receive these fields in a different code page, you must set this property to inform the WebSphere MQ client.

When the WebSphere MQ client forwards these fields of character data to the queue manager, the data in them must be converted if necessary into the code page used by the queue manager. Similarly, when the WebSphere MQ client receives these fields from the queue manager, the data in them must be converted if necessary into the code page in which the XMS client expects to receive the data. The WebSphere MQ client uses this property to perform these data conversions.

By default, the property is not set.

Setting this property is equivalent to setting the MQCCSID environment variable for a WebSphere MQ client that is supporting native WebSphere MQ client applications. For more information about this environment variable, see *WebSphere MQ Clients*.

### ***XMSC\_WMQ\_QUEUE\_MANAGER***

**Data type:**

String

**Property of:**

ConnectionFactory

The name of the queue manager to connect to.

By default, the property is not set.

### ***XMSC\_WMQ\_RECEIVE\_CCSID***

Destination property that sets the target CCSID for queue manager message conversion. The value is ignored unless XMSC\_WMQ\_RECEIVE\_CONVERSION is set to WMQ\_RECEIVE\_CONVERSION\_QMGR.

**Data type:**

Integer

**Value:**

Any positive integer.

The default value is 1208.

Specifying a GMO\_CONVERT value in a message is optional. If a GMO\_CONVERT value is specified, conversion takes place according to the value specified.

### ***XMSC\_WMQ\_RECEIVE\_CONVERSION***

Destination property that determines whether data conversion is going to be performed by the queue manager.

**Data type:**

Integer

**Values:**

XMSC\_WMQ\_RECEIVE\_CONVERSION\_CLIENT\_MSG (DEFAULT): Perform data conversion on the XMS client only. Conversion is always done using codepage 1208.

XMSC\_WMQ\_RECEIVE\_CONVERSION\_QMGR: Perform data conversion on the queue manager before sending a message to the XMS client.

### ***XMSC\_WMQ\_RECEIVE\_EXIT***

**Data type:**

String

**Property of:**

ConnectionFactory

Identifies a channel receive exit to be run.

The value of the property is a string that identifies a channel receive exit and has the following format:

**libraryName**(entryPointName)

where,

- **libraryName** is the full path of the managed exit .dll
- **entryPointName** is the class name qualified by the namespace

For example, C:\MyReceiveExit.dll(MyReceiveExitNameSpace.MyReceiveExitClassName)

By default, the property is not set.

This property is relevant only when an application connects to a queue manager in managed client mode. Also, only managed exits are supported.

### ***XMSC\_WMQ\_RECEIVE\_EXIT\_INIT***

**Data type:**

String

**Property of:**

ConnectionFactory

The user data that is passed to a channel receive exit when it is called.

The value of the property is a string. By default, the property is not set.

This property is relevant only when an application connects to a queue manager in managed client mode and the “[XMSC\\_WMQ\\_RECEIVE\\_EXIT](#)” on page 221 property is set.

### ***XMSC\_WMQ\_RESOLVED\_QUEUE\_MANAGER***

**Data type:**

String

**Property of:**

ConnectionFactory

This property is used to obtain the name of the queue manager to which it is connected.

When used with a CCDT (Client Channel Definition Table), this name might be different from the queue manager name specified in the Connection Factory.

### ***XMSC\_WMQ\_RESOLVED\_QUEUE\_MANAGER\_ID***

**Data type:**

String

**Property of:**

ConnectionFactory

This property is populated with the ID of the queue manager after the connection.

### ***XMSC\_WMQ\_SECURITY\_EXIT***

**Data type:**

String

**Property of:**

ConnectionFactory

Identifies a channel security exit.

The value of the property is a string that identifies a channel security exit and has the following format:

**libraryName**(entryPointName)

where,

- **libraryName** is the full path of the managed exit .dll
- **entryPointName** is the class name qualified by the namespace

For example, C:\MySecurityExit.dll(MySecurityExitNameSpace.MySecurityExitClassName)

The maximum length of the string is 128 characters.

By default, the property is not set.

This property is relevant only when an application connects to a queue manager in managed client mode. Also, only managed exits are supported.

### ***XMSC\_WMQ\_SECURITY\_EXIT\_INIT***

**Data type:**

String

**Property of:**

ConnectionFactory

The user data that is passed to a channel security exit when it is called.

The maximum length of the string of user data is 32 characters.

By default, the property is not set.

This property is relevant only when an application connects to a queue manager in managed client mode and the [“XMSC\\_WMQ\\_SECURITY\\_EXIT”](#) on page 222 property is set.

### ***XMSC\_WMQ\_SEND\_EXIT***

**Data type:**

String

**Property of:**

ConnectionFactory

Identifies a channel send exit.

The value of the property is a string. A channel send exit has the following format:

**libraryName**(entryPointName)

where,

- **libraryName** is the full path of the managed exit .dll
- **entryPointName** is the class name qualified by the namespace

For example, C:\MySendExit.dll(MySendExitNameSpace.MySendExitClassName)

By default, the property is not set.

This property is relevant only when an application connects to a queue manager in managed client mode. Also, only managed exits are supported.

### ***XMSC\_WMQ\_SEND\_EXIT\_INIT***

**Data type:**

String

**Property of:**

ConnectionFactory

The user data that is passed to channel send exits when they are called.

The value of the property is a string of one or more items of user data separated by commas. By default, the property is not set.

The rules for specifying user data that is passed to a sequence of channel send exits, are the same as the rules for specifying user data that is passed to a sequence of channel receive exits. For the rules therefore, see [“XMSC\\_WMQ\\_RECEIVE\\_EXIT\\_INIT”](#) on page 222.

This property is relevant only when an application connects to a queue manager in managed client mode and the “XMSC\_WMQ\_SEND\_EXIT” on page 223 property is set.

### ***XMSC\_WMQ\_SEND\_CHECK\_COUNT***

**Data type:**  
System.Int32

**Property of:**  
ConnectionFactory

The number of send calls to allow between checking for asynchronous put errors, within a single non-transacted XMS session.

By default this property is set to 0.

### ***XMSC\_WMQ\_SHARE\_CONV\_ALLOWED***

**Data type:**  
System.Int32

**Property of:**  
ConnectionFactory

Whether a client connection can share its socket with other top-level XMS connections from the same process to the same queue manager, if the channel definitions match. This property is provided to allow complete isolation of Connections in separate sockets if required for application development, maintenance, or operational reasons. Setting this property merely indicates to XMS to make the underlying socket shared. It does not indicate how many connections shares a single socket. The number of connections sharing a socket is determined by SHARECNV value which is negotiated between WebSphere MQ Client and WebSphere MQ Server.

An application can set the following named constants to set the property:

- XMSC\_WMQ\_SHARE\_CONV\_ALLOWED\_FALSE - Connections do not share a socket.
- XMSC\_WMQ\_SHARE\_CONV\_ALLOWED\_TRUE - Connections share a socket.

By default the property is set to XMSC\_WMQ\_SHARE\_CONV\_ALLOWED\_ENABLED.

This property is relevant only when an application connects to a queue manager in client mode.

### ***XMSC\_WMQ\_SSL\_CERT\_STORES***

**Data type:**  
String

**Property of:**  
ConnectionFactory

The locations of the servers that hold the certificate revocation lists (CRLs) to be used on an SSL connection to a queue manager.

The value of the property is a list of one or more URLs separated by commas. Each URL has the following format:

```
[user[/password]@]ldap://[serveraddress][:portnum][, ...]
```

This format is compatible with, but extended from, the basic MQJMS format.

It is valid to have an empty `serveraddress`. In this case, XMS assumes that the value is the string "localhost".

An example list is:

```
myuser/mypassword@ldap://server1.mycom.com:389  
ldap://server1.mycom.com
```

ldap://  
ldap://:389

For .NET only: From IBM MQ 8.0, managed connections to IBM MQ (WMQ\_CM\_CLIENT) and unmanaged connections to IBM MQ (WMQ\_CM\_CLIENT\_UNMANAGED) both support TLS/SSL connections.

By default, the property is not set.

### Related concepts

[SSL and TLS support for the unmanaged .NET client](#)

[SSL and TLS support for the managed .NET client](#)

## ***XMSC\_WMQ\_SSL\_CIPHER\_SPEC***

### Data type:

String

### Property of:

ConnectionFactory

The name of the CipherSpec to be used on a secure connection to a queue manager.

Cipher specifications that you can use with IBM WebSphere MQ SSL and TLS support are listed in the following table. When you request a personal certificate, you specify a key size for the public and private key pair. The key size that is used during the SSL handshake is the size stored in the certificate unless it is determined by the CipherSpec, as noted in the table. By default, this property is not set.

CipherSpec name	Protocol used	Hash algorithm	Encryption algorithm	Encryption bits	FIPS <sup>1</sup>	Suite B 128 bit	Suite B 192 bit
NULL_MD5	SSL 3.0	MD5	None	0	No	No	No
NULL_SHA	SSL 3.0	SHA-1	None	0	No	No	No
RC4_MD5_EXPORT <sup>2</sup>	SSL 3.0	MD5	RC4	40	No	No	No
RC4_MD5_US	SSL 3.0	MD5	RC4	128	No	No	No
RC4_SHA_US	SSL 3.0	SHA-1	RC4	128	No	No	No
RC2_MD5_EXPORT <sup>2</sup>	SSL 3.0	MD5	RC2	40	No	No	No
DES_SHA_EXPORT <sup>2</sup>	SSL 3.0	SHA-1	DES	56	No	No	No
RC4_56_SHA_EXPORT1024 <sup>3</sup>	SSL 3.0	SHA-1	RC4	56	No	No	No
DES_SHA_EXPORT1024 <sup>3</sup>	SSL 3.0	SHA-1	DES	56	No	No	No
TRIPLE_DES_SHA_US	SSL 3.0	SHA-1	3DES	168	No	No	No
TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	SHA-1	AES	128	Yes	No	No
TLS_RSA_WITH_AES_256_CBC_SHA <sup>4</sup>	TLS 1.0	SHA-1	AES	256	Yes	No	No
TLS_RSA_WITH_DES_CBC_SHA	TLS 1.0	SHA-1	DES	56	No <sup>5</sup>	No	No
TLS_RSA_WITH_3DES_EDE_CBC_SHA <sup>8</sup>	TLS 1.0	SHA-1	3DES	168	Yes	No	No
FIPS_WITH_DES_CBC_SHA	SSL 3.0	SHA-1	DES	56	No <sup>6</sup>	No	No
FIPS_WITH_3DES_EDE_CBC_SHA	SSL 3.0	SHA-1	3DES	168	No <sup>7</sup>	No	No
TLS_RSA_WITH_AES_128_GCM_SHA256 <sup>6</sup>	TLS 1.2	SHA-256	AES	128	Yes	No	No

CipherSpec name	Protocol used	Hash algorithm	Encryption algorithm	Encryption bits	FIPS <sup>1</sup>	Suite B 128 bit	Suite B 192 bit
TLS_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	SHA-384	AES	256	Yes	No	No
TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	SHA-256	AES	128	Yes	No	No
TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2	SHA-256	AES	256	Yes	No	No
ECDHE_ECDSA_RC4_128_SHA256	TLS 1.2	SHA-256	RC4	128	No	No	No
ECDHE_ECDSA_3DES_EDE_CBC_SHA256	TLS 1.2	SHA-256	3DES	168	Yes	No	No
ECDHE_RSA_RC4_128_SHA256	TLS 1.2	SHA-256	RC4	128	No	No	No
ECDHE_RSA_3DES_EDE_CBC_SHA256	TLS 1.2	SHA-256	3DES	168	Yes	No	No
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS 1.2	SHA-256	AES	128	Yes	No	No
ECDHE_ECDSA_AES_256_CBC_SHA384	TLS 1.2	SHA-384	AES	256	Yes	No	No
ECDHE_RSA_AES_128_CBC_SHA256	TLS 1.2	SHA-256	AES	128	Yes	No	No
ECDHE_RSA_AES_256_CBC_SHA384	TLS 1.2	SHA-384	AES	256	Yes	No	No
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS 1.2	SHA-256	AES	128	Yes	Yes	No
ECDHE_ECDSA_AES_256_GCM_SHA384	TLS 1.2	SHA-384	AES	256	Yes	No	Yes
ECDHE_RSA_AES_128_GCM_SHA256	TLS 1.2	SHA-256	AES	128	Yes	No	No
ECDHE_RSA_AES_256_GCM_SHA384	TLS 1.2	SHA-384	AES	256	Yes	No	No
TLS_RSA_WITH_NULL_SHA256	TLS 1.2	SHA-256	None	0	No	No	No
ECDHE_RSA_NULL_SHA256	TLS 1.2	SHA-256	None	0	No	No	No
ECDHE_ECDSA_NULL_SHA256	TLS 1.2	SHA-256	None	0	No	No	No
TLS_RSA_WITH_NULL_NULL	TLS 1.2	None	None	0	No	No	No
TLS_RSA_WITH_RC4_128_SHA256	TLS 1.2	SHA-256	RC4	128	No	No	No

CipherSpec name	Protocol used	Hash algorithm	Encryption algorithm	Encryption bits	FIPS <sup>1</sup>	Suite B 128 bit	Suite B 192 bit
-----------------	---------------	----------------	----------------------	-----------------	-------------------	-----------------	-----------------

**Notes:**

1. Specifies whether the CipherSpec complies with Federal Information Processing Standards (FIPS) 140-2. For an explanation of FIPS and information about how to configure WebSphere MQ for FIPS 140-2 compliant operation, see *Federal Information Processing Standards (FIPS)* in the online IBM WebSphere MQ product documentation.
2. The maximum handshake key size is 512 bits. If either of the certificates exchanged during the SSL handshake has a key size greater than 512 bits, a temporary 512-bit key is generated for use during the handshake.
3. The handshake key size is 1024 bits.
4. This CipherSpec cannot be used to secure a connection from the WebSphere MQ Explorer to a queue manager unless the appropriate unrestricted policy files are applied to the JRE used by the Explorer.
5. This CipherSpec was FIPS 140-2 certified before 19 May 2007.
6. This CipherSpec was FIPS 140-2 certified before 19 May 2007. The name FIPS\_WITH\_DES\_CBC\_SHA is historical and reflects the fact that this CipherSpec was previously (but is no longer) FIPS-compliant. This CipherSpec is deprecated.
7. The name FIPS\_WITH\_3DES\_EDE\_CBC\_SHA is historical and reflects the fact that this CipherSpec was previously (but is no longer) FIPS-compliant. The use of this CipherSpec is deprecated.
8. When WebSphere MQ is configured for FIPS 140-2 compliant operation, this CipherSpec can be used to transfer up to 32 GB of data before the connection is terminated with error AMQ9288. To avoid this error, either avoid using triple DES (which is deprecated), or enable secret key reset when using this CipherSpec in a FIPS 140-2 configuration.

**Related concepts**

[Security](#)

[Data integrity of messages](#)

**Related tasks**

[Specifying CipherSpecs](#)

***XMSC\_WMQ\_SSL\_CIPHER\_SUITE***

**Data type:**

String

**Property of:**

ConnectionFactory

The name of the CipherSuite to be used on an SSL or TLS connection to a queue manager. The protocol used in negotiating the secure connection depends on the specified CipherSuite.

This property has the following canonical values:

- SSL\_RSA\_WITH\_DES\_CBC\_SHA
- SSL\_RSA\_EXPORT1024\_WITH\_DES\_CBC\_SHA
- SSL\_RSA\_FIPS\_WITH\_3DES\_EDE\_CBC\_SHA
- SSL\_RSA\_FIPS\_WITH\_DES\_CBC\_SHA
- SSL\_RSA\_WITH\_NULL\_MD5
- SSL\_RSA\_WITH\_NULL\_SHA
- SSL\_RSA\_EXPORT1024\_WITH\_RC4\_56\_SHA
- SSL\_RSA\_EXPORT\_WITH\_RC4\_40\_MD5

- SSL\_RSA\_WITH\_RC4\_128\_MD5
- SSL\_RSA\_WITH\_RC4\_128\_SHA
- SSL\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA
- SSL\_RSA\_WITH\_AES\_128\_CBC\_SHA
- SSL\_RSA\_WITH\_AES\_256\_CBC\_SHA
- SSL\_RSA\_WITH\_DES\_CBC\_SHA
- SSL\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA

This value can be supplied as an alternative to `XMSC_WMQ_SSL_CIPHER_SPEC`.

If a non-empty value is specified for `XMSC_WMQ_SSL_CIPHER_SPEC`, this value overrides the setting for `XMSC_WMQ_SSL_CIPHER_SUITE`. If `XMSC_WMQ_SSL_CIPHER_SPEC` does not have a value, the value of `XMSC_WMQ_SSL_CIPHER_SUITE` is used as the cipher suite to be given to IBM Global Security Kit (GSKit). In this case, the value is mapped on to the equivalent CipherSpec value, as described in [“CipherSuite and CipherSpec name mappings for connections to a WebSphere MQ queue manager”](#) on page 65.

If both `XMSC_WMQ_SSL_CIPHER_SPEC` and `XMSC_WMQ_SSL_CIPHER_SUITE` are empty, the field `pChDef->SSLCipherSpec` is filled with spaces.

For .NET only: From IBM MQ 8.0, managed connections to IBM MQ (`WMQ_CM_CLIENT`) and unmanaged connections to IBM MQ (`WMQ_CM_CLIENT_UNMANAGED`) both support TLS/SSL connections.

By default, the property is not set.

#### **Related concepts**

[SSL and TLS support for the unmanaged .NET client](#)

[SSL and TLS support for the managed .NET client](#)

### ***XMSC\_WMQ\_SSL\_CRYPTO\_HW***

#### **Data type:**

String

#### **Property of:**

ConnectionFactory

Configuration details for the cryptographic hardware connected to the client system.

This property has the following canonical values:

- GSK\_ACCELERATOR\_RAINBOW\_CS\_OFF
- GSK\_ACCELERATOR\_RAINBOW\_CS\_ON
- GSK\_ACCELERATOR\_NCIPHER\_NF\_OFF
- GSK\_ACCELERATOR\_NCIPHER\_NF\_ON

There is a special format for PKCS11 cryptographic hardware (where `DriverPath`, `TokenLabel`, and `TokenPassword` are user-specified strings):

```
GSK_PKCS11=PKCS#11 DriverPath; PKCS#11 TokenLabel;PKCS#11 TokenPassword
```

XMS does not interpret or alter the contents of the string. It copies the value supplied, up to a limit of 256 single-byte characters, into the `MQSCO.CryptoHardware` field.

For .NET only: From IBM MQ 8.0, managed connections to IBM MQ (`WMQ_CM_CLIENT`) and unmanaged connections to IBM MQ (`WMQ_CM_CLIENT_UNMANAGED`) both support TLS/SSL connections.

By default, the property is not set.

#### **Related concepts**

[SSL and TLS support for the unmanaged .NET client](#)

[SSL and TLS support for the managed .NET client](#)

## ***XMSC\_WMQ\_SSL\_FIPS\_REQUIRED***

**Data type:**

Boolean

**Property of:**

ConnectionFactory

The value of this property determines whether an application can or cannot use non-FIPS compliant cipher suites. If this property is set to true, only FIPS algorithms are used for the client-server connection.

This property can have the following values, which translate to the two canonical values for MQSCO.FipsRequired:

<b>Value</b>	<b>Description</b>	<b>Corresponding value of MQSCO.FipsRequired</b>
false	Any CipherSpec can be used.	MQSSL_FIPS_NO (the default)
true	Only FIPS-certified cryptographic algorithms can be used in the CipherSpec applying to this client connection.	MQSSL_FIPS_YES

XMS copies the relevant value into MQSCO.FipsRequired before calling MQCONN.

The parameter MQSCO.FipsRequired is only available from WebSphere MQ version 6. If WebSphere MQ version 5.3, if this property is set, XMS does not attempt to make the connection to the queue manager, and throws an appropriate exception instead.

For .NET only: From IBM MQ 8.0, managed connections to IBM MQ (WMQ\_CM\_CLIENT) and unmanaged connections to IBM MQ (WMQ\_CM\_CLIENT\_UNMANAGED) both support TLS/SSL connections.

**Related concepts**

[SSL and TLS support for the unmanaged .NET client](#)

[SSL and TLS support for the managed .NET client](#)

## ***XMSC\_WMQ\_SSL\_KEY\_REPOSITORY***

**Data type:**

String

**Property of:**

ConnectionFactory

The location of the key database file in which keys and certificates are stored.

XMS copies the string, up to a limit of 256 single-byte characters, into the MQSCO.KeyRepository field. WebSphere MQ interprets this string as a filename, including the full path.

For .NET only: From IBM MQ 8.0, managed connections to IBM MQ (WMQ\_CM\_CLIENT) and unmanaged connections to IBM MQ (WMQ\_CM\_CLIENT\_UNMANAGED) both support TLS/SSL connections.

By default, the property is not set.

**Related concepts**

[SSL and TLS support for the unmanaged .NET client](#)

[SSL and TLS support for the managed .NET client](#)

## ***XMSC\_WMQ\_SSL\_KEY\_RESETCOUNT***

**Data type:**

System.Int32

**Property of:**

ConnectionFactory

The KeyResetCount represents the total number of unencrypted bytes sent and received within an SSL conversation before the secret key is renegotiated. The number of bytes includes control information sent by the MCA.

XMS copies the value that you supply for this property into MQSCO.KeyResetCount before calling MQCONN.

The parameter MQSCO.KeyResetCount is only available from WebSphere MQ version 6. If WebSphere MQ version 5.3, if this property is set, XMS does not attempt to make the connection to the queue manager, and throws an appropriate exception instead.

For .NET only: From IBM MQ 8.0, managed connections to IBM MQ (WMQ\_CM\_CLIENT) and unmanaged connections to IBM MQ (WMQ\_CM\_CLIENT\_UNMANAGED) both support TLS/SSL connections.

The default value of this property is zero, which means that secret keys are never renegotiated.

**Related concepts**

[SSL and TLS support for the unmanaged .NET client](#)

[SSL and TLS support for the managed .NET client](#)

***XMSC\_WMQ\_SSL\_PEER\_NAME*****Data type:**

String

**Property of:**

ConnectionFactory

The peer name to be used on an SSL connection to a queue manager.

There is no list of canonical values for this property. Instead, you must build this string according to the rules for SSLPEER.

An example of a peer name is:

```
"CN=John Smith, O=IBM ,OU=Test , C=GB"
```

XMS copies the string into the correct single-byte code page, and places the correct values into MQCD.SSLPeerNamePtr and MQCD.SSLPeerNameLength before calling MQCONN.

This property is relevant only if the application connects to a queue manager in client mode.

For .NET only: From IBM MQ 8.0, managed connections to IBM MQ (WMQ\_CM\_CLIENT) and unmanaged connections to IBM MQ (WMQ\_CM\_CLIENT\_UNMANAGED) both support TLS/SSL connections.

By default, the property is not set.

**Related concepts**

[SSL and TLS support for the unmanaged .NET client](#)

[SSL and TLS support for the managed .NET client](#)

**Related reference**

[SSLPEERNAME](#)

***XMSC\_WMQ\_SYNCPOINT\_ALL\_GETS*****Data type:**

System.Boolean

**Property of:**

ConnectionFactory

Whether all messages must be retrieved from queues within sync point control.

The valid values of the property are as follows:

<b>Valid value</b>	<b>Meaning</b>
false	When the circumstances are appropriate, the XMS client can retrieve messages from queues outside of sync point control.
true	The XMS client must retrieve all messages from queues within sync point control.

The default value is false.

### ***XMSC\_WMQ\_TARGET\_CLIENT***

**Data type:**  
System.Int32

**Property of:**  
Destination

**Name used in a URI:**  
targetClient

Whether messages sent to the destination contain an MQRFH2 header.

If an application sends a message containing an MQRFH2 header, the receiving application must be able to handle the header.

The valid values of the property are as follows:

<b>Valid value</b>	<b>Meaning</b>
XMSC_WMQ_TARGET_DEST_JMS	Messages sent to the destination contain an MQRFH2 header. Specify this value if the application is sending the messages to another XMS application, a WebSphere JMS application, or a native WebSphere MQ application that is designed to handle an MQRFH2 header.
XMSC_WMQ_TARGET_DEST_MQ	Messages sent to the destination do not contain an MQRFH2 header. Specify this value if the application is sending the messages to a native WebSphere MQ application that is not designed to handle an MQRFH2 header.

The default value is XMSC\_WMQ\_TARGET\_DEST\_JMS.

### ***XMSC\_WMQ\_TEMP\_Q\_PREFIX***

**Data type:**  
String

**Property of:**  
ConnectionFactory

The prefix used to form the name of the WebSphere MQ dynamic queue that is created when the application creates an XMS temporary queue.

The rules for forming the prefix are the same as the rules for forming the contents of the **DynamicQName** field in an object descriptor, but the last non-blank character must be an asterisk(\*). If the property is not set, the value used is CSQ.\* on z/OS and AMQ.\* on the other platforms. By default, the property is not set.

This property is relevant only in the point-to-point domain.

### ***XMSC\_WMQ\_TEMP\_TOPIC\_PREFIX***

**Data type:**  
String

**Property of:**

ConnectionFactory, Destination

When creating temporary topics, XMS generates a topic string of the form "TEMP/TEMPTOPICPREFIX/unique\_id", or if this property contains the default value, then this string, "TEMP/unique\_id", is generated. Specifying a non-empty value allows specific model queues to be defined for creating the managed queues for subscribers to temporary topics created under this connection.

Any non-null string consisting only of valid characters for a IBM WebSphere MQ topic string is a valid value for this property.

By default this property is set to "" (empty string).

**Note:** This property is relevant only in the publish/subscribe domain.

***XMSC\_WMQ\_TEMPORARY\_MODEL*****Data type:**

String

**Property of:**

ConnectionFactory

The name of the WebSphere MQ model queue from which a dynamic queue is created when the application creates an XMS temporary queue.

The default value of the property is SYSTEM.DEFAULT.MODEL.QUEUE.

This property is relevant only in the point-to-point domain.

***XMSC\_WMQ\_WILDCARD\_FORMAT*****Data type:**

System.Int32

**Property of:**

ConnectionFactory, Destination

This property determines which version of wildcard syntax is to be used.

When using publish/subscribe with IBM WebSphere MQ '\*' and '?' are treated as wildcards. Whereas '#' and '+' are treated as wildcards when using publish subscribe with IBM Integration Bus. This property replaces the XMSC\_WMQ\_BROKER\_VERSION property.

The valid values for this property are:

***XMSC\_WMQ\_WILDCARD\_TOPIC\_ONLY***

Recognizes the topic level wildcards only i.e. '#' and '+' are treated as wildcards. This value is same as XMSC\_WMQ\_BROKER\_V2.

***XMSC\_WMQ\_WILDCARD\_CHAR\_ONLY***

Recognizes the character wildcards only i.e. '\*' and '?' are treated as wildcards. This value is same as XMSC\_WMQ\_BROKER\_V1.

By default this property is set to XMSC\_WMQ\_WILDCARD\_TOPIC\_ONLY.

***XMSC\_WPM\_BUS\_NAME*****Data type:**

String

**Property of:**

ConnectionFactory and Destination

**Name used in a URI:**

busName

For a connection factory, the name of the service integration bus that the application connects to or, for a destination, the name of the service integration bus in which the destination exists.

For a destination that is a topic, this property is the name of the service integration bus in which the associated topic space exists. This topic space is specified by the `XMSC_WPM_TOPIC_SPACE` property.

If the property is not set for a destination, the queue or associated topic space is assumed to exist in the service integration bus to which the application connects.

By default, the property is not set.

### ***XMSC\_WPM\_CONNECTION\_PROTOCOL***

**Data type:**  
System.Int32

**Property of:**  
Connection

The communications protocol used for the connection to the messaging engine. This property is read-only.

The possible values of the property are as follows:

<b>Value</b>	<b>Meaning</b>
XMSC_WPM_CP_HTTP	The connection uses HTTP over TCP/IP.
XMSC_WPM_CP_TCP	The connection uses TCP/IP.

### ***XMSC\_WPM\_CONNECTION\_PROXIMITY***

**Data type:**  
System.Int32

**Property of:**  
ConnectionFactory

The connection proximity setting for the connection. This property determines how close the messaging engine that the application connects to must be to the bootstrap server.

The valid values of the property are as follows:

<b>Valid value</b>	<b>Connection proximity setting</b>
XMSC_WPM_CONNECTION_PROXIMITY_BUS	Bus
XMSC_WPM_CONNECTION_PROXIMITY_CLUSTER	Cluster
XMSC_WPM_CONNECTION_PROXIMITY_HOST	Host
XMSC_WPM_CONNECTION_PROXIMITY_SERVER	Server

The default value is `XMSC_WPM_CONNECTION_PROXIMITY_BUS`.

### ***XMSC\_WPM\_DUR\_SUB\_HOME***

**Data type:**  
String

**Property of:**  
ConnectionFactory

**Name used in a URI:**  
durableSubscriptionHome

The name of the messaging engine where all durable subscriptions for a connection or a destination are managed. Messages to be delivered to the durable subscribers are stored at the publication point of the same messaging engine.

A durable subscription home must be specified for a connection before an application can create a durable subscriber that uses the connection. Any value specified for a destination overrides the value specified for the connection.

By default, the property is not set.

This property is relevant only in the publish/subscribe domain.

### ***XMSC\_WPM\_HOST\_NAME***

**Data type:**

String

**Property of:**

Connection

The host name or IP address of the system that contains the messaging engine to which the application is connected. This property is read-only.

### ***XMSC\_WPM\_LOCAL\_ADDRESS***

**Data type:**

String

**Property of:**

ConnectionFactory

For a connection to a service integration bus, this property specifies the local network interface to be used, or the local port or range of local ports to be used, or both.

The value of the property is a string with the following format:

*[host\_name][(low\_port)[,high\_port]]*

The meanings of the variables are as follows:

***host\_name***

The host name or IP address of the local network interface to be used for the connection.

Providing this information is necessary only if the system on which the application is running has two or more network interfaces and you need to be able to specify which interface must be used for the connection. If the system has only one network interface, only that interface can be used. If the system has two or more network interfaces and you do not specify which interface must be used, the interface is selected at random.

***low\_port***

The number of the local port to be used for the connection.

If *high\_port* is also specified, *low\_port* is interpreted the lowest port number in a range of port numbers.

***high\_port***

The highest port number in a range of port numbers. One of the ports in the specified range must be used for the connection.

Here are some examples of valid values of the property:

JUPITER  
9.20.4.98  
JUPITER(1000)  
9.20.4.98(1000,2000)  
(1000)  
(1000,2000)

By default, the property is not set.

## ***XMSC\_WPM\_ME\_NAME***

**Data type:**

String

**Property of:**

Connection

The name of the messaging engine to which the application is connected. This property is read-only.

## ***XMSC\_WPM\_NON\_PERSISTENT\_MAP***

**Data type:**

System.Int32

**Property of:**

ConnectionFactory

The reliability level of nonpersistent messages that are sent using the connection.

The valid values of the property are as follows:

**Valid value**

XMSC\_WPM\_MAPPING\_AS\_DESTINATION

XMSC\_WPM\_MAPPING\_BEST\_EFFORT\_NON\_PERSISTENT

XMSC\_WPM\_MAPPING\_EXPRESS\_NON\_PERSISTENT

XMSC\_WPM\_MAPPING\_RELIABLE\_NON\_PERSISTENT

XMSC\_WPM\_MAPPING\_RELIABLE\_PERSISTENT

XMSC\_WPM\_MAPPING\_ASSURED\_PERSISTENT

**Reliability level**

Determined by the default reliability level specified for the queue or topic space in the service integration bus

Best effort nonpersistent

Express nonpersistent

Reliable nonpersistent

Reliable persistent

Assured persistent

The default value is XMSC\_WPM\_MAPPING\_EXPRESS\_NON\_PERSISTENT.

## ***XMSC\_WPM\_PERSISTENT\_MAP***

**Data type:**

System.Int32

**Property of:**

ConnectionFactory

The reliability level of persistent messages that are sent using the connection.

The valid values of the property are as follows:

**Valid value**

XMSC\_WPM\_MAPPING\_AS\_DESTINATION

**Reliability level**

Determined by the default reliability level specified for the queue or topic space in the service integration bus

Valid value	Reliability level
XMSC_WPM_MAPPING_BEST_EFFORT_NON_PERSISTENT	Best effort nonpersistent
XMSC_WPM_MAPPING_EXPRESS_NON_PERSISTENT	Express nonpersistent
XMSC_WPM_MAPPING_RELIABLE_NON_PERSISTENT	Reliable nonpersistent
XMSC_WPM_MAPPING_RELIABLE_PERSISTENT	Reliable persistent
XMSC_WPM_MAPPING_ASSURED_PERSISTENT	Assured persistent

The default value is XMSC\_WPM\_MAPPING\_RELIABLE\_PERSISTENT.

### ***XMSC\_WPM\_PORT***

**Data type:**

System.Int32

**Property of:**

Connection

The number of the port listened on by the messaging engine to which the application is connected. This property is read-only.

### ***XMSC\_WPM\_PROVIDER\_ENDPOINTS***

**Data type:**

String

**Property of:**

ConnectionFactory

A sequence of one or more endpoint addresses of bootstrap servers. The endpoint addresses are separated by commas.

A bootstrap server is an application server that is responsible for selecting the messaging engine to which the application connects. The endpoint address of a bootstrap server has the following format:

*host\_name:port\_number:chain\_name*

The meanings of the components of an endpoint address are as follows:

***host\_name***

The host name or IP address of the system on which the bootstrap server resides. If no host name or IP address is specified, the default is localhost.

***port\_number***

The number of the port on which the bootstrap server listens for incoming requests. If no port number is specified, the default is 7276.

***chain\_name***

The name of a bootstrap transport chain used by the bootstrap server. The valid values are as follows:

Valid value	Name of the bootstrap transport chain
XMSC_WPM_BOOTSTRAP_HTTP	BootstrapTunneledMessaging
XMSC_WPM_BOOTSTRAP_HTTPS	BootstrapTunneledSecureMessaging
XMSC_WPM_BOOTSTRAP_SSL	BootstrapSecureMessaging
XMSC_WPM_BOOTSTRAP_TCP	BootstrapBasicMessaging

If no name is specified, the default value is XMSC\_WPM\_BOOTSTRAP\_TCP.

If no endpoint address is specified, the default is localhost:7276:BootstrapBasicMessaging.

### **XMSC\_WPM\_SSL\_CIPHER\_SUITE**

**Data type:**

String

**Property of:**

ConnectionFactory

The name of the CipherSuite to be used on an SSL or TLS connection to a WebSphere Service Integration Bus messaging engine. The protocol used in negotiating the secure connection depends on the specified CipherSuite.

*Table 36. CipherSuite options for connection to a WebSphere Service Integration Bus messaging engine*

<b>Cipher suite</b>	<b>Protocol used</b>
SSL_RSA_WITH_NULL_MD5	SSLv3
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5	SSLv3
SSL_RSA_EXPORT_WITH_RC4_40_MD5	SSLv3
SSL_RSA_WITH_RC4_128_MD5	SSLv3
SSL_RSA_WITH_NULL_SHA	SSLv3
SSL_RSA_EXPORT1024_WITH_RC4_56_SHA	SSLv3
SSL_RSA_WITH_RC4_128_SHA	SSLv3
SSL_RSA_WITH_DES_CBC_SHA	SSLv3
SSL_RSA_EXPORT1024_WITH_DES_CBC_SHA	SSLv3
SSL_RSA_FIPS_WITH_DES_CBC_SHA	SSLv3
SSL_RSA_WITH_3DES_EDE_CBC_SHA	SSLv3
SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA	SSLv3
TLS_RSA_WITH_DES_CBC_SHA	TLSv1
TLS_RSA_WITH_3DES_EDE_CBC_SHA (deprecated)	TLSv1
TLS_RSA_WITH_AES_128_CBC_SHA	TLSv1
TLS_RSA_WITH_AES_256_CBC_SHA	TLSv1

**Note:** TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA and TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA CipherSuites are supported on Windows or Solaris only. (This is dictated by IBM Global Security Kit (GSKit).)

There is no default for this property. If you want to use SSL or TLS, you must specify a value for this property, otherwise your application is not able to connect successfully to the server.

### **XMSC\_WPM\_SSL\_FIPS\_REQUIRED**

**Note:** On UNIX, Linux®, and Windows, IBM MQ provides FIPS 140-2 compliance through the IBM Crypto for C (ICC) cryptographic module. The certificate for this module has been moved to the Historical status. Customers should view the [IBM Crypto for C \(ICC\) certificate](#) and be aware of any advice provided by NIST. A replacement FIPS 140-3 module is currently in progress and its status can be viewed by searching for it in the [NIST CMVP modules in process list](#).

**Data type:**

Boolean

**Property of:**

ConnectionFactory

The value of this property determines whether an application can or cannot use non-FIPS compliant cipher suites. If this property is set to true, only FIPS algorithms are used for the client-server connection. Setting the value of this property to TRUE prevents the application from using non-FIPS compliant cipher suites.

By default, the property is set to FALSE (that is, FIPS mode off).

***XMSC\_WPM\_SSL\_KEY\_REPOSITORY*****Data type:**

String

**Property of:**

ConnectionFactory

A path to the file that is the keyring file containing the public or private keys to be used in the secure connection.

Setting the keyring file property to the special value of XMSC\_WPM\_SSL\_MS\_CERTIFICATE\_STORE specifies the use the Microsoft Windows key database. Using the Microsoft Windows key database, which is found under **Control Panel > Internet Options > Content > Certificates**, removes the need for a separate key file database. Use of this constant on Windows x64 and other platforms is not permitted.

By default, the property is not set.

***XMSC\_WPM\_SSL\_KEYRING\_LABEL*****Data type:**

String

**Property of:**

ConnectionFactory

The certificate to be used when authenticating with the server. If no value is specified, the default certificate is used.

By default, the property is not set.

***XMSC\_WPM\_SSL\_KEYRING\_PW*****Data type:**

String

**Property of:**

ConnectionFactory

The password for the keyring file.

This property can be used as an alternative to using [XMSC\\_WPM\\_SSL\\_KEYRING\\_STASH\\_FILE](#) to configure the password for the keyring file.

By default, the property is not set.

***XMSC\_WPM\_SSL\_KEYRING\_STASH\_FILE*****Data type:**

String

**Property of:**

ConnectionFactory

The name of a binary file containing the password of the key repository file.

This property can be used as an alternative to using [XMSC\\_WPM\\_SSL\\_KEYRING\\_PW](#) to configure the password for the keyring file.

By default, the property is not set.

## ***XMSC\_WPM\_TARGET\_GROUP***

**Data type:**

String

**Property of:**

ConnectionFactory

The name of a target group of messaging engines. The nature of the target group is determined by the [XMSC\\_WPM\\_TARGET\\_TYPE](#) property.

Set this property if you want to restrict the search for a messaging engine to a subgroup of the messaging engines in the service integration bus. If you want your application to be able to connect to any messaging engine in the service integration bus, do not set this property.

By default, the property is not set.

## ***XMSC\_WPM\_TARGET\_SIGNIFICANCE***

**Data type:**

System.Int32

**Property of:**

ConnectionFactory

The significance of the target group of messaging engines.

The valid values of the property are as follows:

<b>Valid value</b>	<b>Meaning</b>
XMSC_WPM_TARGET_SIGNIFICANCE_PREFERRED	A messaging engine in the target group is selected if one is available. Otherwise, a messaging engine outside the target group is selected, provided it is in the same service integration bus.
XMSC_WPM_TARGET_SIGNIFICANCE_REQUIRED	The selected messaging engine must be in the target group. If a messaging engine in the target group is not available, the connection process fails.

The default value of the property is XMSC\_WPM\_TARGET\_SIGNIFICANCE\_PREFERRED.

## ***XMSC\_WPM\_TARGET\_TRANSPORT\_CHAIN***

**Data type:**

String

**Property of:**

ConnectionFactory

The name of the inbound transport chain that the application must use to connect to a messaging engine.

The value of the property can be the name of any inbound transport chain that is available in the application server that hosts the messaging engine. The following named constant is provided for one of the predefined inbound transport chains:

<b>Named constant</b>	<b>Name of transport chain</b>
XMSC_WPM_TARGET_TRANSPORT_CHAIN_BASIC	InboundBasicMessaging

The default value of the property is XMSC\_WPM\_TARGET\_TRANSPORT\_CHAIN\_BASIC.

## ***XMSC\_WPM\_TARGET\_TYPE***

**Data type:**

System.Int32

**Property of:**

ConnectionFactory

The type of the target group of messaging engines. This property determines the nature of the target group identified by the XMSC\_WPM\_TARGET\_GROUP property.

The valid values of the property are as follows:

<b>Valid value</b>	<b>Meaning</b>
XMSC_WPM_TARGET_TYPE_BUSMEMBER	The name of the target group is the name of a bus member. The target group is all the messaging engines in the bus member.
XMSC_WPM_TARGET_TYPE_CUSTOM	The name of the target group is the name of a user-defined group of messaging engines. The target group is all the messaging engines that are registered with the user-defined group.
XMSC_WPM_TARGET_TYPE_ME	The name of the target group is the name of a messaging engine. The target group is the specified messaging engine.

By default, the property is not set.

## ***XMSC\_WPM\_TEMP\_Q\_PREFIX***

**Data type:**

String

**Property of:**

ConnectionFactory

The prefix used to form the name of the temporary queue that is created in the service integration bus when the application creates an XMS temporary queue. The prefix can contain up to 12 characters.

The name of a temporary queue starts with the characters "\_Q" followed by the prefix. The remainder of the name consists of system generated characters.

By default, the property is not set, which means that the name of a temporary queue does not have a prefix.

This property is relevant only in the point-to-point domain.

## ***XMSC\_WPM\_TEMP\_TOPIC\_PREFIX***

**Data type:**

String

**Property of:**

ConnectionFactory

The prefix used to form the name of a temporary topic that is created by the application. The prefix can contain up to 12 characters.

The name of a temporary topic starts with the characters "\_T" followed by the prefix. The remainder of the name consists of system generated characters.

By default, the property is not set, which means that the name of a temporary topic does not have a prefix.

This property is relevant only in the publish/subscribe domain.

## ***XMSC\_WPM\_TOPIC\_SPACE***

**Data type:**

String

**Property of:**

Destination

**Name used in a URI:**

topicSpace

The name of the topic space that contains the topic. Only a destination that is a topic can have this property.

By default, the property is not set, which means that the default topic space is assumed.

This property is relevant only in the publish/subscribe domain.



## Notices

---

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document 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 document. The furnishing of this document does not grant 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:

Intellectual Property Licensing  
Legal and Intellectual Property Law  
IBM Japan, Ltd.  
19-21, Nihonbashi-Hakozakicho, Chuo-ku  
Tokyo 103-8510, 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 publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication 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.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation  
Software Interoperability Coordinator, Department 49XA  
3605 Highway 52 N  
Rochester, MN 55901  
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

#### COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

## Programming interface information

---

Programming interface information, if provided, is intended to help you create application software for use with this program.

This book contains information on intended programming interfaces that allow the customer to write programs to obtain the services of WebSphere MQ.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

**Important:** Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

## Trademarks

---

IBM, the IBM logo, [ibm.com](http://ibm.com)<sup>®</sup>, are trademarks of IBM Corporation, registered in many jurisdictions worldwide. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml). Other product and service names might be trademarks of IBM or other companies.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

This product includes software developed by the Eclipse Project (<http://www.eclipse.org/>).

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.







Part Number:

(1P) P/N: